cryptography

MDPI

Article

Garbled Quantum Computation

Elham Kashefi [1,2] and Petros Wallden [1,*]

[1] School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, UK; ekashefi@staffmail.ed.ac.uk

[2] CNRS LTCI, UPMC–LIP6, 4 Place Jussieu 75252, Paris CEDEX 05, France

* Correspondence: petros.wallden@ed.ac.uk; Tel.: +44-131-651-5631

Academic Editor: Kwangjo Kim
Received: 2 March 2017; Accepted: 30 March 2017; Published: 7 April 2017

Abstract: The universal blind quantum computation protocol (UBQC) enables an almost classical client to delegate a quantum computation to an untrusted quantum server (in the form of a garbled quantum circuit) while the security for the client is unconditional. In this contribution, we explore the possibility of extending the verifiable UBQC, to achieve further functionalities following the analogous research for classical circuits (Yao 1986). First, exploring the asymmetric nature of UBQC (the client preparing only single qubits, while the server runs the entire quantum computation), we present a "Yao"-type protocol for secure two-party quantum computation. Similar to the classical setting, our quantum Yao protocol is secure against a specious (quantum honest-but-curious) garbler, but in our case, against a (fully) malicious evaluator. Unlike the previous work on quantum two-party computation of Dupuis et al., 2010, we do not require any online-quantum communication between the garbler and the evaluator and, thus, no extra cryptographic primitive. This feature will allow us to construct a simple universal one-time compiler for any quantum computation using one-time memory, in a similar way to the classical work of Goldwasser et al., 2008, while more efficiently than the previous work of Broadbent et al., 2013.

Keywords: blind quantum computation; Yao's two-party computation protocol; one-time memory

1. Introduction

Future information and communication networks will consist of both classical and quantum devices, some of which are expected to be dishonest. These devices will have various different functionalities, ranging from simple routers to servers executing quantum algorithms. Within the last few years, anticipating this development has led to the formation and growth of the field of delegated quantum computing [1–6]. Among them is the universal blind quantum computation (UBQC) protocol of [3], which is developed based on the measurement-based quantum computation model (MBQC) [7] that appears as the most promising physical implementation for a networked architecture [8]. In the UBQC framework, the only quantum requirement for the client is the offline creation of random single qubit states, which is a currently available technology and has been demonstrated experimentally [9].

The MBQC model of computation can be viewed as a set of classical instructions steering a quantum computation performed on a highly entangled quantum state. The classical outcomes of the single-system measurements that occur during the computation are in general randomly distributed bits with no significance for the final output of the computation. This enables one to use relatively basic obfuscation techniques in order to prevent an untrusted operator (that implements an MBQC computation) from obtaining access to the true flow of information. This key observation has led to an entirely new approach to quantum verification that exploits cryptographic techniques [4,5,10–17]. The core idea is to encode simple trap computations within a target computation that is run on a remote device. This is done in such a way that the computation is not affected while, at the same

time, revealing no information to the server. The correctness of the overall computation is tested by verifying that the trap computations were done correctly. The latter, being significantly less costly, leads to efficient verification schemes. This approach of quantum verification has been recently used to obtain specific cryptographic primitives, such as quantum one-time program [6] and secure two-party quantum computation [18], which are also the main focus of this paper.

1.1. Our Contribution

We will explore two extensions of the verifiable universal blind quantum computing (VUBQC) protocols, which are built based on measurement-based quantum computation, in order to achieve new functionalities to be implemented in the quantum network setting [8]. The essential novelty of our approach is that the client-server setting allows different participants to have different quantum technological requirements. As a result, in our proposed two-party primitives, one party (garbler or sender) remains as classical as possible (with no need for quantum memory), while it is only the other party (evaluator or receiver) that requires access to a quantum computer. Moreover, the required offline initial quantum communication between the participants is also limited to the exchange of simple single quantum states that can be generated, for example, in any quantum key distribution network (as was proven recently in [19]). Finally, all of the utilised sub-protocols in our schemes could be reduced to the core protocols of offline preparation [19] and verifiable universal blind quantum computation [10,14] that are both proven to be composably secure in the abstract cryptography framework of [20]. For simplicity, we only present the stand-alone security proof for our protocols as we follow the framework of [21] and use simulation-based techniques. We present two new protocols:

1. In Section 3, we present a protocol for secure two-party quantum computation that we refer to as QYao. This protocol involves two parties with asymmetric roles that wish to securely compute any given unitary on a joint quantum input. Similar to the classical protocol of [22], in QYao, an honest-but-curious (formally defined for the quantum setting in [21]) client capable of preparing only random single qubit and performing constant depth classical computation "garbles" the entire unitary. The fully-malicious server/evaluator receives instructions from the garbler and inserts their input either using a (classical) oblivious transfer (OT) or by a quantum input insertion scheme secure against the honest-but-curious garbler. The evaluator performs the computation, extracts its own output qubits and returns the remaining output qubits to the garbler. After the garbler verifies the computation, the encryption keys of the evaluator output qubits are released. Unlike the classical setting, our proposed QYao protocol is interactive, but importantly, only classical online communication is required. The gain we have, on the other hand, is the boosted security since apart from the initial input exchange where classical OT is needed, the rest of the protocol is unconditionally secure. If one could replace the classical OT's in our QYao protocol with new primitives, such as the unconditionally secure relativistic primitives [23], the security would be extended to fully unconditional. In Section 4, we prove the security of the protocol in the ideal real-world paradigm using a simulation-based technique.
2. In Section 5, we follow the classical approach of [24] and make our QYao protocol non-interactive by using the classical hardware primitive "one-time memory" (OTM), which is essentially a non-interactive oblivious transfer. Using OTM, the need for initial OT calls, in our QYao protocol, is also removed, leading to a one-time universal compiler for any given quantum computation. Such one-time quantum programs can be executed only once, where the input can be chosen at any time. The one-time programs have a wide range of applications ranging from program obfuscation, software protection to temporary transfer of cryptographic ability [24]. Classically, the challenge in lifting the Yao protocol for two-party computation to a one-time program is addressing the issue of a malicious adversary. However, our QYao protocol is already secure against a malicious evaluator without any extra primitive or added overhead. Our quantum one-time program is therefore a straightforward translation of QYao. The same technique is applicable to essentially any cryptographic protocols developed in the measurement-based model

that require offline quantum preparation and online classical communications. In any such MBQC computation, there are exponentially many branches of computation (this expansion is the result of the non-deterministic nature of the intermediate single-qubits measurement that occurs during the computation); however, we prove that a single OTM (of constant size) per computation qubit suffices to make the QYao non-interactive. This is due to the fact that in any constant degree graph-state (the underlying entanglement resources for an MBQC computation), the flow of information acts locally. We prove that this is the case for the required entangled recourse in the verifiable UBQC introduced in [25] that is the basis of our QYao protocol. Hence, each classical message between parties, in the interactive case, depends only on the constant number of previous messages, and this allows us to remove this interaction using a simple constant-size OTM.

1.2. Related Works

Deriving new quantum cryptographic protocols for functionalities beyond the secure key exchange [26] is an active field of research (see a recent review for a summary [27]). In particular, our contributions are directly linked to the works on verifiable blind quantum computing (VUBQC) [4,5,10], secure two-party quantum computing [18,21] and quantum one-time program and quantum obfuscation [6,28]. The main focus in VUBQC research is the ability of a limited verifier to certify the correctness of the quantum computation performed by a remote server. Recently, many such protocols have been developed achieving different quantum technological requirement for the verifier or the prover [11,13,15,16,29]. We have used the optimal (from the verifier's point of view) VUBQC that has the additional property of a local and independent trap construction [25]. This property allows us to construct a simple yet generic scheme for the server (receiver, evaluator) input insertion and output extraction that could be applicable to other VUBQC schemes and that might lead to further properties not present in the scheme of [25].

Due to the impossibility results of [6,28], having extra primitives such as OTM is unavoidable in order to achieve program hiding. However, using the MBQC framework with the verification protocol of [25] leads to a simpler procedure for removing the classical interaction compared to the initial work that pioneered this approach [6]. Furthermore, due to the direct utilisation of OTMs instead of classical one-time programs, our scheme could be applicable to other VUBQC protocols that might emerge in the future. Another way of making protocols non-interactive is based on the security of other classical primitives, as was done in [30] using fully-homomorphic encryption. However, to apply this method for non-interactive secure two-party quantum computation, it is important to enable verifiability from the side of the sender, something naturally present in our scheme, but not necessarily in other schemes. Finally, the VUBQC framework, which naturally separates the classical and quantum part of the computation, allows us to construct a client-server scheme for secure two-party quantum computation that unlike the work of [21] removes the requirement of any extra cryptographic primitive, which was an open question in [21]. Here, we need to clarify that, while in the classical Yao with an honest-but-curious garbler the primitive of OT is required for the evaluator's input insertion, in our QYao, this is not the case. This is simply due to the fact that the notion of the quantum specious adversary that was formalised in [21] is more restrictive than classical honest-but-curious in certain cases (see Appendix B). Hence, instead of utilising a classical OT, one could simply devise a quantum communication scheme, as we present here, in order to achieve the same goal of secure input insertion against a (less powerful) adversarial garbler.

Another related issue is that the server-client setting can also be exploited to achieve simpler (implementation-wise) multiparty blind quantum computation protocols, where multiple clients use a single server to jointly and securely perform a quantum computation [31].

Finally, an interesting future direction is to try to view secure two-party quantum computation (2PQC) as a quantum game. The link of classical SMPCand game theory has been analysed, for example, in connection with the issue of fairness and rational players [32,33]. Moreover, the quantum settings

with multiple parties having competing interests has also been considered in a game-theoretic way (e.g., [34] and recently [35]), which leads to the question of using these techniques in the 2PQC case.

2. Preliminaries

2.1. Verifiable Universal Blind Quantum Computation

We will assume that the reader is familiar with the measurement-based quantum computation (MBQC) model [7] that is known to be the same as any gate teleportation model [36,37]. In this section, we introduce MBQC and use it to revise a blind quantum computation (the server performs computation without learning input/output or computation) [3] and a verifiable blind quantum computation (the client can also verify that the computation was performed correctly) [10] protocols. The general idea behind the MBQC model is: start with a large and highly entangled generic multiparty state (the resource state), and then, perform the computation by carrying out single-qubit measurements. To perform a desired quantum computation, each qubit should be measured in a suitable basis, and this basis is (partly) determined by some default measurement angles ϕ_i. There is an order in which the measurements should occur, which is determined by the flow of the computation (see also later), and the basis by which each qubit is measured generally depends on the outcomes of previous measurements (and the default measurement angle ϕ_i). The resource states used are known as graph states, as they can be fully determined by a given graph (see the details in [38]). A way to construct a graph state given the graph description is by assigning to each vertex of the graph a qubit initially prepared in the state $|+\rangle$ and for each edge of the graph to perform a controlled$-Z$ gate for the two adjacent vertices. For completeness, in Appendix A, we give the expression for the measurement angles of each qubit and an example of measurement pattern (graph state and default measurement angles ϕ_i) that implements each gate from a universal set of gates.

If one starts with a graph state where qubits are prepared in a rotated basis $|+_\theta\rangle = 1/\sqrt{2}(|0\rangle + e^{i\theta}|1\rangle)$ instead, then it is possible to perform the same computation with the non-rotated graph state by preforming measurements in a similarly rotated basis. This observation led to the formulation of the universal blind quantum computation (UBQC) protocol [3], which hides the computation in a client-server setting. Here, a client prepares rotated qubits, where the rotation is only known to them. The client sends the qubits to the server, as soon as they are prepared (hence, there is no need for any quantum memory). Finally, the client instructs the server to perform entangling operations according to the graph and to carry out single qubit measurements in suitable angles in order to complete the desired computation (where an extra randomisation r_i of the outcome of the measurements is added). During the protocol, the client receives the outcomes of previous measurements and can classically evaluate the next measurement angles. Due to the unknown rotation and the extra outcome randomisation, the server does not learn what computation they actually perform.

The UBQC protocol can be uplifted to a verification protocol where the client can detect a cheating server. To do so, the client for certain vertices (called dummies) sends states from the set $\{|0\rangle, |1\rangle\}$, which has the same effect as a Z-basis measurement on that vertex. In any graph state, if a vertex is measured in the Z-basis, it results in a new graph where that vertex and all of its adjacent edges are removed. During the protocol, the server does not know for a particular vertex if the client sent a dummy qubit (i.e., a qubit from $\{|0\rangle, |1\rangle\}$) or not. This enables the client to isolate some qubits (disentangled from the rest of the graph). Those qubits have fixed deterministic outcomes if the server followed the instructions honestly. The positions of those isolated qubits are unknown to the server, and the client uses them as traps to test that the server performs the quantum operations that are requested. This technique led to the first universal VUBQC protocol [10] and was subsequently extended to many other protocols depending on what optimised construction was used, what was the required quantum technology and which was the desired level of security. It is clear that the more (independent) traps we have within a graph, the higher the probability of detecting a deviation. In [25],

we gave a construction that, while maintaining an overhead that is linear in the size of the computation, introduced multiple traps and in particular of the same number as the computation qubits. We will be using that construction, not only for efficiency reasons, but because this construction is "local", and revealing to the server partial information about the graph does not compromise the security. This is important for our QYao protocol since, in this case, the server has the input and output and needs to know in which parts of the graph their input/output belongs. The construction is summarised below:

1. We are given a base-graph G that has vertices $v \in V(G)$ and edges $e \in E(G)$.
2. For each vertex v_i, we define a set of three new vertices $P_{v_i} = \{p_1^{v_i}, p_3^{v_i}, p_3^{v_i}\}$. These are called primary vertices.
3. Corresponding to each edge $e(v_i, v_j) \in E(G)$ of the base-graph that connects the base vertices v_i and v_j, we introduce a set of nine edges $E_{e(v_i,v_j)}$ that connect each of the vertices in the set P_{v_i} with each of the vertices in the set P_{v_j}.
4. We replace every edge in the resulted graph with a new vertex connected to the two vertices originally joined by that edge. The new vertices added in this step are called added vertices. This is the dotted triple-graph $DT(G)$.

We can see (Figure 1) that each vertex in the $DT(G)$ corresponds to either a vertex (for primary vertices) or an edge (for added vertices) of the base-graph. The precise edge/vertex of the base-graph to which each vertex $v \in DT(G)$ belongs is called base-location. The nice property of this graph is that one can reduce this graph to three copies of the base-graph by "breaking" some edges (which can be done using dummy qubits). Moreover, the choice of which vertex belongs to each of the three graphs is essentially independent (for different base-locations corresponding to vertices of the base-graph). With this construction we can eventually use one copy of the base-graph for the computation, while make multiple single traps from the two remaining copies (see Figure 1). The choice of which vertex belongs to which graph is called trap-colouring. This is a free choice made by the client, and the fact that the server is ignorant of the actual trap-colouring guarantees the security (see the details in [25]).

Definition 1 (Trap-colouring)**.** *We define trap-colouring to be an assignment of one colour to each of the vertices of the dotted triple-graph that is consistent with the following conditions.*

(i) *Primary vertices are coloured in one of the three colours, white or black (for traps) and green (for computation). There is an exception for input base-locations (see Step (v)).*
(ii) *Added vertices are coloured in one of the four colours white, black, green or red.*
(iii) *In each primary set P_v, there is exactly one vertex of each colour.*
(iv) *Colouring the primary vertices fixes the colours of the added vertices: added vertices that connect primary vertices of different colour are red; added vertices that connect primary vertices of the same colour get that colour.*
(v) *For input base-locations, instead of green, we have a blue vertex (but all other rules, including how to connect with the other vertices, apply in the same way as if it were green).*

For completeness, we give the basic verification protocol from [25] that we use.

Protocol 1 Verifiable universal blind quantum computation using the dotted triple graph (with fault-tolerant encoding); taken from [25].

We assume that a standard labelling of the vertices of the dotted triple-graph $DT(G)$, is known to both the client and the server. The number of qubits is at most $3N(3c+1)$ where c is the maximum degree of the base graph G.

• **Client's resources**

– Client is given a base graph G. The corresponding dotted graph state $|D(G)\rangle$ is generated by graph $D(G)$ that is obtained from G by replacing every edge with a new vertex connected to the two vertices originally joined by that edge.

– Client is given an MBQC measurement pattern $\mathbb{M}_{\text{Comp}}$ which: Applied on the dotted graph state $|D(G)\rangle$ performs the desired computation, in a fault-tolerant way, that can detect or correct errors fewer than $\delta/2$.

– Client generates the dotted triple-graph $DT(G)$, and selects a trap-colouring according to Definition 1 which is done by choosing independently the colours for each set P_v.

– Client for all red vertices will send dummy qubits and thus performs break operation.

– Client chooses the green graph to perform the computation.

– Client for the white graph sends dummy qubits for all added qubits a_w^e and thus generates white isolated qubits at each primary vertex set P_v. Similarly for the black graph the client sends dummy qubits for the primary qubits p_b^v and thus generates black isolated qubits at each added vertex set A_e.

– The dummy qubits position set D is chosen as defined above (fixed by the trap-colouring).

– A binary string $\mathbf{s}$ of length at most $3N(3c+1)$ represents the measurement outcomes. It is initially set to all zero's.

– A sequence of measurement angles, $\phi = (\phi_i)_{1\leq i\leq 3N(3c+1)}$ with $\phi_i \in A = \{0, \pi/4, \cdots, 7\pi/4\}$, consistent with $\mathbb{M}_{\text{Comp}}$. We define $\phi_i'(\phi_i, \mathbf{s})$ to be the measurement angle in MBQC, when corrections due to previous measurement outcomes $\mathbf{s}$ are taken into account (the function depends on the specific base-graph and its flow, see e.g., [3]). We also set $\phi_i' = 0$ for all of the trap and dummy qubits. The Client chooses a measurement order on the dotted base-graph $D(G)$ that is consistent with the flow of the computation (this is known to Server). The measurements within each set P_v, A_e of $DT(G)$ graph are ordered randomly.

– $3N(3c+1)$ random variables θ_i with value taken uniformly at random from A.

– $3N(3c+1)$ random variables r_i and $|D|$ random variable d_i with values taken uniformly at random from $\{0,1\}$.

– A fixed function $C(i, \phi_i, \theta_i, r_i, \mathbf{s}) = \phi_i'(\phi_i, \mathbf{s}) + \theta_i + \pi r_i$ that for each non-output qubit i computes the angle of the measurement of qubit i to be sent to the Server.

• **Initial Step**

– **Client's move:** Client sets all of the values in $\mathbf{s}$ to be 0 and prepares the input qubits as

$$|e\rangle = X^{x_1}Z(\theta_1) \otimes \ldots \otimes X^{x_l}Z(\theta_l)\,|I\rangle$$

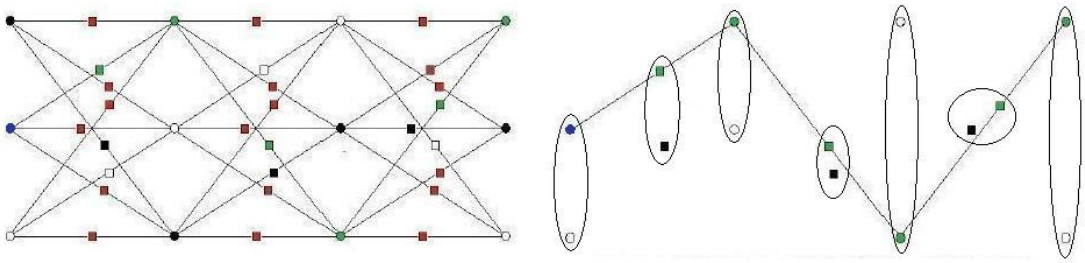
and the remaining qubits in the following form

$$\begin{array}{ll} \forall i \in D & |d_i\rangle \\ \forall i \notin D & \prod_{j\in N_G(i)\cap D} Z^{d_j}\,|+_{\theta_i}\rangle \end{array}$$

and sends the Server all of the $3N(3c+1)$ qubits in the order of the labelling of the vertices of the graph.

– **Server's move:** Server receives $3N(3c+1)$ single qubits and entangles them according to $DT(G)$.

– Continues to Protocol 2



(a) Trap-colouring (b) Computation graph and isolated white and black traps

Figure 1. Dotted triple graph. Circles: primary vertices with the base-location of the vertex of the base-graph; squares: added vertices with the base-location of edges of the base-graph. (**a**) Trap-colouring. Blue: input qubits; green: gate qubits; white/black: trap qubits; red: wiring qubits. The client chooses the colours randomly for each vertex with the base-location of vertex of the base-graph and prepares each qubit individually before sending them one by one to the server to entangle them according to the generic construction. (**b**) After entangling, the breaking operation defined by the wiring qubits will reduce the graph in (a) to the computation graph and for each vertex a corresponding trap/tag qubits.

Protocol 2 Continuing from Protocol 1: VUBQC with DT(G).

• **Step** i : $1 \leq i \leq 3N(3c+1)$
– **Client's move:** Client computes the angle $\delta_i = C(i, \phi_i, \theta_i, r_i, \mathbf{s})$ and sends it to the Server.
– **Server's move:** Server measures qubit i with angle δ_i and sends the Client the result b_i.
– **Client's move:** Client sets the value of s_i in $\mathbf{s}$ to be $b_i + r_i$.

• **Final Step:**
– **Server's move:** Server returns the last layer of qubits (output layer) to the Client.

• **Verification**
– After obtaining the output qubits from the Server, the Client measures the output trap qubits with angle $\delta_t = \theta_t + r_t\pi$ to obtain b_t.
– Client accepts if $b_i = r_i$ for all of the white (primary) and black (added) trap qubits i.

2.2. Two-Party Quantum Protocols

The impossibility of achieving unconditionally secure two-party cryptographic protocols has led to the definition and use of simpler hardware or software primitives to form the basis for the desired functionalities. A one out of n oblivious transfer (OT) is a two-party protocol where one party (Alice) has input n messages $(x_1, \cdots, x_n)$ and the other party (Bob) inputs a number $c \in \{1, \cdots, n\}$ and receives the message x_c with the following guarantees: Bob learns nothing about the other messages $x_i | i \neq c$, and Alice is "oblivious" (does not know) about which message Bob obtained (i.e., does not know the value c) [39]. A hardware token that implements a non-interactive OT is called one-time memory (OTM) [24]. We will utilise a one out of n OTM, where again Alice stores in the OTM n strings $(x_1, \cdots, x_n)$, Bob specifies a value c and the OTM reveals x_c to Bob and then self-destructs; i.e., the remaining strings $x_i | i \neq c$ are lost forever.

The first paper that studied secure two-party quantum computation is [21], and we follow their notations and conventions. We have two parties A, B with registers $\mathcal{A}, \mathcal{B}$ and an extra register $\mathcal{R}$ with $\dim \mathcal{R} = (\dim \mathcal{A} + \dim \mathcal{B})$. The input state is denoted $\rho_{in} \in D(\mathcal{A} \otimes \mathcal{B} \otimes \mathcal{R})$, where $D(\mathcal{A})$ is the set of all possible quantum states in register $\mathcal{A}$. We also denote with $L(\mathcal{A})$ the set of linear mappings from $\mathcal{A}$ to itself, and the superoperator $\phi : L(\mathcal{A}) \to L(\mathcal{B})$ that is completely positive and trace preserving is called a quantum operation. We denote $\mathbb{I}_{\mathcal{A}}$ the identity operator in register $\mathcal{A}$. The ideal output is then given by $\rho_{out} = (U \otimes \mathbb{I}_{\mathcal{R}}) \cdot \rho_{in}$, where for simplicity, we write $U \cdot \rho$ instead of $U\rho U^{\dagger}$. For two states ρ_0, ρ_1, we denote the trace norm distance $\Delta(\rho_0, \rho_1) := \frac{1}{2}\|\rho_0 - \rho_1\|$. If $\Delta(\rho_0, \rho_1) \leq \epsilon$, then any process applied on ρ_0 behaves as for ρ_1, except with probability at most ϵ.

Definition 2 (Taken from [21]). *An n-step two-party strategy with oracle calls is denoted* $\Pi^O = (A, B, O, n)$*:*

1. *input spaces* $\mathcal{A}_0, \mathcal{B}_0$ *and memory spaces* $\mathcal{A}_1, \cdots, \mathcal{A}_n$ *and* $\mathcal{B}_1, \cdots, \mathcal{B}_n$
2. *n-tuple of quantum operations* $(L_1^A, \cdots, L_n^A)$ *and* $(L_1^B, \cdots, L_n^B)$ *such that* $L_i^A : L(\mathcal{A}_{i-1}) \to L(\mathcal{A}_i)$ *and similarly for* L_i^B*.*
3. *n-tuple of oracle operations* $(\mathcal{O}_1, \cdots, \mathcal{O}_n)$ *where each oracle is a global operation for that step,* $\mathcal{O}_i : L(\mathcal{A}_i \otimes \mathcal{B}_i) \to L(\mathcal{A}_i \otimes \mathcal{B}_i)$

The trivial oracle is a communication oracle that transfers some quantum register from one party to another. Protocols that have only such oracles are called the bare model. Other oracles include calls to other cryptographic primitives. The quantum state in each step of the protocol is given by:

$$\begin{aligned} \rho_1(\rho_{in}) &:= (\mathcal{O}_1 \otimes \mathbb{I})(L_1^A \otimes L_1^B \otimes \mathbb{I})(\rho_{in}) \\ \rho_{i+1}(\rho_{in}) &:= (\mathcal{O}_{i+1} \otimes \mathbb{I})(L_{i+1}^A \otimes L_{i+1}^B \otimes \mathbb{I})(\rho_i(\rho_{in})) \end{aligned} \tag{1}$$

The security definitions are based on the ideal functionality of two-party quantum computation (2PQC) that takes a joint input $\rho_{in} \in \mathcal{A}_0 \otimes \mathcal{B}_0$, obtains the state $U \cdot \rho_{in}$ and returns to each party their corresponding quantum registers. A protocol Π_U^O implements the protocol securely, if no possible adversary in any step of the protocol can distinguish whether they interact with the real protocol or with a simulator that has only access to the ideal functionality. When a party is malicious, we add the notation "$\sim$", e.g., $\tilde{A}$.

Definition 3 (Simulator). $\mathcal{S}(\tilde{A}) = \langle (\mathcal{S}_1, \cdots, \mathcal{S}_n), q \rangle$ *is a simulator for adversary* $\tilde{A}$ *in* Π_U^O *if it consists of:*

1. *operations where* $\mathcal{S}_i : L(\mathcal{A}_0) \to L(\tilde{\mathcal{A}}_i)$*,*
2. *sequence of bits* $q \in \{0,1\}^n$ *determining if the simulator calls the ideal functionality at step i (*$q_i = 1$ *calls the ideal functionality).*

For other adversaries, the simulator is defined analogously.
Given input ρ_{in}, the simulated view for step i is defined as:

$$\nu_i(\tilde{A}, \rho_{in}) := \mathrm{Tr}_{\mathcal{B}_0}\left((\mathcal{T}_i \otimes \mathbb{I})(U^{q_i} \otimes \mathbb{I}) \cdot \rho_{in}\right) \tag{2}$$

and similarly for the other party.

Definition 4 (Privacy). *We say that the protocol is* δ*-private if for all adversaries and for all steps i:*

$$\Delta(\nu_i(\tilde{A}, \rho_{in}), \mathrm{Tr}_{\mathcal{B}_i}(\tilde{\rho}_i(\tilde{A}, \rho_{in}))) \leq \delta \tag{3}$$

where $\tilde{\rho}_i(\tilde{A}, \rho_{in})$ *the state of the real protocol with corrupted party* $\tilde{A}$*, at step i.*

In classical cryptography, a type of adversary commonly considered is the "honest-but-curious". This adversary follows the protocol, but also keeps records of its actions and attempts to learn from those more than what it should. This type of weak adversary has been proven very useful in many protocols, since it typically constitutes the first step in constructing protocols secure against more powerful (even fully malicious) adversaries.

Since quantum states cannot be copied, one cannot have a direct analogue of honest-but-curious. Instead, we have the notion of specious adversaries [21], where they can deviate as they wish, but in every step, if requested, should be able to reproduce the honest global state by acting only on their subsystems. More formally:

Definition 5 (Specious). *An adversary $\tilde{A}$ is ϵ-specious if there exists a sequence of operations $(\mathcal{T}_1, \cdots, \mathcal{T}_n)$, where $\mathcal{T}_i : L(\tilde{\mathcal{A}}_i) \to L(\mathcal{A}_i)$, such that:*

$$\Delta\left((\mathcal{T}_i \otimes \mathbb{I})(\tilde{\rho}_i(\tilde{A}, \rho_{in})), \rho_i(\rho_{in})\right) \leq \epsilon \tag{4}$$

Note that this (standard) definition of specious adversary allows for different interpretations that lead to stronger and weaker versions of the adversary. These subtleties are explained in Appendix B. Here, we stress that we take the weaker notion that takes Equation (4) in the stricter sense, while in [21], the authors implicitly used a stronger version. This difference led to our paper evading certain impossibility results mentioned in [21]. A detailed discussion of this is given in Appendix B.

A 2PQC protocol needs to be (by definition) private. Moreover, it may have two extra properties, verification and fairness. Verification means that each party, when they receive their output, not only is sure that nothing leaked about its input, but also it can know whether the outcome it received is correct or corrupted. Fairness is the extra property that no party should be able to obtain its output and after that cause an abort (or a corruption) for the other party. Even in classical 2PC, it is impossible to have fairness against fully-malicious adversaries (without any extra assumption). We consider two-party protocols $\Pi^0 = (C, S, O, n)$ where C denotes the client and S the server.

3. Secure Two-Party Quantum Computation

The first extension of VUBQC that we will explore refers to its use in order to construct a 2PQC protocol similar to the classical Yao protocol of [22]. We refer to this protocol as QYao, where the sender-garbler is called the client and the receiver-evaluator is called the server. As in the Yao protocol, we assume that the client, when adversarial, is specious; however, we make no such assumption for the server, which is assumed to be fully malicious. In [22], the server needs to use OT in order to insert its input. We, instead, use a scheme to insert the quantum input that requires no such functionality. This is possible because we make the assumption that the client is specious, and in specific situations, a specious adversary is weaker than honest-but-curious (see Appendix B.1). For the specific case of classical input/output, one can modify our protocol to make it secure against the classical honest-but-curious adversary by replacing the input injection subprotocol (see below) with OT.

Our QYao protocol provides a one-time verifiable 2PQC similar to the classical setting as recently shown for the original Yao protocol in [40]. The speciousness of the client restricts any possible deviations on their side, while a malicious server would be detected through the hidden traps of the VUBQC protocol. The blindness property of VUBQC also guarantees that the server learns nothing before the client is certain that there was no deviation and returns the suitable keys for the decryption of the output.

The major difference that the QYao protocol has in comparison with regular VUBQC [25] is that the server needs to provide (part of) the input and at the end keeps (part of) the output. There are multiple ways to modify the VUBQC protocol; we use a direct approach at the cost of having two rounds of quantum communication during input preparation and two rounds of quantum communication during output read-out. However, there is no quantum communication during the evaluation stage. Note that if we restrict the protocol to classical input/output, we would avoid all quantum communication apart from the initial and offline sending of pre-rotated qubits from the client to the server (but we would then need OT, as in the classical Yao, for the server to insert its input).

3.1. Server's Input Injection

The qubits composing the DT(G)(see Figure 1) are prepared by the client. This is crucial in order to ensure that the server is blind about the positions of the traps. However, the server should somehow insert their input in the DT(G). For simplicity, we present here the case that there is a single qubit input, but it generalises trivially. To do so, the server encrypts its input using secret keys $(m_{z,i}, m_{x,i})$ and sends the state $X^{m_{x,i}} Z^{m_{z,i}} |\Phi_S\rangle$ to the client, to insert it randomly in the corresponding base-location.

The state $|\Phi_S\rangle$ is assumed to be pure as we have included its purification in the hands of the server. The client encrypts further the server's input by applying an extra random $Z(\theta'_i)$ and an $X^{x'_i}$ correction to obtain the state:

$$X^{x'_i} Z(\theta'_i) X^{m_{x,i}} Z^{m_{z,i}} |\Phi_S\rangle \tag{5}$$

This extra encryption is needed in order to hide the future actual measurement of the input qubit (to be performed by the server) and ensure that no information about the position of the traps is leaked. Trap hiding also requires the client to return two extra qubits for the input insertion (see Figure 2) that will belong to trap graphs (called the white-graph and black-graph). The white qubit (which is a trap qubit) is prepared in a state $|\theta_k\rangle$, while the black qubit (which is a dummy qubit) is prepared in state $|d_j\rangle$. The three qubits will be randomly permuted by the client so that the server does not know which qubit is which. A similar procedure (with no communication from the server) is applied for client's input qubits, as well as for all of the qubits corresponding to the gate computation (see Figure 2).

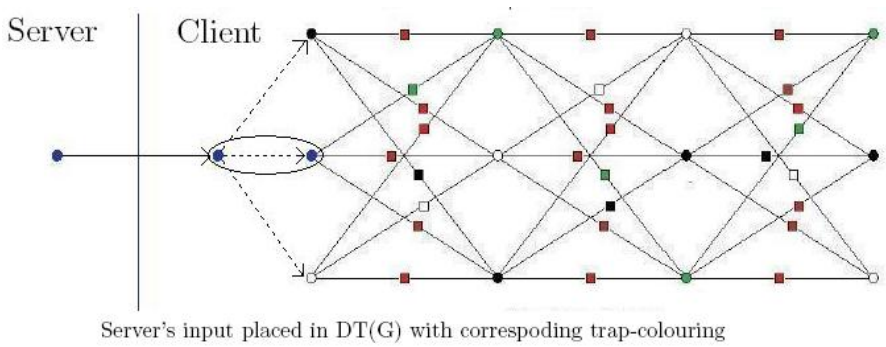


Figure 2. The server gives its input (blue), and the client chooses (randomly) where in the input base-location to place the input. The random choice is highlighted. The trap-colouring is filled correspondingly, after the random choice is made.

Finally, after the server has received all of the qubits, he/she announces the secret keys $(m_{x,i}, m_{z,i})$ for each input i to the client, so that the client can update the encryption for these qubits and have $(x_i, \theta_i) := \left(x'_i + m_{x,i}, (-1)^{m_{x,i}} \theta'_i + \pi m_{z,i}\right)$. With the updated encryption, the client computes the suitable measurement angles δ_i. It is worth pointing out that the key releasing step from server to client could be avoided by using classical OT to compute the measurement angles as a function of the secret parameters of the server $\delta_i(m_{x,i}, m_{z,i})$ for the first two layers (that have dependency on $m_{x,i}, m_{z,i}$). To construct protocols dealing with a malicious client, the use of OTs may be necessary. For our case, however, the client is (weak) specious, and using OTs is not necessary.

3.2. Server's Output Extraction

In the standard VUBQC protocol, the server returns all of the output qubits to the client. The client measures the final layer's traps to check for any deviation and then obtains the output of the computation by decrypting the output computation qubits using their secret keys. In the 2PQC, part of the output (of known base-locations) should remain in the hands of the server. This, however, would not allow the client to check for the related traps (that could have effects on other output qubits). Similar to the input injection, the solution is obtained via an extra layer of encryption by the server followed by a delayed key releasing. The server will encrypt using two classical bits (k_x, k_z): $E_k(|\psi\rangle) = X^{k_x} Z^{k_z} |\psi\rangle$, all of the output qubits that correspond to the server's output base-locations and then return all of the output qubits to the client. Due to the encryption, the client obtains no information about the output of the server, while it now has access to all of the final trap qubits for the verification. The client returns to the server only the computation qubits corresponding to the server's output base-locations, while keeping the traps and all of the qubits from other base-locations.

The server reveals their keys (note that the client, being specious, has not kept the computation output qubits, because it would be impossible to reconstruct the ideal state by acting only on their systems). The client checks all of the traps and, if all of them are correct, reveals the final computation output keys to the server (θ_i, r_i). The server undoes the padding to reveal their output.

3.3. The QYao Protocol

We combine Protocols 3 and 4 with the core VUBQC, Protocol 1, to obtain a secure two-party protocol to compute unitaries given as Protocol 5. An illustration of a simple example is given in Appendix C.1.

Protocol 3 Server's input injection

Setting:

• The server has input $|\Phi_S\rangle$ that corresponds to specific positions I_S of the input layer of the base-graph. For each i qubit, server chooses pair of secret bits $(m_{x,i}, m_{z,i})$.

Instructions:

1. The server sends to the client the states $X^{m_{x,i}} Z^{m_{z,i}} |\Phi_S\rangle$ for all $i \in I_S$.
2. The client prepares all of the states of the DT(G) as in Protocol 1 (see [25]) apart from the computation qubits of the server's input.
3. The client chooses at random x'_i, θ'_i for each of the server's input and obtains the states: $X^{x'_i} Z(\theta'_i) X^{m_{x,i}} Z^{m_{z,i}} |\Phi_S\rangle$.
4. The client mixes the computation qubit of the server's input, with a dummy and a trap qubit to return the three qubits of server's input base-locations.
5. The server, after receiving the qubits, returns to the client the secret bits $(m_{x,i}, m_{z,i})$ for all i of their input.
6. The client computes $x_i := x'_i + m_{x,i}$ and $\theta_i := (-1)^{m_{x,i}} \theta'_i + \pi m_{z,i}$ and uses these (x_i, θ_i) for computing the measurement angles as in Protocol 1.

Outcome:

• The server receives a DT(G), where the quantum input is the joint input of the client and server and the related measurement angles instructions perform the desired unitary operation, if the client is honest.

Protocol 4 Server's output extraction.

Setting:

• The server has the state $\mathrm{Tr}_C(|\psi(f)\rangle)$ at the corresponding positions O_S of their output base-locations of the DT(G). For each i qubit, server chooses pair of secret bits $(m_{x,i}, m_{z,i})$.

Instructions:

1. The server sends to the client the states $X^{m_{x,i}} Z^{m_{z,i}} \mathrm{Tr}_C(|\psi(f)\rangle)$ for all $i \in O_S$.
2. The client keeps the qubits that correspond to traps and dummies in the final layer of server's output, while returns the computation qubits of server's output.
3. The server returns to the client $(m_{x,i}, m_{z,i})$.
4. The client checks the traps and if correct returns to the server the final layer paddings for their output locations $(\theta_i, r_i, r_{j<i})$ and $(m_{x,i}, m_{z,i})$ (that the server sent earlier).
5. The server undoes the final layer paddings and obtains the output.

Outcome:

• The server obtains the computation qubits of their output base-locations unpadded $\mathrm{Tr}_C(U(|\Psi\rangle_C, |\Phi\rangle_S))$.

Theorem 1 (Correctness). *If both the client and server follow the steps of Protocol 5, then the output is correct, and the computation is accepted.*

Proof. If the client and server follow Protocol 5, after Step 1, where the server injected its input, we are in exactly the same situation as in Protocol 1, with (overall) input $\rho_{in} = |\Psi\rangle_C \otimes |\Phi\rangle_S$.

During Step 2, the client and server run exactly Protocol 1, and the correctness follows from the correctness of that protocol; the proof is given in [25]. The positions of dummies result in having isolated traps measured on the correct basis, and thus, there is never an abort. From the remaining qubits, one copy of the dotted base-graph G, where the measurement pattern $\mathbb{M}_{\text{Comp}}$ is applied, results in the state $E_{k^S,k^C}(U \cdot \rho_{in})$. This is the honest global final state, encrypted with keys of both the client (secret parameters) and the server (the padding $(m_{x,i}, m_{z,i})$ from the output extraction protocol, which applies to server's output registers only).

Protocol 5 Secure 2PQC-QYao.

Input:

- The client and server know the unitary operator U that they wish to compute. The client has a description of U in MBQC using resource $|G\rangle$, and maps it to the DT(G) (see Protocol 1 and [25]). For each qubit the client knows the angles ϕ_i and dependencies. The base-locations of the inputs and outputs of both parties are public. The inputs of client and server are denoted correspondingly as $|\Psi\rangle_C$ and $|\Phi\rangle_S$.

Output:

- The client receives the subset C_o of the output qubits of the final quantum state $U(|\Psi\rangle_C, |\Phi\rangle_S)$.
- The server receives the subset S_o of the output qubits of the final quantum state $U(|\Psi\rangle_C, |\Phi\rangle_S)$.

The protocol

1. Client sends all qubits of the DT(G) (choosing the random parameters r_i, θ_i, d_i and trap-colouring), where for base-locations that corresponds to the server's input, the input injection Protocol 3 is used.
2. Server and client follow the verification protocol 1 until the step that the output is returned.
3. Client and server interact according to Protocol 4 so that the server extracts their own output.

During Step 3, the client keeps the registers of its output, while returning the registers of the server's output. The protocol finishes with the server returning its keys $(m_{x,i}, m_{z,i})$ to the client (to check for traps), while the client returns the keys (secret parameters) involved with the final decryption of the server's output registers. □

Since the protocol we give is generic for any quantum computation, the complexity of the 2PQC protocol with respect to the input size depends on the specific computation/algorithm used, and it is against this that it should be compared. At this point, we should stress that the number of qubits required for the 2PQC protocol is exactly the same as those to perform the same computation using VUBQC where all input and output are of the client. From [25], the number of qubits and classical messages to be exchanged is linear to the number of qubits required to perform the quantum computation using MBQC (i.e., without blindness or verification). Finally, the only extra "cost" for 2PQC is the extra quantum communication (and related classical exchanges) during the input injection and the output extraction. This extra communication is simply one extra qubit communication per server input qubits and similarly one extra qubit communication per server output qubit.

Due to the simple composition of input injection and output extraction, the verification property of our QYao is directly inherited from the VUBQC. We prove this first, before presenting and proving the main privacy property of the QYao protocol in the next section (which exploits the verifiability).

Definition 6. *We define a 2PQC protocol to be* ***ϵ-verifiable for the client****, if for any (potentially malicious) server, the probability of obtaining a corrupt output and not an abort is bounded by ϵ. The output of the real protocol with malicious server $\tilde{S}$ is $\tilde{\rho}(\tilde{S}, \rho_{in})$, and we have:*

$$\Delta(\tilde{\rho}(\tilde{S}, \rho_{in}), \rho_{ideal}(\rho'_{in})) \quad \leq \quad \epsilon \tag{6}$$

where

$$\rho_{ideal}(\rho_{in}) \quad := \quad p_{ok}(\mathbb{I}_{\mathcal{H}_C} \otimes \mathcal{C}_{\mathcal{H}_S}) \cdot U \cdot (\rho_{in}) + (1 - p_{ok})(|fail\rangle \langle fail|)$$

and $\mathcal{C}_{\mathcal{H}_S}$ is the deviation that acts on the server's systems after it receives its outcome (a CP-map, but can be purified if we include ancilla). Furthermore, $\rho'_{in} = (\mathbb{I}_{\mathcal{H}_C} \otimes D_{\mathcal{H}_S}) \cdot \rho_{in}$ is an initial state compatible with the client's input, with $D_{\mathcal{H}_S}$ the deviation on the input by the server.

We should note that the server can always choose any input from its side, and the security of the protocol is defined with respect to this "deviated" input. Moreover, since the deviation $\mathcal{C}_{\mathcal{H}_S}$ is performed at the final step of the protocol, we also have that the global state (before that deviation, i.e., at step $n-1$) obeys:

$$\Delta(\tilde{\rho}^{n-1}(\tilde{S}, \rho_{in}), \rho^{n-1}_{ideal}(\rho'_{in})) \quad \leq \quad \epsilon \tag{7}$$

where

$$\rho^{n-1}_{ideal}(\rho_{in}) \quad := \quad p_{ok} U \cdot (\rho_{in}) + (1 - p_{ok})(|fail\rangle \langle fail|)$$

Theorem 2 (ϵ-verification for the client)**.** *Protocol 5 is ϵ-verifiable for the client, where $\epsilon = \left(\frac{8}{9}\right)^d$ and $d = \lceil \frac{\delta}{2(2c+1)} \rceil$, c is the maximum degree of the base graph and δ is the number of errors tolerated on the base graph G.*

Proof. In order to prove the verifiability for the client, we assume that the client is honest, while the server is malicious.

During Step 1 of Protocol 5, the server sends his/her input and also gives the keys of the one-time padding encryption $(m_{x,i}, m_{z,i})$ from the input injection phase. It follows that any deviation on these affects only the computation qubits of the server's input base-location; in other words, it results in a state: $\rho'_{in} = (\mathbb{I}_{\mathcal{H}_C} \otimes D_{\mathcal{H}_S}) \cdot \rho_{in}$

During Step 2, the protocol proceeds exactly as the VUBQC, Protocol 1, with the only difference that the qubits with the base-location of the server's output have an extra encryption with keys known to the server. This means that the client delays the measurement of the traps in those base-locations, until the next step, where it receives from the server these keys. Note, however, that the client can already check all of the past traps and the ones corresponding to base-location of the clients' output.

In Step 3, the server returns the keys $(m_{x,i}, m_{z,i})$ for all of its output base-locations qubit and also receives the computation qubits of those base-locations encrypted with both the client's and server's keys. Any deviation by the server at this stage either returns different (wrong) keys to the client or acts only on the server's output. Returning wrong keys increases the chance of an abort, but in any case, it is equivalent to this deviation happening before the return of the output to the server. The only remaining, extra deviation is a possible deviation acting on the computation output qubits of the server, i.e., a deviation of the form $(\mathbb{I}_{\mathcal{H}_C} \otimes \mathcal{C}_{\mathcal{H}_S})$.

It follows that this protocol has the same verification properties as Protocol 1, with the modified input and (server's part) of the output given by Equation (6). From [25], we have that Protocol 1 is ϵ-verifiable with $\epsilon = \left(\frac{8}{9}\right)^d$ and $d = \lceil \frac{\delta}{2(2c+1)} \rceil$, and this completes the proof. □

It is worth mentioning that this verification property essentially restricts the server to behave similarly as a specious adversary, with the extra ability to abort the protocol.

4. Proof of the Privacy of the QYao Protocol

Recall that we defined a protocol to be secure if no possible adversary in any step of the protocol can distinguish whether it interacts with the real protocol or with a simulator that has access only to the ideal functionality (see Definition 4).

Theorem 3. *The QYao protocol, Protocol 5, which is ϵ_1-verifiable for the client, is $O(\sqrt{\epsilon_2})$-private against an ϵ_2-specious client and ϵ_1-private against a malicious server.*

To prove the theorem, we need to introduce simulators for each step of the protocol and each possible adversary. Below, we define those simulators and prove that they are as close to the real protocol as requested from Theorem 3. Since in our setting, the two parties have different roles and maliciousness, we consider the simulators for each party separately.

4.1. Client's Simulators

The client is ϵ-specious, and this means that for each step i, there exist a map $\mathcal{T}_i : L(\mathcal{H}_{\tilde{V}_i}) \to L(\mathcal{H}_{V_i})$, such that:

$$\Delta\left((\mathcal{T}_i \otimes \mathbb{I})(\tilde{\rho}_i(\tilde{V}, \rho_{in})), \rho_i(\rho_{in})\right) \leq \epsilon \tag{8}$$

Following the proof of the "rushing lemma" of [21], we obtain a similar lemma for each step of the protocol, where it shows that there is no extra information stored in the ancillas of the specious adversary:

Lemma 1 (No-extra information)**.** *Let $\Pi_U = (A, B, n)$ be a correct protocol for the two-party evaluation of U. Let $\tilde{A}$ be any ϵ-specious adversary. Then, there exists an isometry $T_i : \tilde{A}_i \to A_i \otimes \hat{A}$ and a (fixed) mixed state $\hat{\rho}_i \in D(\hat{A}_i)$, such that for all joint input states ρ_{in},*

$$\Delta\left((T_i \otimes \mathbb{I})(\tilde{\rho}_i(\tilde{A}, \rho_{in})), \hat{\rho}_i \otimes \rho_i(\rho_{in})\right) \leq 12\sqrt{2\epsilon} \tag{9}$$

where $\rho_i(\rho_{in})$ is the state in the honest run and $\tilde{\rho}_i(\tilde{A}, \rho_{in})$ is the real state (with the specious adversary $\tilde{A}$).

Proof. This proof follows closely the proof of the "rushing lemma" of [21], where one can find further details. For simplicity, we assume pure ρ_{in} (where it holds in general by convexity). Consider two different states $|\psi_1\rangle, |\psi_2\rangle$ in $\mathcal{A}_0 \otimes \mathcal{B}_0 \otimes \mathcal{R}$, and we extend the space $\mathcal{R}' = \mathcal{R} \otimes \mathcal{R}_2$ with $\mathcal{R}_2 = \text{span}\{|0\rangle, |1\rangle\}$. We define the state $|\psi\rangle = 1/\sqrt{2}(|\psi_1\rangle |1\rangle + |\psi_2\rangle |2\rangle)$. Due to the speciousness of $\tilde{A}$ and using Ulmann's theorem, there is an isometry $T_i : \tilde{\mathcal{A}}_i \to \mathcal{A}_i \otimes \hat{\mathcal{A}}$ and a state $\tilde{\rho} \in D(\hat{\mathcal{A}})$, such that:

$$\Delta\left((T_i \otimes \mathbb{I}) \cdot \tilde{\rho}_i(\tilde{A}, \psi), \tilde{\rho} \otimes \rho_i(\psi)\right) \leq 2\sqrt{2\epsilon} \tag{10}$$

The state $\tilde{\rho}$ in general is not independent of the input ψ, so there are a few more steps required. By noting that the projection and partial trace are trace non-increasing maps (by projecting on the $|1\rangle$ subspace and tracing out the $\mathcal{R}'$ subspace), we obtain:

$$\Delta\left((T_i \otimes \mathbb{I}) \cdot \tilde{\rho}_i(\tilde{A}, \psi_1), \tilde{\rho} \otimes \rho_i(\psi_1)\right) \leq 4\sqrt{2\epsilon} \tag{11}$$

and similarly for ψ_2. We repeat the same using ψ_1, ψ_3 and with initial state $|\psi'\rangle = 1/\sqrt{2}(|\psi_1\rangle |1\rangle + |\psi_3\rangle |2\rangle)$ and obtain:

$$\Delta\left((T_i \otimes \mathbb{I}) \cdot \tilde{\rho}_i(\tilde{A}, \psi_1), \tilde{\rho}' \otimes \rho_i(\psi_1)\right) \leq 4\sqrt{2\epsilon} \tag{12}$$

By the triangular inequality, we get $\Delta(\tilde{\rho}, \tilde{\rho}') \leq 8\sqrt{2\epsilon}$. This means that for any state $|\chi\rangle$, there exists a state $\hat{\rho} \in \hat{\mathcal{A}}$ with $\Delta(\tilde{\rho}, \hat{\rho}) \leq 8\sqrt{2\epsilon}$, such that:

$$\Delta\left((T_i \otimes \mathbb{I}) \cdot \tilde{\rho}_i(\tilde{A}, \chi), \hat{\rho} \otimes \rho_i(\chi)\right) \leq 4\sqrt{2\epsilon} \tag{13}$$

and the lemma follows by using once more the triangle inequality. □

Before introducing the simulators, we give some intuitive arguments. The client, during the input-injection step, receives a fully-one-time padded quantum state, while in later steps (up until the last), it has no legitimate quantum state (all of the quantum states are in the server's side). Therefore, using the above lemma, we can already see that the client has no information about the actual quantum state during any step before the last. This is illustrated by the fact that the simulators for these steps can run the honest protocol with a random (but fixed) input, and the client's view is the same (they cannot distinguish a run with the correct or different input in any step before the last). The simulator for the final step (after receiving the quantum states from the server) is more subtle, as it necessarily involves a call to the ideal functionality.

Proof of Theorem 3.

Simulator upon receiving the server's encoded input: There is a full one-time pad on the server's input; thus, a simulator can use a random fixed state φ^* instead of the real state $\mathrm{Tr}_C(\rho_{in})$, and the client cannot distinguish them.

Simulator after returning the server's input and before receiving output: The simulator runs the honest protocol using instead of ρ_{in} a fixed random input φ^*. The honest state with input χ at this stage is:

$$\begin{aligned}\rho_i(\chi) \quad &:= \\ &(|k_i\rangle\langle k_i| \otimes |b_i\rangle\langle b_i|)_{\mathcal{H}_C} \otimes \left(E_{k_i}(\sigma_i(\chi)) \otimes |d_i\rangle\langle d_i| \otimes \left|+_{\theta_{t_i}}\right\rangle\left\langle+_{\theta_{t_i}}\right|\right)_{\mathcal{H}_S}\end{aligned} \tag{14}$$

where $\sigma_i(\chi)$ is the evolution of the input state, when the gates corresponding to the i-th step have been performed (here, we collectively call the secret parameters used to encrypt the state in each step as k_i). Later, we will be more specific about the different secret parameters. Furthermore, d_i, θ_{t_i} are the dummy and trap qubits of the i-th layer (we can include the qubits of future layers, as well with no difference in the remaining argument). It is easy to see that the reduced state on the client's side $\mathcal{H}_C$ is independent of the input state χ. The simulator is then constructed:

1. The simulator runs the protocol with fixed but random input φ^*, to obtain $\rho_i(\varphi^*)$.
2. The simulator obtains the fixed state $\hat{\rho}_i$ and the isometry T_i, which are both independent of the input.
3. The simulated view is then defined to be:

$$\nu_i(\tilde{C}, \rho_{in}) := \mathrm{Tr}_{\mathcal{H}_{S_i}}\left((T_i^\dagger \otimes \mathbb{I})(\hat{\rho}_i \otimes \rho_i(\varphi^*))\right) \tag{15}$$

We can easily see that:

$$\nu_i(\tilde{C}, \rho_{in}) = \mathrm{Tr}_{\mathcal{H}_{S_i}}\left((T_i^\dagger \otimes \mathbb{I})(\hat{\rho}_i \otimes \rho_i(\rho_{in}))\right) = \mathrm{Tr}_{\mathcal{H}_{S_i}}\left((T_i^\dagger \otimes \mathbb{I})(\hat{\rho}_i \otimes \rho_i(\varphi^*))\right)$$

as $T_i^\dagger$ does not acts on $\mathcal{H}_{S_i}$ and thus commutes with the partial trace, while tracing out $\mathcal{H}_{S_i}$ leaves the reduced state independent from the input (see Equation (14)). Since isometries leave invariant the trace distance, while the partial trace is non-increasing, we use Equation (9), and we have:

$$\Delta\left(\mathrm{Tr}_{\mathcal{H}_{S_i}}\left(\tilde{\rho}_i(\tilde{C}, \rho_{in})\right), \nu_i(\tilde{C}, \rho_{in})\right) \leq 12\sqrt{2\epsilon} \tag{16}$$

Simulator after receiving client's output: First, we need to introduce some notation and conventions:

- By k^S, k^C, we define the padding of the server's and client's output, respectively. In more detail, these keys are functions of the trap-colouring (trap positions) of the last layer, the rotations θ, the secret parameters r's and measurement results b's of the previous two layers. We collectively denote the rest, i.e., all of the other secret keys, as k^R.

- We also assume that the classical measurement outcomes b's are in the Hilbert space of the client, while being classical and public, could be copied to the server's Hilbert space as well (adding this copy does not affect the client's simulators and complicates the notation). Moreover, classical results corresponding to trap measurements are denoted b_t, while other results are simply denoted b. The trap measurements outcomes (in the honest run) are $b_t = r_t$, where r_t is one of the secret parameters included in k^R.
- The final layer (unmeasured) qubits are separated into: (i) output qubits (those of the server and the client; and it includes the purification of their inputs), (ii) dummy qubits collectively denoted d^S, d^C for server/client unmeasured dummies and (iii) trap qubits denoted as θ_t^S, θ_t^C. The dummies and trap qubits are in the tensor product with the rest (in the honest run).
- The Hilbert spaces containing the output qubits (and the related purification of honest run) are denoted $\mathcal{H}_{S_o}, \mathcal{H}_{C_o}$, while the remaining qubits are $\mathcal{H}_S, \mathcal{H}_C$.
- For brevity, we denote $[b] := |b\rangle \langle b|$ and use the analogous notation for other states.

With the above notations, the honest run of the protocol at this step, with some input χ, is:

$$\begin{aligned} \rho_i(\chi) &= \left(E_{k_i^C, k_i^S}(U(\chi))\right)_{\mathcal{H}_{C_o} \otimes \mathcal{H}_{S_o}} \\ &\otimes \left([k^S] \otimes [k^C] \otimes [k^R] \otimes [b] \otimes [b_t] \otimes [d^C] \otimes [\theta_t^C]\right)_{\mathcal{H}_C} \\ &\otimes \left([d^S] \otimes [\theta_t^S]\right)_{\mathcal{H}_S} \end{aligned} \tag{17}$$

We define:

$$\begin{aligned} \varrho(\chi, k^C, k^S) &:= \left(E_{k_i^C, k_i^S}(U(\chi))\right)_{\mathcal{H}_{C_o} \otimes \mathcal{H}_{S_o}} \\ \sigma &:= \left([k^S] \otimes [k^C] \otimes [k^R] \otimes [b] \otimes [b_t] \otimes [d^C] \otimes [\theta_t^C]\right)_{\mathcal{H}_C} \otimes \left([d^S] \otimes [\theta_t^S]\right)_{\mathcal{H}_S} \\ \rho_i(\chi) &= \varrho(\chi, k^C, k^S) \otimes \sigma \end{aligned} \tag{18}$$

It is worth pointing out that σ is independent of the input χ. Now, the simulator is constructed in the following steps:

1. The simulator obtains from the client the choice of secret keys k^C, k^S, k^R (that can be viewed as part of the client's input).
2. The simulator sets $b_t = r_t$. For the other measurement outcomes b, the simulator chooses randomly a bit value (to ensure that the b's obtained are indistinguishable from the real honest protocol, the simulator can run the full protocol with some random input φ. Since in the honest run, the values of b's do not depend on (are not correlated to) the input, the outcomes that the simulator returns would be indistinguishable from those of a run with the correct input. Finally, speciousness ensures that this state is also close to the real protocol).
3. The simulator, using the previous steps, constructs the state σ.
4. The simulator uses the state $\hat{\rho}_i$ from Lemma 1 for the i-th step, which is a fixed state (independent of the input).
5. The simulator calls the ideal functionality and receives the state $\mathrm{Tr}_{\mathcal{H}_{S_o}}(\rho_i(\rho_{in}))$.
6. The simulator uses the keys and the state received from the ideal functionality and constructs $\mathrm{Tr}_{\mathcal{H}_{S_o}}\left(\varrho(\rho_{in}, k^C, k^S)\right)$.
7. The simulator obtains the operator $\mathcal{T}_i$ from the definition of specious and constructs the isometry T_i acting from $\mathcal{H}_{\tilde{C}} \to \mathcal{H}_C \otimes \mathcal{H}_{\hat{C}}$.
8. The simulated view is then:

$$\nu(\tilde{C}, \rho_{in}) := \mathrm{Tr}_{\mathcal{H}_S}\left((T_i^\dagger \otimes \mathbb{I})\left(\hat{\rho}_i \otimes \mathrm{Tr}_{\mathcal{H}_{S_o}}\left(\varrho(\rho_{in}, k^C, k^S)\right) \otimes \sigma\right)\right) \tag{19}$$

It is clear that in all of this construction, ρ_{in} appears only through $\mathrm{Tr}_{\mathcal{H}_{S_o}}(\rho_i(\rho_{in}))$, which is the ideal output of the client. Now, we prove that this simulated view is δ-close to the view of the client in the real protocol.

We start from Equation (9), and we obtain:

$$\Delta\left(\mathrm{Tr}_{\mathcal{H}_S,\mathcal{H}_{S_o}}\left(\tilde{\rho}_i(\tilde{C},\rho_{in})\right),\mathrm{Tr}_{\mathcal{H}_S,\mathcal{H}_{S_o}}\left(\left(T_i^\dagger\otimes\mathbb{I}\right)\left(\hat{\rho}_i\otimes\rho_i(\rho_{in})\right)\right)\right)\leq 12\sqrt{2\epsilon}$$

Now, since $T_i^\dagger$ does not act on either of the server's Hilbert spaces, it commutes with the partial trace (see Appendix D). We can then see that:

$$\nu(\tilde{C},\rho_{in})=\mathrm{Tr}_{\mathcal{H}_S,\mathcal{H}_{S_o}}\left(\left(T_i^\dagger\otimes\mathbb{I}\right)\left(\hat{\rho}_i\otimes\rho_i(\rho_{in})\right)\right) \tag{20}$$

and this concludes the first part of the proof, as we have a $(12\sqrt{2\epsilon})$-private protocol.

□

4.2. Server's Simulators

Our QYao protocol is secure against a fully-malicious server that can deviate in any possible way and can also cause an abort. We will use the fact that the QYao protocol is ϵ-verifiable for the client, i.e., Equations (6) and (7) hold. In many classical protocols, to prove security against malicious adversaries, one has to restrict his or her actions to essentially honest-but-curious adversaries. Here, the verification property plays such a role, as the condition obtained can be used in a similar way as the speciousness of the client.

We can see that, before the client sends the secret parameters, the server is totally blind, i.e., the server's reduced state at all times is the totally mixed state (for all qubits in the DT(G), including their own input qubits, after they are injected). This is highlighted by the fact that one can run the full protocol (before releasing the keys), without choosing the (client's) input (see simulator below). After receiving the keys from the client, to simulate the server's view, we need a call to the ideal functionality and, at this point, to use Equation (6).

Continuation of the proof of Theorem 3:

Simulator before receiving keys: The simulator, instead of sending qubits in one of these states $\{|+_\theta\rangle,|0\rangle,|1\rangle\}$, sends one side of an EPRpair $|\psi\rangle=\frac{1}{\sqrt{2}}(|01\rangle+|10\rangle)$ for each qubit to the server; then chooses an angle δ_i at random for each of the measurement angles. The simulator can measure (if it wishes) its qubits (of the EPR pairs) in a suitable way and angles that insert any input it wishes and performs any computation. This can be done after all measurements of the server have taken place (see [14]). It follows that the server cannot obtain any information about the client's input.

Simulator after receiving keys: The simulator is constructed using the following steps:

1. The simulator prepares multiple Bell states $|\psi\rangle=\frac{1}{\sqrt{2}}(|01\rangle+|10\rangle)$, sends the one qubit of each pair to the server and instructs them to entangle the qubits as it would in the normal protocol.
2. The simulator for each qubit chooses randomly an angle δ_i and instructs the server to measure in this angle.
3. The simulator obtains from the server's (malicious) strategy the parameter p_{ok} (see Equation (6)).
4. With probability $(1-p_{ok})$, the simulator returns an abort. Otherwise, it performs the remaining steps.
5. The simulator calls the ideal functionality and obtains the state $\mathrm{Tr}_{\mathcal{H}_C}(U(\rho'_{in}))$, where $\rho'_{in}=(\mathbb{I}_{\mathcal{H}_C}\otimes D_{\mathcal{H}_S})\cdot\rho_{in}$ is the deviated input that the corrupted server inputs.
6. The simulator encrypts the outcome using k^S, and sends the output qubits of the server back in the state: $E_{k^S}\left(\mathrm{Tr}_{\mathcal{H}_C}(U(\rho'_{in}))\right)$
7. The simulator returns the keys k^S to the server.

The last step of the protocol is that the server decrypts its output. The decryption is denoted as $D_{k^S}(\cdot)$, where we have by definition that $D_{k^S}(E_{k^S}(\cdot)) = \mathbb{I}$. It follows that the server's view of the real protocol after the key exchange is:

$$\mathrm{Tr}_{\mathcal{H}_C}\left(E_{k^S}\left(\tilde{\rho}^{n-1}(\tilde{S}, \rho'_{in})\right)\right) \tag{21}$$

From Equation (7), we obtain:

$$\Delta\left(\mathrm{Tr}_{\mathcal{H}_C}\left(E_{k^S}\left(\tilde{\rho}^{n-1}(\tilde{S}, \rho_{in})\right)\right), \mathrm{Tr}_{\mathcal{H}_C}\left(E_{k^S}(\rho^{n-1}_{ideal}(\rho'_{in}))\right)\right) \leq \epsilon \tag{22}$$

as the partial trace is a distance non-increasing operation. Using the definition of $\rho^{n-1}_{ideal}(\rho'_{in})$ from Equation (7), we see that:

$$\begin{aligned}
\mathrm{Tr}_{\mathcal{H}_C}\left(E_{k^S}(\rho^{n-1}_{ideal}(\rho'_{in}))\right) &= \mathrm{Tr}_{\mathcal{H}_C}\left(E_{k^S}\left(p_{ok}U\cdot(\rho'_{in}) + (1-p_{ok})(|fail\rangle\langle fail|)\right)\right) \\
\mathrm{Tr}_{\mathcal{H}_C}\left(E_{k^S}(\rho^{n-1}_{ideal}(\rho'_{in}))\right) &= \nu(\tilde{P}, \rho_{in})
\end{aligned} \tag{23}$$

We have now proven that the simulated view is ϵ-close to the real view of the server, just after the key exchange, thus proving that the protocol is ϵ-private against malicious server $\tilde{S}$. □

5. Non-Interactive QYao

Following the classical approach of [24], we exploit our QYao protocol to construct the one-time program. To do so, we simply need to remove the classical online communication of the QYao protocol using the classical hardware primitive of secure "one-time memory" (OTM), which is essentially a non-interactive oblivious transfer. The obtained one-time quantum programs can be executed only once, where the input can be chosen at any time. Classically, the challenge in lifting the Yao protocol for two-party computation to the one-time program was the issue of the malicious adversary. However, our QYao protocol is already secure against a malicious evaluator without any extra primitive or added overhead.

Recall that the interaction in our QYao is required for two reasons. First, from the server's perspective, this is done to obtain the measurement angles δ_i that perform the correct computation, while these could not be computed offline as they depend on the measurement outcomes $b_{j<i}$ of certain qubits measured before i, but after the preparation stage (and on secret parameters $(\theta_i, r_i, r_{j<i}, T)$ and computation angle ϕ_i that are known from start). Second, from the client's perspective, the results of measurements needed to be provide with a "proof" that the output is correct, by testing for deviation from the trap outcomes.

Removing the interaction: An obvious solution for the first issue raised is to have the client compute δ_i for all combinations of previous outcomes $b_{j<i}$ and then store in an OTM the different values, while the server chooses and learns the entry of the OTM corresponding to the outcomes obtained. This solution, at first, appears to suffer from an efficiency issue as one may think that for each qubit, the client needs to compute δ_i for all combinations of past outcomes, which grows exponentially (as would the size of the OTM used). However, a closer look at the dependencies of corrections in MBQC for the typical graphs used shows that the measurement angle of qubit i depends on at most a constant number of qubits. Those dependencies are within the two previous "layers" (past neighbours or past neighbours of past neighbours). This is evident from the flow construction [41,42], which guarantees that corrections can be done, and the explicit form of dependencies involves only neighbours and next-to neighbours in the graph G. A flow is defined by a function $(f : O^c \to I^c)$ from measured qubits to non-input qubits and a partial order $(\preceq)$ over the vertices of the graph such that $\forall i : i \preceq f(i)$ and $\forall j \in N_G(i) : f(i) \preceq j$, where $N_G(i)$ denotes the neighbours of i in G. Each qubit i is X-dependent on $f^{-1}(i)$ and Z-dependent on all qubits j, such that $i \in N_G(j)$.

Definition 7 (Past of qubit i)**.** *We define $P_i = Z_i \cup X_i$ to be the set of qubits j that have X or Z dependency on i.*

Definition 8 (Influence-past of qubit i)**.** *We define influence-past c_i of qubit i to be an assignment of an outcome $b_j \in \{0,1\}$ for all qubits $j \in P_i$.*

For each influence-past c_i, there exists a unique value of $\delta_i(c_i)$. While c_i provides all of the necessary dependencies for the client to compute δ_i, this is not known to the server. This could be problematic, as the server is expected to open the OTM using the past outcomes with a labelling consistent with its own knowledge. The true dependencies depend on the actual flow of the computation, which is hidden from the server, as the positions of dummies (that break the DT(G) to the three graphs) are not known to the server. From here on, we restrict attention to DT(G). This is resolved by defining:

Definition 9 (Extended-past of qubit i)**.** *We define the extended past EP_i of qubit i to be the set of all qubits that for some trap-colouring are in the past of i.*

Similarly we define the extended-influence-past (EIP) of qubit i (it is clear that δ_i has trivial dependency on all outcomes of EIP that are not in the actual influence-past). The extended past has also finite cardinality, as it is evident that the only qubits that can affect a qubit with base-location i are those that belong to (the finite) base-locations that contain the qubits that have true dependency (i.e., neighbours or next-to neighbours base-locations).

Ensuring verification: The interaction is also important from the clients' perspective, in order to verify that the server did not deviate. Again, one could naively suggest that the server can return all of the measurement outcomes at the end. However, this suffers from a potential attack (not present in normal VUBQC). The underlying reason is that there is nothing to force the server to commit that the b_i's that will return are the same as the ones used for selecting measuring angles. For example, the server measures Qubit 1 at δ_1 and obtains b_1 (which will be returned at the end), but uses $b'_1 = b_1 \oplus 1$ when opening the OTM corresponding to qubit j where $1 \in EP_j$. If qubit j is a computation qubit, this leads to measuring at a wrong angle, while if qubit j is a trap qubit, it has no (real) dependence on b_1; thus, the angle recovered is the correct one, and the server never causes an abort, while corrupting the computation. To avoid this problem, we introduce a method that uses some flag bit-string l_i.

The client provides for each qubit the angle δ_i and a flag bit-string l_i of length m. This flag bit-string is a (one-time-padded) flag of whether a trap failed in the (extended) past of qubit i or not. At the final step of the protocol, the server returns the final layer qubits and all of the flags l_i. Here, we will denote $b_{j,i}$ to mean the outcome of qubit j measurement that the client uses for the computation of the measurement angle for qubit i, i.e., the value of b_j selected by the server when opening the i-th OTM.

For each qubit i, the client chooses uniformly at random an m-bit string l_i^0 to denote that none of its past qubits ($j \in P_i$) was a failed trap. The string l_i^0 will be called the "accept" flag, while all other strings $l_i \neq l_i^0$ are the "reject" flags. The OTMs are prepared according to Protocol 6.

According to this protocol, the flag that the server obtains when opening the OTM at qubit i, is rejected if and only if in the extended past of this qubit, there is a failed trap outcome (we can see that returning a wrong outcome for some measurement b_i implies opening an entry of the OTM with the wrong flag only if the qubit is a trap one (as in regular VUBQC), while it still returns the accept flag if it is a dummy or computation qubit). Here, we should note that even if the client knows that one flag is a reject flag (i.e., has one particular $l_i \neq l_i^0$), the probability of guessing the correct flag is only $\epsilon_f := (2^m - 1)^{-1}$. This, intuitively, will force the client to return the flag obtained from the OTM (or abort with high probability), provided that m is chosen suitably. We can now give the non-interactive QYao Protocol 7 (see Appendix C.2 for a simple example).

Protocol 6 Preparation of OTMs for non-interactive QYao.

Description of OTM's: We will use an (at most) one out of K OTM, where $K = 2^{\max_i |EP_i|}$. Note, that the first layer does not need an OTM, since (i) we give the angle directly (does not depend on anything) and (ii) there is no trap in the previous layer and thus no need to give a flag. Moreover, for the last layer, while the qubits are not measured (and thus there is no corresponding angle δ) we will use an OTM to obtain the last flags (that correspond to testing traps in the previous layers).

- **Labels** of the cells of the OTM at i: Each qubit is labelled according to the outcomes of the extended past qubits $b_{j;i}|j \in EP_i$, in other words according to the extended influence past c_i.
- **Content** of the cells of the OTM at i: In each cell of the OTM we have a pair of numbers (We could view this as a single string with the convention that the first three bits give δ and the remaining the flag.) (δ_i, l_i). The δ_i is the unique correct δ for the particular outcomes of past qubits $b_j|j \in P_i$.

$$\delta_i(c_i) = (-1)^{s_i^{X_i}}\phi_i + \theta_i + \pi(r_i \oplus s_i^{Z_i}) \tag{24}$$

The flag l_i depends on the outcomes b_j of the extended past $j \in EP_i$. In particular:

1. If for all traps t in the extended past of i, i.e., $t \in EP_i$, $b_{t;i} = r_t$, we return l_i^0, i.e., accept flag.
2. Otherwise, we return a random string $l_i \neq l_i^0$, i.e., reject flag.

Protocol 7 Non-interactive QYao.

Assumptions
Client and server want to jointly compute a unitary as in Protocol 5. The client has N OTM's that are 1-out-of-K, i.e., one OTM per qubit, with sufficient entries to store a pair of (δ_i, l_i) measurement angle and flag bit-string (of length at least $m \geq \log(\frac{1}{\epsilon} + 1)$), for each extended influence past of the qubit.
Protocol

1. Client and server interact according to Protocol 3 to obtain the server's input locations. Client also sends the qubits of DT(G) after choosing secret parameters (r_i, θ_i, d_i and trap-colouring).
2. The client, for each qubit and each extended influence past, computes $\delta_i(c_i)$. Then prepares one OTM per qubit as described in Protocol 6.
3. Server performs the measurements according to the first layer of angles received δ_i (directly as in Protocol 3). Then open the next layer OTM's using the outcomes b_i of their measurements. Uses the new measurement angle revealed δ_j, while records the flag bit-string l_j. Iterates until the last layer OTM is opened (and the second last layer is measured). The final layer OTM's return only a flag.
4. Server and client interact according to Protocol 4 so that the server obtains their output. The only difference is that the client, before returning the keys, in order to accept checks the flags (instead of checking the trap outcomes of measured qubits) and the final layer traps.

This protocol requires exactly the same number of qubits as Protocol 5. The only extra "cost" is that we require one OTM per qubit. The size of the labels of the OTM is one-out-of-K (where $K = \max_i |EP_i|$), while the size of the content of each cell is $3 + m \geq 3 + \log(\frac{1}{\epsilon} + 1)$ bits as described in Protocol 6.

Theorem 4. *Protocol 7 is ϵ-verifiable for the client, with ϵ the same as the VUBQC protocol that is used.*

Proof. This proof consists of three stages. First, similarly to the proof of theorem 2, we show that the verifiability for the client of the non-interactive QYao reduces to the verifiability of the same VUBQC protocol with modifications for the server's input and output. The second stage is to observe that the optimal strategy for an adversarial server is to return the flags from the opened OTMs. This makes the verifiability property for this protocol identical to an interactive protocol with the only modification that the server can return different values for the measurement outcome b_i depending on which future

qubit the client needs the outcome ($b_{i;j}$). The final stage, is to show that this modified (interactive) verification protocol is ϵ-verifiable with the same ϵ as Protocol 1.

Stage 1: Following the proof of Theorem 2, we can see that the non-interactive QYao Protocol 7 is ϵ-verifiable for the client if the corresponding non-interactive VUBQC protocol that is used during Step 3 of Protocol 7 is ϵ-verifiable, with deviated input ρ'_{in} and a final deviation $(\mathbb{I}_{\mathcal{H}_C} \otimes \mathcal{C}_{\mathcal{H}_S})$ on the server's output (computation) qubits.

Stage 2: If the server attempts to guess a flag (given the extra knowledge of one bit-string $l_i \neq l_i^0$), it causes an abort with high probability $(1 - \epsilon_f$, where $\epsilon_f := (2^m - 1)^{-1})$. However, since by assumption $m \geq \log(\frac{1}{\epsilon} + 1)$, the probability of an abort is so high that it makes the protocol trivially verifiable, as it is ϵ-close to the ideal state Equation (6) with $(1 - p_{ok}) = (1 - \epsilon_f)$.

It follows that the adversary (trying to maximising its cheating chances) should return the flags found in the OTMs. This is equivalent to an interactive VUBQC protocol, for which= (i) the server for each qubit i returns multiple values of the measurement outcome $b_{i;j}$, one for each j in the extended future qubit of i, (ii) the client uses those outcomes to compute the δ_i's and (iii) the client aborts only when it receives at least one trap outcome wrong $b_{t;j} \neq r_t$ for any j.

Stage 3: We should now show the ϵ-verifiability of the modified interactive VUBQC protocol described above. The proof follows the same steps of the proof of the verifiability of Protocol 1 in [25]. The first steps exploit the blindness to reduce the possible attacks (of the adversarial server) to a convex combination of Pauli attacks. Then, it is noted that since the computation is encoded in an error-correcting code (that corrects up to $\delta/2$ errors), there is a minimum number ($d = \lceil \frac{\delta}{2(2c+1)} \rceil$) of independent base-locations that need to be corrupted to cause an error.

For the proof of verifiability, as in [10,25], we make the assumption that if the minimum number of attacks that could corrupt the computation occurs, then the computation is corrupted. This is clearly not true, but is sufficient to provide a bound on the probability of corrupt and accept (which is the "bad" case that the server manages to cheat). Then, given this minimum number of attacks, the probability of abort is computed and found to be greater than $1 - \epsilon$, and thus, the protocol is verifiable.

In our case, there is the difference that for each measured qubit b_i of the original protocol, we have multiple qubits $b_{i;j}$. Once again, we make the assumption of minimum corrupted qubits; we need at least $d = \lceil \frac{\delta}{2(2c+1)} \rceil$ independent base-locations; and we can allow for each base-location to corrupt a single $b_{i;j}$ for one specific j. However, this does not change the probability of hitting a trap, as it suffices to give the wrong value $b_{t,j}$ for one j to cause an abort. We then obtain again that the probability of an abort (in the minimum corruptible attack) is at least $(1 - \epsilon)$ (for $\epsilon = \left(\frac{8}{9}\right)^d$), and the protocol is indeed ϵ-verifiable. □

Theorem 5. *Protocol 7 that is ϵ_1-verifiable for the client is $O(\sqrt{\epsilon_2})$-private against an ϵ_2-specious client and ϵ_1-private against a malicious server.*

Proof. We need simulators for the client only during input injection and output extraction (since the client does not participate in the evaluation phase, in the non-interactive protocol). During these steps, the simulators are identical to those in Protocol 5.

The server's simulators until before the output extraction can be identical with the ones of Protocol 5, where the interaction is replaced with the preparation of OTMs. It is important to note that all OTMs can be constructed with no information about the client's input and thus from the simulator. Finally, the simulator for the final step of Protocol 5 needs to only use the property that the protocol is ϵ-verifiable for the client. We proved in Theorem 4 that this is the case for Protocol 7, and thus, we can use the same simulator. □

6. Conclusions

We gave a protocol for 2PQC, secure against specious adversaries. Our protocol differs significantly from [21] since it has an asymmetric treatment of the two parties. There are two approaches

for classical S2PC, that of Yao [22] and that of Goldreich et al. [43]. While our protocol can be seen as the quantum version of [22] (that also has asymmetry between parties), the protocol of [21] is a quantum version of [43]. In the quantum case, our approach provides the extra advantage of being more practical in terms of technological requirements. This is because the client requires only basic quantum technologies (preparing single qubits), and there is no online quantum communication among the parties.

Furthermore, due to a subtle difference in the definition of specious adversaries, our protocol requires no extra primitives (see the details in Appendix B). Typically, considering weaker adversaries such as the specious adversaries is a first step towards protocols secure against malicious adversaries. This is why it is not crucial whether we take the stronger or weaker form of the specious adversaries. There are standard techniques, such as the cut-and-choose technique and the GMWcompiler [43], that turn protocols from secure against honest-but-curious to secure against malicious adversaries. However, the quantum generalisations are considerably more involved, as these techniques cannot be directly applied due to no-cloning and quantum inseparability. We have recently extended our results towards this direction, in the specific case of classical input/output, by generalising the cut-and-choose technique [44].

Finally, we obtained a one-time compiler for any quantum computation using one-time memories. This was done following the classical work of Goldwasser et al. [24] and more efficiently than the quantum work of Broadbent et al. [6]. An important difficulty to use the classical Yao protocol directly to construct one-time programs is the case where the evaluator is malicious (rather than honest-but-curious). However, in our QYao, the protocol is secure against an evaluator that is fully malicious, and thus, we can obtain the one-time program directly once we remove the interaction (which is present in the quantum case). To overcome this difficulty, we used OTMs to replace the interaction and introduced an extra "flag" register to ensure the security of the protocol.

Acknowledgments: We would like to thank Frédéric Dupuis for discussions on the definition of specious adversaries. Funding from EPSRCGrants EP/N003829/1 and EP/M013243/1 is acknowledged.

Author Contributions: Both authors contributed to developing the theory and writing the paper.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

UBQC	Universal blind quantum computation
MBQC	Measurement-based quantum computation
VBQC	Verifiable blind quantum computation
QYao	Quantum "Yao" protocol
OT	Oblivious transfer
OTM	One-time memory
DT(G)	Dotted triple graph
2PQC	Secure two-party quantum computation

Appendix A. Measurement-Based Quantum Computation

An MBQC pattern is fully characterised by the graph (representing the entangled state), default measurement angles ϕ_i, an order of measurements (determined by the flow) and a set of corrections that determine the actual measurement angle of each qubit (modify the default measurement angles with respect to previous measurement outcomes).

For the brickwork state, or any subset of the square lattice state, the flow $f(i)$ takes a qubit to the same row, next column, e.g., $i = (k,l)$, then $f(i) = (k+1,l)$. Given a flow $f(\cdot)$ and measurement outcomes s_j, the actual (corrected) measurement angle is given:

$$\phi_i'(\phi_i, s_{j<i}) := \phi_i(-1)^{s_{f^{-1}(i)}} + \pi \sum_{j|f(j)\in N_G(i)\setminus f(i)} s_j \tag{A1}$$

where $N_G(i)$ denotes the set of neighbours (in the graph) of vertex i. It is easy to see that the corrected angle for qubit i has an X-correction (i.e., a (-1) factor before ϕ_i) from the one qubit $f^{-1}(i)$ and Z-corrections (i.e., π addition) from some neighbours of neighbours, which for the brickwork state is at most two (but in any case is constant in number).

We present here diagrams taken from [3] showing how to translate a universal set of gates to the MBQC measurement patterns using the brickwork graphs.

Appendix B. On Specious Adversaries Definitions

The definition of specious, as given by Equation (4), includes $\rho_i(\rho_{in})$, which is the honest state in step i, for any input state ρ_{in}. There are two subtle issues with what this means, (i) in relation with what are the possible input states ρ_{in} and (ii) in relation to whether the random secret parameters are considered the input of the computation or not. If we take the most restrictive case, it leads to a very weak adversary. In [21], there are some impossibility results that stem from taking a (slightly) stronger form of this adversary. Here, we see separately these two issues and finally discuss the differences with [21].

Appendix B.1. Restricting to Classical Inputs

The first important point is that we observe that a specious adversary, under certain conditions, could be weaker than an honest-but-curious classical adversary. A specious adversary is allowed to do actions/operations that for any (allowed) input can be "undone" by actions on his/her side if there is an audit.

Now, we consider a (trivial) example that specious is weaker than classical honest-but-curious. Assume that as the first step of a protocol, a party (that is specious adversary) $\tilde{A}$ receives a one-time padded quantum input $E_k(|\psi\rangle_S)$ of the other party. As the second step, the $\tilde{A}$ returns the padded quantum state back. If the input is considered to be a general (unknown) quantum state, a specious adversary cannot do the following action before returning the system S:

$$(\wedge X)_{SA} E_k(|\psi\rangle_S) \otimes |0\rangle_A \tag{A2}$$

There is no map that $\tilde{A}$ can apply to their private system/ancilla A alone and obtain the correct state $\rho_i(\rho_{in})$, because for a general $E_k(|\psi\rangle_S)$, the resulting joint state is entangled.

However, imagine that we are actually considering a classical protocol, which means that the input is in computational basis, i.e., either $|0\rangle$ or $|1\rangle$. In that case, the $\wedge X$ simply copies the ciphertext $E_k(|\psi\rangle_S)$, which is exactly the action that an honest-but-curious classical adversary can do (Note that in this specific case, the resulting state is no longer an entangled state and thus could be recovered by acting only on system S. Nonetheless, the definition of specious requires recovering the correct state at each step for any possible input.). It is exactly because of this property (that specious under certain conditions is weaker than classical honest-but-curious) that we can avoid using OT for inserting the input of the server (unlike the Yao protocol). In generalisations where we will have stronger adversaries, we will once again need to have OT for the input insertion.

Appendix B.2. About the Secret Random Parameters

The second important subtlety of the specious definition is related to the secret (random) parameters that the adversary can choose. Specious adversaries should be able to recover the global state $\rho_i(\rho_{in})$ in any step. However, this definition may be somewhat ambiguous. In general, the quantum state in any step is also a function of secret random parameters of the two parties k_A, k_B, i.e., we have $\rho_i(\rho_{in}, k_A, k_B)$. We can (and generally do) treat the secret keys as part of the input of

the two parties (i.e., part of ρ_{in}), but in this case, the specious adversary is essentially requested to reconstruct precisely the state $\rho_i(\rho_{in}, k_A, k_B)$ at step i.

However, we can imagine an adversary $\tilde{A}$ that could reproduce the state $\rho_i(\rho_{in}, k'_A, k_B)$ instead, i.e., reproduce a state that would be correct for step i, if the secret parameter were k'_A instead of the k_A that was given at the start. In this case, the adversary $\tilde{A}$ is not specious with the standard definition that we use. On the other hand, intuitively, since k_A is a secret parameter (not known by anyone but $\tilde{A}$ until this step), it should not matter what is this value, and reproducing $\rho_i(\rho_{in}, k'_A, k_B)$ should be sufficient for a version of quantum honest-but-curious.

We will define the strong specious adversary to be the adversary that is required to have CP maps $\mathcal{T}_i$ acting only on their subsystem, such that they can reproduce the state $\rho_i(\rho_{in}, k_A, k_B)$ for at least one secret key k_A (not determined in advance and thus of their choice).

We will again give a simple example to make the distinction of these two flavours of specious adversaries. Imagine a protocol that a (strong or weak) specious adversary $\tilde{A}$ is supposed to return an unknown state $|\psi\rangle$ padded with its key $E_{k_A}(|\psi\rangle)$. A strong specious adversary can cheat by keeping the state $|\psi\rangle$ and returning instead the one side of an EPR pair (and keep the other side). Then, if an audit occurs, the adversary can use its side of the EPR pair and teleport the state $|\psi\rangle$ back, where the resulting state at the honest side is $E_{k_m}(|\psi\rangle)$, with k_m being (randomly) determined by the outcome of the Bell measurement that teleports the state. A weak specious adversary, on the other hand, cannot do this. The state that is supposed to be returned needs to be padded with the key k_A that is fixed from the start of the protocol, while k_m is randomly determined during the audit (by the Bell measurement outcome).

Appendix B.3. Regarding Secure SWAP No-Go Theorem

It is not difficult to see that a protocol that is secure against the standard (weak) specious adversary can be made secure against the strong specious adversary by modifying the protocol to request that in every step a new random (secret) parameter appears, a commitment to its value is made. This means that requesting an adversary to be weak specious is practically equivalent to a strong specious adversary where commitments are allowed.

Here, it is worth mentioning that in [21], while not explicitly stated, their protocol was secure against the stronger version of specious adversaries (unlike our protocol). This difference in the definition of specious adversaries (implicit assumption of the stronger adversaries) also resolves the apparent contradiction of our result (no secure primitive needed) with their no-go theorem (a secure SWAPis needed for 2PQC even for specious adversaries). As is mentioned in [21], their no-go would not hold if commitments were possible, which is exactly what our weaker definition essentially permits.

Appendix C. Examples

We will now give two simple examples (one for Protocol 5 and one for Protocol 7) to illustrate how the generic protocols given work practically.

Appendix C.1. Example for Protocol 5

We consider a simple example for Protocol 5 where each of the clients/servers has a single qubit input/output, and the unitary/computation to be implemented is a CNOT gate. The MBQC pattern for this gate is given in Figure A1b. The base-graph required is a "brick" from the brickwork state of [3] and consists of 10 qubits. To use the VUBQC protocol of [25], we need the DT(G) of that graph, the construction that is summarised in Section 2.1, resulting in the coloured DT(G) of Figure 1a. In Figure A2, we see the dotted base-graph, while in Figure A3, we can see the resulting DT(G) construction after the dummies have been removed, leaving us with the dotted base-graph of Figure A2 along with the one trap/tag qubit for each computation qubit (as was done in the general case in Figure 1b). The white trap qubit corresponds to primary vertices (qubits of the initial base-graph of Figure A1b), while black trap qubits correspond to added vertices (those that resulted

from "dotting" the initial base-graph). Note that in Figure A3, the computation qubits are entangled (connected with edges), while the trap/tag qubits are isolated. The server, from his/her view, cannot distinguish whether one qubit belongs to the dotted base-graph (i.e., is used for computation) or is the related trap/tag qubit (denoted in the same box in Figure A3). In the next example (Appendix C.2), the base-graph is simpler, and we will give the construction starting from the base-graph and resulting in the dotted base-graph with tags in greater detail.

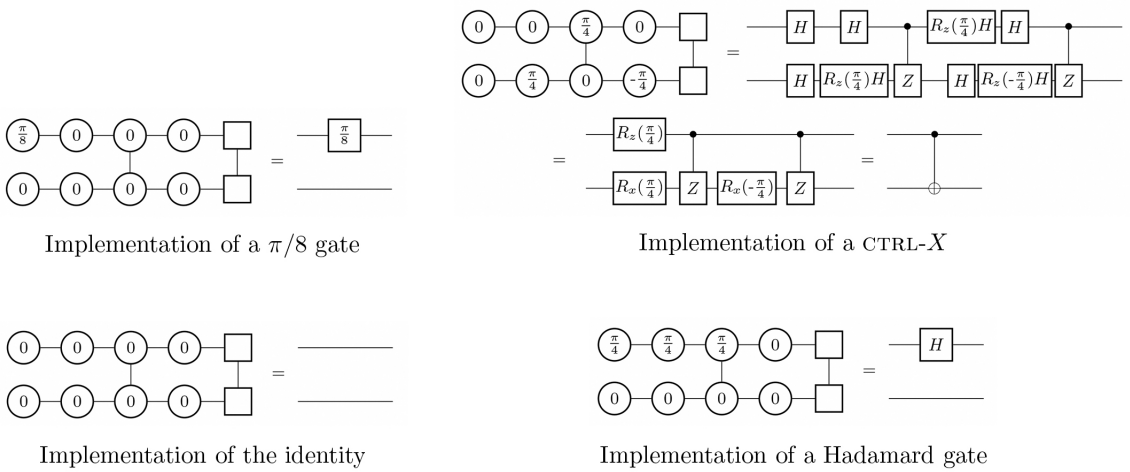


Figure A1. MBQC patterns for different gates: (**a**) $\pi/8$ gate, (**b**) CNOTgate, (**c**) identity gate and (**d**) Hadamard gate.

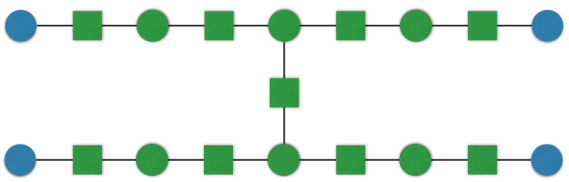


Figure A2. Dotted base-graph for a CNOTgate.

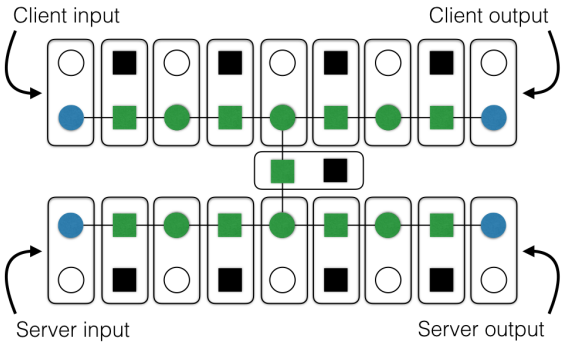


Figure A3. Dotted base-graph for a CNOT gate along with isolated traps/tags. The input/output of the client/server is coloured blue.

Finally, we note that in UBQC, the measurement angle that the client instructs the server to measure is given: $\delta_i = \phi'_i(\phi_i, b_{j<i}) + \theta_i + \pi r_i = C(i, \phi_i, \theta_i, r_i, \mathbf{s})$.

The protocol starts with the server's input injection, where the server gives his/her input (bottom left blue qubit in Figure A3) one-time padded to the client as described in Section 3.1. The client then permutes it with a trap and a dummy qubit (see Figure 2) and returns the triplet to the server. Now,

after removing the dummies (that play no role), the server's view is that of Figure A3, where he/she is not aware which qubit was his/her input and which qubit is a trap/tag. The same holds for the other qubits of the MBQC pattern.

The client instructs the server to measure the qubits at angles δ_i. The distribution of these angles is uniformly random within the set $\{0, \pi/4, 2\pi/4, \cdots, 7\pi/4\}$ irrespective of the actual computation, and thus, the server (from his/her perspective) is totally blind to what computation is performed or of which qubits are computation and which are trap/tag qubits.

The client, from his/her view, taking into account the pre-rotations θ_i and corrections from previous measurements in reality is actually asking the server to measure the computation qubits in the angles of Figure A1b and thus performing the CNOT gate. At the same time, the instruction for the trap/tag qubits is to measure them in the basis that was prepared $|\pm_\theta\rangle$ and thus return a deterministic outcome (since they are isolated from the rest qubits).

Finally is the server's output extraction part (see Section 3.2). Here, the server one-time pads both his/her output (bottom right blue qubit) and the corresponding trap/tag (since he/she is not aware of which is which) and returns them to the client. The client knows the position of the server's output (but not the padding, so it obtains no information). The client keeps the trap/tag, but returns the real output to the server. The protocol is completed with the exchange of keys (as described in Section 3.2), where the client can test that the final traps are indeed correct (so no deviation occurred before that), and the server obtains the key to his/her output and recovers it.

Appendix C.2. Example for Protocol 7

In this example, we are not concerned with the input/output constructions, as these are similar to Protocol 5. Instead, we give the simplest example of a base-graph to illustrate what influence-past and extended influence-past means and how the flags and the OTMs are constructed.

We start with the simplest base-graph of two qubits given in Figure A4a. Following the DT(G) procedure (described in Section 2.1), we have the DT(G) of Figure A4b, where we label the qubits. Note, that for each of the base-graph qubits (Figure A4a), we have three copies (circles in Figure A4b); all qubits of the one base-location are connected with all qubits of the other base-location (nine edges in total); and an added qubit (square) is added to each of these (nine) edges. This is the view that the server has, since he/she is not aware of the trap-colouring (i.e., of which qubit is computation, trap or dummy).

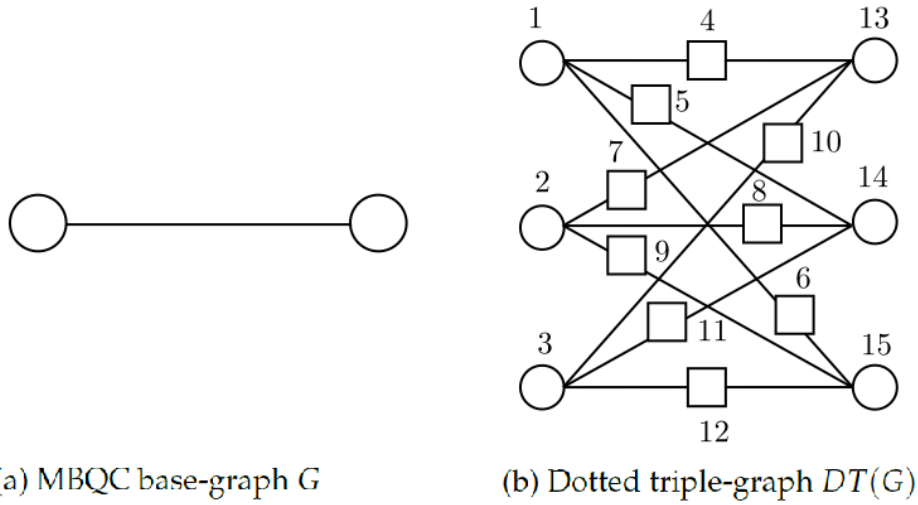


Figure A4. (**a**) Simple MBQC base-graph. (**b**) Corresponding dotted triple-graph with labelled qubits (view of the server).

The view of the client (that knows the trap-colouring) is given in Figure A5a, where we remove the dummies in Figure A5b to keep only the relevant information. This, of course, is just an example of one consistent trap-colouring.

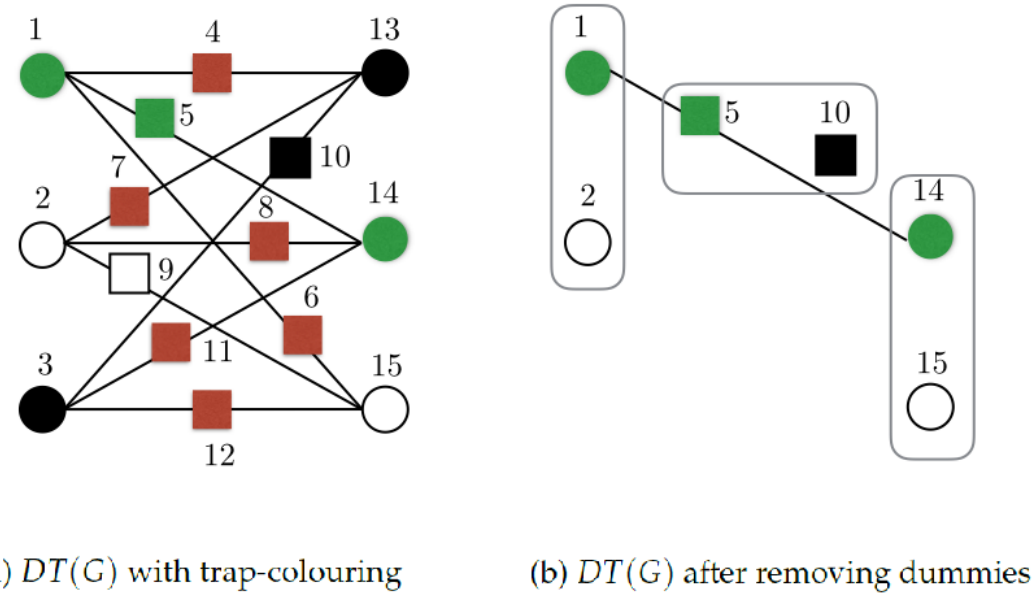


Figure A5. (**a**) Labelled dotted triple-graph with trap colouring (view of client). (**b**) Labelled dotted triple-graph when removing dummy qubits (essential information from client's view).

Now, influence-past is defined in terms of which qubit results in a Z or X correction (in the MBQC terms). It is clear that corrections exist only for qubits that are truly entangled (i.e., the green qubits $\{1,5,14\}$). In particular, qubit $\{5\}$ has X-dependency from qubit $\{1\}$, which is its influence-past. Qubit $\{14\}$ has X-dependency from qubit $\{5\}$ and Z-dependency from qubit $\{1\}$, i.e., its influence-past is $\{1,5\}$. The influence-past of isolated qubits is the empty set.

The extended influence-past includes all of the qubits that for some trap-colouring could be at the influence-past, and it is a superset of the influence-past. In other words, all of the qubits that belong to the same base-location as the true influence-past are always included. We consider the last layer qubits $\{13,14,15\}$, and we summarise their influence-pasts and extended influence-pasts in this table:

Qubit	Influence-past	Extended influence-past	Non-trivial flag dependence
13	$\emptyset$	$\{1,2,3,4,7,10\}$	$\{2,10\}$
14	$\{1,5\}$	$\{1,2,3,5,8,11\}$	$\{2\}$
15	$\emptyset$	$\{1,2,3,6,9,12\}$	$\{2\}$

Now, we describe the OTMs. First, the cells of each OTM are labelled according to different outcomes of qubits in the extended influence-past. In other words, for the OTM for Qubit 13, we will have one cell for each string of measurement outcomes of the qubits $\{1,2,3,4,7,10\}$, i.e., 2^6 cells (see the table). Within each cell of the OTM, there will be the measurement angle corresponding to these outcomes (in general, the corrected measurement angle ϕ_i' depends on outcomes of qubits in the actual influence-past). Moreover, in each cell, there is a flag bit-string of length m (see Section 5). There is one string l_i^0 that corresponds to the accept flag. This flag is contained in all cells except for those that contain a wrong result at one (past) trap/tag. Trap/tag qubits (in the example $\{2,10,15\}$) have a deterministic outcome (in an honest run). Therefore, in all cells of an OTM in which the outcome of one of this trap is wrong, a flag $l_i \neq l_i^0$ is returned. For our example, we can see from the last column of the table which qubits outcomes may result to the non-accept flag in the related cell of the OTM.

Finally, we stress again that in the above table, the influence-past and non-trivial flag dependence is not known to the server, while the extended influence-past is.

Appendix D. Proof of Commuting

We prove here that the partial trace over one system commutes with any operation to the non-traced-out system. We express a generic density matrix $\rho_{AB} \in \mathcal{H}_{AB}$ with respect to some fixed orthonormal basis as:

$$\rho_{AB} = \sum_{i,i',j,j'} \rho_{ii',jj'} \left|i\right\rangle_A \left\langle i'\right| \otimes \left|j\right\rangle_B \left\langle j'\right| \tag{A3}$$

Similarly, a general operator T_B acting on system B is given by:

$$T_B = \sum_{k,k'} T_{kk'} \left|k\right\rangle_B \left\langle k'\right| \tag{A4}$$

It is easy to see that:

$$\mathrm{Tr}_A(\mathbb{I}_A \otimes T_B \cdot \rho_{AB}) = T_B \cdot \mathrm{Tr}_A(\rho_{AB}) \tag{A5}$$

which proves that the partial trace (over system A) commutes with any operation on the non-traced-out system B. We can see this since the left-hand side equals:

$$\begin{aligned} \text{l.h.s} &= \sum_{\tilde{i},i,i',j,j',k,k'} \left\langle \tilde{i}\right|_A T_{kk'} \rho_{ii',jj'} \left|i\right\rangle_A \left\langle i'\right| \otimes \left(\left|k\right\rangle_B \left\langle k'\right| \left|j\right\rangle_B \left\langle j'\right|\right) \left|\tilde{i}\right\rangle_A \\ &= \sum_{i,j,j',k} T_{kj} \rho_{ii,jj'} (\left|k\right\rangle_B \left\langle j'\right|) \delta_{i,i'} \delta_{i,\tilde{i}} \delta_{j,k'} \end{aligned} \tag{A6}$$

using the orthonormality of the bases used (i.e., $\left\langle i\right| \left|i'\right\rangle = \delta_{i,i'}$ etc.). The right-hand side is:

$$\begin{aligned} \text{r.h.s} &= \sum_{\tilde{i},i,i',j,j',k,k'} (T_{kk'} \left|k\right\rangle_B \left\langle k'\right|) \cdot \left(\left\langle \tilde{i}\right|_A \rho_{ii',jj'} \left(\left|i\right\rangle_A \left\langle i'\right| \otimes \left|j\right\rangle_B \left\langle j'\right|\right) \left|\tilde{i}\right\rangle_A \right) \\ &= \sum_{i,j,j',k} T_{kj} \rho_{ii,jj'} (\left|k\right\rangle_B \left\langle j'\right|) \delta_{i,i'} \delta_{i,\tilde{i}} \delta_{j,k'} \end{aligned} \tag{A7}$$

which completes the proof.

References

1. Childs, A. Secure assisted quantum computation. *Quant. Inf. Compt.* **2005**, *5*, 456.
2. Arrighi, P.; Salvail, L. Blind Quantum Computation. *Int. J. Quant. Inf.* **2006**, *4*, 883–898.
3. Broadbent, A.; Fitzsimons, J.; Kashefi, E. Universal blind quantum computation. In Proceedings of the 50th Annual Symposium on Foundations of Computer Science, Atlanta, GA, USA, 25–27 October 2009; pp. 517–526.
4. Aharonov, D.; Ben-Or, M.; Eban, E. Interactive Proofs for Quantum Computations. In Proceedings of Innovations in Computer Science 2010, Beijing, China, 5–7 January 2010; pp. 453–469.
5. Reichardt, B.W.; Unger, R.F.; Vazirani, U. Classical command of quantum systems. *Nature* **2013**, *496*, 456–460.
6. Broadbent, A.; Gutoski, G.; Stebila, D. Quantum One-Time Programs. In *Advances in Cryptology—CRYPTO 2013*; Canetti, R., Garay, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Voume 8043, pp. 344–360.
7. Raussendorf, R.; Briegel, H.J. A One-Way Quantum Computer. *Phys. Rev. Lett.* **2001**, *86*, 5188–5191.
8. Nickerson, N.H.; Fitzsimons, J.F.; Benjamin, S.C. Freely scalable quantum technologies using cells of 5-to-50 qubits with very lossy and noisy photonic links. *Phys. Rev. X* **2014**, *4*, 041041.
9. Barz, S.; Kashefi, E.; Broadbent, A.; Fitzsimons, J.F.; Zeilinger, A.; Walther, P. Demonstration of blind quantum computing. *Science* **2012**, *335*, 303.

10. Fitzsimons, J.F.; Kashefi, E. Unconditionally verifiable blind computation. *arXiv* **2012**, arXiv:1203.5217.
11. Kapourniotis, T.; Kashefi, E.; Datta, A. Blindness and Verification of Quantum Computation with One Pure Qubit. In Proceedings of the 9th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2014), Singapore, 21–23 May 2014, pp. 176–204.
12. Gheorghiu, A.; Kashefi, E.; Wallden, P. Robustness and device independence of verifiable blind quantum computing. *New J. Phys.* **2015**, *17*, 083040.
13. Kapourniotis, T.; Dunjko, V.; Kashefi, E. On optimising quantum communication in verifiable quantum computing, 2015. In Proceedings of the 15th Asian Quantum Information Science Conference (AQISC 2015), Seoul, Korea, 25–28 August 2015, pp. 23–25.
14. Dunjko, V.; Fitzsimons, J.F.; Portmann, C.; Renner, R. Composable Security of Delegated Quantum Computation. In *Advances in Cryptology*; Springer: Kaoshiung, Taiwan, 2014.
15. McKague, M. Interactive proofs for BQP via self-tested graph states. *arXiv* **2013**, arXiv:1309.5675.
16. Broadbent, A. How to verify a quantum computation.*arXiv* **2015**, arXiv:1509.09180.
17. Gheorghiu, A.; Wallden, P.; Kashefi, E. Rigidity of quantum steering and one-sided device-independent verifiable quantum computation. *New J. Phys.* **2017**, *19*, 023043.
18. Dupuis, F.; Nielsen, J.B.; Salvail, L. Actively secure two-party evaluation of any quantum operation. In *Advances in Cryptology—CRYPTO 2012*; Springer: Santa Barbara, CA, USA, 2012; pp. 794–811.
19. Dunjko, V.; Kashefi, E. Blind quantum computing with two almost identical states. *arXiv* **2016**, arXiv:1604.01586.
20. Maurer, U.; Renner, R. Abstract cryptography. In *Innovations in Computer Science*; Citeseer: Beijing, China, 2011.
21. Dupuis, F.; Nielsen, J.B.; Salvail, L. Secure two-party quantum evaluation of unitaries against specious adversaries. In *Advances in Cryptology—CRYPTO 2010*; Springer: Santa Barbara, CA, USA, 2010; pp. 685–706.
22. Yao, A. How to generate and exchange secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science, Toronto, ON, Canada, 27–29 October 1986; pp. 162–167.
23. Kent, A. Unconditionally secure bit commitment. *Phys. Rev. Lett.* **1999**, *83*, 1447.
24. Goldwasser, S.; Kalai, Y.T.; Rothblum, G.N. One-time programs. In *Advances in Cryptology—CRYPTO 2008*; Springer: Santa Barbara, CA, USA, 2008; pp. 39–56.
25. Kashefi, E.; Wallden, P. Optimised resource construction for verifiable quantum computation. *J. Phys. A Math. Theor.* **2017**, *50*, 145306.
26. Bennett, C.; Brassard, G.; Ekert, A. Quantum cryptography. In *Progress in Atomic physics Neutrinos and Gravitation, Proceedings of the XXVIIth Rencontre de Moriond, Les Arcs, Savoie, France, 25 January–1 February 1992*; Atlantica Séguier Frontières: Biarritz, France, 1992; p. 371.
27. Broadbent, A.; Schaffner, C. Quantum cryptography beyond quantum key distribution. *Des. Codes Cryptogr.* **2016**, *78*, 351–382.
28. Alagic, G.; Fefferman, B. On quantum obfuscation. *arXiv* **2016**, arXiv:1602.01771.
29. Hayashi, M.; Morimae, T. Verifiable measurement-only blind quantum computing with stabilizer testing. *arXiv* **2015**, arXiv:1505.07535.
30. Dulek, Y.; Schaffner, C.; Speelman, F. Quantum homomorphic encryption for polynomial-sized circuits. *arXiv* **2016**, arXiv:1603.09717.
31. Kashefi, E.; Pappa, A. Blind multiparty quantum computing. **2016**, in preparation.
32. Abraham, I.; Dolev, D.; Gonen, R.; Halpern, J. Distributed Computing Meets Game Theory: Robust Mechanisms for Rational Secret Sharing and Multiparty Computation. In Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing, Denver, CO, USA, 23–26 July 2006; pp. 53–62.
33. Kol, G.; Naor, M. Cryptography and game theory: Designing protocols for exchanging information. In *Theory of Cryptography*; Springer: New York, NY, USA, 2008; pp. 320–339.
34. Eisert, J.; Wilkens, M.; Lewenstein, M. Quantum Games and Quantum Strategies. *Phys. Rev. Lett.* **1999**, *83*, 3077–3080.
35. Pappa, A.; Kumar, N.; Lawson, T.; Santha, M.; Zhang, S.; Diamanti, E.; Kerenidis, I. Nonlocality and Conflicting Interest Games. *Phys. Rev. Lett.* **2015**, *114*, 020401.
36. Childs, A.M.; Leung, D.W.; Nielsen, M.A. Unified derivations of measurement-based schemes for quantum computation. *Phys. Rev. A* **2005**, *71*, 032318.
37. Danos, V.; Kashefi, E.; Panangaden, P. The Measurement Calculus. *J. ACM* **2007**, *54*, 8.

38. Hein, M.; Eisert, J.; Briegel, H.J. Multiparty entanglement in graph states. *Phys. Rev. A* **2004**, *69*, 062311.
39. Naor, M.; Pinkas, B. Oblivious transfer and polynomial evaluation. In Proceedings of the Thirty-First Annual ACM Symposium on Theory of computing, Atlanta, GA, USA, 1–4 May 1999; pp. 245–254.
40. Gennaro, R.; Gentry, C.; Parno, B. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology—CRYPTO 2010*; Springer: Santa Barbara, CA, USA, 2010; pp. 465–482.
41. Danos, V.; Kashefi, E. Determinism in the one-way model. *Phys. Rev. A* **2006**, *74*, 052310.
42. Browne, D.E.; Kashefi, E.; Mhalla, M.; Perdrix, S. Generalized flow and determinism in measurement-based quantum computation. *New J. Phys.* **2007**, *9*, 250.
43. Goldreich, O.; Micali, S.; Wigderson, A. How to Play ANY Mental Game. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 25–27 May 1987; pp. 218–229.
44. Kashefi, E.; Music, L.; Wallden, P. The Quantum Cut-and-Choose Technique and Quantum Two-Party Computation. *arXiv* **2017**, arXiv:1703.03754.

© 2017 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).