Succinct Blind Quantum Computation using a Random Oracle

Jiayu Zhang
Boston University
The United States
jyz16@bu.edu

ABSTRACT

In the universal blind quantum computation problem, a client wants to make use of a single quantum server to evaluate $C|0\rangle$ where C is an arbitrary quantum circuit while keeping C secret. The client's goal is to use as few resources as possible. This problem, first raised by Broadbent, Fitzsimons and Kashefi [FOCS 2009], has become fundamental to the study of quantum cryptography, not only because of its own importance, but also because it provides a testbed for new techniques that can be later applied to related problems (for example, quantum computation verification). Known protocols on this problem are mainly either information-theoretically (IT) secure or based on trapdoor assumptions (public key encryptions).

In this paper we study how the availability of symmetric-key primitives, modeled by a random oracle, changes the complexity of universal blind quantum computation. We give a new universal blind quantum computation protocol. Similar to previous works on IT-secure protocols (for example, Broadbent-Fitzsimons-Kashefi), our protocol can be divided into two phases. In the first phase the client prepares some quantum gadgets with relatively simple quantum gates and sends them to the server, and in the second phase the client is entirely classical — it does not even need quantum storage. Crucially, the protocol's first phase is *succinct*, that is, its complexity is independent of the circuit size. Given the security parameter κ, its complexity is only a fixed polynomial of κ, and can be used to evaluate any circuit (or several circuits) of size up to a subexponential of κ. In contrast, known schemes either require the client to perform quantum computations that scale with the size of the circuit, or require trapdoor assumptions.

CCS CONCEPTS

• **Theory of computation → Cryptographic protocols**; • **Security and privacy → Mathematical foundations of cryptography**.

KEYWORDS

Quantum Cryptography; Blind Quantum Computation; Random Oracle

ACM Reference Format:
Jiayu Zhang. 2021. Succinct Blind Quantum Computation using a Random Oracle. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21), June 21–25, 2021, Virtual, Italy.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3406325.3451082

STOC '21, June 21–25, 2021, Virtual, Italy

ACM ISBN 978-1-4503-8053-9/21/06.
https://doi.org/10.1145/3406325.3451082

1 INTRODUCTION

1.1 Problem Background

In the universal blind quantum computation problem, a client wants to make use of a single quantum server to evaluate a quantum circuit C secretly, where C can be arbitrary (up to a subexponential size). The protocol should at least satisfy the following requirements:

(1) (Correctness) When the server is honest, the client holds $C|\varphi\rangle$ in the end of the protocol with probability negligibly close to 1.
(2) (Security) For any adversarial server, which might be unbounded, polynomial, etc, depending on the setting, it cannot distinguish whether the current protocol is run on input C, or run on input $0^{|C|}$.
(3) (Efficiency) When the protocol is run honestly, the client and the server should be in polynomial time.

For simplicity, we choose $|\varphi\rangle = |0\rangle$ for the presentation below. The protocol can be adopted for the general cases.

This problem is important for two reasons.

First, the problem itself is fundamental. The related problems in the classical world, like delegation of computation, multiparty computation or homomorphic encryption, are all well-studied and fundamental problems with a very long history. The blind quantum computation problem is important for the same reasons. In quantum world there is one more reason to study this problem: It's very possible that the quantum computers will mainly be used as a cloud service. If a client wants to use the power of a remote quantum server and simultaneously wants to keep its data or circuits secret, a blind quantum computation protocol is needed.

Second, blind quantum computation is a testbed for new techniques. Historically, techniques developed for blind quantum computation have subsequently been useful for many other problems, including quantum computation verification, multiparty quantum computation, certifiable randomness, zero knowledge proofs for QMA and so on. For example, the UBQC protocol [3] for blind quantum computation became the foundation of the UVBQC protocol for quantum computation verification [8]; the trapdoor claw-free function techniques [14] led to a series of works for quantum computation verification [15], certifiable randomness [2], zero-knowledge arguments [19] and so on.

In classical world, this problem was studied under the names of two party computation and fully homomorphic encryption. We note that these concepts are not the same, but they are closely related and aiming at the same goal: to delegate the computation while keep it (or the data) secure. There are two fundamental constructions in classical world: one is the garbled circuit, or garbled table, raised by

Yao [20]; another construction is the fully homomorphic encryption [9], or FHE.

1.2 Previous Works and Motivating Questions

Previous protocols for universal blind quantum computation require either the execution of many quantum gates — proportional to the size of the circuit [3] — but not computational assumptions; or the existence of trapdoor cryptographic primitives, such as the quantum hardness of learning with errors (LWE [18]) [14]. (See Table 1 for a summary of existing works.) There are also protocols that use two separated quantum servers [3] and some protocols that are not universal [4, 13, 17, 21]; in this paper we focus on the universal protocols using a single quantum server.

- One representative of information-theoretically (IT) secure protocols is the UBQC protocol [3]. This protocol is based on the measurement-based quantum computation. It contains an offline phase and an online phase. In the offline phase the client sends many quantum gadgets to the server. These quantum gadgets can be prepared using single-qubit gates, but the total number of gadgets is linear in the size of the circuit to be evaluated, which is prohibitive. This protocol became the basis of many later works.
- Some early representatives of computationally-secure protocols include [7], which is based on LWE. Then in [14] a classical fully homomorphic encryption for quantum circuits was constructed. That protocol is based on the classical FHE and a new technique based on a primitive called "trapdoor claw-free functions". Both primitives were constructed from LWE.

One way to classify these assumptions is through the "Impagliazzo's Five Worlds"[10]. IT-secure protocols remain secure in all of these worlds, since it does not rely on any computational assumption; and the trapdoor assumptions, FHE, LWE and many "fancy" cryptographic schemes and protocols are secure only in Cryptomania (a world in which trapdoor primitives exist).

Minicrypt, intuitively, is the world where symmetric cryptography (for example, pseudorandom generators) exists but public key cryptography (trapdoor functions) is not possible. Our motivation is to understand what sorts of cryptography are possible in the quantum analogue of Minicrypt. We work with an abstraction, the QROM, which assumes (1) all parties have oracle access to a common function which is chosen uniformly at random; (2) the adversary is unbounded, but can only makes polynomial number of quantum random oracle queries. This setting allows for symmetric-type primitives (one-way functions, pseudorandom generators, collision-free hash functions), but excludes "public key primitives". By the "Random Oracle Methodology", once we design a protocol in this setting, we can replace the random oracle by an appropriate hash function or symmetric key encryption scheme. (We note that although this setting itself is formal, the instantiation of protocols proved secure in this setting can be subtle: there do exist some constructions that are not possible to be instantiated [5]. However, the usage of the random oracle as an ideal model of hash functions or symmetric key encryption schemes is wide-spread, and has greatly helped the design of cryptographic protocols in the past three decades [12].)

Besides the theoretical motivation, the protocols in this setting have the following advantage: currently there are few choices for post-quantum secure public key encryption schemes[11, 18]. If we want to instantiate some more specific and stronger primitives, like trapdoor claw-free functions or FHE, currently the only known way is through the lattice-based cryptography (for example, LWE). On the other hand, there are many choices for symmetric key primitives, and the protocols can remain to be sound even if lattice-based cryptography is broken.

The design of delegation-style quantum protocols in this setting is not well-understood. As far as we know, except the works on IT-secure protocols, the only work is [21], which designed a quantum delegation (blind quantum computation) protocol for a useful but specific circuit family. Thus we ask the following question:

How does the availability of symmetric-key cryptographic primitives (modeled by a random oracle) change the complexity of universal blind quantum computation?

Another factor that we will consider is the "client side quantum computation". Existing works assume either the client side quantum gates can be linear to the circuit size (*during the whole protocol*), or the client is classical; little is known for the setting between them, which is, to allow the client to run succinct quantum operations, which can depend on the security parameter, but should be independent of the circuit size. Thus, we can ask the following question:

How can we design a universal blind quantum computation protocol in which the client side quantum operations are "succinct" (that is, independent of the size of the circuit to be evaluated)?

Thus we want to design a protocol that is more efficient than the IT-secure protocol in terms of the client side quantum operations (here we do not care about the classical computation and communication as long as they are polynomial size), and does not use any public key primitives. None of the existing techniques works for this setting and we need to develop new techniques and a new protocol.

1.3 Our Results

In this paper we prove the following:

Theorem 1.1. *There exists a universal blind quantum computation protocol for circuits of size up to a fixed subexponential function of the security parameter such that*

- *It contains an offline phase and an online phase. In the offline phase the client prepares and sends some quantum gadgets to the server, and in the online phase the client is completely classical.*
- *The total number of quantum gates to prepare these quantum gadgets is at most a fixed polynomial of the security parameter, independent to the size of the circuit to be evaluated.*
- *The classical computation, communication and the server-side quantum computation are bounded by a fixed polynomial of the security parameter and the size of the circuit to be evaluated.*
- *The protocol is secure in the quantum random oracle model against any unbounded malicious server whose number of queries to the random oracle is bounded by a fixed subexponential function of the security parameter.*

Thus, based on our work, together with previous works [3, 14] we can complete the following table (Table 1) about the different tradeoffs between client side quantum resources and assumptions. Thus we claim our result reveals a more complete cryptographic picture for single-server quantum (blind) delegation problem.

Our result required the development of a set of new techniques for protocol design and security proof. Section 1.4 provides a brief technical overview. As discussed before, new techniques in blind quantum computation often led to protocols for many related problems. We hope the techniques and protocols developed here will also lead to advances on a range of related problems.

1.4 A Top-down Overview of Our Techniques

1.4.1 Two-Step Construction via Remote Gadget Preparation. How can the client allow the server to evaluate $O(|C|)$ gates using only succinct quantum computation? In our protocol, the client will first prepare $\text{poly}(\kappa)$ *gadgets* (poly is a fixed polynomial), then use classical interactions to allow the server to *expand* them to $O(|C|)$ gadgets "securely". Here the *gadget* is defined to be the states in the form of $|y_0\rangle + |y_1\rangle$, where y_0, y_1 are random different strings, or *keys*. The client holds the keys, and the server should hold the state.[1]

This step — the preparation and expansion of gadgets — is called *remote gadget preparation*. Let's give the correctness and security definition informally below (see the full paper [22] for the formal definition):

Definition 1.1 (Remote gadget preparation, informal)**.** The correctness and security of remote gadget preparation are defined as follows.

- (Correctness) When the output number is L, the protocol should output key set $\{y_b^{(i)}\}_{i\in[L],b\in\{0,1\}}$ on the client side and gadgets $\otimes_{i=1}^{L}(|y_0^{(i)}\rangle + |y_1^{(i)}\rangle)$ on the server side.
  And we say it's an $N \to L$ protocol if the protocol starts from N gadgets (the server holds $\otimes_{i=1}^{N}(|x_0^{(i)}\rangle + |x_1^{(i)}\rangle)$ and the client knows all the keys) and ends with L gadgets in the honest setting.
- (Security) For any key pair (say, the i-th keys), a malicious server could not output both keys (which means $y_0^{(i)}||y_1^{(i)}$) with non-negligible probability from the protocol's output state, even if the following information is provided additionally (which makes the adversary more powerful):
  - All the other keys that are not at index-i.
  - The hash tags of $y_0^{(i)}$ and $y_1^{(i)}$.

Which means, after the protocol completes, at least one key in each key pair should be unpredictable, and this unpredictability is not correlated to the unpredictability of the other key pairs.

Then we will construct our universal blind quantum computation protocol that satisfies Theorem 1.1 (denoted SuccUBQC) as follows:

Outline 1. *Design of the* SuccUBQC *protocol:*

(1) *Remote gadget preparation: (1) the client sends some initial gadgets to the server, whose size and length are succinct; (2) the client uses classical interactions to allow the server to expand the number of gadgets securely to* $\Theta(|C|)$.
(2) *Blind quantum computation execution: using the gadgets output from the previous step, the client and the server evaluate* C *using only classical interactions.*

The construction of the secure remote gadget preparation protocol (the first step in Outline 1) is the most difficult step. The second step is relatively easier but still non-trivial.

1.4.2 Remote Gadget Preparation via Weak Security. The first step of Outline 1 are achieved as follows: we will define the *weak security* of the remote gadget preparation. We will first construct a weakly-secure protocol, then *amplify* it to a fully secure one. First we need to develop a framework for the design of different subprotocols. This is (mainly) captured by the notion of *weak security*. Informally speaking, in weak security the adversary is possible to output both keys in some key pair with bounded (but not necessarily negligible) probability. (For comparison, in the full security, the adversary is not possible to output both keys — except with negligible probability).

Definition 1.2 (Weak security of a state, informal)**.** We say a state is $(2^\eta, C)$-SC-secure for a key pair K if for any adversary with query number at most 2^η, with access to the hash tags of keys in K, the norm of outputting both keys in K from this state is at most C.

Here "SC" means "simultanously compute", which is the notion that we define to characterize the weak security.

Then a protocol is weakly secure if the output has some reasonable weak security for each out key pair. (Similar to Definition 1.1, the weak security of each output key pair should still hold when the other key pairs are revealed to the adversary, which prevent some potential correlation between weak security of different pairs.)

Then the overall flow of the construction of the secure remote gadget preparation protocol (step 1 of Outline 1) is as follows [2]:

Flow of Construction 1. *Protocol construction for the step 1 of Outline 1:*

(1) *Construct a weakly-secure remote gadget preparation protocol such that it can (asymptotically multiplicatively) generate more gadgets than it consumes.*
(2) *Use some amplification techniques to amplify it to a secure remote gadget preparation protocol.*

Our work can be seen as the design of a series of subprotocols with different tradeoffs for correctness, (weak) security, etc, and these subprotocols, when combined together, can achieve what we want.

Below we give a brief introduction to each step above.

Weakly Secure Protocol Step. The goal in this step is to create more gadgets (possibly with weak security) from some input gadgets. First, we consider the simplest case, generating two gadgets using one input gadget, remotely:

$$|x_0\rangle + |x_1\rangle \to (|y_0\rangle + |y_1\rangle) \otimes (|y_0'\rangle + |y_1'\rangle)$$

Which stands for a protocol in the following style:

(0) Initially the server holds $|x_0\rangle + |x_1\rangle$.
(1) The client provides some classical information to the server.
(2) The server uses these information to do the transformation above.

[1] This form of states was also previously used in several papers like [2].

[2] This is only a construction flow, we don't mean there is a two-step protocol.

Table 1: Different tradeoff between client side quantum operations and assumptions in quantum computation delegation problem. "Succinct" means it's at most a fixed polynomial of the security parameter; and "Linear" means it's linear to the size of the circuit to be evaluated.

Client side quantum computation	IT-secure	QROM (Idealized symmetric key primitives)	LWE (Public key encryption with functionalities)
Classical	May be impossible [1]	Unknown	[14]
Succinct	Unknown	**This paper**	
Linear	[3]		[7]

And we want the outputs to have some (possibly weak) security in the malicious setting.

First, when the input and output number are the same, a similar form of quantum-to-quantum transformation can be enabled by providing some additional classical information called *reversible look-up tables*. As the preliminary, we give a simple example on encoding one-to-one gadget transformation and two-to-two gadget transformation:

Example 1.1. To transform $|x_0\rangle + |x_1\rangle$ to $|y_0\rangle + |y_1\rangle$ while preserving the security of keys (in the sense of being unable to computing both keys in each key pair), the client can provide the following ciphertexts to the server:

$$\text{Forward table: } \mathsf{Enc}_{x_0}(y_0), \mathsf{Enc}_{x_1}(y_1)$$

$$\text{Backward table: } \mathsf{Enc}_{y_0}(x_0), \mathsf{Enc}_{y_1}(x_1)$$

And the server-side operation goes as follows:

$$|x_0\rangle + |x_1\rangle \tag{1}$$

$$\xrightarrow{\text{decrypt forward table in superposition}} |x_0\rangle|y_0\rangle + |x_1\rangle|y_1\rangle \tag{2}$$

$$\xrightarrow{\text{decrypt backward table in superposition}} |y_0\rangle + |y_1\rangle \tag{3}$$

$$\tag{4}$$

Example 1.2. To transform $(|x_0\rangle + |x_1\rangle) \otimes (|x_0'\rangle + |x_1'\rangle)$ to $(|y_0\rangle + |y_1\rangle) \otimes (|y_0'\rangle + |y_1'\rangle)$ while preserving the security of keys (in the sense of being unable to computing both keys in each key pair), the client can provide the following ciphertexts to the server:

$$\text{Forward table: } \mathsf{Enc}_{x_b x'_{b'}}(y_b y'_{b'}) \text{ for each } b, b' \tag{5}$$

$$\text{Backward table: } \mathsf{Enc}_{y_b y'_{b'}}(x_b x'_{b'}) \text{ for each } b, b' \tag{6}$$

And the server-side operation goes similarly:

$$(|x_0\rangle + |x_1\rangle) \otimes (|x_0'\rangle + |x_1'\rangle) \tag{7}$$

$$\xrightarrow{\text{decrypt forward table}} \sum_{b,b'} |x_b\rangle|x'_{b'}\rangle|y_b\rangle|y'_{b'}\rangle \tag{8}$$

$$\xrightarrow{\text{decrypt backward table}} (|y_0\rangle + |y_1\rangle) \otimes (|y_0'\rangle + |y_1'\rangle) \tag{9}$$

$$\tag{10}$$

Here the input state contains four components in the standard basis, and the output state also has four components. When we do the back-and-forth encryption, we need to find a one-to-one correspondence between the input components and output components. And we note there is some freedom here: in (5)(6) we match the input component to the output component with the same index; however we can also match them in some other way. This will be useful later.

But it's not easy to achieve a one-to-two gadget transformation directly here. We will introduce several ideas to achieve this transformation securely. First we rewrite this one-to-two mapping to an equivalent form, a two-to-two mapping where one of the input gadget is described classically[3] (here we relabel the superscripts to make them consistent with later sections):

$$\underbrace{|\{x_0^{(2)}, x_1^{(2)}\}\rangle}_{\text{classical}} \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle) \to (|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle) \tag{11}$$

Which is an abbreviation of some protocol in the following style:

(0) Initially the server holds $|x_0^{(3)}\rangle + |x_1^{(3)}\rangle$.

(1) The first gadget $\{x_0^{(2)}, x_1^{(2)}\}$ is directly provided to the server in the form of its classical description, and the server can prepare the gadget on its own. Note that this step only uses classical communication. Then the client can send some information to complete this transformation using the previous two-to-two gadget transformation:

$$|\{x_0^{(2)}, x_1^{(2)}\}\rangle \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle) \tag{12}$$

$$\to (|x_0^{(2)}\rangle + |x_1^{(2)}\rangle) \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle) \tag{13}$$

$$\to (|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle) \tag{14}$$

Now the honest behavior is supported, but the security breaks down: providing both keys to the server will allow the server to get both keys in some output key pair.

We will use rescue the security by combining the following two ideas and get the first weakly-secure gadget increasing protocol.

- A secret bit-wise permutation on the output keys;
- Usage of (a revised) Hadamard test and two different types of encodings of the two-to-two mapping.

The combination of these two ideas will restrict the adversary's behavior powerfully and give the protocol weak security. This achieves a fundamental and important step in this work.

The first key step is, instead of implementing this transformation directly, we seek for a transformation to the following state as an intermediate step:

$$|\{x_0^{(2)}, x_1^{(2)}\}\rangle \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle) \to \pi((|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle))$$

[3] It's possible to encode the mapping as $1 \leftrightarrow 2$ mapping directly, but we choose to encode it in this way for nicer honest setting behavior.

where π is a random bit-wise permutation sampled by the client, kept secret from the adversary. Suppose each output key has length κ_{out}, then π permutes the bit positions of these $2\kappa_{out}$ bits.

The secrecy of π will be a key ingredient on implementing this mapping securely. But the client still needs to reveal it to the server to allow it to de-permute the gadgets in the end, which seems to be a dilemma. Now we introduce the second idea: the realization of the mapping above will make use of an extra helper gadget, and a subprotocol called *padded Hadamard test*. This padded Hadamard test is a padded variant of the Hadamard test in [2]. We observe that, this revised Hadamard test has several powerful properties, one of which informally say, if such a test is executed between the client and the server, if the server wants to pass the test with high probability, it loses the ability to predict the keys from the post-test state — a property that we call *unpredictability restriction*. With this property in mind, we can use this subprotocol as a switch that controls when it's safe to reveal the permutation. Now the transformation goes as follows, where we use $|x_0^{\text{helper}}\rangle + |x_1^{\text{helper}}\rangle$ to denote the helper gadget:

$$(|x_0^{\text{helper}}\rangle + |x_1^{\text{helper}}\rangle) \otimes |\{x_0^{(2)}, x_1^{(2)}\}\rangle \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle) \tag{15}$$

$$\to (|x_0^{\text{helper}}\rangle + |x_1^{\text{helper}}\rangle) \otimes \pi((|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle)) \tag{16}$$

$$\text{(Hadamard test on } |x_0^{\text{helper}}\rangle + |x_1^{\text{helper}}\rangle) \tag{17}$$

$$\to \pi((|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle)) \tag{18}$$

$$\text{(Client reveals } \pi \text{ if test passes)} \tag{19}$$

$$\to (|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle) \tag{20}$$

Then in each time step something is secret in the adversary's viewpoint. Before the test the bit-wise permutation is hidden, and after the test, the adversary is not able to have good predictability on $\{x_0^{\text{helper}}, x_1^{\text{helper}}\}$ anymore. The security of the protocol relies on this fact.

Now we go to the construction of (15)→(16). Expanding it a little bit, it is

$$(|x_0^{\text{helper}}\rangle + |x_1^{\text{helper}}\rangle) \otimes \underbrace{|\{x_0^{(2)}, x_1^{(2)}\}\rangle}_{\text{given classically}} \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle) \tag{21}$$

$$\xrightarrow[\text{forward table encoding}]{\text{backward table encoding}} \begin{matrix} x^{(2)}x^{(3)} \text{ under } \pi(y^{(2)}y^{(3)}) \\ \pi(y^{(2)}y^{(3)}) \text{ under } x^{(2)}x^{(3)} \end{matrix} \tag{22}$$

$$(|x_0^{\text{helper}}\rangle + |x_1^{\text{helper}}\rangle) \otimes \underbrace{\pi((|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle))}_{\text{reversibly encoded part}} \tag{23}$$

Which follows Example 1.2 and adds the permutation π on the output keys part. However, this is still not sufficient to guarantee the weak security on both of the output keys. But there is still some freedom on the design of mapping in (22), and we will make use of it. Here the final ingredient is, when we encode the mapping above, we design the underlying mapping for the reversibly encoded part carefully. Note that the reversible encoding of two gadgets to two gadgets have multiple ways of encoding. We consider two different encodings for the reversibly encoded part: the *CNOT-style mapping* and the *identity-style mapping*:

Identity-style mapping: for each $b, c \in \{0, 1\}$,

$$e_0 := \left[\begin{matrix} \text{backward table encoding} & x_b^{(2)}x_c^{(3)} \text{ under } \pi(y_b^{(2)}y_c^{(3)}) \\ \text{forward table encoding} & \pi(y_b^{(2)}y_c^{(3)}) \text{ under } x_b^{(2)}x_c^{(3)} \end{matrix} \right]$$

CNOT-style mapping: $b, c \in \{0, 1\}$

$$e_1 := \left[\begin{matrix} \text{backward table encoding} & x_b^{(2)}x_c^{(3)} \text{ under } \pi(y_b^{(2)}y_{b+c}^{(3)}) \\ \text{forward table encoding} & \pi(y_b^{(2)}y_{b+c}^{(3)}) \text{ under } x_b^{(2)}x_c^{(3)} \end{matrix} \right]$$

Here we use e_0 and e_1 to denote the encryption results of these mappings. We note both encodings support (15)→(16), but neither can make it secure when used alone. The idea is to use both, and associate them to different *branches* of the helper gadget. That is, the client sends the following to the server for step (15)→(16):

$$\text{Enc}_{x_0^{\text{helper}}}(e_0) || \text{Enc}_{x_1^{\text{helper}}}(e_1) \tag{24}$$

And this can be analyzed as follows:

- The honest behavior is supported. The honest server can first decrypt e_0 and e_1 coherently using the helper gadget and (24):

$$|x_0^{\text{helper}}\rangle + |x_1^{\text{helper}}\rangle \xrightarrow{\text{decrypt (24)}} |x_0^{\text{helper}}\rangle |e_0\rangle + |x_1^{\text{helper}}\rangle |e_1\rangle \tag{25}$$

  Then since both e_0 and e_1 support the two-to-two mapping on the reversibly encoded part, server could still implement the honest mapping coherently on keys with superscript (2)(3).
  Finally (25) is applied again to erase the $|e\rangle$ register and the helper gadget is recovered.
- For the security, we explore a powerful property of our Hadamard test called *coherency restriction*, which put a restriction that the behavior of the adversary on two branches of the helper gadget should not be too different. (Here "branch" means either $|x_b^{help}\rangle$ tensoring the other parts for each $b = 0, 1$.) But on the other hand, these two branches restrict the adversary's behavior in different ways (and here the bit-wise permutation also comes in to restrict the adversary's operation), which restricts a cheating server's behavior powerfully.
  Putting it in a more intuitive way, what the protocol achieved can be explained as follows. Without the helper gadget, the adversary can get either e_0 or e_1 in the clear. However, after we introduce the helper gadget and encrypt these two reversible encodings in the form of (24), the server can only use them in the form of the right hand side of (25). And any meaningful attack of the adversary will collapse of the two branches of (25) — which means the adversary will not be able to recover the helper gadget and will not be able to pass the Hadamard test. We will give a more formal proof in Section 3.

In this way we can construct a weakly secure protocol, which is a basic subroutine in our paper. Overall speaking, what we have achieved could be understood as follows. With the cost of one gadget (the helper gadget), the input gadget on the third wire in (21) is "technically teared up" into two gadgets, with securities weaker than the input. (The recovery of the security will be done in the later amplification step.)

But the simple weakly secure protocol above is still not gadget-increasing. The reason is when we save one gadget, we also consume one. But this problem can be solved through a parallel-repetition-style step, and note the helper gadget (consumed in the padded Hadamard test) can be shared in each table. Then we get an $1+n \to 2n$ protocol, which is provable to be both gadget-increasing and weakly-secure.

Amplification Step. After we complete the first step in the Flow of Construction 1, we move to the amplification part. We give it an overview here and put the full description in the full version.

Roughly speaking, we call our technique *repeat-and-combine*. The *repeat* part is a parallel repetition of the weakly secure protocol on many different blocks, and the server is required to pass on all the blocks. The overall protocol still asymptotically doubles the number of gadgets. And this gives the upper-level protocol better weak security than the underlying protocol.

The main challenge is to go from weak security to normal (exponential) security. The *combine* technique combines multiple gadgets into one gadget to reduce the server's ability to compute both keys in a key pair. Let's give a minimal example of our *combine* technique, where only two key pairs are combined.

Suppose the server holds $(|x_0\rangle + |x_1\rangle) \otimes (|x_0'\rangle + |x_1'\rangle)$, while the client knows all the keys. Additionally suppose the server knows the hash tags of all the keys. Then it can make a measurement on the xor of the indexes:

$$(|x_0\rangle + |x_1\rangle) \otimes (|x_0'\rangle + |x_1'\rangle) \to$$

$$(output = 0)(|x_0\rangle\,|x_0'\rangle + |x_1\rangle\,|x_1'\rangle) \quad (output = 1)(|x_0\rangle\,|x_1'\rangle + |x_1\rangle\,|x_0'\rangle)$$

then it reports the measurement result to the client, and the client can compute and store the new output key pairs ($\{x_0x_0', x_1x_1'\}$ or $\{x_0x_1', x_1x_0'\}$). Intuitively, if the server can output both keys in the output key pair, intuitively it means it not only know both x_0 and x_1, but also know x_0' and x_1'. Thus we can hope the adversary's ability of computing both keys for the new key pair is proportional to the multiplication of the corresponding security bounds for the two input key pairs. And if we continue this combination sequentially and combine $\sqrt{\kappa}$ key pairs one-by-one, we can hope this parameter goes down to an exponentially small value.

However the story is not that simple. As far as we know, such a simple combination does not always imply the multiplicativity of the bounds of the adversary's ability of computing both keys. To solve this problem, we add an additional layer — called SecurityRefreshing layer — in the middle of each round of the combination process. This additional layer can be used to "strengthen the security" in each round. It makes use of (and consumes) some "freshly secure" gadgets, but the consumption is small and it can refresh the security of a large number of key pairs simultaneously. And we can prove, the new protocol, with the *combine* technique and this additional layer, is exponentially secure (in the sense of Definition 1.1).

This is still not the end of the story. The combination part decreases the number of gadgets multiplicatively by a factor of $\kappa^{O(1)}$ and thus we need to do more to make it gadget-increasing again! The solution is, before we do this *repeat* and *combine*, we need to first self-compose the $1 + n \to 2n$ protocol to boost the gadget-expansion ratio (defined by output gadget number divided by input gadget number) from 2 to $\tilde{\Theta}(\kappa)$. Then since we can only combine $\sqrt{\kappa}$ gadgets in the combine process into one gadget the overall gadget expansion ratio is still $\tilde{\Theta}(\sqrt{\kappa}) > 2$.

Finally we get a remote gadget preparation protocol that is gadget-increasing (with gadget expansion ratio > 2) and secure (not just weakly-secure). Intuitively we can simply run this protocol again and again to increase the number of gadgets until we get enough gadgets. Again, we make use of the SecurityRefreshing layer to bypass the obstacles in the security proof.

1.4.3 Security Proof Techniques. Formally proving the security, especially for the *combine* technique part, turns out to be technically challenging. One of the most important idea is a series of *state decomposition lemmas* for the quantum random oracle model, which might be of independent interest. These lemmas serve as a bridge from weak security to normal security in our setting.

Let's give a simple, informal example for that. See Section 2.1 for more detailed definitions for notations. Assume the joint purified state of all the parties is described by a normalized state $|\varphi\rangle$ (which means, we first use a cq-state to describe the state of all the parties, where the server's inner state is the quantum part; and then we use the environment to purify all the randomness in the client side, random oracle side, etc). And we want to study the server's ability to predict a single key stored in some client-side register, denoted as x_0. The condition is, assume for any server-side operation $\mathcal{U}$ which makes at most 2^κ queries to the random oracle, there is

$$|P_{x_0}\mathcal{U}\,|\varphi\rangle\,| \le A \tag{26}$$

(Again, the server should know some hash tag of x_0.)

Then we can prove, (technically nontrivially,) the state, together with some server-side ancillas, can be decomposed into the linear sum of two states $|\phi\rangle + |\chi\rangle$ where

- $|\phi\rangle$ is $(2^{O(\kappa)}, 2^{-O(\kappa)})$-unpredictable for x_0 — which means, for any server-side operation $\mathcal{U}$ which makes at most $2^{O(\kappa)}$ queries to the random oracle, there is $|P_{x_0}\mathcal{U}\,|\phi\rangle\,| \le 2^{-O(\kappa)}$. Compare to (26), the right hand side of (26) becomes exponentially small.
- The norm of $|\chi\rangle$ can be bounded: $|\,|\chi\rangle\,| \le (\sqrt{2}+1)A$.
- Both states can be written in a well-behaved form using $|\varphi\rangle$.

Furthermore, this decomposition, in some cases, can be iterated. This will be useful in the security proof of the *combine* technique.

1.4.4 From Remote Gadget Preparation to SuccUBQC. Now we give an informal overview of how to reduce the universal blind quantum computation problem to the remote gadget preparation problem.

Recall that in BFK's UBQC protocol [3], to delegate a circuit C, the client needs to prepare the state $|+_{\theta^i}\rangle$, $\theta^i \in_r \{n\pi/4 : n = 0, \cdots 7\}$ for each gate g^i in C. Thus the total number of client side quantum computation is linear in $|C|$. One natural idea is to delegate the preparation of these states further; such a primitive for preparing secret single qubit states is abstracted and formalized into a concept called *8-basis qfactory* [6]. However there is an important difference of our setting from [6] here: we cannot delegate the preparation of these single qubit state "from scratch"; instead, we make use of the output of the remote gadget preparation protocol.

Here we only informally describe our protocol, which simplifies it a lot. First, the client and the server run a remote gadget preparation protocol, with output number $L := \Theta(|C|)$. The honest server will get the gadgets $\otimes_{i=1}^{L}(|y_0^{(i)}\rangle + |y_1^{(i)}\rangle)$. Then our qfactory protocol (simplified due to the assumption that the gadgets are in the honest form) works as follows, which transforms $|y_0^{(i)}\rangle + |y_1^{(i)}\rangle$ into a single qubit state (which can then be used in the BFK protocol). One interesting trick is the usage of phase lookup table, which is given in [21] and it's convenient to use it for 8-basis qfactory.

$$|y_0^{(i)}\rangle + |y_1^{(i)}\rangle \tag{27}$$

$$(\textit{Phase lookup table}) \Rightarrow |0\rangle\,|y_0^{(i)}\rangle + e^{\mathrm{i}\tilde{\theta}^i}\,|1\rangle\,|y_1^{(i)}\rangle \tag{28}$$

$$(\tilde{\theta}^i = \theta_2^i\pi/2 + \theta_3^i\pi/4,\ \theta_2^i, \theta_3^i \in_r \{0,1\}^2) \tag{29}$$

$$(\textit{Hadamard } [15]) \Rightarrow |+_{\theta^i}\rangle\,(\theta^i = \theta_1^i\pi + \theta_2^i\pi/2 + \theta_3^i\pi/4) \tag{30}$$

1.5 Discussions

This result naturally gives rise to the following questions:

(1) How can we use these techniques on other problems? (for example, quantum computation verification, or zero-knowledge proof.)
(2) Is it possible to do universal blind quantum computation using completely classical client and quantum random oracle model, and make it secure again any unbounded adversary which only makes polynomial number of queries? Is it possible to do universal blind quantum computation using succinct client side quantum computation without relying on any assumptions? We conjecture the answer is no, but we need a formal proof for it.
(3) Is it possible to directly base the protocol on standard model assumptions (quantum-secure oneway functions, or hash functions?)
(4) Is it possible to make it efficient enough for practical usage? Although the complexities of the client and server are polynomial time, the polynomials are very big and not practical. We hope further works can improve on the actual complexities.

One intuitive way to think about the future direction is through Table 1. There are many unknown cells in this table, and the completion of this table will be interesting. And one interesting thing is: similar (although not the same) tradeoffs also appear in many other problems, not restricted to the delegation-style quantum protocols. For example, in the classical world, for the "secure key agreement" problem, symmetric key encryption scheme allows two parties to expand succinct size of pre-shared keys; to achieve key agreement without pre-shared keys, public key encryption is necessary. Thus we wonder how fundamental it is in quantum (or not only quantum) cryptography.

2 PRELIMINARIES AND NOTATIONS

2.1 Basics of Quantum Computation and Basic Notations

We refer to [16] for a basic understanding of quantum computation. In this paper we use $|\cdot|$ to denote the norm of a state. And we use P_S to denote the projection onto a subspace S.

The protocol in our work contains many parties including the client, the server, the random oracle, and the environment. Some parties are classical and some party is quantum. Thus the joint state of all the parties can always be described as a cq-state. Then we introduce the notion of "purified joint state", which purifies this state and provides a brief way to describe the joint state:

Definition 2.1 (Purified joint state)**.** We say "purified joint state" of many parties to mean a pure state defined as follows. Consider the inner state of all the parties in the protocol we consider. (Including client, server, random oracle.) This can be described as a cq-state where the server is the quantum party and the client and the random oracle are classical. However, we can purify the classical randomness in this cq-state with the environment and it becomes a highly entangled pure state among client, server, random oracle, and the environment. This allows us to use Dirac symbol to describe the inner state of all the parties briefly.

Thus when we describe the joint state using this purified notation, when we discuss some keys known by the client, like $K = \{x_0, x_1\}$, we are actually discussing client side registers that store these keys — which are in quantum state after the purification. So there is an abuse of notation: in the purified notation x_0, x_1 means the client side key register, while in the usual notation (like when we discuss the honest behavior) they are just key values.

In this purified notion, some common operation like "the client sends a message to the server" should be seen as a quantum operation on the state. We introduce a notation for it.

Notation 2.1. Suppose $|\varphi\rangle$ is a purified joint state of a client, a server, and some other parties. X is either a client side system or a classical algorithm that takes some client side systems as inputs. We use $|\varphi\rangle \odot X$ to denote the following operation: The X register (if X is an algorithm, we compute it and put the result in a register of the same name) is copied to some server side unused register.

Finally, we clarify the projection operator notation on this purified joint state notion.

Notation 2.2. When we P_{x_0} on a purified joint state where x_0 denote a client's key system, it means the projection of some system onto the space that is equal to the content of x_0 register.

And if we run a protocol $ProtocolName(K; Parameters)$ on input purified joint state $|\varphi\rangle$ against adversary Adv, the final state can be written as $|\varphi'\rangle = ProtocolName_{\mathsf{Adv}}(K; Parameters) \circ |\varphi\rangle$. And the passing space can be written as $P_{pass}\,|\varphi'\rangle$.

2.2 Basics of Quantum Cryptography

We refer to the full version for some basic quantum cryptography. In this work, we use the IND-CPA security for the security definition.

We also refer to the full version for an introduction to the quantum random oracle model. We use H to denote the random oracle queries.

Notation 2.3. The oracle query number of a quantum operation $\mathcal{U}$ is denoted as $|\mathcal{U}|$.

Notation 2.4. The hash tag of a key x is defined as $H(\mathtt{tag}||x)$, where $\mathtt{tag}$ is an unsued special symbol in the alphabet.

Finally we introduce the following notation to simplify the operation of picking a pair of keys from a key set:

Notation 2.5. For $K := \{x_b^{(i)}\}_{i\in[L],b\in\{0,1\}}$, the following notation extracts the key pair with a specific superscript: $K^{(i)} = \{x_b^{(i)}\}_{b\in\{0,1\}}$.

3 PROTOCOL DESIGN FRAMEWORK

In this section we give the framework for constructing and studying different subprotocols.

3.1 SC-security and Well-behaveness

Below we formalize the "SC-security", which describes the adversary's ability to compute both keys from some joint state of all the parties. This will be the foundation of our security framework.

See Definition 2.1, Notation 2.1, 2.3, 2.4 for the notions and symbols appeared in the definition.

Definition 3.1 (SC-security)**.** Suppose the purified joint state of all the parties is $|\varphi\rangle$. We say $|\varphi\rangle$ is $(2^\kappa, A)$-SC-secure for keys $K = \{x_0, x_1\}$ [4] given Z (which is either a client side system or a classical algorithm that takes some client side systems as inputs) if:

For any server side operation $\mathcal{U}$ with query number $|\mathcal{U}| \le 2^\kappa$, (note that $\mathcal{U}$ can introduce server-side ancilla qubits in the zero state, which is inherent in the expression below,)

$$|P_{x_0||x_1}\mathcal{U}(|\varphi\rangle \odot Z \odot Tag)| \le A$$

where:

- Tag denotes the set of hash tags of the keys in K.
- $P_{x_0||x_1}$ projects onto the space where $\mathcal{U}$'s outcome register is equal to the concatenation of client side register x_0 and x_1.

Below we introduce the notion of the representability of a state, which leads to the well-behaveness of a state. This helps us rules out some "ill-behaved cases" like a state that contains the xor of all the random oracle content.

Definition 3.2 (Representability of a state)**.** Let's use $|\mathtt{init}\rangle$ to denote the purified version of the joint state of the initial state (when no party does anything). We say a purified joint state $|\varphi\rangle$ is $(2^{\alpha_1}, 2^{\alpha_2})$-representable from $|\mathtt{init}\rangle$ if

$$|\varphi\rangle = \sum_{i=1}^{2^{\alpha_1}} \mathcal{P}_i |\mathtt{init}\rangle \tag{31}$$

, and $\forall i$, $\mathcal{P}_i$ can contain unitaries, projections and RO queries, and the query number $|\mathcal{P}_i| \le 2^{\alpha_2}$.

Let's introduce the following notation for the set of "well-behaved states", which will be used frequently later. Similarly we use $|\mathtt{init}\rangle$ to denote the purified version of the initial joint state.

[4] Here x_0, x_1 should be considered as client side systems that has already been purified.

Notation 3.1. Define the well-behaved state $\mathcal{WBS}(D)$ to be the set of joint purified states (denoted as $|\varphi\rangle$) such that:

$|\varphi\rangle$ is $(2^{\alpha_1}, 2^{\alpha_2})$-representable from $|\mathtt{init}\rangle$. $\alpha_1, \alpha_2, \log(1/|\,|\varphi\rangle\,|) \le D$.

So D is a bound that controls the well-behaveness condition.

Intuitively, if a state can be prepared using reasonable number of random oracle queries, it could not contain something like the xor of all the random oracle contents.

3.2 Complete Definition of the Weak Security of Remote Gadget Preparation

Since we have defined the SC-security and well-behaveness property, we are prepared to introduce the full definition of the weak security. Again we refer to Section 2.1 for the notations inside it.

Definition 3.3. We say an $N \to L$ remote gadget preparation protocol run on security parameter κ has *weak security transform parameter* $(2^\eta, C) \to p \mid (2^{\eta'}, C')$ for input state in $\mathcal{WBS}(D)$ against adversaries[5] of query number $\le 2^\kappa$ if a statement in the following form holds for the protocol:

> Suppose the input keys are $K = \{x_b^{(i)}\}_{i\in[N],b\in\{0,1\}}$. Suppose the initial state, described by the normalized purified joint state $|\varphi\rangle$, satisfies the following conditions:
>
> - (Input security) $\forall i \in [N]$, $|\varphi\rangle$ is $(2^\eta, C)$-SC-secure for $K^{(i)}$ given $K - K^{(i)}$
> - (Input well-behavenss) $|\varphi\rangle \in \mathcal{WBS}(D)$
>
> For any adversary Adv of query number $|\mathsf{Adv}| \le 2^\kappa$, the final state when the protocol completes, denoted as
>
> $$|\varphi'\rangle = ProtocolName_{\mathsf{Adv}}(K; Parameters) \circ |\varphi\rangle$$
>
> , (and correspondingly, output keys are $K_{out} = \{y_b^{(i)}\}_{i\in[L],b\in\{0,1\}}$) at least one of the followings is true:
>
> - (Passing probability) $|P_{pass}\,|\varphi'\rangle\,| \le p$
> - (Output security) $\forall i \in [L]$, $P_{pass}\,|\varphi'\rangle$ is $(2^{\eta'}, C')$-SC-secure for $K_{out}^{(i)}$ given $K_{out} - K_{out}^{(i)}$.

We can see this definition captures how the security, in terms of the SC-security (Definition 3.1), evolves during the protocol.

And it naturally generalizes to the multi-input-key setting:

Definition 3.4. Suppose the remote gadget preparation protocol is denoted by

$$ProtocolName_{\mathsf{Adv}}(K_1, K_2; Parameters)$$

. We say the protocol has weak security transform parameter $((2^\eta, C), (2^{\eta_2}, C_2)) \to p \mid (2^{\eta'}, C')$ for input state in $\mathcal{WBS}(D)$ against adversaries of query number $\le 2^\kappa$ if a statement similar to the one shown in Definition 3.3 holds, with the following differences: the first condition is replaced by the following conditions:

[5] Note that this is the adversary during the protocol. When we discuss the output security, there is another adversary hidden in the SC-security definition. These two adversaries can be different.

Suppose K_1 has N_1 pairs of keys and K_2 has N_2 pairs of keys. $\forall i \in [N_1], |\varphi\rangle$ is $(2^\eta, C)$-SC-secure for $K_1^{(i)}$ given $(K_1 - K_1^{(i)}) \cup K_2$; and $\forall i \in [N_2], |\varphi\rangle$ is $(2^{\eta_2}, C_2)$-SC-secure for $K_2^{(i)}$ given $(K_2 - K_2^{(i)}) \cup K_1$.

4 WEAKLY SECURE PROTOCOLS

In the introduction we already give an informal introduction to the construction of weakly secure protocol. Here we formalize this part and give a complete protocol. This will be the building block of all the later subprotocols. We will not discuss the remaining steps and the amplification part in details; we refer to the full version.

4.1 $1 + 1 \to 2$ Protocol with Weak Security (When the Input State is Honest)

In this subsection we formalize the protocol in the introduction, and discuss its security.

There are two blocks of it that need to be formalized: the reversible encoding used in (15)→(16) and the Hadamard test in (17). First, we formalize the reversible lookup table used in the protocol:

4.1.1 Reversible Encoding. As the preparation, we introduce a notation for the reversible lookup table:

Definition 4.1 (Notation for classical lookup tables)**.**

$$\mathsf{LT}(\forall b : x_b \to y_{g(b)}; \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}}, \underbrace{\kappa_{\text{tag}}}_{\substack{\text{tag}\\\text{length}}})$$

is defined as the lookup table that maps x_b to $y_{g(b)}$, where $\{x_b\}$ and $\{y_b\}$ are two sets of keys (here two symbols whose only difference is the subscript have the same string length), which means, a table where each row is $\mathsf{Enc}_{x_b}(y_{g(b)}; \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}}, \underbrace{\kappa_{\text{tag}}}_{\substack{\text{tag}\\\text{length}}})$.[6]

And we also use this notation for multi-input multi-output gates: for example, for a Toffoli gate where the input keys are $K = \{x_b^{(1)}, x_b^{(2)}, x_b^{(3)}\}_{b\in\{0,1\}}$ and the output keys are $K_{out} = \{y_b^{(1)}, y_b^{(2)}, y_b^{(3)}\}_{b\in\{0,1\}}$, the notation for the lookup table is

$$\mathsf{LT}(\forall b_1, b_2, b_3 \in \{0,1\}^3 : x_{b_1}^{(1)}, x_{b_2}^{(2)}, x_{b_3}^{(3)} \to y_{b_1}^{(1)}, y_{b_2}^{(2)}, y_{b_1 b_2 \oplus b_3}^{(3)}; \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}}, \underbrace{\kappa_{\text{tag}}}_{\substack{\text{tag}\\\text{length}}})$$

, and each row in this lookup table is

$$\mathsf{Enc}_{x_{b_1}^{(1)}||x_{b_2}^{(2)}||x_{b_3}^{(3)}}(y_{b_1}^{(1)}||y_{b_2}^{(2)}||y_{b_1 b_2 \oplus b_3}^{(3)}; \ell, \kappa_{tag})$$

[6] Note this Enc should support key verification procedure. One example construction is as follows. Consider $\mathsf{Enc}_k(p, \ell, \kappa_{out})$ The client samples $R_1 \leftarrow \{0,1\}^l$, $R_2 \leftarrow \{0,1\}^l$, output

$$((R_1, H(R_1||k) \oplus p), (R_2, H(R_2||k)))$$

. The first part is the ciphertext and the second part is the key tag. The length of the random oracle output in the first part is the same as the length of p and the length of the random oracle output of the second part is κ_{tag}.

Definition 4.2 (Definition and notation for reversible lookup tables)**.**

$$\mathsf{RevLT}(\forall b : x_b \leftrightarrow y_{g(b)}; \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}})$$

is defined as the reversible lookup table that maps x_b to $y_{g(b)}$ and vice versa, which is, the combination of the following two lookup tables:

- Forward table: $\mathsf{LT}(\forall b : x_b \to y_{g(b)}; \ell)$, where the tag length is the same as the length of keys $y_{g(b)}$.
- Backward table: $\mathsf{LT}(\forall b : y_{g(b)} \to x_b; \ell)$, where the tag length is the same as the length of keys x_b.

As Definition 4.1, this notation can be applied in the multi-key case and the keys in different wires are concatenated before feeding into the Enc operation. The tag length is the same as the total output length.

Definition 4.3 (Simplified notations for some reversible lookup tables)**.** For a reversible lookup table on input key set K and the output key set K', if the type of the gate is implicit, when there is no ambiguity, we can use $\mathsf{RevLT}(K \leftrightarrow K_{out}; \ell)$ to denote the reversible lookup table that maps the keys in K to the corresponding keys in K_{out}, and back.

Then we can construct the lookup table we construct in the introduction:

Definition 4.4. $\mathsf{RobustRLT}(K_{\text{help}}, K_{in} \leftrightarrow K_{out}, \pi; \ell)$, where

- $K_{\text{help}} = \{x_b^{\text{help}}\}_{b\in\{0,1\}}$, $K_{in} = \{x_b^{(2)}, x_b^{(3)}\}_{b\in\{0,1\}}$,$K_{out} = \{y_b^{(2)}, y_b^{(3)}\}_{b\in\{0,1\}}$, and the keys with the same symbol and superscript have the same length; $y_b^{(2)}$ and $y_b^{(3)}$ have the same length.
- π is a bit-wise permutation on the strings of length $2\kappa_{out}$, κ_{out} is the key length of $y_b^{(2)}$.
- ℓ is the padding length.

is defined as follows.

First consider the two reversible encoding between $K_{in}^{(2,3)}$ and K_{out}: (Below is their encryption structure.)

$$\text{(Identity-style) } \mathsf{RevLT}_{b_1=0} : \mathsf{RevLT}(\forall b_2, b_3 \in \{0,1\}^2 : x_{b_2}^{(2)}||x_{b_3}^{(3)} \leftrightarrow \pi(y_{b_2}^{(2)}||y_{b_3}^{(3)}); \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}})$$

$$\text{(CNOT-style) } \mathsf{RevLT}_{b_1=1} : \mathsf{RevLT}(\forall b_2, b_3 \in \{0,1\}^2 : x_{b_2}^{(2)}||x_{b_3}^{(3)} \leftrightarrow \pi(y_{b_2}^{(2)}||y_{b_2+b_3}^{(3)}); \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}})$$

Then the RobustRLT is defined as

$$\mathsf{Enc}_{x_0^{\text{help}}}(\mathsf{RevLT}_{b_1=0}; \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}}, \underbrace{\ell}_{\substack{\text{tag}\\\text{length}}}) \quad (32)$$

$$||\mathsf{Enc}_{x_1^{\text{help}}}(\mathsf{RevLT}_{b_1=1}; \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}}, \underbrace{\ell}_{\substack{\text{tag}\\\text{length}}})$$

4.1.2 Hadamard Test. Then the (padded) Hadamard test is formalized as follows:

Definition 4.5 (Padded Hadamard test)**.** The padded Hadamard test

$$\mathsf{PadHadamard}(K; \underbrace{\ell}_{\substack{\text{padding}\\\text{length}}}, \underbrace{\kappa_{\text{out}}}_{\substack{\text{output}\\\text{length}}})$$

on $K = \{x_0, x_1\}$ is defined as follows:

(1) The client samples $pad \leftarrow_r \{0,1\}^l$ and sends R to the server.
(2) The server returns d such that $d \cdot (x_0 || H(pad||x_0)) = d \cdot (x_1 || H(pad||x_1))$ where $H(pad||x_b)$ has length κ_{out}, and d is not all zero on the last κ_{out} bits. The client checks the server's response.

The honest server can pass this test by making Hadamard measurement on $|x_0\rangle |H(pad||x_0)\rangle + |x_1\rangle |H(pad||x_1)\rangle$.

Let's first show some intuition behind our technique. Consider an adversary that can pass this test with high probability. And we want to understand what this fact can tell us about the server's state. It seems to be a hard problem: different from the standard basis measurement in [15], which directly tells us the state of the server before the measurement, Hadamard basis measurement does not give us anything like that. However, it tells us what the server **can not do** after the test:

Lemma 4.1 (Unpredictability restriction, informal)**.** *For $K = \{x_0, x_1\}$, suppose (1)the initial state is sufficiently SC-secure for K; (2)the adversary can pass the padded Hadamard test with high probability, then for any $b \in \{0,1\}$, the adversary can only compute x_b from the post-test state with small probability. (Which means, the output state has some level of ANY-security.)*

The main idea of the proof is a trick on the time-order-switching of two measurements.

Idea of the proof. Suppose the adversary can pass the padded Hadamard test with high probability. Then it proceeds and try to compute x_0 or x_1. Since these two measurements commute (the measurement of getting the output for the Hadamard test, and the measurement that tries to compute x_b), imagine that this adversary first measures and gets one of x_0 and x_1 and then sends out the result for the padded Hadamard test, the probability of passing the test should not change. On the other hand, if the adversary gets one of x_0 or x_1, by the SC-security of the initial state it cannot get the other key, which implies it should be very hard to pass the padded Hadamard test. (Note that the unpadded Hadamard test does not guarantee this!) □

We note the Hadamard test does not guarantee that the server throws away the keys in K^{help} completely: the server can cheat with some probability. But we can see this subprotocol does provide some level of security, which is sufficient for our purpose.

Another property that we will use is the *coherency restriction*: starting from a initial state that has a two-branch form, the behavior of the corresponding two outcome branches should behave coherently on any efficient adversarial attack:

Lemma 4.2 (Coherency restriction, informal)**.** *For $K = \{x_0, x_1\}$, suppose (1)the initial state is sufficiently SC-secure for K; (2) the initial state has the form of $|x_0\rangle |\cdots\rangle + |x_1\rangle |\cdots\rangle$; (3)the adversary can pass the padded Hadamard test with high probability, then denote $|x_0\rangle |\cdots\rangle$ and $|x_1\rangle |\cdots\rangle$ as two* branches *of the state, and denote $|\varphi_0'\rangle$ and $|\varphi_1'\rangle$ as the corresponding outcome state of these two branches. Then for any adversarial operation on the post-test state that ends with a projective measurement, for any measurement result, the probability of getting this result on these two branches should not be too far away from each other.*

This gives the adversary a strong restriction since this lemma intuitively tells us its behavior on these two branches should be "coherent". This lemma will be a key property on proving the security of our basic weakly-secure gadget-increasing protocol. Below we give it a more formal description.

Lemma 4.3 (Coherency Restriction, informal)**.** *The following statement is true for sufficiently large security parameter κ:*

Consider the normalized initial state

$$|\varphi\rangle = \textit{purified joint state of } |x_0\rangle |\cdots\rangle + |x_1\rangle |\cdots\rangle \tag{33}$$

such that

- *(SC-security) The state is $(2^{\eta}, 2^{-\eta})$-SC-secure for K.*
- *(Well-behaveness) It is sufficiently well-behaved in the sense of Notation 3.1.*

Then the following conclusion holds for any $C > 2^{-\kappa/10}$:

For any adversary Adv *of query number $|\mathsf{Adv}| \le 2^{\kappa}$, for the post-execution state, which is*

$$|\varphi'\rangle = \mathsf{PadHadamard}_{\mathsf{Adv}}(K; \underbrace{\ell}_{\substack{\textit{padding}\\\textit{length}}}, \underbrace{\kappa_{out}}_{\substack{\textit{output}\\\textit{length}}}) \circ |\varphi\rangle$$

at least one of the following is true:

- *(Small passing probability) $|P_{pass}\, |\varphi'\rangle\,| \le (1 - C^2)$.*
- *(Coherency of two branches) For any server-side operation $\mathcal{D}$ with query number $|\mathcal{D}| \le 2^{\eta/4}$, define the two components of the post-test state:*

$$b \in \{0,1\}, |\varphi_b'\rangle := \mathsf{PadHadamard}_{\mathsf{Adv}}(K; \ell, \kappa_{out}) \circ |\varphi_b\rangle$$

($|\varphi_b\rangle :=$ the purified joint state of $|x_b\rangle |\cdots\rangle$ in (33).) And define

$$p = |P_S \mathcal{D}(|\varphi'\rangle)|$$
$$p_0 = |P_S \mathcal{D}(|\varphi_0'\rangle)|$$
$$p_1 = |P_S \mathcal{D}(|\varphi_1'\rangle)|$$

where P_S is a server-side projection onto some system S. Then at least one of the following two is true:

$$\textit{(S is not a significant outcome) } p \le 5C \tag{34}$$

$$\textit{(Coherency) } \min\{p_0, p_1\} \ge \frac{p}{6} \tag{35}$$

We will give a fully quantitive formalism in the full version. Note that this coherency restriction can also derive the unpredictability restriction for this specific input form.

This coherency restriction property can be proved using a similar idea as the unpredictability restriction. Let's give an overview with a similar quantification level as Lemma 4.3.

Overview of the proof. Suppose the adversary can pass the test with high probability: $|P_{pass}|\varphi'\rangle| > (1-C^2)$.

Consider the following adversary and client: the adversary does not measure d, but sends out the d register in quantum state directly. The client holds d, then the server runs $\mathcal{D}$; then the adversary makes a projection on S; finally the client measures d. Since the two measurements (d, and S) commute the norm of the passing part should be the same.

We can show

$$\forall b \in \{0,1\}, \quad |P_{pass}P_S\mathcal{D} \circ \mathsf{PadHadamard} \circ |\varphi_b\rangle| \tag{36}$$

$$\leq \frac{1}{\sqrt{2}}|P_S\mathcal{D} \circ \mathsf{PadHadamard} \circ |\varphi_b\rangle| + 2^{-\eta/4+10} \tag{37}$$

Intuitively that's because if the adversary gets x_b, by the SC-security of the initial state it cannot predict x_{1-b}. Then it could not query $H(pad||x_{1-b})$ that appears in the test (here we rely on the padding procedure since without the padding the adversary could potentially know this value initially). Then this can derive (after some calculation) that the adversary can only pass with probability close to $1/2$ — which is (37).

Suppose $p > 5C$, and without loss of generality, suppose $p_0 < p/6$. Thus $p_1 < 7p/6$. Then it further implies

$$|P_{pass}\mathcal{D}(|\varphi'\rangle)| \tag{38}$$

$$= \sqrt{1-|P_{fail}\mathcal{D}(|\varphi'\rangle)|^2} \tag{39}$$

$$\leq \sqrt{1-(p^2-|P_{pass}P_S\mathcal{D}(|\varphi'\rangle)|^2)} \tag{40}$$

$$< \sqrt{1-p^2+(\frac{1}{\sqrt{2}}\cdot(\frac{1}{6}p+\frac{7}{6}p)+2^{-\eta/4+11})^2} < 1-C^2 \tag{41}$$

which is a contradiction. □

4.1.3 Protocol Design. Now we can formalize our first remote gadget preparation protocol described in the introduction. We name it as $\mathsf{GdgPrep}^{Basic}$.

Protocol 1. $\mathsf{GdgPrep}^{Basic}(K^{help}, K^{(3)}; \ell, \kappa_{out})$, *where* $K^{help} = \{x_b^{help}\}_{b\in\{0,1\}}$, $K^{(3)} = \{x_b^{(3)}\}_{b\in\{0,1\}}$. ℓ *is the padding length and* κ_{out} *is the output key length:*

For an honest server, the initial state is $|\varphi\rangle = (|x_0^{help}\rangle + |x_1^{help}\rangle) \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle)$.

(1) *the client samples*
- *A permutation* π *of* $[2\kappa_{out}]$. *We will view this as acting bit-wisely on* $\{0,1\}^{2\kappa_{out}}$.
- *a pair of different (input) keys* $K^{(2)} = \{x_0^{(2)}, x_1^{(2)}\}$ *with the same length as* $x_b^{(3)}$;
- 2 *pairs of different (output) keys* $K_{out} = \{y_b^{(2)}, y_b^{(3)}\}_{b\in\{0,1\}}$ *with key length* κ_{out}.

(2) *The client computes*

$$\mathsf{RobustRLT}(K^{help}, K^{(2,3)} \leftrightarrow K_{out}, \pi; \underbrace{\ell}_{\substack{padding\\length}})$$

and sends it together with $K^{(2)}$ *to the server.*

(3) *An honest server should implement the following mapping:*

$$|\varphi\rangle = (|x_0^{help}\rangle + |x_1^{help}\rangle) \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle) \tag{42}$$

$$\to (|x_0^{help}\rangle + |x_1^{help}\rangle) \otimes \pi((|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle)) \tag{43}$$

(4) *The client and the server run the padded Hadamard test on* K^{help}. *The server can use* $|x_0^{help}\rangle + |x_1^{help}\rangle$ *to pass the test, as described in Definition 4.5. Reject if the server does not pass this test.*

(5) *The client sends out* π.

(6) *The server removes the permutation and gets* $(|y_0^{(2)}\rangle + |y_1^{(2)}\rangle) \otimes (|y_0^{(3)}\rangle + |y_1^{(3)}\rangle)$.

Correctness. This protocol transforms 2 gadgets to 2 gadgets.

Efficiency. Both the client and the honest server run in polynomial time (on the key size and the parameters).

The security statement for $\mathsf{GdgPrep}^{Basic}$ is given below. To focus on the most simplified cases, we will only care about the case where the input state is fully honest. This also allows us to omit the well-behaveness requirement in the full weak security formalism (see Definition 3.3). We put the general case in the full version.

See Section 2.1 and Definition 3.4 for the notations.

Lemma 4.4. *There exist constants* $A, B \geq 1$ *such that the following statement is true for sufficiently large security parameter* κ*:*

For keys $K^{help} = \{x_b^{help}\}_{b\in\{0,1\}}, K^{(3)} = \{x_b^{(3)}\}_{b\in\{0,1\}}$ *which are both a pair of keys, protocol*

$$\mathsf{GdgPrep}^{Basic}(K^{help}, K^{(3)}; \underbrace{\ell}_{\substack{padding\\length}}, \underbrace{\kappa_{out}}_{\substack{output\\length}})$$

has weak security transform parameter

$$((2^\eta, 2^{-\eta}), (2^\eta, 4C)) \to (1-C^2) \mid (2^{\eta/B}, AC)$$

for states defined below against adversaries of query number $\leq 2^\kappa$ *when the following input state form and inequalities are satisfied.*

$$|\varphi\rangle = \text{ purified joint state of } (|x_0^{help}\rangle + |x_1^{help}\rangle) \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle) \tag{44}$$

The inequalities are as follows:

(1) *(Sufficient security on the inputs)* $\eta \geq \kappa \cdot B$.

(2) *(Sufficient padding length, output key length)* $\ell \geq 4\eta$, $\kappa_{out} > \ell + 4\eta$

(3) *For simplicity, additionally assume* $\frac{1}{9} > C > 2^{-\sqrt{\kappa}}$

4.2 Intuition of the Proof of Lemma 4.4, the Easier Part

Let's first give some intuition of why this lemma is true. Similarly, see Section 2.1 for the notations.

Note that there are two output keys in Lemma 4.4, corresponding to superscript "(2)" and "(3)" in the protocol. Here we first consider Lemma 4.4 for the "(3)" output key pair, which is easier.

Below we use $|\varphi\rangle$ to denote the purified joint state of $|\varphi\rangle = (|x_0^{\text{help}}\rangle + |x_1^{\text{help}}\rangle) \otimes (|x_0^{(3)}\rangle + |x_1^{(3)}\rangle)$. We can denote the output state

as

$$|\varphi'\rangle = \mathsf{GdgPrep}^{Basic}(K^{\mathrm{help}}, K^{(3)}; \ell, \kappa_{\mathrm{out}}) \circ |\varphi\rangle$$

(1) First we assume the first case does not hold (thus $|P_{pass}\,|\varphi'\rangle\,| > (1-C^2)$) thus we need to prove

$P_{pass}\,|\varphi'\rangle$ is $(2^{O(\eta)}, O(C))$-SC-secure for $K^{(3)}_{out}$ given $K^{(2)}_{out}$.

(2) Denote $|\varphi\rangle = |\varphi_0\rangle + |\varphi_1\rangle$ where $|\varphi_b\rangle$ is the purified joint state of $|\varphi\rangle = |x^{\mathrm{help}}_b\rangle \otimes (|x^{(3)}_0\rangle + |x^{(3)}_1\rangle)$. We define

$$|\varphi'_b\rangle := \mathsf{GdgPrep}^{Basic}(K^{\mathrm{help}}, K^{(3)}; \ell, \kappa_{\mathrm{out}}) \circ |\varphi_b\rangle \tag{45}$$

thus $|\varphi'\rangle = |\varphi'_0\rangle + |\varphi'_1\rangle$. Thus we can study these two *branches* separately and calculate the SC-security by the triangle inequality.
Recall that $|\varphi_0\rangle$ corresponds to the identity-style branch and $|\varphi_1\rangle$ corresponds to the CNOT-style branch.

(3) First, by analyzing the encryption structure of the RobustRLT we can prove the adversary can't compute $y^{(3)}_0||y^{(3)}_1$ on the $|\varphi_0\rangle$ branch with big norm. The reason is intuitively as follows:
Write $|\varphi_0\rangle = |\varphi_{00}\rangle + |\varphi_{01}\rangle$, where $|\varphi_{00}\rangle$ is the purified joint state of $|x^{help}_0\rangle\,|x^{(3)}_0\rangle$ and $|\varphi_{01}\rangle$ is defined correspondingly. On the $|\varphi_{00}\rangle$ branch of the initial state the adversary can't compute $y^{(3)}_1$ with big norm, while on the $|\varphi_{01}\rangle$ part of the initial state the adversary can't compute $y^{(3)}_0$ with big norm. To see it clearly, without loss of generality, assume the initial state is (the purified joint state of) $|x^{\mathrm{help}}_0\rangle\,|x^{(3)}_0\rangle$:
Then in the next step of the protocol the adversary gets the RobustRLT and $K^{(2)}$. In this setting, the adversary can only decrypt the following rows:

$$(\text{Given } x^{\mathrm{help}}_0) : x^{(2)}_0, x^{(3)}_0 \leftrightarrow \pi(y^{(2)}_0||y^{(3)}_0) \tag{46}$$

$$(\text{Given } x^{\mathrm{help}}_0) : x^{(2)}_1, x^{(3)}_0 \leftrightarrow \pi(y^{(2)}_1||y^{(3)}_0) \tag{47}$$

which means, the adversary can only decrypt two rows in the forward table and two rows in the backward table. Each of the $\leftrightarrow$ symbol represents two rows: $\rightarrow$ for the forward table and $\leftarrow$ for the backward table.
What's more, even if π is revealed to the adversary, and when $K^{(2)}_{out}$ is given additionally, the what the adversary knows is still limited to:

$$x^{\mathrm{help}}_0, x^{(2)}_0, x^{(2)}_1, x^{(3)}_0, y^{(2)}_0, y^{(2)}_1, y^{(3)}_0$$

And no matter how the adversary composes them, it cannot decrypt other rows than (46)(47).
Thus we can see $|\varphi'_{00}\rangle$ (the final state on the $|\varphi_{00}\rangle$ branch) does not allow the prediction of $y^{(3)}_1$, even if the adversary is additionally given $K^{(2)}_{out}$. Thus $|\varphi'_{00}\rangle$ is $(2^{\Theta(\eta)}, O(C))$-SC-secure for $K^{(3)}_{out}$ given $K^{(2)}_{out}$. Similarly the final state corresponding to the $|\varphi_{01}\rangle$ branch is unpredictable for $y^{(3)}_1$ given $K^{(2)}_{out}$ and thus is $(2^{\Theta(\eta)}, O(C))$-SC-secure for $K^{(3)}_{out}$ given $K^{(2)}_{out}$. Thus the $|\varphi'_0\rangle$ is $(2^{\Theta(\eta)}, O(C))$-SC-secure for $K^{(3)}_{out}$ given $K^{(2)}_{out}$.

(4) How can we argue about the whole state $|\varphi'\rangle$ from the properties of the $|\varphi'_0\rangle$ branch? Here the property of the Hadamard test comes in: by the coherency restriction (Lemma 4.3) we know if on the $|\varphi_0\rangle$ part the adversary is hard to compute $y^{(3)}_0||y^{(3)}_1$, it's hard to compute it with $|\varphi_1\rangle$ with big norm either, otherwise we can choose the distinguisher in Lemma 4.3 to be the operator that compute $y^{(3)}_0||y^{(3)}_1$ and get a contradiction. (Recall that we assumed $|P_{pass}\,|\varphi'\rangle\,|$.)

Then how about the case for the other output key pair? The proof for the "(2)" case is more challenging. We need to make use of the permutation in the lookup table: we can prove that, the security of the "(3) " case, in some sense, is "mixed" with the (2) case. Which means, if the adversary can compute the keys on output wire "(2)", the property of this permuted lookup table will imply the adversary can also compute some part of the keys at "(3)" under some situation. Then we can use a similar argument as the "(3)" case above. We will give a more detailed proof outline in the next subsection.

4.3 Intuition of the Proof of Lemma 4.4, the Harder Part

Below we give some intuitions on the (2) case of Lemma 4.4. See the full paper for the full proof. See Section 2.1 for some notations.

(1) Similar to the (3) case, we assume the first case does not hold (thus $|P_{pass}\,|\varphi'\rangle\,| > (1-C^2)$) thus we need to prove $P_{pass}\,|\varphi'\rangle$ is $(2^{O(\eta)}, O(C))$-SC-secure for $K^{(2)}_{out}$ given $K^{(3)}_{out}$.

(2) Let's first introduce some symbols. If an adversary wants to break this SC-security statement, the adversary needs to compute $y^{(2)}_0||y^{(2)}_1$. Suppose the adversary's operation is $\mathcal{U}$, and denote:

$$p := |P_{y^{(2)}_0||y^{(2)}_1}\mathcal{U}(|\varphi'\rangle \text{ together with } K^{(3)}_{out})| \tag{48}$$

We want to get an upper bound for p.

(3) The first step[7] is to make use of the *unpredictability restriction* property of our Hadamard test for K^{help}. To formalize these intuitions we will consider the *blinded* adversary where the adversary's queries in $\mathcal{U}$ are replaced by queries to a blinded oracle where $H(\cdots||K^{\mathrm{help}}||\cdots)$ part are blinded. Which means, consider an oracle that on inputs in the form of

$$\underbrace{\cdots}_{\text{length } \ell}\,||x^{\mathrm{help}}_b||\,\underbrace{\cdots}_{\text{any length}}$$

is independently random from H and the same as H otherwise. Since this is under purified joint states the blinded oracles are also entangled with the key register.
We can prove, the output of using this blinded oracle (use $\mathcal{U}^{blind}$ to denote the operation that replace H in U by the blinded one) does not differ too much from the output of using $\mathcal{U}$:

$$|P_{y^{(2)}_0||y^{(2)}_1}\mathcal{U}^{blind}(|\varphi'\rangle \text{ together with } K^{(3)}_{out})| \geq O(p) - O(C) \tag{49}$$

Note that this step is not needed in the (3) case but is crucial in the (2) case.

[7] We skip one step compared to the full version. Here we need an additional step to deal with the hash tags of the keys, which is hidden in (48). Here we simply make this step implicit.

(4) Then, similar to the proof of the (3) case, by applying Lemma 4.3[8] we know, the adversary should be able to compute $y_0^{(2)}||y_1^{(2)}$ on the $|\varphi_1\rangle$ part of the initial state with a not-too-small norm:

$$|P_{y_0^{(2)}||y_1^{(2)}}\mathcal{U}^{blind}(|\varphi_1'\rangle \text{ together with } K_{out}^{(3)})| \geq O(p) - O(C) \quad (50)$$

$|\varphi_1'\rangle$ is defined as (45), the same as the (3) case.

A note on the proof structure. We will see, in our proof, we keep deriving lower bounds in the form of $O(p) - O(C)$ for different expressions. Once we reach an expression that can also be upper-bounded by $O(C)$, we get $O(C) \geq O(p) - O(C)$ thus $p \leq AC$ for some constant A. We note that the whole structure might be counter-intuitive, since we are not trying to prove the adversary cannot do something in some settings; instead, we are proving the adversary can do something in different settings, assuming (48). (Thus (49)(50) are all "$\geq$" inequalities.) Finally we reach something that can be bounded in the other direction and complete the proof.

(5) Where can we go from (50)? The next idea is to consider what will happen if the client replaces the π in the fifth step of the protocol by a random permutation. We would like to prove, if (50) holds, which means the adversary can compute the description of $K_{out}^{(2)}$ when the real permutation is provided in the fifth step, the same operation will also compute a set of *fake keys* when the permutation in the fifth step is a random *fake permutation*.

Note that in (50) we only consider the $|\varphi_1\rangle$ branch of the input, and similar to the (3) case, $|\varphi_1\rangle = |\varphi_{10}\rangle + |\varphi_{11}\rangle$. (Recall that $|\varphi_0\rangle$ corresponds to the identity-style branch and $|\varphi_1\rangle$ corresponds to the CNOT-style branch.)

Informally and without loss of generality let's consider the case where the input is $|\varphi_{10}\rangle$. The adversary already knows or can decrypt the followings directly from the lookup table:

$$(\text{Given } x_1^{\text{help}}) : x_0^{(2)}, x_0^{(3)} \leftrightarrow \pi(y_0^{(2)}||y_0^{(3)}) \quad (51)$$

$$(\text{Given } x_1^{\text{help}}) : x_1^{(2)}, x_0^{(3)} \leftrightarrow \pi(y_1^{(2)}||y_1^{(3)}) \quad (52)$$

(51) is what the adversary can get before it gets π. Since π is hidden before the 5th step in Protocol 1, the server can't extract $y_b^{(w)}, w \in \{2,3\}$ from the right of (51), and the right of (51) looks (almost) the same as two independently random strings.

In the 5th step of the protocol the client provides π. Then since the goal of the adversary is to output $y_0^{(2)}||y_1^{(2)}$, it has to extract $y_0^{(2)}$ from (51) and extract $y_1^{(2)}$ from (52):

$$(\text{Given } x_1^{\text{help}}) : x_0^{(2)}, x_0^{(3)} \leftrightarrow \pi(y_0^{(2)}||y_0^{(3)}) \xrightarrow{\pi} y_0^{(2)} \quad (53)$$

$$(\text{Given } x_1^{\text{help}}) : x_1^{(2)}, x_0^{(3)} \leftrightarrow \pi(y_1^{(2)}||y_1^{(3)}) \xrightarrow{\pi} y_1^{(2)} \quad (54)$$

Imagine that in the 5th step, instead of providing the real permutation π, the client provides a different π'. The key observation is, from (53)(54), the server can't distinguish this *fake permutation* π' from the real π using only the decrypted plaintext in (51). In the server's viewpoint, $\pi(y_0^{(2)}||y_0^{(3)}), \pi(y_1^{(2)}||y_1^{(3)})$ are (almost) just two strings whose bits are all independently random. Thus if some adversary can do the extraction shown in (53)(54), if the client chooses to provide a different π', it should also be able to compute the *fake keys* $K_{out}^{fake-0-(2)}, K_{out}^{fake-0-(3)}$, defined as the key pairs that have the same length with $K_{out}^{(2)}, K_{out}^{(3)}$, and satisfy the following equation:

$$\forall w \in \{2,3\}, K_{out}^{fake-0-(w)} = \{y_b^{fake-0-(w)}\}_{b\in\{0,1\}} \text{ satisfy :}$$

$$\forall b \in \{0,1\}, \pi'(y_b^{fake-0-(2)}||y_b^{fake-0-(3)}) = \pi(y_b^{(2)}||y_b^{(3)})$$

Writing it in the form of (53)(54), in the server's viewpoint, it gets the following:

$$(\text{Given } x_1^{\text{help}}) : x_0^{(2)}, x_0^{(3)} \leftrightarrow perm(y_0^{(2)}||y_0^{(3)}) \xrightarrow{\pi'}$$

$$y_0^{fake-0-(2)} := \text{first half of } \pi'^{-1}(\pi(y_0^{(2)}||y_0^{(3)}))$$

$$(\text{Given } x_1^{\text{help}}) : x_1^{(2)}, x_0^{(3)} \leftrightarrow \pi(y_1^{(2)}||y_1^{(3)}) \xrightarrow{\pi'}$$

$$y_1^{fake-0-(2)} := \text{first half of } \pi'^{-1}(\pi(y_1^{(2)}||y_1^{(3)}))$$

Formally speaking, it implies

$$q_1^{blind,fake,0} := \quad (55)$$

$$|P_{y_0^{fake-0-(2)}||y_1^{fake-0-(2)}}\mathcal{U}^{blind}($$

$$|\phi_1'\rangle \text{ together with } K_{out}^{fake-0-(3)})|$$

$$\geq O(p) - O(C)$$

where $|\phi_1'\rangle$ is the output state when the initial state is $|\varphi_1\rangle$ and the permutation provided is π'.

Note. There is one more detail missing above: in the argument above we implicitly assume the adversary can only decrypt at most two rows in the forward table and two rows in the backward table ((53)(54)). But why is this still true when the π is provided to the server? The reason is, we have already "blinded" the server's operation $\mathcal{U}$! In other words, before the adversary knows π, this property is guaranteed by the fact that the adversary does not know π; after the π is provided, the adversary has already been blinded, and this property still holds.

(6) On the other hand, we can prove, these fake-keys cannot be simultaneously computed in the $|\varphi_0\rangle$ branch of the input:

$$q_0^{blind,fake,0} := \quad (56)$$

$$|P_{y_0^{fake-0-(2)}||y_1^{fake-0-(2)}}\mathcal{U}^{blind}($$

$$|\phi_0'\rangle \text{ together with } K_{out}^{fake-0-(3)})|$$

$$\leq O(C)$$

where $|\phi_0'\rangle$ is the output state when the initial state is $|\varphi_0\rangle$ and the permutation provided is π'.

Then (55)(56) provide a potential violation of the coherency restriction. Applying Lemma 4.3 again gives $p \leq O(C)$ and

[8] Lemma 4.3 does not consider the existence of further communication after the test, but here the permutation π is sent out after the test. But we can still apply the Lemma 4.3. The trick is to temporarily consider the client side register that stores π as a server-side register when we apply the lemma. And we note that even if π is on the server side, the conditions of applying Lemma 4.3 still holds since the SC-security of the helper gadget does not rely on the secrecy of π.

completes the proof. (Again, there is additional communication after the test. We refer to the footnote in the third step of this outline for why we can do it.)

ACKNOWLEDGEMENT

This work is supported by NSF award 1763786. We would like to give sincere thanks to Prof. Adam Smith for his advising. And the author would like to thank anonymous reviewers, Hezi Zhang, Thomas Vidick and Tomoyuki Morimae for useful comments.

REFERENCES

[1] Scott Aaronson, Alexandru Cojocaru, Alexandru Gheorghiu, and Elham Kashefi. 2019. Complexity-Theoretic Limitations on Blind Delegated Quantum Computation. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece (LIPIcs, Vol. 132)*, Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:13. https://doi.org/10.4230/LIPIcs.ICALP.2019.6

[2] Zvika Brakerski, Paul Christiano, Urmila Mahadev, Umesh V. Vazirani, and Thomas Vidick. 2018. A Cryptographic Test of Quantumness and Certifiable Randomness from a Single Quantum Device. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. 320–331. https://doi.org/10.1109/FOCS.2018.00038

[3] Anne Broadbent, Joseph Fitzsimons, and Elham Kashefi. 2009. Universal Blind Quantum Computation. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '09)*. IEEE Computer Society, Washington, DC, USA, 517–526. https://doi.org/10.1109/FOCS.2009.36

[4] Anne Broadbent and Stacey Jeffery. 2015. Quantum Homomorphic Encryption for Circuits of Low T-gate Complexity. In *Advances in Cryptology – CRYPTO 2015*, Rosario Gennaro and Matthew Robshaw (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 609–629. https://doi.org/10.1007/978-3-662-48000-7_30

[5] Ran Canetti, Oded Goldreich, and Shai Halevi. 2004. The Random Oracle Methodology, Revisited. *J. ACM* 51, 4 (July 2004), 557–594. https://doi.org/10.1145/1008731.1008734

[6] Alexandru Cojocaru, Léo Colisson, Elham Kashefi, and Petros Wallden. 2019. QFactory: Classically-Instructed Remote Secret Qubits Preparation. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11921)*, Steven D. Galbraith and Shiho Moriai (Eds.). Springer, 615–645. https://doi.org/10.1007/978-3-030-34578-5_22

[7] Yfke Dulek, Christian Schaffner, and Florian Speelman. 2016. Quantum Homomorphic Encryption for Polynomial-Sized Circuits. In *Proceedings, Part III, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9816*. Springer-Verlag, Berlin, Heidelberg, 3–32. https://doi.org/10.1007/978-3-662-53015-3_1

[8] Joseph F. Fitzsimons and Elham Kashefi. 2017. Unconditionally verifiable blind quantum computation. *Phys. Rev. A* 96 (Jul 2017), 012303. Issue 1. https://doi.org/10.1103/PhysRevA.96.012303

[9] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing* (Bethesda, MD, USA) *(STOC '09)*. Association for Computing Machinery, New York, NY, USA, 169–178. https://doi.org/10.1145/1536414.1536440

[10] R. Impagliazzo. 1995. A personal view of average-case complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*. 134–147. https://doi.org/10.1109/SCT.1995.514853

[11] David Jao and Luca De Feo. 2011. Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In *Post-Quantum Cryptography*, Bo-Yin Yang (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 19–34. https://doi.org/10.1007/978-3-642-25405-5_2

[12] Neal Koblitz and Alfred J. Menezes. 2015. The random oracle model: a twenty-year retrospective. *Designs, Codes and Cryptography* 77, 2 (2015), 587–610. https://doi.org/10.1007/s10623-015-0094-2

[13] Ching-Yi Lai and Kai-Min Chung. 2018. On Statistically-Secure Quantum Homomorphic Encryption. *Quantum Info. Comput.* 18, 9–10 (Aug. 2018), 785–794. https://doi.org/10.26421/QIC18.9-10-4

[14] Urmila Mahadev. 2018. Classical Homomorphic Encryption for Quantum Circuits. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, Mikkel Thorup (Ed.). IEEE Computer Society, 332–338. https://doi.org/10.1109/FOCS.2018.00039

[15] Urmila Mahadev. 2018. Classical Verification of Quantum Computations. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, Mikkel Thorup (Ed.). IEEE Computer Society, 259–267. https://doi.org/10.1109/FOCS.2018.00033

[16] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press, New York, NY, USA.

[17] Yingkai Ouyang, Si-Hui Tan, and Joseph F. Fitzsimons. 2018. Quantum homomorphic encryption from quantum codes. *Phys. Rev. A* 98 (Oct 2018), 042334. Issue 4. https://doi.org/10.1103/PhysRevA.98.042334

[18] Oded Regev. 2009. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM* 56, 6, Article 34 (Sept. 2009), 40 pages. https://doi.org/10.1145/1568318.1568324

[19] Thomas Vidick and Tina Zhang. 2019. Classical zero-knowledge arguments for quantum computations. *TQC Proceedings* (2019). https://eprint.iacr.org/2019/194

[20] A. C. Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. 162–167. https://doi.org/10.1109/SFCS.1986.25

[21] Jiayu Zhang. 2019. Delegating Quantum Computation in the Quantum Random Oracle Model. In *Theory of Cryptography*, Dennis Hofheinz and Alon Rosen (Eds.). Springer International Publishing, Cham, 30–60. https://doi.org/10.1007/978-3-030-36033-7_2

[22] Jiayu Zhang. 2020. Succinct Blind Quantum Computation Using a Random Oracle. (2020). http://arxiv.org/abs/2004.12621