A Quantum Approach For Reducing Communications in Classical Secure Computations with Long Outputs

Jiayu Zhang*[1]

[1]Zhongguancun Laboratory

April 4, 2025

Abstract

How could quantum cryptography help us achieve what are not achievable in classical cryptography? In this work we study the classical cryptographic problem that two parties would like to perform secure computations *with long outputs*. As a basic primitive and example, we first consider the following problem which we call *secure function sampling* with long outputs: suppose $f : \{0,1\}^n \to \{0,1\}^m$ is a public, efficient classical function, where m is big; Alice would like to sample x from its domain and sends $f(x)$ to Bob; what Bob knows should be no more than $f(x)$ even if it behaves maliciously. Classical cryptography, like FHE and succinct arguments [25, 14, 34, 30], allows us to achieve this task within communication complexity $O(n+m)$; could we achieve this task with communication complexity independent of m?

In this work, we first design a quantum cryptographic protocol that achieves secure function sampling with approximate security, within $O(n)$ communication (omitting the dependency on the security parameter and error tolerance). We also prove the classical impossibility using techniques in [30], which means that our protocol indeed achieves a type of quantum advantage. Building on the secure function sampling protocol, we further construct protocols for general secure two-party computations [51, 27] with approximate security, with communication complexity only depending on the input length and the targeted security. In terms of the assumptions, we construct protocols for these problems assuming only the existence of collapsing hash functions [48]; what's more, we also construct a classical-channel protocol for these problems additionally assuming the existence of noisy trapdoor claw-free functions [11, 13].

Contents

*zhangjy@zgclab.edu.cn

1 Introduction

1.1 Background

The rapid development of quantum information science leads to the rapid development of quantum cryptography. One remarkable feature about quantum cryptography is that it could allow us to achieve what are not achievable in classical cryptography. [42, 45, 52] For example:

- Quantum key distribution [42] achieves information-theoretic secure key exchange, which is not possible classically.
- Quantum cryptography allows us to make cryptographic primitives unclonable [52, 45].
- Other examples include test of quantumness [11, 13, 32, 50], certified deletion [16, 46], position verification [17], etc.

Discovering new quantum advantage in cryptography is of significant theoretical interest and may even have practical impacts. In this work we focus on the quantum approach for a class of classical cryptographic tasks, which we introduce below.

1.1.1 Secure computations with long outputs

A central topic of cryptography is the secure computation. An example is the Yao's millionaires problem [27]: two millionaires want to compare who is richer, but they do not want to reveal how wealthy they are; so they want to use a cryptographic protocol to solve the problem. The Yao's

millionaires problem is an example of the general *secure two-party computation* problem, which is as follows. Suppose Alice holds a private input x_A and Bob holds a private input x_B, and Alice would like to learn $f_A(x_A, x_B)$ and Bob would like to learn $f_B(x_A, x_B)$. Here f_A, f_B are efficient functions (which are described by, for example, Turing machines or polynomial-size circuits). We would like to design cryptographic protocols that at least satisfy three requirements, the correctness (or called completeness), security (or called soundness) and efficiency, which are roughly as follows:

- (Correctness) When both parties follow the protocol (which is called the honest behavior or the honest setting), Alice gets $f_A(x_A, x_B)$ and Bob gets $f_B(x_A, x_B)$.
- (Security) When some party deviate from the protocol (which is called the malicious setting), the overall behavior of the protocol is as if the protocol is executed by some trusted third party, or called ideal functionality. An adversary is either caught cheating or only learns what it should learn from the trusted third party.

  We note that this discussion is informal and formalizing the security requires some works. And there are also other settings and definitions for the security; here we focus on the security in the malicious setting.
- (Efficiency) Both parties should run in polynomial time.

The study of secure two-party and multi-party computation has become a central topic in cryptography; people have developed many important theories and techniques for its different settings, including garbled circuits, fully homomorphic encryption (FHE), succinct arguments [51, 25, 14, 34], etc.

One important factor in protocol design is the *communication complexity*, that is, the total size of communications. For simplicity of the introduction let's consider the setting where Alice and Bob want to perform secure two-party computation on a public, efficient classical function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$. In classical cryptography, one typical solution for reducing communications is based on FHE and succinct arguments, which achieves communication complexity $O(n + m)$ (omitting the dependency on the security parameter); we refer to [6, 22, 30] for its constructions. What's remarkable for these protocols is that the communication complexity is independent of the time complexity of evaluating f, which shows significant advantage when the time complexity of evaluating f is much bigger than $n + m$.

However, there is still an undesirable output-length dependency in the classical solutions. This means that when the function to be evaluated has long outputs (that is, when m is big), the communication complexity will still be big. Indeed, as proved in [30], this output-length dependency is unavoidable if we want the security in the malicious setting. We note that this output-length dependency is avoided by the following trivially insecure protocol: in the example above, both parties simply reveal their inputs x_A and x_B and compute $f(x_A, x_B)$ on their own. This roughly means that, in the world of classical cryptography, we have to pay the price of the output-length dependency to get good security in general secure two-party computations.

However, quantum cryptography gives us the possibility for overcoming the limitations of classical cryptography. In this background, we ask:

Could quantum cryptography help to reduce communications in secure computations with long outputs?

1.2 Our Contributions

In this work we achieve a type of quantum advantage in the problem above. In summary, we design quantum cryptographic protocols for classical secure computation problems with approximate security, with communication complexity independent of both the function evaluation time and the output length. We first focus on a special problem called secure function sampling, and then study the general secure two-party computations problem. Below we elaborate our results.

1.2.1 A quantum advantage in reducing communications in secure function sampling

We first formalize and study a problem called *secure function sampling* (SFS). SFS is a special case of the general secure two-party computation problem (2PC).

Definitions of secure function sampling The SFS problem is informally defined as follows. We elaborate some missing details in Section 2.2 and the formal definition is given in Section 4.1.

Definition 1.1 (Secure function sampling (SFS), informal)**.** Consider a protocol between two parties, the client and the server. Suppose $f : \{0,1\}^n \to \{0,1\}^m$ is a public, efficient classical function (which is described by a polynomial time classical Turing machine or polynomial-size classical circuits). The protocol should satisfy the following requirements:

- (Correctness) When both parties are honest, in the end of the protocol the client gets a random $x \in \{0,1\}^n$ and the server gets the corresponding $f(x)$.
- (Security) Even if the server is malicious, what it could get from the protocol is no more than one of the following two cases:
  - either the server only gets $f(x)$ as in the honest setting,
  - or the server is caught cheating by the client and gets nothing.

  In this work we consider an approximate variant of the security, where the error tolerance is denoted by ϵ.
- (Efficiency) Both parties should run in polynomial time.

And our goal is to design a protocol for the SFS problem above so that the communication complexity only depends on the input length n and the targeted security.

The setting of SFS is restricted compared to the general 2PC problem; one remarkable restriction is that the $x \in \{0,1\}^n$ is sampled during the protocol as the client-side outputs instead of controlled by the client as client-side inputs. Note that the fact that the primitive is restricted makes the classical impossibility result stronger. What's more, later we will construct protocols for 2PC based on our protocol for SFS.

Classical impossibility By generalizing the results and techniques in [30], we show that the output length dependency is unavoidable for classical SFS protocol, even if we allow the security to be approximate:

Theorem 1.1 (Informal)**.** *Assuming the existence of one-way functions, there exists a family of functions f such that there is no classical protocol that achieves SFS for f with security error tolerance $\epsilon < 1 - O(1)$ within communication complexity* $\mathsf{poly}(n)$.

Quantum protocol Then we construct a quantum cryptographic protocol that solves the SFS problem above.

Theorem 1.2 (Informal)**.** *Assuming the existence of collapsing hash functions, there exists a quantum cryptographic protocol (Protocol 8) for the SFS problem that is ϵ-secure and has communication complexity (including quantum and classical communications)* $\mathsf{poly}(n, 1/\epsilon)$.

Here the collapsing hash function [48] is a quantum generalization of the collision-resistant hash function. It's plausible to assume common cryptographic hash functions, like the SHA-3 [33], satisfy this property [48]; collapsing hash function could also be constructed from the LWE assumption [47]. See Section 3.3.3 for a more detailed discussion.

Finally we note that in the theorem statement above we omit some desirable properties of our protocol. In particular, in our protocol the client-side computation is also succinct (that is, $\mathsf{poly}(n, 1/\epsilon)$).

1.2.2 Protocols for the secure two-party computations

Building on our SFS protocol, we construct protocols for 2PC. As before, we allow for approximate security, and aim at communication complexity that only depends on the input length and the targeted security. We further consider two different settings: in one setting we allow the quantum communication, and in the other setting we only allow the classical communication and local quantum computation. Our results are as follows.

Theorem 1.3 (Informal)**.** *Assuming the existence of collapsing hash functions, there exists a quantum cryptographic protocol (Protocol 12) for the 2PC problem that is ϵ-secure and has communication complexity (including quantum and classical communications)* $\mathsf{poly}(n, 1/\epsilon)$.

Theorem 1.4 (Informal)**.** *Assuming the existence of NTCF and collapsing hash functions, there exists a quantum cryptographic protocol over classical channel (Protocol 14) for the 2PC problem that is ϵ-secure and has communication complexity* $\mathsf{poly}(n, 1/\epsilon)$.

Here NTCF stands for noisy trapdoor claw-free functions [11, 57, 1, 13]; in particular we do not need the adaptive hardcore bit property. The NTCF assumption could be instantiated by the LWE assumption [11, 13]. (See Section 3.3.7 for details.)

1.3 Brief Technical Overview

We give a brief technical overview in this section; a more detailed technical overview is given in Section 2. We focus on two parts in this brief overview: (1) how do we construct the SFS protocol (which corresponds to Theorem 1.2)? (2) how do we go from the SFS protocol to the general secure two-party computations protocol (which corresponds to Theorem 1.3)?

1.3.1 Construction of the SFS protocol

We first give a toy protocol to illustrate the idea behind our protocol.

Very roughly speaking, we would like to design a protocol that works as follows:

1. The client first uses some way to send the function input x to the server, which allows the server to evaluate f and get $f(x)$;

2. Then the client deletes all the information other than the function output $f(x)$.

We first consider the following toy protocol.

1. The client randomly samples $x_0, x_1 \in \{0,1\}^n$ randomly and sends their superposition
$$\frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle)$$
to the server.

2. The server evaluates f and gets the state
$$\frac{1}{\sqrt{2}}(|x_0\rangle\,|f(x_0)\rangle + |x_1\rangle\,|f(x_1)\rangle)$$
The server sends back the input register (holding $|x_0\rangle, |x_1\rangle$) to the client.

3. The client measures this register to collapse the state. If the server is honest the final outputs achieve SFS for f.

We note that this protocol hasn't achieved what we want in several senses; especially we do not have security against malicious servers: the server could simply store the function inputs in step 2 and break the security.

Starting from this toy protocol, we will do a series of revisions and finally construct an SFS protocol. For a brief technical overview, we discuss three ideas in our constructions:

- We will make use of the Hadamard test to test the state and certify the server's behavior. Note that in the toy protocol above, instead of asking the server to send back the input register, the client could ask the server to measure the input register on Hadamard basis; suppose the server's measurement result is $d^{(in)} \in \{0,1\}^n$. Then if the client further asks the server to measure the output register on the Hadamard basis and get $d^{(out)} \in \{0,1\}^m$, the following relation should be satisfied:
$$d^{(in)} \cdot (x_0 + x_1) + d^{(out)} \cdot (f(x_0) + f(x_1)) \equiv 0 \mod 2$$
This provides a powerful test for certifying the server's operations. But a significant drawback is, the honest behavior is affected: if we revise the protocol in this way, once the input register is measured on the Hadamard basis, the client could not know which input (either x_0 or x_1) it should choose corresponding to the server-side state. In later constructions we will combine these ideas and introduce new ideas to design protocols that could both control the malicious server's behavior and preserve the honest setting behavior.

- In the construction of the full protocol, we will need to first construct a forward-succinct protocol, which means, the client-to-server communication complexity only depends on n and the targeted security; then we further compile the protocol to make the server-to-client communication succinct. We formalize this protocol for reducing the server-to-client communication as a primitive that we call "succinct testing for two-party relations", and its construction is a variant of the answer-reduction compiler in [8].

- We will view the SFS problem as a quantum cryptographic primitive called *remote state preparation with verifiability* (RSPV), and use the corresponding framework, techniques and results [26, 12, 57] to work on this problem.

In Section 2.2 to 2.5 we discuss our ideas in more detail.

1.3.2 Construction of the general two-party computations protocol

After the construction of the SFS protocol, we would like to further construct the protocol for general secure two-party computations. We want the communication complexity to be succinct (that is, polynomial in the input size and the targeted security, even when the outputs are long); what's more, we only want to assume the existence of collapsing hash functions. Although we allow for approximate security and could use quantum communication, the construction still requires a significant amount of works.

For a brief technical overview, we show an important step of our construction: building on our SFS protocol, we construct a protocol called *secure function value preparation* (SFVP).

The notion of SFVP is similar to SFS with the following difference: in SFVP the function input x is provided by the client as its private input (for comparison, in SFS x is sampled and the client does not control it). To construct SFVP from SFS, we are going to use a classical cryptographic primitive called *Yao's garbled circuits*.

Background on Yao's garbled circuits Given a circuit C and an input x of the circuit, Yao's garbled circuit allows us to encode the circuit and input as follows:

1. Sample a random string r; the length of r scales with the length of x.
2. Encode C by r and denote the output as $\hat{C}$. We call this the circuit encoding.
3. Encode x by r and denote the output as $\hat{x}$. We call this the input encoding.

Then if the server is given $\hat{C}, \hat{x}$, it could recover $C(x)$; what's more, what the server could know from $\hat{C}, \hat{x}$ is no more than $C(x)$. Finally the Yao's garbled circuits could be constructed assuming only one-way functions, which are implied by collapsing hash functions.

Construction of SFVP Using Yao's garbled circuits and SFS, our SFVP protocol goes as follows:

1. Using SFS to sample the circuit encoding; the client gets a random r and the server gets the corresponding $\hat{C}_f$, where C_f is the circuit description of f.
2. The client uses r to compute and send the input encoding $\hat{x}$ to the server.

In other words, the first two steps in the garbling scheme above are combined and implemented using SFS; note that the communication is succinct since r is succinct.[1]

Finally we briefly discuss other main obstacles and our techniques for constructing a general two-party computations protocol in succinct communication.

- A 2PC protocol needs to take private inputs from both parties; intuitively we could see the the second step in the SFVP protocol above (the client sends the input encoding to the server) does not work any more. To solve this problem, we make use of the oneway-function-based 2PC protocol constructed in [7, 28]; since this 2PC step is only applied on the input encoding part the communication complexity is still succinct regardless of which 2PC protocol we use here.

[1] The size of C_f does not matter as long as f has a succinct description (for example, by a Turing machine): in SFS protocol the client does not need to explicitly compute the circuit description C_f.

- In the SFVP protocol above there is no security guarantee against a malicious client; this means that a malicious client could send some arbitrary value to the server without being detected. We will make use of several cryptographic primitives including the statistically-binding commitment, succinct zero-knowledge argument-of-knowledge, and collapsing hash functions to achieve the soundness against a malicious client.
- In the construction of 2PC protocol we will also encounter a technical issue that seems related to the adaptive security of garbling [31]. To bypass this issue our protocol actually works on the garbling of the garbling of the function.
- Finally we construct the classical-channel 2PC protocol by compiling our quantum-channel 2PC protocol with existing RSPV constructions [12, 57] (corresponding to Theorem 1.4).

1.4 Related Works

Other works on reducing communications In classical cryptography there are a plenty of works for reducing communications in various secure computation problems. These include fully homomorphic encryption (FHE) [25, 14], succinct arguments [34], succinct garbling [35], laconic function evaluation [43], etc. Many of these primitives have achieved succinctness in their settings and definitions. But for our problem, SFS and general two-party computations for functions with long outputs, none of these primitives could make the communication complexity independent to both the function evaluation time and the output length. (As discussed before, this is impossible to achieve in classical cryptography.)

For succinctness in quantum cryptographic protocols, there are a series of works on succinct arguments for QMA [8, 38, 29, 19, 24]. Note that the problem setting of succinct arguments for QMA is different from our work: in succinct arguments for QMA we consider quantum circuits and the inputs and outputs are all public. Another work that discusses succinctness is [55], where the succinctness refers to the size of quantum communication in the protocol (where the size of classical communication is not counted).

Deletion in quantum cryptographic protocols Our construction of the forward-succinct protocols contains a sense of revocation or deletion, which is a quantum phenomenon: recall that the client first sends the input to the server to allow the server to do something, and then this part could be measured on the Hadamard basis and gets destroyed. A series of existing works also make use of some types of deletion or revocation to achieve nontrivial cryptographic tasks, for example, [46, 16, 9, 54]. As far as we know, our usage of this quantum phenomenon is different from these existing works on both the targeted problems and the technical level.

Other quantum advantages There are various types of quantum advantages. Examples include quantum algorithms like the Shor algorithm and what have been discussed in Section 1.1. Other examples include quantum advantage in shallow circuits [15], quantum advantage in communication complexity (without considering the security) [3]. As far as we know, the quantum advantage in the communication complexity of secure computations has not been studied before.

Previous versions This version has been significantly improved compared to its previous versions in 2023 [56]. Many limitations of the protocols in the previous versions have been overcome: previous versions only handle constant error tolerance and this version could achieve inverse-polynomial error

tolerance; this version contains a proof of classical impossibility[2]; this version also contains protocols for secure two-party computations (some of them rely on a result coming out after the previous versions [57]).

1.5 Summary and Discussion

The big message of this work is that we study and discover a new type of quantum advantage in cryptography: we show that for the problem of secure function sampling with approximate security, there exists a quantum cryptographic protocol that achieves this task with communication complexity independent of both the function evaluation time and the output length, while it's impossible to achieve this task solely by classical cryptography. Furthermore, we generalize our results from the secure function sampling problem to the general secure two-party computation problem.

Our results lead to a series of open problems. For example:

- Could we further generalize the problem setting (for example, to quantum circuits) or further weaken the assumptions (for example, to post-quantum one-way functions)?
- We note that for some types of functions, when we do not care about the security, the communication complexity could be even smaller than the input length. This sugguests that there might be some further room for reducing communications in secure computations: could we design secure computation protocols with communication complexity $O(CC(f))$ (where $CC(f)$ denotes the communication complexity of f without considering the security)?
- In our constructions we only achieve approximate security. Could we overcome this issue and make the security loss negligible?
- Could our protocols be made practical?

Acknowledgement

We thank Prabhanjan Ananth, Zhengfeng Ji and anonymous reviewers for discussions and comments. This work is done in Zhongguancun Laboratory.

2 Technical Overview

In this section we give a detailed technical overview for the whole constructions.

2.1 Overview and Organizations

Below we first give a high-level overview, including organizations of this section and the whole paper.

[2]We thank Prabhanjan Ananth for pointing to paper [30]

High-level overview and organization of this section The secure function sampling (SFS) problem is introduced in Section 1.2.1. First, although the goal of the problem is purely classical, to design its quantum protocol, we will first view it as a quantum cryptographic problem called *remote state preparation with verifiability* (RSPV). In Section 2.2 we elaborate the notions of SFS, RSPV and their relations.

In Section 2.3, 2.4 and 2.5 we give an overview for our construction of the quantum protocol for SFS. We first present a forward-succinct protocol, which means, the client-to-server communication is succinct; then we show how to revise it to a fully succinct protocol by a variant of the compiler in [8].

In Section 2.6 we discuss the classical impossibility. The technique is an adaptation of [30].

In Section 2.7 we show how to construct protocols for general two-party computations with succinct communication.

Organization of the whole paper The main content is organized in a slightly different order from this technical overview.

In Section 3 we review the preliminaries of our works.

In Section 4 we formalize the SFS problem, prove the classical impossibility and prepare for the constructions.

In Section 5 and Section 6 we give our construction of the quantum protocol for SFS. We first formalize the protocol that reduces the server-to-client communication (which is a variant of the compiler in [8]), and then construct the full protocol.

In Section 7 we construct our protocols for general two-party computations.

2.2 Secure Function Sampling and Remote State Preparation with Verifiability

As said before, to construct the SFS protocol, we will first view it as an RSPV problem and use the related tools. Below we first explain some more details in the description of the SFS problem, and then review the notion of RSPV and connect SFS to this notion.

Details of the SFS problem The SFS problem is informally introduced in Section 1.2.1. One missing detail is that, in the problem modeling the protocol will take an additional parameter κ called the security parameter, which describes the security level of the protocol. We further clarify some naming and notation: note that the client-side outputs have a special part called flag, whose value is in $\{\mathsf{pass}, \mathsf{fail}\}$, where pass means the protocol completes successfully and both parties should get the desired outputs, and fail means the server is caught cheating (or called aborted, or rejected).

Furthermore, the security is described by a security definition paradigm called simulation-based security. Recall that in Section 1.2.1 we describe the security of SFS as follows: "what it (the malicious server) gets from the protocol is no more than one of the following two cases: ...". Here the "one of the following two cases: ..." is called the ideal functionality of the problem; "what it gets from the protocol" is the real execution of the protocol. Then the formal security statement goes as follows:

$$\forall \text{ poly-time adversarial server } \mathsf{Adv}, \exists \text{ poly-time simulator } \mathsf{Sim} \text{ working on the server side}$$

$$\text{such that } \mathrm{Real}^{\mathsf{Adv}} \approx^{ind} \mathsf{Sim} \circ \mathrm{Ideal}. \tag{1}$$

Here $\text{Real}^{\mathsf{Adv}}$ denotes the real execution against adversary (malicious server) Adv, $\mathsf{Sim} \circ \text{Ideal}$ is the ideal functionality composed with the simulator (note that there might be interaction between Sim and Ideal), and $\approx^{ind}$ denotes the computational indistinguishability. Here the introduction of the simulator, together with the computational indistinguishability, formalizes what it means by saying "... is no more than ..." in the informal claim.

Finally we discuss the approximate security. This is defined by taking the indistinguishability in (1) to be approximate. In other words, no polynomial-time distinguisher could distinguish the two sides of (1) with probability advantage more than ϵ.

We elaborate the details of the security definition paradigm in Section 3.2.

A review of remote state preparation with verifiability (RSPV) RSPV is an important primitive in quantum cryptography. As its basic form, consider a finite set of states $\{|\varphi_1\rangle, |\varphi_2\rangle, \cdots |\varphi_D\rangle\}$. The client would like to sample a random state $|\varphi_i\rangle$ from this set and send this state to the server, so that in the end the client knows the state description (which could be the index i here), while the server only holds the state. In the malicious setting, what the malicious server could get should be no more than one of the following two cases: (1) the server gets the state as in the honest behavior; (2) the server is caught cheating and gets nothing. Existing works give RSPV constructions for several specific state families [26, 21, 54, 2, 57]. We give a more detailed review for RSPV in Section 3.3.8.

[57] provides a framework for working on RSPV; this work will build on this framework.

SFS, RSPV and preRSPV We first note that SFS could be viewed as a variant of RSPV. The problem setting is basically the same: in SFS the server would like to get a classical value $f(x)$; note that classical states could be viewed as a special case of quantum states. Then the state family (the $\{|\varphi_1\rangle, |\varphi_2\rangle, \cdots |\varphi_D\rangle\}$ above) corresponds to the values of f over all the input x.

Then as discussed in [57], its construction could be reduced to a notion called preRSPV, as follows. The preRSPV is defined to be a pair of protocols $(\pi_{\text{test}}, \pi_{\text{comp}})$; these two protocols run in the same set-up and stand for the test mode and the computation mode correspondingly. The soundness (or called security or verifiability) of preRSPV is roughly defined as follows: for any malicious server, if it passes in the test-mode protocol, then the computation-mode protocol achieves what we want (that is, prepare the target output states). On the one hand, a preRSPV could be amplified to an RSPV protocol by sequential repetition, as shown in [57]; on the other hand, preRSPV is usually easier to work on when we construct protocols from scratch.

Below we will construct the preRSPV for our problem. Then it could be amplified to an RSPV, which is exactly a protocol for SFS.

2.3 Construction of the Forward-succinct Protocol for Certain Functions

As said before, we will first construct forward-succinct protocols, which means, the size of the client-to-server communication is $\mathsf{poly}(n, \kappa, 1/\epsilon)$. We will construct a preRSPV (as described in Section 2.2) protocol that is forward-succinct.

In this section we will first consider a toy protocol, given below in Toy Protocol 1, 2. Toy Protocol 1, 2 is a forward-succinct preRSPV protocol for functions with certain desirable properties. In Section 2.4 we will further revise the protocol and get a forward-succinct preRSPV protocol for general functions.

Below we first discuss the protocol in Section 2.3.1, and then discuss the intuition for its soundness in Section 2.3.2.

2.3.1 Protocol design

As said before, in preRSPV we consider a pair of protocols, corresponding to the computation mode and the test mode. We first present the computation-mode protocol.

Toy Protocol 1 (Computation-mode)**.** 1. The client samples and stores two different strings $x_0, x_1 \in \{0,1\}^n$; then the client prepares and sends

$$\frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle) \tag{2}$$

to the server.

2. The server evaluates f on the received state and gets the state

$$\frac{1}{\sqrt{2}}(|x_0\rangle\,|f(x_0)\rangle + |x_1\rangle\,|f(x_1)\rangle).$$

The server sends back the input register (holding $|x_0\rangle, |x_1\rangle$) through a quantum channel back to the client.

Now the joint state between the client and the server is as follows:

$$\frac{1}{\sqrt{2}}(\underbrace{|x_0\rangle}_{\text{client}}\underbrace{|f(x_0)\rangle}_{\text{server}} + \underbrace{|x_1\rangle}_{\text{client}}\underbrace{|f(x_1)\rangle}_{\text{server}}). \tag{3}$$

3. The client measures its quantum state and stores the outcome (either x_0 or x_1) as x. The server holds the corresponding $f(x)$.

We note that Toy Protocol 1 alone already achieves something: for a server that is honest during the protocol execution but becomes malicious after the protocol completes, the security already holds.[3] In the world of classical cryptography this is called the honest-but-curious security; although this type of security is actually trivial to achieve in the quantum world, this is a good beginning for explaining the intuition.

To get the security against a malicious server, we need the following test-mode protocol.

Toy Protocol 2 (Test-mode)**.** 1, 2. The step 1 and step 2 are the same as Toy Protocol 1.

3. The client randomly chooses to execute one of the following two tests with the server:

   - (Computational basis test) The client measures its state on the computational basis and asks the server to measure its state on the computational basis and send back the result. Suppose the client's outcome is x and the server's response is y. The client checks $f(x) = y$ and rejects if the test fails.

[3]We thank Prabhanjan Ananth for raising Toy Protocol 1 as a simplification of the original constructionand giving this interesting observation in a personal discussion.

- (Hadamard basis test) The client measures its state on the Hadamard basis and asks the server to measure its state on the Hadamard basis and send back the result. Suppose the client's outcome is $d^{(in)}$ and the server's response is $d^{(out)}$. The client checks (below $\cdot$ denotes vector inner product)

$$d^{(in)} \neq 0^n \quad \wedge \quad d^{(in)} \cdot (x_0 + x_1) + d^{(out)} \cdot (f(x_0) + f(x_1)) \equiv 0 \mod 2 \tag{4}$$

  and rejects if the test fails.[4]

In other words, starting from state (3), the client will choose to test the server in either computational basis or in the Hadamard basis. We note that some forms of computational basis test and Hadamard basis test have also been used in existing works like [11, 37], but our problem settings, ideas and constructions are different from these existing works.

How could a malicious server attack this protocol? The server might try to store something more than $f(x)$ (for example, x) before sending back the input register (in the second step of the protocol); here the test-mode part is designed to catch it cheating. Recall the preRSPV soundness: if the adversary could pass the test-mode protocol with high probability, then in the computation-mode protocol what it could get is no more than the honest execution output $f(x)$. Below we explain the intuition of why these protocols satisfy this soundness, under the condition that f satisfies certain desirable property.

2.3.2 Intuition for the soundness

We first note that if the server could pass the tests in the test-mode protocol, then starting from the output state of the second step, it should pass both the computational basis test and Hadamard basis test.

Denote the joint state of the client and the server after the second step of the protocol as $|\phi\rangle$. Let's first see what we could say on it under the condition that the server could pass the computational basis test. This implies that the server could recover $f(x)$ on its own side; in other words, we could say $|\phi\rangle$ is in the following form up to a server-side operation:

$$|\phi\rangle \approx \underbrace{|x_0\rangle}_{\text{client}} \underbrace{|f(x_0)\rangle\,|\varphi_0\rangle}_{\text{server}} + \underbrace{|x_1\rangle}_{\text{client}} \underbrace{|f(x_1)\rangle\,|\varphi_1\rangle}_{\text{server}} \tag{5}$$

We could compare (5) with the state from the honest execution, (3): roughly speaking, what remains to be done is to show that for each $b \in \{0,1\}$, $|\varphi_b\rangle$ could be simulated from solely $f(x_b)$. We will make use of the condition that the server could pass the Hadamard test starting from (5), under the extra condition that f satisfies certain desirable property (elaborated below).

Denote the two terms on the right hand side of (5) as $|\phi_0\rangle, |\phi_1\rangle$, and let's call them the 0-branch and the 1-branch; thus $|\phi\rangle \approx |\phi_0\rangle + |\phi_1\rangle$. First it's intuitive to see that these two branches correspond to the two branches in state (2) correspondingly; if the state (2) is replaced by a single branch the state (5) will also become the corresponding branch. Then let's assume the function f has the desirable property that when the protocol starts with a normalized single-branch state (that is, (2) is replaced by a normalized single branch, which is either $|x_0\rangle$ or $|x_1\rangle$), the server could only pass the Hadamard test with $\frac{1}{2}$ probability.[5]

[4] As a convention, below we may omit "the client rejects if the test fails" when we say the client checks something.

[5] For example when f is a random function it satisfies this condition: without loss of generality consider the

So on the one hand each branch of $|\phi\rangle$ has norm only $\frac{1}{\sqrt{2}}$ and each branch (after normalization) could only pass the Hadamard test with probability $\frac{1}{2}$; on the other hand $|\phi\rangle$ passes with high probability (that is, close to 1). By linear algebra calculations we get that the passing space and failing space of these states in the Hadamard test should interfere in the right way:

$$\Pi_{\mathsf{pass}} O_{HT} |\phi_0\rangle \approx \Pi_{\mathsf{pass}} O_{HT} |\phi_1\rangle \tag{6}$$

$$\Pi_{\mathsf{fail}} O_{HT} |\phi_0\rangle \approx -\Pi_{\mathsf{fail}} O_{HT} |\phi_1\rangle \tag{7}$$

where $\Pi_{\mathsf{pass}}, \Pi_{\mathsf{fail}}$ is the projection onto the passing and failing space, O_{HT} is the server's operation in the Hadamard test.

(6)(7) implies that $|\phi_0\rangle$ could be mapped to $|\phi_1\rangle$ (and vice versa) by the following operation:

1. Apply O_{HT};
2. Add a (-1) phase on $d = (d^{(in)}, d^{(out)})$ that satisfies $d^{(in)} \cdot (x_0 + x_1) + d^{(out)} \cdot (f(x_0) + f(x_1)) \equiv 1 \mod 2$;
3. Apply O_{HT}^{-1}.

Recall that we would like to show that for each $b \in \{0, 1\}$, $|\varphi_b\rangle$ could be simulated from solely $f(x_b)$; without loss of generality consider how to simulate $|\varphi_1\rangle$ from solely $f(x_1)$. Let's roughly show why this is true.

First by the discussion above $|\varphi_1\rangle$ could be prepared from $|\phi_0\rangle$ by applying the three-step operation above and disgarding other reigsters. But the step 2 of the three-step operation above makes use of x_0 which is not what we want. To proceed note that the step 2 above could be written as the composition of the following two operations:

- Add a (-1) phase on $d^{(in)}$ that safisfies $d^{(in)} \cdot (x_0 + x_1) \equiv 1 \mod 2$;
- Add a (-1) phase on $d^{(out)}$ that satisfies $d^{(out)} \cdot (f(x_0) + f(x_1)) \equiv 1 \mod 2$.

Note that we only care about $|\varphi_1\rangle$, which lies on the server side of $|\phi_1\rangle$, so we could skip the $d^{(in)}$ part in the operations above.

Finally note that the state $|\phi_0\rangle$ contains basically no information about x_1 and could be simulated by sampling a random x_0. In summary, the operation that simulates $|\varphi_1\rangle$ from solely $f(x_1)$ is as follows:

1. Sample a freshly random x_0 and simulate the corresponding $|\phi_0\rangle$;
2. Apply O_{HT};
3. Add a (-1) phase on $d^{(out)}$ that satisfies $d^{(out)} \cdot (f(x_0) + f(x_1)) \equiv 1 \mod 2$;
4. Apply O_{HT}^{-1}.
5. Keep only the registers corresponding to $|\varphi_1\rangle$.

0-branch, then note that the 0-branch contains basically no information on the value of x_1 or $f(x_1)$, so it's hard for the server to come up with a $d = (d^{(in)}, d^{(out)})$ that passes the check in (4) with probability better than random guessing.

2.4 Construction of the Forward-succinct Protocol for General Functions

In the previous section we present a toy protocol for forward-succinct preRSPV. But the protocol in the previous section has several limitations:

- In Section 2.3.2 we assume an extra condition on the function f; what we want is a protocol for general functions.
- Some properties of Toy Protocol 1, 2 are obstacles for further improving the protocol. These include:
  - The protocol contains backward quantum communication.
  - The initial state (2) does not have an existing RSPV protocol from standard assumptions.

In this section we show how to construct a forward-succinct preRSPV that solves the problem above.

Below we first discuss our protocol in Section 2.4.1, and then discuss its soundness proof in Section 2.4.2.

2.4.1 Protocol design

To solve the problem above, we revise the Toy Protocol 1, 2 as follows:

- To support general functions, we introduce an input padding $(r_0^{(in)}, r_1^{(in)})$ where each of them is sampled from $\{0,1\}^\kappa$. This part is padded into (2) and the Hadamard test is also updated.
- To remove the backward quantum communication, we let the server measure the input register (holding x_0, x_1 in (3)) together with the input padding on the Hadamard basis in step 2, and send back the results. The side effect is, in the computation mode (step 3 of Protocol 1) the client could not know whether it should keep x_0 or x_1. To solve the problem, an output padding $(r_0^{(out)}, r_1^{(out)})$ is introduced and padded into the state (2); this part will be used to test and identify the state on the computational basis.
- Finally, we start with the state $\frac{1}{\sqrt{2}}(|0\rangle |x_0\rangle |r_0^{(in)}\rangle |r_0^{(out)}\rangle + |1\rangle |x_1\rangle |r_1^{(in)}\rangle |r_1^{(out)}\rangle)$ for state (2) (that is, adding a bit holding branch subscripts $0, 1$). By [57] there exists an RSPV protocol for this state from standard assumptions.

In summary, the new protocol is roughly as follows.

Protocol 1 (Test-mode)**.** 1. The client samples and stores the following strings: function inputs $x_0, x_1 \in \{0,1\}^n$; input padding $r_0^{(in)}, r_1^{(in)} \in \{0,1\}^\kappa$; output padding $r_0^{(out)}, r_1^{(out)} \in \{0,1\}^\kappa$. Then the client prepares and sends

$$\frac{1}{\sqrt{2}}(|0\rangle |x_0\rangle |r_0^{(in)}\rangle |r_0^{(out)}\rangle + |1\rangle |x_1\rangle |r_1^{(in)}\rangle |r_1^{(out)}\rangle) \tag{8}$$

to the server.

2. The server evaluates f on the received state and gets the state

$$\frac{1}{\sqrt{2}}(|0\rangle |x_0\rangle |r_0^{(in)}\rangle |r_0^{(out)}\rangle |f(x_0)\rangle + |1\rangle |x_1\rangle |r_1^{(in)}\rangle |r_1^{(out)}\rangle |f(x_1)\rangle).$$

The server measures the registers holding function inputs and input paddings on the Hadamard basis; denote the measurement results as $d^{(in)} \in \{0,1\}^n, d^{(inpad)} \in \{0,1\}^\kappa$. The server sends back these results to the client.

The client checks $d^{(inpad)} \neq 0^\kappa$. Then it stores $\theta_0 = d^{(in)} \cdot x_0 + d^{(inpad)} \cdot r_0^{(in)} \mod 2$ and $\theta_1 = d^{(in)} \cdot x_1 + d^{(inpad)} \cdot r_1^{(in)} \mod 2$.

Now the remaining state on the server side is :

$$\frac{1}{\sqrt{2}}((-1)^{\theta_0} |0\rangle |r_0^{(out)}\rangle |f(x_0)\rangle + (-1)^{\theta_1} |1\rangle |r_1^{(out)}\rangle |f(x_1)\rangle). \tag{9}$$

3. The client randomly chooses to execute one of the following two tests with the server:

   - (Computational basis test) The client asks the server to measure its remaining state on the computational basis and send back the result. The client checks the server's response is in $\{0||r_0^{(out)}||f(x_0), 1||r_1^{(out)}||f(x_1)\}$.
   - (Hadamard basis test) The client asks the server to measure its remaining state on the Hadamard basis and send back the result. Suppose the server's response is $d^{(sub)} \in \{0,1\}, d^{(outpad)} \in \{0,1\}^\kappa, d^{(out)} \in \{0,1\}^m$. The client checks

$$d^{(sub)} + d^{(outpad)} \cdot (r_0^{(out)} + r_1^{(out)}) + d^{(out)} \cdot (f(x_0) + f(x_1)) \equiv \theta_1 - \theta_0 \mod 2. \tag{10}$$

Protocol 2 (Computation-mode)**.** 1, 2. The step 1 and step 2 are the same as Protocol 1.

3. The client asks the server to measure the first two registers (registers holding $|0\rangle |r_0^{(out)}\rangle$ and $|1\rangle |r_1^{(out)}\rangle$) on the computational basis and send back the result.

   The client checks the server's response is either $0||r_0^{(out)}$ or $1||r_1^{(out)}$. The client chooses x_0 as its final outputs in the former case and chooses x_1 as its final outputs in the later case. The server-side remaining state is the corresponding function output.

2.4.2 Discussion on the soundness

Let's first intuitively see how these changes overcome the limitation in Toy Protocol 1, 2. First, with the input padding, regardless of what functions we are working on, an adversary running on a single branch could only pass with $\frac{1}{2}$ probability: note that $d^{(inpad)} \cdot r_0^{(in)}$ (or $d^{(inpad)} \cdot r_1^{(in)}$) has completely randomized the right hand side of (10). Then for the effect of the output padding, intuitively if the server outputs $r_b^{(out)}$ correctly, it means that the state has been collapsed to the b-branch.

The formal proofs are more complicated than this intuitive discussion. To prove the soundness we make use of a security proof guideline in a previous work on RSPV [54]. As in [54], we define different forms of states including *basis-honest form*, *basis-phase correspondance form* and *phase-honest form*, which define stricter and stricter restrictions on the form of states; we then analyze different components of the protocol and show that these different components restrict the form of states step by step. We give a more detailed overview of the proof in Section 6.2.1.

2.5 Construction of the Fully Succinct Protocol

In this section we discuss how to further revise the forward-succinct protocol in the previous section to get a fully succinct protocol, which proves Theorem 1.2. Note that the reason that Protocol 1, 2 is not fully succinct is that the server-to-client communication is long. Below we present a protocol for reducing the server-to-client communication; the construction is a variant of the compiler in [8]. In Section 5 we formalize our protocol and prove its (post-quantum) soundness.

Problem setting Suppose the client holds x and the server claims to the client that it holds a witness w such that $R(x, w) = 0$, where R is a public efficient classical function. The client would like to use a protocol to verify the server's claim in succinct communication; we call this primitive succinct testing for two-party relations. Note that the setting may seem similar to the succinct arguments problem (see Section 3.3.5), but here x is a client-side input (instead of a public one), so succinct arguments do not apply in a direct way. Simply letting the client reveal x and applying succinct arguments do not work either: in this way it is only verified that the server knows such a w after it knows x, but we would like to verify that the server holds w in the very beginning.

Note that such a primitive is sufficient to compile the Protocol 1, 2 to make the server-to-client communication succinct: all the server-to-client communications in Protocol 1, 2 are used for client-side checking of an NP relation (for example, (10)), so these communications together with the checkings could be replaced by callings to the succinct testing protocol. An additional note is that we do not care about the secrecy of x during the protocol; we call it "testing" to suggest the slightly unusual setting that the input x is private in the beginning but is allowed to be revealed during the protocol.

Protocol construction The protocol is roughly as follows.

1. (Commit) The server commits to w using a collapsing hash function h: it computes $h(w) = c$ and sends c to the client.
2. (Prove) The client and the server execute a succinct arguments of knowledge for the statement "$\exists w$ such that $h(w) = c$".
3. (Reveal) The client reveals x to the server.
4. (Prove) The client and the server execute a succinct arguments of knowledge for the statement "$\exists w$ such that $f(x, w) = 0 \wedge h(w) = c$".

The soundness is intuitive: by committing to w, an arguments of knowledge for the statement about w in step 4 should mean that the server knows this w when it does the commitment.

Comparison to existing works We note that the overall structure of our protocol is not new: [8] gives a compiler for reducing the server-to-client communication, and the protocol has the following similar structure: the server does a commitment, both parties execute a succinct argument of knowledge, the client reveals something, and both parties execute a succinct argument of knowledge for both the statement and the commitment. But our problem setting and protocol are technically different from [8]: in [8] the client-side messages are all instance-independent (they are freshly new random bits) and in our problem x is the protocol input (but is allowed to be revealed). To handle these technical differences we re-formalize the protocol.

2.6 Classical Impossibility

In this section we discuss the proof of the classical impossibility. The formal proof is given in Section 4.2.

The proof is an adaptation of the proof in [30]. [30] proves that for the "secure function evaluation" problem it's impossible to construct a classical cryptographic protocol with succinct communication. Our secure function sampling problem is different from the secure function evaluation problem in [30] in the following ways: (1) SFS considers functions with a single input, while in [30] the functions receive private inputs from both parties; (2) in our setting the function input x is sampled during the protocol as a client-side output while in [30] the function inputs are provided by the parties; (3) we allow approximate security while [30] considers normal (negligible soundness error) security. We note that these differences actually make our result stronger. Technically we adapt the ideas and techniques in [30] to our setting and handle technical differences.

Below we give an overview of the idea of the proof. We will prove that when the function is a pseudorandom generator (PRG) this task is classically impossible; note that PRG expands a short seed to a long pseudorandom output string. The proof of impossibility is by a compression argument. Note that for classical protocols, the server-side outputs could be deterministically predicted given the following information: (1) the server's operations, the initial state and the random coins; (2) the client-to-server communication. For constructing the compressor, the first part could be hard-coded and we could focus on the second part. If such an SFS protocol with succinct communication exists, it means that the PRG outputs could be efficiently compressed to something succinct (this is by considering an adversary that outputs the transcripts and calling the simulation-based security) and could be efficiently decompressed. By the properties of PRG this is impossible.

2.7 Protocols for Secure Two-party Computations

In this section we give an overview on how to construct protocols for secure two-party computations based on the SFS protocol.

The roadmap is as follows:

1. We first generalize the function family from classical deterministic functions to classical randomized functions.
2. We then construct a protocol for the setting where the function input is provided as the client-side private input (instead of sampled during the protocol). We call it *secure function value preparation* (SFVP).
3. We then construct a secure two-party computation protocol with succinct communication. An important step for achieving this is how to make the protocol sound against the malicious client.
4. We then show how to compile the protocol with quantum communication to a protocol with fully classical communication.

Note that each step builds upon or is a revision of previous ones. Below we elaborate each step.

2.7.1 From deterministic functions to randomized functions

An efficient randomized classical function could be equivalently modeled by a deterministic function on the function input and the random coins; let's denote it as $f(x;r) : \{0,1\}^n \times \{0,1\}^L \to \{0,1\}^m$.

Note that the number of random coins L could be much bigger than n, and we want a protocol whose communication complexity is also independent to the number of random coins.

The solution is to take a PRG to simulate the random coins: we consider the function $g(x, s) := f(x; \mathrm{PRG}(s))$. Note that the inputs of g are succinct and running our SFS protocol for g gives a protocol with succinct communication. By the property of PRG the outputs of this protocol are indistinguishable[6] to the desired outputs of SFS of f.

2.7.2 Secure function value preparation: from sampled input to client-chosen input

In the original setting of SFS the client could not control the function input x. We then consider the setting where x is a client-side input and the client would like to transmit $f(x)$ to the server; we call it secure function value preparation (SFVP) and design a protocol SFVP for this problem. Our idea is as follows.

We make use of a cryptographic primitive called Yao's garbled circuit [51]: For a classical circuit C and input x, Yao's garbled circuit transforms them into $\mathsf{GarbleC}(C, r)$ and $\mathsf{GarbleI}(x, r)$ where r is the shared random coins; let's call them the circuit encoding and input encoding correspondingly. (We note that $\mathsf{GarbleC}$ could itself be a randomized function for fixed (C, r); so a deterministic function description of $\mathsf{GarbleC}$ could be written as $\mathsf{GarbleC}(C, r; r_{\mathrm{GCin}})$.) This primitive has the following properties:

- The server could recover $C(x)$ given $\mathsf{GarbleC}(C, r)$ and $\mathsf{GarbleI}(x, r)$.
- $\mathsf{GarbleC}(C, r)$ and $\mathsf{GarbleI}(x, r)$ could be simulated from solely $C(x)$; intuitively this means that when the server is given $\mathsf{GarbleC}(C, r)$ and $\mathsf{GarbleI}(x, r)$, what it knows is no more than $C(x)$.
- As shown in the notations, the computation of $\mathsf{GarbleC}(C, r)$ does not use x and the computation of $\mathsf{GarbleI}(x, r)$ does not use C. What these two algorithms share are some random coins r. This property is called "decomposable" in garbled circuit primitive. We additionally note that the shared coins r is also succinct: its size only depends on the length of x and the security parameter κ.

Now we could construct the SFVP protocol for f and the client-side input x as follows.

1. The client and the server execute an SFS protocol for function $g(r) := \mathsf{GarbleC}(C_f, r)$, where C_f is the circuit description of f. The client should get a random r and the server gets the corresponding $\mathsf{GarbleC}(C_f, r)$.
2. The client computes and sends $\mathsf{GarbleI}(x, r)$ to the server.

Note that the Yao's garbled circuits could be constructed from one-way functions, which is implied by collapsing hash functions.

2.7.3 Generalization to two-party computations

Now we would like to further generalize our protocols to the general two-party computations, where both parties could have private inputs and private outputs. To overcome the obstacles we need a combination of several ideas, which we discuss below.

[6] Note that here the definition of the completeness is also based on indistinguishability.

Use two-party computations to transmit input encoding For introducing the first idea we first consider a single function $f(x_A, x_B)$ as the outputs, which takes input x_A from Alice and x_B from Bob. If we simply use the SFVP protocol, the circuit encoding step could be executed normally while the input encoding step needs to take inputs from both parties. How could Alice and Bob do this securely? The first idea is as follows: the transmission of input encoding, which is $\mathsf{Garblel}((x_A, x_B), r)$, is performed by a two-party computation. Note that this step is succinct since it's only about x_A, x_B, r, which are all succinct; the computation of Garblel is also relatively easy. Then this two-party computation step could be achieved quantumly assuming only one-way functions by existing works [7, 28].

Obstacles in the malicious setting Recall that we are working in the malicious setting where the parties could deviate from the honest behavior maliciously. Examining the security of the constructions so far, there are still two unsolved obstacles for achieving general secure two-party computations:

- In the study of SFS we only care about the security against a malicious server; in two-party computations either Alice or Bob could be malicious. So we need some type of security guarantee against a malicious client in the SFVP protocol.
- A more subtle obstacle is that in the proof of simulation-based security of garbling-based protocol, if the adversary first receives the circuit encoding, the simulator needs to simulate the circuit encoding first; then the simulator does not have an easy way to simulate the input encoding so that the simulated circuit encoding and simulated input encoding still decode to $C(x)$.[7]

Our ideas are as follows:

- We design a new secure function value preparation protocol that also achieves a type of security against a malicious client. The construction makes use of the combination of statistically-binding computationally-hiding string commitments, collapsing hash functions and post-quantum succinct zero-knowledge arguments of knowledge (succinct-ZK-AoK) to control the client's behavior. This protocol is denoted as SFVP2.
- In the construction of the two-party computation protocol both parties will first transmit the input encoding and then transmit the circuit encoding (with SFVP2). Note that since SFVP2 itself is based on SFVP so we are actually performing the garbling technique twice: in SFVP we first transmit the circuit encoding and then transmit the input encoding, while in the two-party computation protocol we first transmit the input encoding and then the circuit encoding.

Below we elaborate our ideas and constructions.

Secure function value preparation with security against malicious clients The SFVP2 protocol relies on Naor's statistically-binding string commitment scheme [39], denoted by Commit (see Section 3.3.2 for details): to commit to a string x, the receiver first samples public randomness pp, and then the sender samples r and calculates $\mathsf{Commit}(x, r, pp)$. To open the commitment the sender simply reveals r.

[7] We suspect this obstacle is related to the notion of adaptive security of garbled circuits.

The setting of $\mathsf{SFVP2}$ is as follows, which is slightly unusual. Initially the client chooses x as the protocol input; the client also holds a random string r as the commitment randomness. The server holds the commitment $com = \mathsf{Commit}(x, r, pp)$. pp is randomly sampled and is public. In summary we assume the client has already committed to its function input before the protocol. The security against malicious clients is defined using the simulation-based paradigm, under the condition that the initial setup is done honestly. In more detail, roughly speaking, the soundness against malicious client says that what the client sends to the server should be exactly $f(x)$ where x is the committed string; otherwise the client will be caught cheating.

Now the construction of $\mathsf{SFVP2}$ is roughly as follows.

1. Both parties execute the SFVP protocol to send $f(x)$ to the server.
2. The server uses a collapsing hash function h to calculate $h(f(x)) = c$. The server sends h, c to the client.
3. Using a succinct-ZK-AoK, the client proves the following statement to the server: "$\exists(x, r)$ such that $h(f(x)) = c \wedge \mathsf{Commit}(x, r, pp) = com$".

Note that all the basic primitives used above could be constructed from collapsing hash functions [39, 20, 34]. The security against malicious clients is intuitive: by the commitment and the proof-of-knowledge the client holds x; note that the statistically-binding property of the commitment guarantees that x is unique. Then by the $h(f(x)) = c$ and the collision-resistance of h we know what the server gets should be $f(x)$. The ZK property is used to preserve the security against malicious servers.

Finally we note that the intuitive discussion of the client-side soundness above skipped an important obstacle: note that the discussion above implicitly assumes that the client passes the PoK with probability close to 1; in general either aborting or non-aborting could happen with some probability and constructing a simulator that dealing with both cases is not easy. Here our solution is to further revise the protocol above as follows: the server will repeat the third step above for $\mathsf{poly}(1/\epsilon)$ rounds (where ϵ is the error tolerance on the approximate soundness), and requires that the client should pass the checking in each round. Then the simulator could do a cut-and-choose: if the simulator randomly picks a round in them, with high probability the client should pass the PoK verification with high probability conditioned on it passes in all the previous rounds.

Construction of the secure two-party computations protocol Now we could construct our secure two-party computations protocol as follows. We consider the general setting where there are both Alice-side output function $f_A(x_A, x_B)$ and Bob-side output function $f_B(x_A, x_B)$.

1. Both parties execute a two-party computation for the following two tasks:
   - Transmit the suitable input encoding;
   - Commit to the randomness used in the input encoding.
2. Alice uses $\mathsf{SFVP2}$ to transmit the circuit encoding of f_B to Bob. An honest Bob could decode and get $f_B(x_A, x_B)$.
3. Bob uses $\mathsf{SFVP2}$ to transmit the circuit encoding of f_A to Alice. An honest Alice could decode and get $f_A(x_A, x_B)$.

Finally we note that to prove the simulation-based soundness we need a stronger form of soundness for the garbled circuit scheme: instead of letting one simulator simulate both the input encoding and circuit encoding, the input encoding is now simulated first by an input encoding simulator and the circuit encoding is simulated later by a circuit encoding simulator (given the simulated input encoding and the circuit outputs). Luckily Yao's garbled circuit satisfies this slightly stronger soundness requirement.

2.7.4 De-quantizing the communication

Finally we ask: is it possible to construct protocols for secure two-party computations using classical channel and local quantum computations? Note that in our protocol the quantum communication comes from the following two steps:

- In SFS protocol the client prepares and sends states in the form of $\frac{1}{\sqrt{2}}(|0\rangle |x_0\rangle |r_0^{(in)}\rangle |r_0^{(out)}\rangle + |1\rangle |x_1\rangle |r_1^{(in)}\rangle |r_1^{(out)}\rangle)$ to the server.
- We make use of secure two-party computations from one-way functions [7, 28], in which one party prepares and sends BB84 states to the other during the protocol.

We make use of existing RSPV protocols for these states to replace these prepare-and-send steps. By [57] there exist RSPV protocols for these types of states assuming NTCF. Suppose in the original protocol Alice prepares and sends some states to Bob; after this step is replaced by the corresponding RSPV protocols, the soundness of the new protocol is preserved by the following reason:

- When Bob is malicious, by the soundness of RSPV what the malicious Bob could do is no more than the ideal functionality (caught cheating or honest behavior) up to an approximation error.
- When Alice is malicious, note that the effect of the RSPV protocols is no more than the following operations: Alice simulates and prepares the joint output states of RSPV and sends the simulated Bob's part to Bob. (Note that RSPV protocols do not take private inputs from Bob.) If the original protocol is sound against malicious Alice the new protocol should also be sound.

3 Preliminaries

In this section we review and clarify basic notions and notations, modeling and basic properties of cryptographic protocols, and important cryptographic primitives.

Note that many choices of notions and notations, including their definitions, would be similar to [57].

3.1 Notations and Basics

We first clarify basic mathematical notations.

Notation 3.1. We use $\mathbb{N}$ to denote the natural numbers (positive integers). We use 1^κ to denote the string $\underbrace{111\cdots1}_{\text{for } \kappa \text{ times}}$. We use $[n]$ to denote the set $\{1, 2, \cdots n\}$. We use $\leftarrow_r$ to denote random sampling.

We use $||$ to denote string concatenation. We use $|\cdot|$ to denote the Euclidean norm of a vector. We also use $|C|$ to denote the size of a circuit C. We use $a \equiv b \mod m$ to denote the congruence modulo m. We use $\cdot$ to denote the inner product between vectors. We use $\mathrm{Dist}(S)$ to denote the set of probability distribution on S. We write $a \approx_\epsilon b$ when $|a-b| \le \epsilon$ where a, b are real numbers.

We refer to [41, 49] for basics of quantum computation. Below we review or clarify notations.

Notation 3.2. We use Dirac notation $|\varphi\rangle$ to denote pure states. $|\varphi\rangle \in \mathcal{H}$ means that $|\varphi\rangle$ is in the Hilbert space $\mathcal{H}$. We call $|\varphi\rangle$ normalized if $\||\varphi\rangle\| = 1$.

We use $\mathrm{D}(\mathcal{H})$ to denote the set of density operators on a Hilbert space $\mathcal{H}$. This could be seen as the quantum analog of probability distribution in the classical world.

Quantum operations on pure states are described by unitaries. The output state of applying a unitary quantum operation U on quantum state $|\varphi\rangle$ is $U|\varphi\rangle$.

In general, quantum operations are described by completely-positive trace-preserving (CPTP) maps on density operators. The output state of applying a quantum operation $\mathcal{E}$ on quantum state ρ is denoted as $\mathcal{E}(\rho)$.

We work on registers, which is very convenient.

Notation 3.3. A quantum register is a labeled Hilbert space. We use bold font (like $\boldsymbol{S}$) to denote registers; the corresponding Hilbert space is $\mathcal{H}_{\boldsymbol{S}}$.

Then we use $\otimes$ to denote the tensor product and $\mathrm{tr}_{\boldsymbol{R}}$ to denote the partial trace that traces out register $\boldsymbol{R}$.

The joint state on a classical register and a quantum register is call a cq-state, which could also be described by density operators.

We use $\Pi_v^{\boldsymbol{reg}}$ to denote the projection onto the space that the register $\boldsymbol{reg}$ is in value v, and use $\Pi_S^{\boldsymbol{reg}}$ to denote the projection onto the space that the value of register $\boldsymbol{reg}$ is in set S.

Notation 3.4. We write $|\varphi\rangle \approx_\epsilon |\phi\rangle$ if $\||\varphi\rangle - |\phi\rangle\| \le \epsilon$.

We write $\rho \approx_\epsilon \sigma$ if $\mathrm{TD}(\rho, \sigma) \le \epsilon$ where TD denotes the trace distance between $\rho, \sigma \in \mathrm{D}(\mathcal{H})$.

Fact 1 (Purification)**.** *For each $\rho \in D(\mathcal{H}_{\boldsymbol{S}})$, there exists $|\phi\rangle \in \mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{R}}$ such that $\mathrm{tr}_{\boldsymbol{R}}(|\phi\rangle\langle\phi|) = \rho$. We say $|\phi\rangle$ is a purification of ρ.*

If $|\phi\rangle, |\phi'\rangle \in \mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{R}}$ are two purification of ρ, then there exists a unitary U operating on $\mathcal{H}_{\boldsymbol{R}}$ such that $U|\phi\rangle = |\phi'\rangle$.

If $|\phi\rangle, |\phi'\rangle \in \mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{R}}$ are purifications of ρ, ρ' correspondingly, $|\phi\rangle \approx_\epsilon |\phi'\rangle$, then $\rho \approx_\epsilon \rho'$.

We refer to [5] for basics of complexity theory. Below we review or clarify notions and notations.

Notation 3.5. We use poly to denote a polynomial function (that is, $O(n^c)$ and $\Omega(n^{-c})$ where $c \in \mathbb{N}$), and use negl to denote a negligible function (that is, converges to 0 faster than any inverse polynomial).

Notation 3.6. We say a family of classical functions $f = (f_n)_{n\in\mathbb{N}}$, $f_n : \{0,1\}^n \to \{0,1\}$ is classical polynomial-time computable if it could be computed by a classical Turing machine TM in polynomial time.

This notion also applies to function families with both input length parameters and output length parameters: we say $f = (f_{n,m})_{n\in\mathbb{N},m\in\mathbb{N}}$, $f_{n,m} : \{0,1\}^n \to \{0,1\}^m$ is classical polynomial-time computable if the same definition holds.

Notation 3.7. A classical circuit C could be described by $((g_i)_{i\in[|C|]}, \text{topo})$ where $(g_i)_{i\in[|C|]}$ is a sequence of elementary classical gates and topo is the topology of the circuit (which includes the circuit graph and the identification of input wires and the output wires).

A quantum circuit is defined in the same way with $(g_i)_{i\in[|C|]}$ being a sequence of elementary quantum gates.

We use $C(x)$ to denote the outputs of evaluating a circuit C on x, and use $C(\rho)$ to denote the outputs of evaluating a quantum circuit C on initial state ρ.

Notation 3.8. We use $\Pr[C(\rho) \to o]$ to denote the probability that running a quantum circuit C on state ρ gives output o.

Notation 3.9. We say a family of circuit $C = (C_\kappa)_{\kappa\in\mathbb{N}}$ is polynomial-size if $|C_\kappa| = \mathsf{poly}(\kappa)$ and say it's polynomial-time uniform if the circuit description is classical polynomial-time computable.

Then we review important complexity classes.

Notation 3.10. We say a language L is in complexity class P if it could be decided by a classical polynomial-time Turing machine.

We say a language L is in complexity class NP if there exists a classical polynomial-time Turing machine TM:

- (Completeness) For each $x \in L$, there exists a witness w such that $TM(x, w) = 1$.
- (Soundness) For each $x \notin L$, for any w there is $TM(x, w) = 0$.

We say $(L_{\text{yes}}, L_{\text{no}})$ is in complexity class QMA if there exists a polynomial-time uniform family of quantum circuits V:

- (Completeness) For each $H \in L_{\text{yes}}$, there exists a quantum state w such that $\Pr[V(H, w) \to 1] \geq \frac{2}{3}$.
- (Soundness) For each $H \in L_{\text{no}}$, for any quantum state w there is $\Pr[V(H, w) \to 0] \geq \frac{2}{3}$.

Finally we note that there are non-uniform variants of these complexity classes that take advice strings (or advice states). For example:

We say a $(L_{\text{yes}}, L_{\text{no}})$ is in complexity class BQP/qpoly if there exists a polynomial-time uniform family of quantum circuits V and a family of polynomial-size advice state $(v_\kappa)_{\kappa\in\mathbb{N}}$:

- (Completeness) For each $C \in L_{\text{yes}}$, $\Pr[V(C, v_{|C|}) \to 1] \geq \frac{2}{3}$.
- (Soundness) For each $C \in L_{\text{no}}$, $\Pr[V(C, v_{|C|}) \to 0] \geq \frac{2}{3}$.

Convention 1. Notation 3.6, 3.9 formalize what we mean when we say a function is efficient. Note that for classical functions its efficiency could be described by either Turing machines (Notation 3.6) or circuits (Notation 3.9), while for quantum circuits we usually do not work on the quantum analog of Turing machines but work on quantum circuits instead.

As a convention, in this paper we consider non-uniform version of these notions by default. For example, when we consider efficient quantum operations, we are considering a polynomial-time uniform family of quantum circuits V together with a family of polynomial-size advice states $(v_\kappa)_{\kappa\in\mathbb{N}}$.

Finally the following fact is about the description of efficient classical randomized functions.

Fact 2. *An efficient classical randomized function* $\{0,1\}^n \to Dist(\{0,1\}^m)$ *could be described by a deterministic function on the inputs and random coins:* $f : \{0,1\}^n \times \{0,1\}^L \to \{0,1\}^m$ *where* L *is the number of gates for evaluating the function.*

Note that when we say efficient functions we consider deterministic functions by default (but in this paper we will also encounter lots of randomized functions). Another convention is that when we use two different representations of efficient randomized functions (as in Fact 2) we use semicolon to seprate the random coins: for example, the randomized function $f(x), x \in \{0,1\}^n$ as in Fact 2 could be represented by a deterministic function $f(x;r), x \in \{0,1\}^n, r \in \{0,1\}^L$.

3.2 Basics of Cryptography and Quantum Cryptography

Below we review the hierarchy of notions for modeling the cryptographic protocols in classical and quantum cryptography.

3.2.1 Modeling

Value and registers As said before, it's convenient to work on registers for describing quantum states. For classical data, we could either describe them by values or say they're in some classical registers.

Interactive protocols In this work we only need to consider interactive protocols between two parties; we call them either client-server or Alice-Bob. For modeling the security we also need to consider an environment, which describes everything else outside the protocol execution.

The protocol execution is synchronous, that is, the execution of protocols could be divided into cycles of four phases: the client (or Alice) does some local operations; the client sends messages to the server (or Bob); the server does some local operations; the server sends messages back to the client.

The operation of a protocol against an adversary (for example, a malicious server) could be described as

$$\text{ProtocolName}^{\mathsf{Adv}}(\text{vals}, \boldsymbol{regs}) \tag{11}$$

Suppose the initial state (on the registers considered) is ρ_0, then the output state could be described as

$$\text{ProtocolName}^{\mathsf{Adv}}(\text{vals}, \boldsymbol{regs})(\rho_0)$$

We say a protocol π is a classical protocol if the operations in the honest setting are fully classical.

Inputs and outputs Besides the inputs and outputs related to the tasks, the following inputs and outputs are commonly used. The protocol could take an additional input 1^κ, which is called the security parameter, for formalizing the security (see below). For many problems the output registers will contain a special register $\boldsymbol{flag}$, whose value is in $\{\mathsf{pass}, \mathsf{fail}\}$, where pass means the protocol completes successfully (we also say the client accepts the results in the client-server setting) and fail means the malicious party is caught cheating (we also say the protocol aborts or the client rejects in the client-server setting).

We use Π_{pass} to denote the projection on register $\boldsymbol{flag}$ to the passing space.

Modeling of cryptographic primitives One typical template of definitions of cryptographic primitives is as follows. We would like the protocol to satisfy at least the following requirements:

- (Correctness, or called completeness) When both parties are honest, the protocol completes successfully and the task is achieved.
- (Security, or called soundness) The protocol holds some security properties against a malicious attacker. For example: for any malicious attacker, if the task is not achieved correctly, the attacker is caught cheating.
- (Efficiency) Both parties run in polynomial time.

For the security, there are two settings:

- Security in the classical world: the adversary is completely classical.
- Security in the (post-)quantum[8] world: the adversary could be quantum.

In this paper we consider security in the (post-)quantum world by default.

Formalizing the definition of the security is nontrivial. A paradigm for formalizing the security is based on the simulation, which we discuss below.

3.2.2 Simulation-based security definition paradigm

We first review the notions and notations of state family and indistinguishability.

Convention 2. We note that when we study the security, we consider a family of states parameterized by the security parameter κ, for example, $(\rho_\kappa)_{\kappa\in\mathbb{N}}$. For notation simplicity we simply write ρ and make the security parameter and the underlying state family implicit. We note that when we talk about efficient operations there is also an implicit security parameter κ; so when we talk about $D(\rho)$ where D is an efficient quantum operation, we are actually talking about $(D(1^\kappa)(\rho_\kappa))_{\kappa\in\mathbb{N}}$. We omit these parameters for simplicity.

Definition 3.1. We write $\rho \approx_\epsilon^{\text{ind}} \sigma$ if for any efficient quantum operation D, there exists a negligible function negl such that

$$|\Pr[D(\rho)\to 0] - \Pr[D(\sigma)\to 0]| \le \epsilon + \mathsf{negl}(\kappa).$$

We say ρ is ϵ-indistinguishable to σ and D is called the distinguisher.

As reviewed in Section 2.2, the simulation-based security definition paradigm considers the indistinguishability between the "real world" (real execution) and the "ideal world" (ideal functionality). To formalize the security, we first need to define the ideal functionality Ideal. One typical template of definition for the ideal functionality is as follows:

Example 3.1. Consider the client-server setting in this example. Ideal receives a bit b from the server and it works as follows:

- If $b = 0$, Ideal sets $\boldsymbol{flag}$ to be pass and sets the outputs to be the honest behavior outputs.

[8] "Post-quantum" means the protocol is classical while the adversary is quantum.

- If $b = 1$, Ideal sets $\boldsymbol{flag}$ to be fail.

Then as discussed in Section 2.2, the simulation-based security is defined to be the indistinguishability of the real execution (that is, (11)) and the ideal functionality composed with the simulation (that is, $\mathsf{Sim} \circ \text{Ideal}$); see also equation (1) in Section 2.2. If we expand the expression, it goes as follows:

Example 3.2. Consider the client-server setting as in Example 3.1. We say a protocol π is ϵ-sound if the following holds.

For any efficient quantum adversary Adv there exist efficient quantum operations $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$ there is

$$\pi^{\mathsf{Adv}}(\rho_0) \approx_\epsilon^{\text{ind}} \underbrace{\mathsf{Sim}_1}_{\text{server side}} (\text{Ideal}(\underbrace{\mathsf{Sim}_0}_{\text{server side and outputs } b} (\rho_0))) \tag{12}$$

Here $\boldsymbol{S}$ is the register corresponding to the server, $\boldsymbol{E}$ is the environment. Note that the distinguisher in (12) has access to all the inputs, the client-side and server-side outputs and the environment.

We note that in this paper we need to consider approximate security, as in the definitions above. And we note that in (12) the operations implicitly take the public parameters as their parameters, for example, the security parameter κ; the error tolerance ϵ could also be part of the public parameters.

3.3 Important Cryptographic Primitives

In this section we review basic cryptographic primitives and related results that will be used in our work.

3.3.1 One-way functions and pseudorandom generators

One-way functions and pseudorandom generators are both very basic primitives in cryptography. [33] One-way functions are functions that are easy to compute but hard to invert. Formally:

Definition 3.2 (One-way function)**.** We say an efficient function family $(f_\kappa)_{\kappa\in\mathbb{N}}, f_\kappa : \{0,1\}^\kappa \to \{0,1\}^*$ is a family of post-quantum one-way function if the following holds: for any efficient quantum operation Adv, there is a negligible function $\mathsf{negl}(\kappa)$ such that

$$\Pr_{x\leftarrow\{0,1\}^\kappa}[f(\mathsf{Adv}(f(x))) = f(x)] \leq \mathsf{negl}(\kappa)$$

Pseudorandom genrators (PRG) are functions whose outputs are indistinguishable to random outputs. Formally:

Definition 3.3 (PRG)**.** We say an efficient function family $\mathrm{PRG}(1^\kappa, 1^L) : \{0,1\}^\kappa \to \{0,1\}^L$ is a family of post-quantum pseudorandom generator if the following holds: for any efficient quantum operation Adv, for any $L = \mathsf{poly}(\kappa)$, there is a negligible function $\mathsf{negl}(\kappa)$ such that

$$\Pr\begin{bmatrix} s \leftarrow_r \{0,1\}^\kappa \\ \mathsf{Adv}(\mathrm{PRG}(1^\kappa, 1^L)(s)) \to 0 \end{bmatrix} \approx_{\mathsf{negl}(\kappa)} \Pr\begin{bmatrix} u \leftarrow_r \{0,1\}^L \\ \mathsf{Adv}(u) \to 0 \end{bmatrix} \tag{13}$$

Notation 3.11. Note that the notation in (13) denotes the probability of the last row when the variables are sampled as above.

Another notation to express it is

$$\mathrm{Dist}\begin{bmatrix} s \leftarrow_r \{0,1\}^\kappa \\ \mathrm{PRG}(1^\kappa, 1^L)(s) \end{bmatrix} \approx^{\mathrm{ind}} \mathrm{Dist}\begin{bmatrix} u \leftarrow_r \{0,1\}^L \\ u \end{bmatrix} \tag{14}$$

which denotes the indistinguishability between the distributions of the last rows.

A well-known fact is [53]:

Theorem 3.1. *The existence of post-quantum one-way functions is equivalent to the existence of post-quantum PRGs.*

Convention 3. In this paper we often omit "post-quantum" and simply say, for example, one-way functions or PRGs.

3.3.2 Commitments

Commitment is a very basic primitive in cryptography. There are different ways to formalize this primitive and we formalize it following the Naor's construction [39].

Definition 3.4. We say an efficient function family $\mathsf{Commit}(1^n, 1^\kappa) : \{0,1\}^n \times \{0,1\}^{n\kappa} \times \{0,1\}^{3n\kappa}$ is a statistically-binding computationally-hiding commitment scheme if it satisfies:

- (Computationally hiding) For any efficient quantum adversary Adv, for any $n = \mathsf{poly}(\kappa)$, there is a negligible function $\mathsf{negl}(\kappa)$ such that

$$\begin{aligned} &\Pr\begin{bmatrix} \mathsf{Adv} \to (m_0, m_1, pp) \in \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{3n\kappa} \\ r \leftarrow_r \{0,1\}^{n\kappa} \\ \mathsf{Adv}(\mathsf{Commit}(m_0, r, pp)) = 0 \end{bmatrix} \\ \approx_{\mathsf{negl}(\kappa)} &\Pr\begin{bmatrix} \mathsf{Adv} \to (m_0, m_1, pp) \in \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{3n\kappa} \\ r \leftarrow_r \{0,1\}^{n\kappa} \\ \mathsf{Adv}(\mathsf{Commit}(m_1, r, pp)) = 0 \end{bmatrix} \end{aligned}$$

- (Statistically binding) For any $n = \mathsf{poly}(\kappa)$ there is a negligible function $\mathsf{negl}(\kappa)$ such that

$$\Pr\begin{bmatrix} pp \leftarrow_r \{0,1\}^{3n\kappa} \\ \exists (m_0, r_0, m_1, r_1) : \mathsf{Commit}(m_0, r_0, pp) = \mathsf{Commit}(m_1, r_1, pp) \wedge m_0 \neq m_1 \end{bmatrix} \leq \mathsf{negl}(\kappa) \tag{15}$$

Here we call pp the public randomness and r the sender-side randomness.

Theorem 3.2 (By [39])**.** *There exists a statistically-binding computationally-hiding commitment scheme assuming the existence of one-way functions.*

A typical commitment protocol based on Commit is as follows.

1. Committing phase:
   (a) The receiver samples $pp \leftarrow_r \{0,1\}^{3n\kappa}$ and sends it to the sender.

(b) The sender samples and stores $r \leftarrow_r \{0,1\}^{n\kappa}$. The sender computes and sends $\mathsf{Commit}(x, r, pp)$ to the receiver as the commitment.

2. Opening phase:

   (a) The sender sends r to the receiver.

   (b) The receiver checks $\mathsf{Commit}(x, r, pp) = com$.

3.3.3 Collapsing hash functions

Collapsing hash functions are the quantum generalization of the collision-resistant hash functions. For convenience we first review the collision-resistant hash functions.

Definition 3.5 (Collision-resistant hash functions)**.** Suppose $\mathsf{Hash}(1^n, 1^\kappa)$ is an efficient procedure that samples functions from $\{0,1\}^n$ to $\{0,1\}^\kappa$; we additionally require that the description length of h sampled from $\mathsf{Hash}(1^n, 1^\kappa)$ is $\mathsf{poly}(\kappa)$ (note that the description could be based on Turing machines). We say Hash is a family of collision-resistant hash functions if the following holds: for any efficient quantum adversary Adv, for any $n = \mathsf{poly}(\kappa)$, there exists a negligible function $\mathsf{negl}(\kappa)$ such that

$$\Pr\begin{bmatrix} h \leftarrow_r \mathsf{Hash}(1^n, 1^\kappa) \\ \mathsf{Adv}(h) \to (x_0, x_1) \\ h(x_0) = h(x_1) \end{bmatrix} \leq \mathsf{negl}(\kappa)$$

Note that although Definition 3.5 is already stated in the post-quantum setting, this definition is usually not sufficient when we need the quantum analog of collision-resistance. What we need is the following notion, the collapsing hash functions [48]:

Definition 3.6. Consider Hash whose basic set-up is as in Definition 3.5. Suppose $\boldsymbol{X}$ is a quantum register holding n qubits. Define the following two operations:

- Define $\mathsf{Eval}(h, \boldsymbol{X})$ as follows, where h is a function from $\{0,1\}^n$ to $\{0,1\}^\kappa$: $\mathsf{Eval}(h, \boldsymbol{X})$ evaluates h on $\boldsymbol{X}$ and measures and disgards the results (that is, only $\boldsymbol{X}$ is kept in the end).
- Define $\mathsf{Collapse}(\boldsymbol{X})$ as the operation that collapses $\boldsymbol{X}$ on the standard basis.

We say Hash is a family of collapsing hash functions if the following holds: for any efficient quantum adversary Adv, for any $n = \mathsf{poly}(\kappa)$, there exists a negligible function $\mathsf{negl}(\kappa)$ such that

$$\Pr\begin{bmatrix} h \leftarrow_r \mathsf{Hash}(1^n, 1^\kappa) \\ \mathsf{Adv}(h) \to \rho \in \mathrm{D}(\mathcal{H}_{\boldsymbol{X}} \otimes \mathcal{H}_{\boldsymbol{S}}) \\ \mathsf{Eval}(h, \boldsymbol{X}) \\ \mathsf{Adv}(h, \boldsymbol{X}, \boldsymbol{S}) \to 0 \end{bmatrix} \approx_{\mathsf{negl}(\kappa)} \Pr\begin{bmatrix} h \leftarrow_r \mathsf{Hash}(1^n, 1^\kappa) \\ \mathsf{Adv}(h) \to \rho \in \mathrm{D}(\mathcal{H}_{\boldsymbol{X}} \otimes \mathcal{H}_{\boldsymbol{S}}) \\ \mathsf{Collapse}(\boldsymbol{X}) \\ \mathsf{Adv}(h, \boldsymbol{X}, \boldsymbol{S}) \to 0 \end{bmatrix}$$

The following results are from existing works [47, 48].

Theorem 3.3. *If* Hash *is collapsing, then it's collision-resistant.*

Theorem 3.4. *Existence of collapsing hash functions implies the existence of one-way functions.*

Theorem 3.5. *There exists a family of collapsing hash functions assuming the LWE assumption.*

The LWE [44] assumption is a widely-accepted assumption with security against quantum computations.

Theorem 3.6. *There exists a family of collapsing hash functions in the quantum random oracle model.*

The quantum random oracle model (QROM) [10] is a widely-used idealized model for modeling hash functions. Although the constructions in QROM are not always instantiable [18, 23], it provides a good evidence that hash functions used in reality, like SHA-3, are indeed collapsing.

3.3.4 Secure two-party computations

Below we review the notion of secure two-party computations [27]. We first formalize its set-up. Here we consider its general form, where both parties have private inputs and both parties receive private outputs.

Set-up 1. The set-up for general secure two-party computations is as follows. We consider a protocol between Alice and Bob.

The inputs of the protocol are:

- Public parameters: problem sizes $1^{n_A}, 1^{n_B}, 1^{m_A}, 1^{m_B}$, where A means Alice, B means Bob, n means the input size and m means the output size; security parameter κ.
- Public efficient classical functions $f_A : \{0,1\}^{n_A} \times \{0,1\}^{n_B} \to \{0,1\}^{m_A}$, $f_B : \{0,1\}^{n_A} \times \{0,1\}^{n_B} \to \{0,1\}^{m_B}$.
- Alice-side input $x_A \in \{0,1\}^{n_A}$, Bob-side input $x_B \in \{0,1\}^{n_B}$.

The outputs of the protocol are stored in the following registers: Alice-side $\boldsymbol{flag}^{(A)}$ holding values in $\{\mathsf{pass}, \mathsf{fail}\}$, $\boldsymbol{y}^{(A)}$ holding values in $\{0,1\}^{m_A}$, Bob-side $\boldsymbol{flag}^{(B)}$ holding values in $\{\mathsf{pass}, \mathsf{fail}\}$, $\boldsymbol{y}^{(B)}$ holding values in $\{0,1\}^{m_B}$.

Either Alice or Bob could be malicious. For modeling the initial states in the malicious setting, when Alice is malicious, suppose the Alice-side registers are denoted by $\boldsymbol{S}^{(A)}$; when Bob is malicious, suppose the Bob-side registers are denoted by $\boldsymbol{S}^{(B)}$. The environment is denoted by $\boldsymbol{E}$.

The completeness and soundness are in Definition 3.7, 3.8 below. The efficiency is defined in the common way.

Definition 3.7. We say a protocol under Set-up 1 is complete if when both parties are honest, the output state of the protocol is negligibly close to the following state: $\boldsymbol{flag}^{(A)}, \boldsymbol{flag}^{(B)}$ holds the value pass, $\boldsymbol{y}^{(A)}$ holds $f_A(x^{(A)}, x^{(B)})$, $\boldsymbol{y}^{(B)}$ holds $f_B(x^{(A)}, x^{(B)})$.

Definition 3.8. To define the soundness under Set-up 1, we first define $\text{2PCIdeal}^{(A)}, \text{2PCIdeal}^{(B)}$ as follows.

$\text{2PCIdeal}^{(B)}$ works as follows. It interacts with the honest Alice and malicious Bob. Note that the Alice's private inputs are prepared in advanced but Bob's private inputs could be adversarily generated.

1. $\text{2PCIdeal}^{(B)}$ receives a bit $b \in \{0,1\}$ from Bob, then:
   - (Early abort option) If $b = 1$, $\text{2PCIdeal}^{(B)}$ sets $\boldsymbol{flag}^{(A)}$ to be fail.

- If $b = 0$, 2PCIdeal$^{(B)}$ takes $x^{(A)}$ from Alice's inputs and take $x^{(B)}$ from Bob. Then 2PCIdeal$^{(B)}$ stores $f_B(x^{(A)}, x^{(B)})$ in $\boldsymbol{y}^{(B)}$. Then:
  (a) (Early abort option) 2PCIdeal$^{(B)}$ receives a bit $b' \in \{0,1\}$ from Bob, then:
    – If $b = 1$, 2PCIdeal$^{(B)}$ sets $\boldsymbol{flag}^{(A)}$ to be fail.
    – If $b = 0$, 2PCIdeal$^{(B)}$ sets $\boldsymbol{flag}^{(A)}$ to be pass, and stores $f_A(x^{(A)}, x^{(B)})$ in $\boldsymbol{y}^{(A)}$.

2PCIdeal$^{(B)}$ is defined similarly by swapping the role of Alice and Bob in the definition above.

We say a protocol π under Set-up 1 is sound if:

- (Soundness against malicious Bob) For any efficient quantum adversary Adv playing the role of Bob, there exist efficient quantum operations Sim such that for any polynomial-size inputs $1^{n_A}, 1^{n_B}, 1^{m_A}, 1^{m_B}, f_A, f_B, x^{(A)}$ and any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}^{(B)}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

$$\pi^{\mathsf{Adv}}(\rho_0) \approx^{\mathrm{ind}} (\underbrace{\mathsf{Sim}}_{\text{working on the server side and interacting with 2PCIdeal}^{(B)}} \circ \text{2PCIdeal}^{(B)})(\rho_0) \quad (16)$$

  We note that the distinguisher in (16) has access to all the inputs above, client-side states, server-side states and the environment.

- (Security against malicious Alice) Similar to the definition above by swapping Alice and Bob in the definition.

Convention 4. We make the following convention for working on protocol design and analysis.

First, in this paper, we use "Set-up" to organize the set-up of protocols and notions, as in this section. Then when we design protocols, simply refering to a set-up gives lots of information including the inputs, output registers and security definitions. For readability we still review the inputs when we describe protocols.

For simplicity of expressions, we might omit the inputs that are public and given by values. Note that in (16) above we already make the inputs $1^{n_A}, f_A$, etc implicit for simplicity.

We need the following theorem in our work, which is proved in [7, 28].

Theorem 3.7. *Assuming the existence of one-way functions, there exists a quantum protocol for general secure two-party computations.*

3.3.5 Succinct arguments, succinct-AoK and succinct-ZK-AoK

In this section we review the notion of succinct arguments, succinct arguments of knowledge (succinct-AoK) and succinct zero-knowledge arguments of knowledge (succinct-ZK-AoK) for NP. [34, 20]

Set-up 2. The set-up for succinct arguments / succinct-AoK / succinct-ZK-AoK for NP is as follows. We consider a classical protocol between a client and a server.

The protocol takes the following public parameters and inputs: problem sizes $1^n, 1^m$, security parameter 1^κ, an efficient classical relation $R : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, an input $x \in \{0,1\}^n$.

The server holds a classical register $\boldsymbol{S}_w$ whose value is in $\{0,1\}^m$.

The client-side output register is $\boldsymbol{flag}$ whose value is in $\{\mathsf{pass}, \mathsf{fail}\}$.

For succinct arguments and succinct-AoK the server is the malicious party. For succinct-ZK-AoK both parties could be malicious. For modeling the initial states in the malicious server setting,

suppose the overall server-side registers are denoted by $\boldsymbol{S}$. For modeling the initial states in the malicious client setting, the client-side registers are denoted by $\boldsymbol{C}$ and the environment is denoted by $\boldsymbol{E}$.

The completeness is in Definition 3.9 below. The soundness and zero-knowledge property is as follows.

- If we consider the succinct arguments primitive, the soundness is in Definition 3.10;
- if we consider the succinct-AoK, the soundness is in Definition 3.11 below; if we consider the succinct-ZK-AoK, the soundness is in Definition 3.11 below and the zero-knowledge property is in Definition 3.12 below.

The efficiency is defined in the common way. We additionally want the property that the size of the client-side computation is $\mathsf{poly}(n, \kappa)$.

Definition 3.9. We say a protocol under Set-up 2 is complete if when the server is honest and initially $\boldsymbol{S}_w$ holds a value w that satisfies $R(x, w) = 1$, the client accepts with probability $\geq 1 - \mathsf{negl}(\kappa)$.

Definition 3.10. We say a protocol under Set-up 2 (for succinct arguments primitive) is sound if the following holds. For any efficient quantum adversary Adv as the server, any polynomial-size inputs $1^n, 1^m, R, x$, any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}})$, if there is no w that satisfies $R(x, w) = 1$, the client rejects with probability $\geq 1 - \mathsf{negl}(\kappa)$.

Definition 3.11. We say a protocol under Set-up 2 (for succinct-AoK primitive) is (δ, ϵ)-sound if the following holds. For any efficient quantum adversary Adv as the server there exists an efficient quantum operation Ext such that for any polynomial-size inputs $1^n, 1^m, R, x$, for any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}})$, if the client accepts with probability $\geq 1 - \delta$, there is

$$\mathrm{tr}(\Pi^{\boldsymbol{S}_w}_{w:R(x,w)=1} \mathsf{Ext}(\rho_0)) \geq 1 - \epsilon - \mathsf{negl}(\kappa)$$

Definition 3.12. We say a protocol π under Set-up 2 (for succinct-ZK-AoK primitive) is zero-knowledge if the following holds. For any efficient quantum adversary Adv as the client there exists an efficient quantum operation Sim such that for any polynomial-size inputs $1^n, 1^m, R, x$, for any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{C}} \otimes \mathcal{H}_{\boldsymbol{E}})$ and any w in register $\boldsymbol{S}_w$ such that $R(x, w) = 1$:

$$\pi^{\mathsf{Adv}}(\rho_0) \approx^{\mathrm{ind}} \underbrace{\mathsf{Sim}}_{\text{client side}} (\rho_0) \otimes \underbrace{|w\rangle\langle w|}_{\boldsymbol{S}_w} \tag{17}$$

Note that the distinguisher in (17) also has access to all the inputs, which are omitted in the expression[9].

By Kilian's work [34] and the proof of its post-quantum security [20], there is:

Theorem 3.8. *Assuming the existence of collapsing hash functions, there exists a succinct-ZK-AoK protocol for NP that is $(\delta, O(\delta))$-sound for all $\delta > 0$.*

We note that the δ, ϵ parameters in our set-up are different from the ϵ parameter in [20]: we define the parameters to be the failing errors, while [20] defines ϵ to be the success probability.

[9]Note that since the statement is quantified with "for all inputs", whether or not we explicitly give these inputs to the distinguisher actually do not matter: the distinguisher could always hardcode them. In the later content we will not care about this detail.

3.3.6 Yao's garbled circuits

Yao's garbled circuits is a basic and famous construction in secure computations. [51] It achieves a notion called decomposable randomized encoding. [4] In this section we review these notions.

Definition 3.13. A decomposable randomized encoding scheme is defined to be efficient classical algorithms $(\mathsf{GarbleC}, \mathsf{GarbleI}, \mathsf{Dec})$ where:

- $\mathsf{GarbleI}$ is a deterministic algorithm that takes $1^n, 1^\kappa$ as parameters, takes $x \in \{0,1\}^n, r \in \{0,1\}^{n\times\kappa}$ and calculates the encoding of x using r. This part is called the input encoding and r is the shared randomness used in both $\mathsf{GarbleI}$ and $\mathsf{GarbleC}$.
- Use $\mathcal{C}_n$ to denote the set of circuits with input length n. $\mathsf{GarbleC}$ takes $1^n, 1^\kappa$ as parameters, takes $C \in \mathcal{C}_n, r \in \{0,1\}^{n\times\kappa}$ and calculates the encoding of C using r. This part is called the circuit encoding. Note that $\mathsf{GarbleC}$ itself could be a randomized algorithm; its deterministic function description that makes its internal random coins explicit could be written as $\mathsf{GarbleC}(C, r; r_{\mathrm{GCin}})$ (see Fact 2).
- Dec is a deterministic algorithm that takes the outputs of $\mathsf{GarbleC}(C, r)$ and $\mathsf{GarbleI}(x, r)$ as inputs and outputs $C(x)$.

Here "decomposable" means that the encoding of $C(x)$ could be decomposed into the input encoding and the circuit encoding separately. What $\mathsf{GarbleC}$ and $\mathsf{GarbleI}$ share is only the shared random coins r.

And they satisfy the following properties:

- (Completeness) For any polynomial-size 1^n, any x, C:
$$\Pr_{r\leftarrow\{0,1\}^{n\kappa}}[\mathsf{Dec}(\mathsf{GarbleC}(C, r), \mathsf{GarbleI}(x, r)) = C(x)] \geq 1 - \mathsf{negl}(\kappa)$$
- (Soundness) There exists an efficient classical algorithm Sim such that for any polynomial-size 1^n, any x, C:
$$\mathsf{Sim}(C(x)) \approx^{\mathrm{ind}} \mathrm{Dist}\begin{bmatrix} r \leftarrow_r \{0,1\}^{n\kappa} \\ (\mathsf{GarbleC}(C, r), \mathsf{GarbleI}(x, r)) \end{bmatrix} \tag{18}$$

By Yao's famous construction [51] there is:

Theorem 3.9. *Assuming the existence of one-way functions, Yao's Garbled Circuits a decomposable randomized encoding scheme.*

As discussed in Section 2.7.3, in this work we need a relatively stronger form of soundness for Yao's scheme, which is formalized below.

Definition 3.14. We say a garbling scheme under Definition 3.13 is sound with input-first simulation if there exist efficient classical algorithms $\mathsf{Sim}_{\mathrm{ie}}, \mathsf{Sim}_{\mathrm{ce}}$ such that for any polynomial-size 1^n, any x, C:

$$\mathrm{Dist}\begin{bmatrix} \tilde{x} \leftarrow \mathsf{Sim}_{\mathrm{ie}}() \\ (\tilde{x}, \mathsf{Sim}_{\mathrm{ce}}(\tilde{x}, C(x))) \end{bmatrix} \approx^{\mathrm{ind}} \mathrm{Dist}\begin{bmatrix} r \leftarrow_r \{0,1\}^{n\kappa} \\ \hat{x} := \mathsf{GarbleI}(x, r) \\ (\hat{x}, \mathsf{GarbleC}(C, r)) \end{bmatrix} \tag{19}$$

Note that we use $:=$ to define new variables in the expressions.

Theorem 3.10. *Assuming the existence of one-way functions, Yao's Garbled Circuits a decomposable randomized encoding scheme with soundness under Definition 3.14.*

3.3.7 Trapdoor claw-free functions

(Noisy) trapdoor claw-free functions (NTCF/TCF) is an important technique in quantum cryptography. It is raised in [11, 37] for the test-of-quantumness and CVQC problem. There are some variants of definitions raised for different problems, for example, NTCF/TCF with the adaptive hardcore bit property [11, 26], or extended NTCF [37]. Below we review the definition of NTCF without the adaptive hardcore bit property [13, 57].

Definition 3.15 (NTCF, without the adaptive hardcore bit property)**.** The NTCF is defined to be a class of polynomial time algorithms as below, which take security parameter 1^κ as the parameter. KeyGen is a sampling algorithm. Dec, CHK are deterministic algorithms. Eval is allowed to be a sampling algorithm. poly' is a polynomial that determines the the range size.

$$\mathsf{KeyGen}(1^\kappa) \to (\mathrm{sk}, \mathrm{pk}),$$

$$\mathsf{Eval}_{\mathrm{pk}} : \{0,1\} \times \{0,1\}^\kappa \to \{0,1\}^{\mathsf{poly}'(\kappa)},$$

$$\mathsf{Dec}_{\mathrm{sk}} : \{0,1\} \times \{0,1\}^{\mathsf{poly}'(\kappa)} \to \{0,1\}^\kappa \cup \{\perp\},$$

$$\mathsf{CHK}_{\mathrm{pk}} : \{0,1\} \times \{0,1\}^\kappa \times \{0,1\}^{\mathsf{poly}'(\kappa)} \to \{\mathsf{true}, \mathsf{false}\}$$

And they satisfy the following properties:

- (Correctness)
  - (Noisy 2-to-1) For all possible $(\mathrm{sk}, \mathrm{pk})$ in the range of $\mathsf{KeyGen}(1^\kappa)$ there exists a subnormalized probability distribution $(p_y)_{y \in \{0,1\}^{\mathsf{poly}'(\kappa)}}$ that satisfies: for any y such that $p_y \neq 0$, $\forall b \in \{0,1\}$, there is $\mathsf{Dec}_{\mathrm{sk}}(b, y) \neq \perp$, and

$$\mathsf{Eval}_{\mathrm{pk}}(|+\rangle^{\otimes(1+\kappa)}) \approx_{\mathsf{negl}(\kappa)} \sum_{y: p_y \neq 0} \frac{1}{\sqrt{2}}(|\mathsf{Dec}_{\mathrm{sk}}(0, y)\rangle + |\mathsf{Dec}_{\mathrm{sk}}(1, y)\rangle) \otimes \sqrt{p_y}\,|y\rangle \tag{20}$$

  - (Correctness of CHK) For all possible $(\mathrm{sk}, \mathrm{pk})$ in the range of $\mathsf{KeyGen}(1^\kappa)$, $\forall x \in \{0,1\}^\kappa, \forall b \in \{0,1\}$:

$$\mathsf{CHK}_{\mathrm{pk}}(b, x, y) = \mathsf{true} \Leftrightarrow \mathsf{Dec}_{\mathrm{sk}}(b, y) = x$$

- (Claw-free) For any BQP adversary Adv,

$$\Pr\begin{bmatrix}(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa), \\ \mathsf{Adv}(\mathrm{pk}) \to (x_0, x_1, y): \quad x_0 \neq \perp, x_1 \neq \perp, x_0 \neq x_1 \\ \mathsf{Dec}_{\mathrm{sk}}(0, y) = x_0, \mathsf{Dec}_{\mathrm{sk}}(1, y) = x_1\end{bmatrix} \leq \mathsf{negl}(\kappa) \tag{21}$$

By [11, 13] we have the following:

Theorem 3.11. *NTCF could be instantiated from the LWE assumption.*

3.3.8 Remote state preparation with verifiability

Remote state preparation with verifiability (RSPV) [57] is an important primitive in quantum cryptography. Famous works on RSPV include [26], where a classical channel RSPV protocol for $|+_\theta\rangle$ states is constructed. [57] gives a framework for working on RSPV and gives more protocols. Below we review the notions and results that are related to our work.

We first review a common set-up for RSPV. Below we repeat the definitions used in [57].

Set-up 3. A typical set-up for RSPV is as follows. We consider a protocol between a client and a server. We consider a tuple of states $(|\varphi_1\rangle, |\varphi_2\rangle, \cdots |\varphi_D\rangle)$.

The protocol takes the following parameters: security parameter 1^κ, approximation error parameter $1^{1/\epsilon}$.

The outputs of the protocol are stored in the following registers: the client-side $\boldsymbol{flag}$ holding value in $\{\mathsf{pass}, \mathsf{fail}\}$, the client-side classical register $\boldsymbol{D}$ holding value in $[D]$, the server-side quantum register $\boldsymbol{Q}$ whose dimension is suitable for holding the states.

The server is the malicious party. For modeling the initial states in the malicious setting, suppose the server-side registers are denoted by $\boldsymbol{S}$ and the environment is denoted by $\boldsymbol{E}$.

The completeness and soundness are in Definition 3.16, 3.17 below. The efficiency is defined in the common way.

Definition 3.16 (Completeness)**.** We say a protocol under Set-up 3 is complete if when the server is honest the output state of the protocol is negligibly close to the following state: $\boldsymbol{flag}$ holds the value pass, $\boldsymbol{D}$ holds a random $i \in [D]$, $\boldsymbol{Q}$ holds the corresponding $|\varphi_i\rangle$.

Below we use ρ_{tar} to denote this output state on $\boldsymbol{D}, \boldsymbol{Q}$ in the honest setting.

Definition 3.17 (Soundness)**.** To define the soundness under Set-up 3, we first define RSPVIdeal as follows.

RSPVIdeal receives a bit b from the server, then:

- If $b = 0$, it sets $\boldsymbol{flag}$ to be pass, samples $i \in [D]$ and stores it in $\boldsymbol{D}$, and stores $|\varphi_i\rangle$ in $\boldsymbol{Q}$.
- If $b = 1$, it sets $\boldsymbol{flag}$ to be fail.

We say a protocol π under Set-up 3 is approximately sound if:

For any efficient quantum adversary Adv, there exist efficient quantum operations $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

$$\pi^{\mathsf{Adv}}(\rho_0) \approx_\epsilon^{\mathrm{ind}} \underbrace{\mathsf{Sim}_1}_{\text{on } \boldsymbol{S},\boldsymbol{Q}} (\mathrm{RSPVIdeal}(\underbrace{\mathsf{Sim}_0}_{\text{on } \boldsymbol{S} \text{ and outputs } b} (\rho_0)))$$

We note that (1) there are some variants of soundness definitions for RSPV. The definition we use here corresponds to Definition 3.3 in [57], which follows the standard form of the simulation-based paradigm. (2) Since the error tolerance parameter is already part of the set-up so we could simply say "approximately sound" in the definition without ambiguity; we could also say it's ϵ-sound for convenience.

In the framework of [57], there is a useful notion called preRSPV. Below we review this notion.

Set-up 4. A typical set-up of preRSPV is as follows. The basic set-up is similar to Set-up 3 with the following differences: we are considering a pair of protocols $(\pi_{\text{test}}, \pi_{\text{comp}})$ instead of a single protocol; and we do not model $1^{1/\epsilon}$ as part of the protocol inputs[10].

The completeness and soundness are in Definition 3.18, 3.19 below.

Definition 3.18. We say $(\pi_{\text{test}}, \pi_{\text{comp}})$ under Set-up 4 is complete if:

- In π_{test}, when the server is honest, the passing probability is negligibly close to 1.
- In π_{comp}, when the server is honest, the output state of the protocol is as required in Definition 3.16.

Definition 3.19. We say $(\pi_{\text{test}}, \pi_{\text{comp}})$ under Set-up 4 is (δ, ϵ)-sound if:

For any efficient quantum adversary Adv, there exists an efficient quantum operation Sim such that for any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

If

$$\mathrm{tr}(\Pi_{\mathsf{pass}}(\pi_{\text{test}}^{\mathsf{Adv}}(\rho_0))) \geq 1 - \delta$$

then

$$\Pi_{\mathsf{pass}}(\pi_{\text{comp}}^{\mathsf{Adv}}(\rho_0)) \approx_{\epsilon}^{\text{ind}} \Pi_{\mathsf{pass}}(\underbrace{\mathsf{Sim}}_{\boldsymbol{S},\boldsymbol{Q},\boldsymbol{flag}} (\underbrace{\rho_{tar}}_{\boldsymbol{D},\boldsymbol{Q}} \otimes \rho_0))$$

Note that here we only consider indistinguishability on the passing space, which is chosen for some convenience reason in [57]. As shown in [57], there is a protocol template that amplifies a preRSPV to an RSPV by a suitable sequential repetition:

Below we assume $(\pi_{\text{test}}, \pi_{\text{comp}})$ is a preRSPV under Set-up 4 that is (δ, ϵ_0)-sound.

Protocol 3. A temporary protocol π_{temp} is as follows. The protocol inputs are: public parameters $1^{1/\epsilon}, 1^{\kappa}$. It is required that $\epsilon > \epsilon_0$.

1. Define $L = \frac{512}{\delta(\epsilon-\epsilon_0)^3}, p = \frac{\epsilon-\epsilon_0}{8}$.

   For each $i \in [L]$:

   (a) The client randomly chooses $\text{mode}^{(i)} = \text{test}$ with probability p or $\text{mode}^{(i)} = \text{comp}$ with probability $1 - p$.

   (b) The client executes $\pi_{\text{mode}^{(i)}}$ with the server. Store the outputs in $\boldsymbol{flag}^{(i)}, \boldsymbol{D}^{(i)}, \boldsymbol{Q}^{(i)}$.

2. The client sets $\boldsymbol{flag}$ to be fail if any of $\boldsymbol{flag}^{(i)}$ is fail. Otherwise it sets $\boldsymbol{flag}$ to be pass; and it randomly chooses $i \in [L]$ such that $\text{mode}^{(i)} = \text{comp}$ and sends i to the server; both parties use the states in $\boldsymbol{D}^{(i)}, \boldsymbol{Q}^{(i)}$ as the outputs.

Protocol 4. An RSPV protocol under Set-up 3 is as follows. The protocol inputs are: public parameters $1^{1/\epsilon}, 1^{\kappa}$. It is required that $\epsilon > \epsilon_0$.

1. Define $L = \frac{1728}{(\epsilon-\epsilon_0)^3}$.

   For each $i \in [L]$:

[10] Instead, ϵ directly appears in the soundness definition.

(a) The client and the server execute π_{temp} with approximation error tolerance $\frac{\epsilon+\epsilon_0}{2}$. Store the outputs in register $\boldsymbol{flag}^{(i)}, \boldsymbol{D}^{(i)}, \boldsymbol{Q}^{(i)}$.

2. The client sets $\boldsymbol{flag}$ to be fail if any of $\boldsymbol{flag}^{(i)}$ is fail. Otherwise it sets $\boldsymbol{flag}$ to be pass; and it randomly chooses $i \in [L]$ and sends i to the server; both parties use the states in $\boldsymbol{D}^{(i)}, \boldsymbol{Q}^{(i)}$ as the outputs.

The completeness and efficiency are from the protocol description.

Theorem 3.12. *Protocol 4 is ϵ-sound.*

Proof. The amplification is the amplification procedure used in [57]. Note that the amplification from preRSPV to Protocol 3 is what [57] calls "preRSPV to RSPV amplification". The amplification to Protocol 4 is to amplify from the passing-space simulation to all-space simulation, which is proved in Section 3.1.1 of [57]. □

Finally we review the following theorems about the concrete constructions of RSPV protocols, which are also needed in our work.

Theorem 3.13 (By [57])**.** *Assuming the existence of NTCF, there exists a classical-channel RSPV protocol for state family* $\{\frac{1}{\sqrt{2}}(|0\rangle |x_0\rangle + |1\rangle |x_1\rangle) : x_0, x_1 \in \{0,1\}^n\}$.

Theorem 3.14 (By [12])**.** *Assuming the existence of NTCF, there exists a classical-channel RSPV protocol for BB84 states.*

Note that the NTCF above does not assume the adaptive hardcore bit property, as formalized in Definition 3.15.

4 Secure Function Sampling with Long Outputs

In this section we formalize the notion of secure function sampling (SFS), prove its classical impossibility, and prepare for the construction of its quantum protocols.

4.1 Definition of Secure Function Sampling

In this section we formalize the notion of SFS. Below we first formalize its set-up.

Set-up 5. The set-up for SFS is as follows. We consider a protocol between a client and a server.

The inputs of the protocol are all public, which are: problem sizes $1^n, 1^m$, security parameter 1^κ, error tolerance parameter $1^{1/\epsilon}$, an efficient classical function $f : \{0,1\}^n \to \{0,1\}^m$.

The outputs of the protocol are as follows. The client-side output registers are $\boldsymbol{flag}$, which holds a value in $\{\text{pass}, \text{fail}\}$, and $\boldsymbol{x}^{(out)}$, which holds a string in $\{0,1\}^n$; the server-side output register is $\boldsymbol{Q}^{(out)}$, which holds a string in $\{0,1\}^m$.

The server is the malicious party. For modeling the initial states in the malicious setting, suppose the server-side registers are denoted by $\boldsymbol{S}$ and the environment is denoted by $\boldsymbol{E}$.

The completeness and soundness are in Definition 4.1, 4.2 below. The efficiency is defined in the common way.

As discussed in the introduction, in this paper we would like to construct an SFS protocol in succinct communication.

The completeness of SFS is as follows.

Definition 4.1 (Completeness of SFS)**.** We say a protocol under Set-up 5 is complete if when the server is honest, the output state of the protocol is negligibly close to the following state: the register $\boldsymbol{flag}$ holds the value pass, the register $\boldsymbol{x}^{(out)}$ holds a random classical string $x \leftarrow_r \{0,1\}^n$, the register $\boldsymbol{Q}^{(out)}$ holds the corresponding classical string $f(x)$.

We use ρ_{tar} to denote the output state on $\boldsymbol{x}^{(out)}, \boldsymbol{Q}^{(out)}$ in the honest setting.

The soundness of SFS is as follows.

Definition 4.2 (Soundness of SFS)**.** To define the soundness, we first define SFSIdeal as follows.

SFSIdeal receives a bit $b \in \{0,1\}$ from the server, then:

- If $b = 0$, SFSIdeal sets $\boldsymbol{flag}$ to be pass, and it randomly samples $x \leftarrow_r \{0,1\}^n$, stores x in $\boldsymbol{x}^{(out)}$ and stores $f(x)$ in $\boldsymbol{Q}^{(out)}$.
- If $b = 1$, SFSIdeal sets $\boldsymbol{flag}$ to be fail.

Then we define the soundness as follows. We say a protocol π under Set-up 5 is ϵ-sound if:

For any efficient quantum adversary Adv, there exist efficient quantum operations $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any polynomial-size inputs $1^n, 1^m, 1^{1/\epsilon}, f$ and any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

$$\pi^{\mathsf{Adv}}(\rho_0) \approx^{\mathrm{ind}}_{\epsilon} \underbrace{\mathsf{Sim}_1}_{\text{on } \boldsymbol{S},\boldsymbol{Q}^{(out)}} (\mathrm{SFSIdeal}(\underbrace{\mathsf{Sim}_0}_{\text{on } \boldsymbol{S} \text{ and outputs } b} (\rho_0))) \tag{22}$$

4.2 Classical Impossibility

In this section we prove the SFS primitive is not achievable by a classical protocol. As described in Section 2.6, the impossibility result considers a pseudorandom generator (PRG) as the function f. Below we state the theorem.

Theorem 4.1. *Suppose a classical protocol π in Set-up 5 is complete, efficient and has communication complexity $\mathsf{poly}(n,\kappa)$. Then the following statement could* not *be true:*

For any efficient quantum adversary Adv, there exist efficient quantum operations $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any polynomial-size $1^n, 1^m$, take $\epsilon < 1 - O(1)$ and $f = PRG(1^n, 1^m)$, for any initial state $\rho_0 \in D(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

$$\pi^{\mathsf{Adv}}(f)(\rho_0) \approx^{ind}_{\epsilon} \underbrace{\mathsf{Sim}_1}_{on\ \boldsymbol{S},\boldsymbol{Q}^{(out)}} (SFSIdeal(f)(\underbrace{\mathsf{Sim}_0}_{on\ \boldsymbol{S}\ and\ outputs\ b} (\rho_0))) \tag{23}$$

The following corollary reduces the existence of PRG to one-way functions and encapsulates the theorem using Set-up 5:

Corollary 4.2. *Assuming the existence of post-quantum one-way functions, there does not exist a classical SFS protocol as described in Set-up 5 with communication complexity $\mathsf{poly}(n,\kappa,1/\epsilon)$.*

The overview of the proof is given in Section 2.6. The formal proof is as follows.

Proof of Theorem 4.1. Suppose there exists a classical protocol π that is complete, efficient, has communication complexity $\mathsf{poly}(n,\kappa)$ and satisfies (23). We prove that's impossible.

Consider an adversary Adv as follows: Adv follows the honest behavior of the protocol, but keeps the transcript of each step. In the end it outputs both the honest setting outputs and the transcripts. Suppose the output register holding the transcripts part is $\boldsymbol{t}$, which holds a classical string in $\{0,1\}^{\mathsf{poly}(n,\kappa)}$. By the completeness, for any n, m, ρ_0, the output states of $\pi^{\mathsf{Adv}}(f)(\rho_0)$ should have the following form: the client holds a random $x \leftarrow_r \{0,1\}^n$, the server holds the corresponding $\mathrm{PRG}(x) \in \{0,1\}^m$ in register $\boldsymbol{Q}^{(out)}$, and $\boldsymbol{t}$ holds the transcripts; $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$ is not affected. Finally we use $r \in \{0,1\}^L$ to denote the random coins used by Adv and use $\mathsf{Adv}(r)$ to denote the adversary running with random coin r (recall that π is a classical protocol so this could be done); then the discussion above holds with probability $1 - \mathsf{negl}(\kappa)$ for adversary $\mathsf{Adv}(r)$ with a random $r \leftarrow_r \{0,1\}^L$.

Below we will construct $\mathsf{Compressor} : \{0,1\}^m \times \{0,1\}^L \to \{0,1\}^{\mathsf{poly}(n,\kappa)}$ and $\mathsf{Decompressor} : \{0,1\}^{\mathsf{poly}(n,\kappa)} \times \{0,1\}^L \to \{0,1\}^m$. Intuitively they could be understood as a compressor that compresses the PRG outputs to the transcripts in $\boldsymbol{t}$ and a decompressor that decompresses $\boldsymbol{t}$ to PRG outputs.

- By (23) against adversary $\mathsf{Adv}(r), r \in \{0,1\}^L$, there exists efficient quantum operations $(\mathsf{Sim}_0(r), \mathsf{Sim}_1(r))$ such that (23) holds.

  Define $\mathsf{Compressor} : \{0,1\}^m \times \{0,1\}^L \to \{0,1\}^{\mathsf{poly}(n,\kappa)}$ as follows. We use $y \in \{0,1\}^m$ to denote its first input and use $r \in \{0,1\}^L$ to denote its second input. $\mathsf{Compressor}$ first prepares the state $\mathsf{Sim}_0(r)(\rho_0)$, and measures to collapse the register holding b; then it stores y in register $\boldsymbol{Q}^{(out)}$; then it applies $\mathsf{Sim}_1(r)$; in the end it keeps the states in $\boldsymbol{t}$ and disgards all the other parts. By (23) we have

$$\mathrm{Dist}\begin{bmatrix} x \leftarrow_r \{0,1\}^n, r \leftarrow_r \{0,1\}^L \\ (x, \mathsf{Compressor}(\mathrm{PRG}(x), r)) \end{bmatrix} \approx_{\epsilon}^{\mathrm{ind}} \mathrm{Dist}\begin{bmatrix} \text{exec } \pi^{\mathsf{Adv}}(f)(\rho_0) \\ \text{outputs } \boldsymbol{x}^{(out)}, \boldsymbol{t} \end{bmatrix} \tag{24}$$

- Note that π is a classical protocol so there is an efficient algorithm that recovers the outputs given the adversary's code together with its random coins, the initial state, and the messages from the client to the server.

  Define $\mathsf{Decompressor} : \{0,1\}^{\mathsf{poly}(n,\kappa)} \times \{0,1\}^L \to \{0,1\}^m$ as follows. $\mathsf{Decompressor}$ interprets its first input as the messages from the client to the server and interprets its second input as the server's random coins, simulates the protocol execution and outputs the outputs of π on $\boldsymbol{Q}^{(out)}$. Note that the initial state ρ_0 is not used in the operation of Adv and the code of Adv could be hard-coded. Then we have:

$$\mathrm{Dist}\begin{bmatrix} r \leftarrow_r \{0,1\}^L \\ \text{exec } \pi^{\mathsf{Adv}(r)}(f)(\rho_0) \\ (x,t) := \text{values in } \boldsymbol{x}^{(out)}, \boldsymbol{t} \\ (x, \mathsf{Decompressor}(t, r)) \end{bmatrix} \approx_{\mathsf{negl}(\kappa)} \mathrm{Dist}\begin{bmatrix} x \leftarrow_r \{0,1\}^n \\ (x, \mathrm{PRG}(x)) \end{bmatrix} \tag{25}$$

Combining (24)(25) we have

$$\mathrm{Dist}\begin{bmatrix} x \leftarrow_r \{0,1\}^n, r \leftarrow_r \{0,1\}^L \\ (x, \mathsf{Decompressor}(\mathsf{Compressor}(\mathrm{PRG}(x), r), r)) \end{bmatrix} \approx_{\epsilon}^{\mathrm{ind}} \mathrm{Dist}\begin{bmatrix} x \leftarrow_r \{0,1\}^n \\ (x, \mathrm{PRG}(x)) \end{bmatrix}$$

Note that PRG is a determinisitic function. This directly implies that

$$\Pr\left[\begin{matrix} x \leftarrow_r \{0,1\}^n, r \leftarrow_r \{0,1\}^L \\ \mathsf{Decompressor}(\mathsf{Compressor}(\mathrm{PRG}(x), r), r) = \mathrm{PRG}(x) \end{matrix}\right] \geq 1 - \epsilon - \mathsf{negl}(\kappa)$$

Apply the property of PRG we get

$$\Pr\left[\begin{matrix} u \leftarrow_r \{0,1\}^m, r \leftarrow_r \{0,1\}^L \\ \mathsf{Decompressor}(\mathsf{Compressor}(u, r), r) = u \end{matrix}\right] \geq 1 - \epsilon - \mathsf{negl}(\kappa)$$

which is impossible information-theoretically when $m > \kappa + \mathsf{poly}(n, \kappa)$ and $1 - \epsilon \geq O(1)$. This completes the proof. □

Note We note that Definition 4.2 is stated in a setting where the adversary could be quantum; thus Theorem 4.1 is about the impossibility of the post-quantum security (classical protocol against quantum adversaries). We could also state the impossibility result in a setting where the adversary is assumed to be classical; our impossibility proof also works in this setting.

4.3 Secure Function Sampling via RSPV

We will construct the quantum protocol for SFS in Section 5, 6. This section is a preparation for the construction.

As discussed in Section 2.2, we will view SFS as an RSPV problem, and make use of an intermediate notion called preRSPV. Below we formalize the set-up of the corresponding preRSPV.

Set-up 6. The basic set-up is similar to Set-up 5 with the following differences: we are considering a pair of protocols $(\mathsf{SFSTest}, \mathsf{SFSComp})$ instead of a single protocol; and we do not model $1^{1/\epsilon}$ as part of the protocol inputs.

The completeness and soundness are in Definition 4.3, 4.4 below.

Definition 4.3. We say $(\mathsf{SFSTest}, \mathsf{SFSComp})$ under Set-up 6 is complete if:

- In $\mathsf{SFSTest}$, when the server is honest, the passing probability is negligibly close to 1.
- In $\mathsf{SFSComp}$, when the server is honest, the otuput state of the protocol is as required in Definition 4.1.

Definition 4.4. We say $(\mathsf{SFSTest}, \mathsf{SFSComp})$ under Set-up 6 is (δ, ϵ)-sound if:

For any efficient quantum adversary Adv, there exists an efficient quantum operation Sim such that for any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

If

$$\mathrm{tr}(\Pi_{\mathsf{pass}}(\mathsf{SFSTest}^{\mathsf{Adv}}(\rho_0))) > 1 - \delta$$

then

$$\Pi_{\mathsf{pass}}(\mathsf{SFSComp}^{\mathsf{Adv}}(\rho_0)) \approx_{\epsilon}^{\mathrm{ind}} \Pi_{\mathsf{pass}}(\underbrace{\mathsf{Sim}}_{\boldsymbol{S},\boldsymbol{Q}^{(out)},\boldsymbol{flag}} (\underbrace{\rho_{tar}}_{\boldsymbol{x}^{(out)},\boldsymbol{Q}^{(out)}} \otimes \rho_0))$$

where ρ_{tar} is as defined in Definition 4.1.

As discussed in Section 3.3.8, the preRSPV could be amplified to an RSPV, which is the SFS that we formalized in Set-up 5. (Note that although the set-ups are slightly different in the sense that Set-up 4 considers a finite set of states while Set-up 6 works on a function f, the amplification still works.)

Then we briefly discuss the organization of Section 5 and 6, where we construct our SFS protocol. We will first formalize the succinct testing for two-party relations in Section 5, and then formalize the full SFS protocol in Section 6. In Section 6 we first give the preRSPV $(\mathsf{SFSTest}, \mathsf{SFSComp})$ in Section 6.1.1, and then amplify it to an SFS in Section 6.1.2. The intuitions of the protocols are described in Section 2. The overview of the security proof of $(\mathsf{SFSTest}, \mathsf{SFSComp})$ is given in Section 6.2.1, which makes use of the notions in [54]. The security proof is completed in Section 6.2.2 and 6.2.3.

Finally we discuss some conventions that might be implicit in later constructions and proofs. First we could review Convention 1, 2, 3, 4. We additionally note that in the protocol description we usually omit the explicit statement in the form of "set $\boldsymbol{flag}$ to be fail if any step fails" for simplicity: for example, if we are constructing Protocol A and this Protocol A calls Protocol B, we omit "set $\boldsymbol{flag}$ to be fail if Protocol B fails".

5 Protocol for Reducing Server-to-Client Communications

In this section we formalize the primitive that we call succinct testing for two-party relations, and construct the protocol. As said before, this protocol is a variant of the compiler in [8]. In Section 5.1 we formalize the problem set-up; in Section 5.2 we give the protocol construction.

5.1 Problem Set-up

Notation 5.1. Recall that a relation between S_1 and S_2 is a function $S_1 \times S_2 \to \{0,1\}$. The efficiency is defined in the same way as Definition 3.6, 3.9.

In this section we formalize the notion of succinct testing for two-party relations. Below we first formalize its set-up.

Set-up 7. The set-up for succinct testing for two-party relations protocols is as follows. We consider a protocol between a client and a server.

The inputs, outputs and registers are similar to Set-up 2 with the following difference: $x \in \{0,1\}^n$ is the client-side input instead of a public input.

The completeness and soundness are in Definition 5.1, 5.2 below. The efficiency is defined in the common way. We additionally want the property that the communication complexity is $\mathsf{poly}(n, \kappa)$.

Definition 5.1 (Completeness)**.** We say a protocol under Set-up 7 is complete if when the server is honest and initially $\boldsymbol{S}_w$ holds a value w that satisfies $R(x,w) = 1$, the client accepts with probability $\geq 1 - \mathsf{negl}(\kappa)$.

Definition 5.2 (Soundness)**.** We say a protocol under Set-up 7 is (δ, ϵ)-sound if the following holds. For any malicious server Adv there exists an efficient quantum operation Ext such that for any polynomial-size inputs $1^n, 1^m, R, x$, for any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}})$, if the client accepts with probability $\geq 1 - \delta$, there is

$$\mathrm{tr}(\Pi^{\boldsymbol{S}_w}_{w:R(x,w)=1} \mathsf{Ext}(\rho_0)) \geq 1 - \epsilon - \mathsf{negl}(\kappa)$$

5.2 Protocol Design

Our succinct testing protocol is as follows. The protocol uses a family of collapsing hash functions as discussed in Section 3.3.3. Then recall that there exists a succinct argument of knowledge protocol for NP assuming collapsing hash functions (see Section 3.3.5).

The succinct testing protocol ($\mathsf{succTest}$) is as follows.

Protocol 5 ($\mathsf{succTest}$)**.** This protocol works under Set-up 7. The protocol inputs are: public parameters $1^n, 1^m, 1^\kappa$, public relation $R : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, client-side input $x \in \{0,1\}^n$.

Suppose Hash is a family of collapsing hash functions. Suppose SAoK is a succinct arguments of knowledge protocol for NP that is $(\delta, O(\delta))$-sound for all $\delta > 0$.

1. The client samples $h \leftarrow \mathsf{Hash}(1^m, 1^\kappa)$ and sends it to the server.
2. The server computes $h(w) = c$ and sends c to the client.
3. Using SAoK, the server proves the following statement to the client:
$$\exists w : h(w) = c$$
4. The client reveals x to the server.
5. Using SAoK, the server proves the following statement to the client:
$$\exists w : R(x, w) = 1 \wedge h(w) = c$$

The correctness and the efficiency are from the protocol description. The communication complexity is $\mathsf{poly}(n, \kappa)$. The assumption used in the protocol is the collapsing hash functions, used in both the commitment (step 2) and the succinct arguments of knowledge protocol.

Below we state the soundness.

Theorem 5.1. *For any $\delta > 0$, $\mathsf{succTest}$ is $(\delta, O(\delta))$-sound.*

Proof. Consider an efficient quantum adversary Adv. Suppose on some inputs and initial state ρ_0 the protocol passes with probability $\geq 1 - \delta$. Then the adversary should pass with probability $\geq 1 - \delta$ in both the SAoK in step 3 and step 5.

Use ρ_2 to denote the joint state between the server and the client. Apply the soundness of SAoK to step 3 we know that there exists an efficient quantum operation Ext_1 working on the server side such that

$$\mathrm{tr}(\Pi^{\boldsymbol{S}_w}_{w:h(w)=c} \mathsf{Ext}_1(\rho_2)) \geq 1 - O(\delta) - \mathsf{negl}(\kappa) \tag{26}$$

without loss of generality we could assume Ext_1 is a unitary operation.

Apply the soundness of SAoK to step 5 we know that there exists an efficient quantum operation Ext_2 which takes x as input and works on the server side such that

$$\mathrm{tr}(\Pi^{\boldsymbol{S}_w}_{w:R(x,w)=1 \wedge h(w)=c} \mathsf{Ext}_2(x)(\rho_2)) \geq 1 - O(\delta) - \mathsf{negl}(\kappa) \tag{27}$$

Now we apply the collapsing property of the hash functions to the state in (26). Use M to denote the operation that measure the register $\boldsymbol{S}_w$ and stores the results in register $\boldsymbol{S}_{\mathrm{temp}}$; here $\boldsymbol{S}_{\mathrm{temp}}$ is a new

register that is not touched by either the adversary or the extractor. Then by the collapsing property we know $\mathsf{Ext}_1(\rho_2)$ is $O(\delta)$-indistinguishable to $\mathsf{M}(\mathsf{Ext}_1(\rho_2))$ against all the efficient distinguishers that do not touch $\boldsymbol{S}_{\mathrm{temp}}$. Substituting this to (27) we get

$$\mathrm{tr}(\Pi^{\boldsymbol{S}_w}_{w:R(x,w)=1\wedge h(w)=c}\mathsf{Ext}_2(x)(\mathsf{Ext}_1^{-1}(\mathsf{M}(\mathsf{Ext}_1(\rho_2))))) \geq 1-O(\delta)-\mathsf{negl}(\kappa) \tag{28}$$

Then since h is collision-resistant the values in $\boldsymbol{S}_w$ and $\boldsymbol{S}_{\mathrm{temp}}$ should be the same on the space that $h(w)=c$:

$$\mathrm{tr}(\Pi^{\boldsymbol{S}_w,\boldsymbol{S}_{\mathrm{temp}}}_{(w_1,w_2):w_1=w_2}\Pi^{\boldsymbol{S}_w}_{w:R(x,w)=1\wedge h(w)=c}\mathsf{Ext}_2(x)(\mathsf{Ext}_1^{-1}(\mathsf{M}(\mathsf{Ext}_1(\rho_2))))) \geq 1-O(\delta)-\mathsf{negl}(\kappa) \tag{29}$$

This further implies that the value in $\boldsymbol{S}_{\mathrm{temp}}$ should also satisfy $R(x,w)=1$ with high probability, which further implies:

$$\mathrm{tr}(\Pi^{\boldsymbol{S}_w}_{w:R(x,w)=1\wedge h(w)=c}(\mathsf{Ext}_1(\rho_2))) \geq 1-O(\delta)-\mathsf{negl}(\kappa) \tag{30}$$

Thus we could define the overall extractor Ext as follows: Ext simulates the first and second step of $\mathsf{succTest}$ on its own and applies Ext_1 to extract the witness to register $\boldsymbol{S}_w$. Then (30) implies $\mathrm{tr}(\Pi^{\boldsymbol{S}_w}_{w:R(x,w)=1}(\mathsf{Ext}(\rho_0))) \geq 1-O(\delta)-\mathsf{negl}(\kappa)$ which is what we want. □

6 Construction of Our Secure Function Sampling Protocol

In this section we construct our SFS protocol. In Section 6.1 we give the protocol construction. In Section 6.2 we give the security proof.

6.1 Protocol Design

As said before, we first construct a preRSPV, and then amplify it to an RSPV which is exactly an SFS.

6.1.1 Construction of the preRSPV

The preRSPV $(\mathsf{SFSTest},\mathsf{SFSComp})$ is as follows. The protocols works under Set-up 6. The protocol inputs are: public parameters $1^n,1^m,1^\kappa$, public function $f:\{0,1\}^n\to\{0,1\}^m$.

Protocol 6 ($\mathsf{SFSTest}$)**.** 1. The client samples and stores the following strings: function inputs $x_0,x_1\in\{0,1\}^n$; input padding $r_0^{(in)},r_1^{(in)}\in\{0,1\}^\kappa$; output padding $r_0^{(out)},r_1^{(out)}\in\{0,1\}^\kappa$. Then the client prepares and sends

$$\frac{1}{\sqrt{2}}(|0\rangle|x_0\rangle|r_0^{(in)}\rangle|r_0^{(out)}\rangle+|1\rangle|x_1\rangle|r_1^{(in)}\rangle|r_1^{(out)}\rangle) \tag{31}$$

to the server.

2. The server evaluates f on the received state and gets the state

$$\frac{1}{\sqrt{2}}(\underbrace{|0\rangle}_{\boldsymbol{Q}^{(sub)}}\underbrace{|x_0\rangle}_{\boldsymbol{Q}^{(in)}}\underbrace{|r_0^{(in)}\rangle}_{\boldsymbol{Q}^{(inpad)}}\underbrace{|r_0^{(out)}\rangle}_{\boldsymbol{Q}^{(outpad)}}\underbrace{|f(x_0)\rangle}_{\boldsymbol{Q}^{(out)}}+|1\rangle|x_1\rangle|r_1^{(in)}\rangle|r_1^{(out)}\rangle|f(x_1)\rangle).$$

The server measures $\boldsymbol{Q}^{(in)}$ and $\boldsymbol{Q}^{(inpad)}$ on the Hadamard basis; denote the measurement results as $d^{(in)} \in \{0,1\}^n, d^{(inpad)} \in \{0,1\}^\kappa$. The server sends back these results to the client.

The client checks $d^{(inpad)} \neq 0^\kappa$. Then it calculates and stores

$$\theta_0 = d^{(in)} \cdot x_0 + d^{(inpad)} \cdot r_0^{(in)} \mod 2 \tag{32}$$

$$\theta_1 = d^{(in)} \cdot x_1 + d^{(inpad)} \cdot r_1^{(in)} \mod 2 \tag{33}$$

Now the remaining state on the server side is :

$$\frac{1}{\sqrt{2}}((-1)^{\theta_0} |0\rangle |r_0^{(out)}\rangle |f(x_0)\rangle + (-1)^{\theta_1} |1\rangle |r_1^{(out)}\rangle |f(x_1)\rangle). \tag{34}$$

3. The client randomly chooses to execute one of the following two tests with the server:

   - (Computational basis test) The client asks the server to measure $\boldsymbol{Q}^{(sub)}$, $\boldsymbol{Q}^{(outpad)}$ and $\boldsymbol{Q}^{(out)}$ on the computational basis; denote the measurement results as c. The client and the server execute $\mathsf{succTest}$ for the following relation

   $$c \in \{0||r_0^{(out)}||f(x_0), 1||r_1^{(out)}||f(x_1)\}.$$

   - (Hadamard basis test) The client asks the server to measure $\boldsymbol{Q}^{(sub)}$, $\boldsymbol{Q}^{(outpad)}$ and $\boldsymbol{Q}^{(out)}$ on the Hadamard basis; denote the measurement results as $d^{(sub)} \in \{0,1\}, d^{(outpad)} \in \{0,1\}^\kappa, d^{(out)} \in \{0,1\}^m$. The client and the server execute $\mathsf{succTest}$ for the following relation

   $$d^{(sub)} + d^{(outpad)} \cdot (r_0^{(out)} + r_1^{(out)}) + d^{(out)} \cdot (f(x_0) + f(x_1)) \equiv \theta_1 - \theta_0 \mod 2. \tag{35}$$

Protocol 7 ($\mathsf{SFSComp}$)**.** 1, 2. The step 1 and step 2 are the same as Protocol 6.

3. The client asks the server to measure $\boldsymbol{Q}^{(sub)}$ and $\boldsymbol{Q}^{(outpad)}$ on the computational basis; denote the measurement results as c. The server sends back c to the client.

   The client checks $c \in \{0||r_0^{(out)}, 1||r_1^{(out)}\}$. The client chooses x_0 as the client-side outputs if $c = 0||r_0^{(out)}$ and chooses x_1 as the client-side outputs if $c = 1||r_1^{(out)}$. The server-side outputs are stored in $\boldsymbol{Q}^{(out)}$.

The correctness and efficiency are from the protocol description; the communication complexity is $\mathsf{poly}(n, \kappa)$. The assumption used in the protocol is the collapsing hash functions, used in the calling to the succinct testing protocol.

The soundness is stated below.

Theorem 6.1. *For any $\delta > 0$, $(\mathsf{SFSTest}, \mathsf{SFSComp})$ is $(\delta, \mathsf{poly}(\delta))$-sound.*

The proof of Theorem 6.1 is given in Section 6.2.

6.1.2 Amplification from preRSPV to SFS

Below we further amplify the preRSPV to an RSPV which is exactly an SFS. (See Section 4.3 and Section 3.3.8 for related discussions).

The SFS (or RSPV) protocol SFS is as follows.

Protocol 8 (SFS)**.** This protocol works under Set-up 6. The protocol inputs are: public parameters $1^n, 1^m, 1^\kappa, 1^{1/\epsilon}$, public function $f : \{0,1\}^n \to \{0,1\}^m$.

The protocol is by applying the amplification procedure in Protocol 3, 4 to $(\mathsf{SFSTest}, \mathsf{SFSComp})$.

The correctness and the efficiency are from the protocol description. The communication complexity is $\mathsf{poly}(n, \kappa, 1/\epsilon)$. The assumption used in the protocol is the collapsing hash functions.

Below we state and prove the security.

Theorem 6.2. *Protocol 8 is ϵ-sound.*

Proof. By combining Theorem 6.1 and Theorem 3.12. □

What remains to be done is the proof of Theorem 6.1, which is given in the rest of this section.

6.2 Security Proofs

In this section we prove Theorem 6.1.

6.2.1 An overview of the proof of Theorem 6.1

As discussed in Section 2.4.2, to prove Theorem 6.1, we take an approach based on the notions in [54]. We will define a series of "forms" of states, and analyze different components of the protocol $(\mathsf{SFSTest}, \mathsf{SFSComp})$. We will prove these different components of the protocol restrict the forms of states step by step. Below we elaborate our approach.

First, to analyze the security, we work on the purified joint state between the client and the server, that is, we without loss of generality assume all the states are purified by some reference registers; this makes the security proof easier to work on.

Then we focus on the states by the end of the step 2 of the protocol. Note that the honest form of the state is given in (34), which is:

$$\frac{1}{\sqrt{2}}((-1)^{\theta_0}|0\rangle|r_0^{(out)}\rangle|f(x_0)\rangle + (-1)^{\theta_1}|1\rangle|r_1^{(out)}\rangle|f(x_1)\rangle). \tag{36}$$

We would like to prove the server's state in the malicious setting should also be close to this state in certain sense. To achieve this we define *basis-honest form* and *basis-phase correspondence form*, which are both sets of states, roughly as follows:

- The basis-honest form contains states in the form of

  $$|0\rangle|r_0^{(out)}\rangle|f(x_0)\rangle|\psi_0\rangle + |1\rangle|r_1^{(out)}\rangle|f(x_1)\rangle|\psi_1\rangle$$

  That is, the server-side registers $\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}$ store the right values as in the honest behavior, but there is no control on the adversary's state on the other registers. Note that we omit the client-side registers and the phases haven't been considered.

- The basis-phase correspondence form contains states in the form of

$$\sum_{\theta_0,\theta_1\in\{0,1\}^2} \underbrace{|\theta_0\rangle\,|\theta_1\rangle}_{\text{client side}}(|0\rangle\,|r_0^{(out)}\rangle\,|f(x_0)\rangle\,|\psi_{0,\theta_0}\rangle + |1\rangle\,|r_1^{(out)}\rangle\,|f(x_1)\rangle\,|\psi_{1,\theta_1}\rangle)$$

  Here we take the phase information into consideration and require that the 0-branch does not depend on the value of θ_1 and the 1-branch does not depend on the value of θ_0.

Based on these notions, our approach for analyzing the output state of the step 2 goes as follows:

1. If the server could pass the computational basis test from a state $|\psi\rangle$, then $|\psi\rangle$ is roughly (in certain sense) in the basis-honest form up to an isometry.
2. If $|\psi\rangle$ is prepared by the first two steps of $\mathsf{SFSTest}$ and is in the basis-honest form, then $|\psi\rangle$ is roughly (in certain sense) in a basis-phase correspondence form.
3. If $|\psi\rangle$ is in a basis-phase correspondence form and satisfies certain efficiently-preparable property, if it could pass the Hadamard basis test, then $|\psi\rangle$ is roughly (in certain sense) the state (34).

In summary different components of $\mathsf{SFSTest}$ controls the forms of states step by step. (Note that some form of efficiently preparable condition is needed to go through.)

Once we have control on the form of output states by the step 2 of $(\mathsf{SFSTest}, \mathsf{SFSComp})$, the analysis of the remaining steps is relatively easy. The third step of $\mathsf{SFSComp}$ basically forces the server to collapse the state; from the values of the output padding the client could keep the right function input x, while the server only holds the corresponding $f(x)$.

Comparison to [54] We note that many notions are from [54] and we compare both works as follows.

- On the one hand the technical difficulties are much simpler than that of [54]. [54] needs to analyze several tests and to reduce the technical works [54] sacrifices, for example, composable security and efficient simulation. Here we do not need to sacrifice these desirable properties.
- On the other hand since we want our protocol to be (sequentially) composable we need careful technical works to construct the simulator.

6.2.2 Definitions and statements

Set-up 8. The set-up for analyzing the output state by the step 2 of $(\mathsf{SFSTest}, \mathsf{SFSComp})$ is as follows.

The register naming and set-up is as follows:

- The initial state is $|\varphi_{\text{init}}\rangle \in \mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}}$.
- The client holds registers $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1$, $\boldsymbol{r}_0^{(out)}$, $\boldsymbol{r}_1^{(out)}$. The values in these registers are x_0, x_1, θ_0, etc, as used in Protocol 6, 7; so the domain of values in these registers follow the description in the protocol. In the honest setting these values should be all uniformly random.

  Use $\boldsymbol{x}$ to denote $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ and $\boldsymbol{\theta}, \boldsymbol{r}^{(in)}, \boldsymbol{r}^{(out)}$ are defined similarly. Use $\boldsymbol{d}^{(\text{step 2})}$ to denote the client side registers holding $(d^{(in)}, d^{(inpad)})$.

- The server holds registers $\boldsymbol{Q}^{(sub)}$, $\boldsymbol{Q}^{(outpad)}$, $\boldsymbol{Q}^{(out)}$. In the honest setting the server's state should be (34).
- We could assume the following client-side registers have been disgarded: $\boldsymbol{r}^{(in)}, \boldsymbol{d}^{(\text{step 2})}$.
- We consider a purification system (or called reference system) $\boldsymbol{R}$, which contains the purification system for each classical registers (for example, $\boldsymbol{x}$ and $\boldsymbol{r}^{(out)}$); we also assume the disgarded registers are part of $\boldsymbol{R}$. Denote the purification corresponding to $\boldsymbol{\theta}$ as $\boldsymbol{R}_{\boldsymbol{\theta}}$.

Then we define Π_0 to be the projection onto the space that the value of $\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}$ is equal to $0||\boldsymbol{r}_0^{(out)}||f(\boldsymbol{x}_0)$ (where we use the register symbol to denote the corresponding values in them). Similarly define Π_1 to be the projection onto the space that the value of $\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}$ is equal to $1||\boldsymbol{r}_1^{(out)}||f(\boldsymbol{x}_1)$. Thus equation (34) is invariant under the operation of $(\Pi_0+\Pi_1)$.

We name different components of the protocol as follows. We use FirstTwoStep to denote the first two steps of (SFSTest, SFSComp); use SecondStep to denote the second step of (SFSTest, SFSComp); use ComputationalTest to denote the computational basis test; use HadamardTest to denote the Hadamard basis test; use ThirdStepComp to denote the third step of SFSComp.

Finally we define some special purified joint states. Note that the first step of (SFSTest, SFSComp) the client prepares and sends a state to the server; denote the purification of this state as

$$|\varphi_{\text{step 1}}\rangle = \sum_{x_0,x_1,r_0^{(in)},r_1^{(in)},r_0^{(out)},r_1^{(out)}\in\{0,1\}^{2n+4\kappa}} \frac{1}{\sqrt{2^{2n+4\kappa}}} \underbrace{|x_0,x_1,r_0^{(in)},r_1^{(in)},r_0^{(out)},r_1^{(out)}\rangle}_{\text{client-side registers } \boldsymbol{x},\boldsymbol{r}^{(in)},\boldsymbol{r}^{(out)}} \otimes$$

$$\frac{1}{\sqrt{2}}(\underbrace{|0\rangle\,|x_0\rangle\,|r_0^{(in)}\rangle\,|r_0^{(out)}\rangle}_{\text{server-side registers } \boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(in)}\boldsymbol{Q}^{(inpad)}\boldsymbol{Q}^{(outpad)}} + |1\rangle\,|x_1\rangle\,|r_1^{(in)}\rangle\,|r_1^{(out)}\rangle) \underbrace{|\cdots\rangle}_{\text{purification system in } \boldsymbol{R}}$$

Then define the target purified joint state after step 2 as

$$|\varphi_{\text{tar step 2}}\rangle = \sum_{x_0,x_1\theta_0,\theta_1,r_0^{(out)},r_1^{(out)}\in\{0,1\}^{2n+2\kappa+2}} \frac{1}{\sqrt{2^{2n+2\kappa+2}}} \underbrace{|x_0,x_1,\theta_0,\theta_1,r_0^{(out)},r_1^{(out)}\rangle}_{\text{client-side registers } \boldsymbol{x},\boldsymbol{\theta},\boldsymbol{r}^{(out)}} \otimes$$

$$\frac{1}{\sqrt{2}}((-1)^{\theta_0} \underbrace{|0\rangle\,|r_0^{(out)}\rangle\,|f(x_0)\rangle}_{\text{server-side registers } \boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}} +(-1)^{\theta_1}|1\rangle\,|r_1^{(out)}\rangle\,|f(x_1)\rangle) \underbrace{|\cdots\rangle}_{\text{purification system in } \boldsymbol{R}}$$

Definition 6.1. Now we introduce the notions for different forms of states as follows.

- We say a state $|\psi\rangle$ in Set-up 8 is in basis-honest form if
$$(\Pi_0+\Pi_1)|\psi\rangle = |\psi\rangle\,. \tag{37}$$
We call the two terms on the right hand side of (37) as the 0-branch and the 1-branch.
- We say a state $|\psi\rangle$ in Set-up 8 is in basis-phase correspondence form if $|\psi\rangle$ is in the basis-honest form, and for each $b\in\{0,1\}$, its b-branch, denoted as $|\psi_b\rangle$, could be written as
$$|\psi_b\rangle = \sum_{\theta_0,\theta_1\in\{0,1\}^2} \underbrace{|\theta_0\rangle\,|\theta_1\rangle}_{\text{client-side registers } \boldsymbol{\theta}_0,\boldsymbol{\theta}_1} |\psi_{b,\theta_b}\rangle \underbrace{|\theta_0\rangle\,|\theta_1\rangle}_{\text{purification register } \boldsymbol{R}_{\boldsymbol{\theta}}} \tag{38}$$
Note that $|\psi_{b,\theta_b}\rangle$ is the same for different values of θ_{1-b}. Thus we say the 0-branch is independent to the value of $\boldsymbol{\theta}_1$ and the 1-branch is independent to the value of $\boldsymbol{\theta}_0$.

To state the lemmas we need an efficiently-preparable notion.

Definition 6.2. We say a state family σ could be efficiently preparable from ρ if there exists a family of efficient quantum operations O such that $\sigma = O(\rho)$.

Finally we define the closeness between states up to an operation on register $\boldsymbol{R}$:

Definition 6.3. We say $|\psi_1\rangle \approx_\epsilon^{\text{up to } \boldsymbol{R}} |\psi_2\rangle$ if there exists an operation U that solely works on $\boldsymbol{R}$ such that $U|\psi_1\rangle \approx_\epsilon |\psi_2\rangle$.

Fact 3. *$|\psi_1\rangle \approx_\epsilon^{up\ to\ \boldsymbol{R}} |\psi_2\rangle$, then $\mathrm{tr}_{\boldsymbol{R}}(|\psi_1\rangle\langle\psi_1|) \approx_\epsilon^{ind} \mathrm{tr}_{\boldsymbol{R}}(|\psi_2\rangle\langle\psi_2|)$.*

Below we state our lemmas for proving Theorem 6.1.

Lemma 6.3. *If an efficient quantum adversary Adv working on the server-side of a state $|\psi\rangle$ could pass the computational-basis test with probability $\geq 1-\delta$, then there exists an efficient quantum operation O working on the server side such that $|(\Pi_0+\Pi_1)O|\psi\rangle|^2 \geq 1-\mathsf{poly}(\delta)$.*

Lemma 6.4. *Suppose Adv is an efficient quantum adversary playing the role of the server. Suppose $|\psi\rangle = \mathsf{SecondStep}^{\mathsf{Adv}}(|\varphi_{step\ 1}\rangle \otimes |\varphi_{init}\rangle)$ satisfies $|\Pi_{\mathsf{pass}}(\Pi_0+\Pi_1)|\psi\rangle|^2 \geq 1-\mathsf{poly}(\delta)$. Then there exists a state $|\tilde{\psi}\rangle$ that satisfies:*

- *$|\tilde{\psi}\rangle$ is efficiently-preparable from $|\varphi_{init}\rangle$.*
- *$|\psi\rangle \approx_{\mathsf{poly}(\delta)+\mathsf{negl}(\kappa)}^{up\ to\ \boldsymbol{R}} |\tilde{\psi}\rangle$. (Note that this together with the condition implies $|\Pi_{\mathsf{pass}}(\Pi_0+\Pi_1)|\tilde{\psi}\rangle|^2 \geq 1-\mathsf{poly}(\delta)$.)*
- *$\Pi_{\mathsf{pass}}(\Pi_0+\Pi_1)|\tilde{\psi}\rangle$ is in a basis-phase correspondence form.*

Lemma 6.5. *Suppose $|\psi\rangle$ is $\mathsf{poly}(\delta)$-close to a basis-phase correspondence form and is efficiently preparable from $|\varphi_{init}\rangle$. If an efficient quantum adversary working on the server-side of $|\psi\rangle$ could pass the Hadamard-basis test with probability $\geq 1-\delta$, then there exists an efficient quantum operation Sim that satisfies*

$$|\psi\rangle \approx_{\mathsf{poly}(\delta)+\mathsf{negl}(\kappa)} (\mathsf{I} \otimes \underbrace{\mathsf{Sim}}_{server\ side})(|\varphi_{tar\ step\ 2}\rangle \otimes |\varphi_{init}\rangle)$$

Lemma 6.6. *For any efficient quantum operation Adv playing the role of the server there exists an efficient quantum operation Sim playing the role of the server such that*

$$\Pi_{\mathsf{pass}}(\mathsf{ThirdStepComp}^{\mathsf{Adv}}(|\varphi_{tar\ step\ 2}\rangle\langle\varphi_{tar\ step\ 2}| \otimes |\varphi_{init}\rangle\langle\varphi_{init}|)) \approx_{\mathsf{poly}(\delta)+\mathsf{negl}(\kappa)}^{ind} \Pi_{\mathsf{pass}}(\mathsf{Sim}(\rho_{tar} \otimes |\varphi_{init}\rangle\langle\varphi_{init}|)) \tag{39}$$

(Note that ρ_{tar} is defined in Definition 4.1. See also Definition 4.3 for the related security definition.)

6.2.3 Formal Security Proofs

Proof of Lemma 6.3. By the soundness of $\mathsf{succTest}$ we know there exists a server-side operation O that extracts a string in $\{0||r_0^{(out)}||f(x_0), 1||r_1^{(out)}||f(x_1)\}$ to the register $\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}$, which completes the proof. □

Proof of Lemma 6.4. Consider the state $|\psi_{mid}\rangle$ as follows:

1. Starting from the state $|\varphi_{\text{step }1}\rangle \otimes |\varphi_{\text{init}}\rangle$, copy the values in $\boldsymbol{Q}^{(in)}$ to an auxiliary register $\boldsymbol{aux}$.

2. Execute $\mathsf{SecondStep}^{\mathsf{Adv}}$ on $|\varphi_{\text{step 1}}\rangle \otimes |\varphi_{\text{init}}\rangle$.

3. Use the values in $\boldsymbol{aux}$ to uncompute $\boldsymbol{Q}^{(in)}$; swap $\boldsymbol{aux}$ to $\boldsymbol{Q}^{(in)}$.

We claim that

$$\Pi_{\mathsf{pass}}(\Pi_0 + \Pi_1)\,|\psi\rangle \approx_{\mathsf{negl}(\kappa)} \Pi_{\mathsf{pass}}(\Pi_0 + \Pi_1)\,|\psi_{mid}\rangle \tag{40}$$

(40) holds because if we expand both $\Pi_{\mathsf{pass}}(\Pi_0 + \Pi_1)\,|\psi\rangle$ and $\Pi_{\mathsf{pass}}(\Pi_0 + \Pi_1)\,|\psi_{mid}\rangle$ by branches of $|\varphi_{\text{step 1}}\rangle$, the norm of their difference is bounded by $|\Pi_1 \mathsf{Adv} \Pi_0(|\varphi_{\text{step 1}}\rangle \otimes |\varphi_{\text{init}}\rangle)| + |\Pi_0 \mathsf{Adv} \Pi_1(|\varphi_{\text{step 1}}\rangle \otimes |\varphi_{\text{init}}\rangle)|$, which is $\mathsf{negl}(\kappa)$. Note that this together with the condition that $|\Pi_{\mathsf{pass}}(\Pi_0 + \Pi_1)\,|\psi\rangle\,|^2 \geq 1 - \mathsf{poly}(\delta)$ implies that $|\psi\rangle \approx_{\mathsf{poly}(\delta)+\mathsf{negl}(\kappa)} |\psi_{mid}\rangle$.

The construction of the $|\psi_{mid}\rangle$ is to enforce that for each $b \in \{0,1\}$, the b-branch of $\Pi_{\mathsf{pass}}\,|\psi_{mid}\rangle$ only comes from the b-branch of $|\varphi_{\text{step 1}}\rangle$ thus there is no information about the other branch thus should be independent to the value of $\boldsymbol{\theta}_{1-b}$. But this intuition is not exactly true: the reason is, θ_{1-b} is computed from $d^{(\text{step 2})}$ and $x_{1-b}, r_{1-b}^{(in)}$; although the $x_{1-b}, r_{1-b}^{(in)}$ registers (together with their purification registers) are initially in a product state from the other parts on the b-branch, the measurement of $d^{(\text{step 2})}$ and calculation of θ_{1-b} could create some entanglement between these registers. But note that the registers holding $d^{(\text{step 2})}$, $r^{(in)}$, and the purification register $\boldsymbol{R}_{\boldsymbol{\theta}}$ are all in $\boldsymbol{R}$; thus as long as we could find an operation AlignR that operates only on these registers that de-entangles $\boldsymbol{\theta}_{1-b}$, we could still prove the lemma.

Now we construct an operation AlignR that operates on $\boldsymbol{r}^{(in)}$, $\boldsymbol{d}^{(\text{step 2})}$ and $\boldsymbol{R}_{\boldsymbol{\theta}}$ such that $\Pi_{\mathsf{pass}}(\Pi_0 + \Pi_1)\mathsf{AlignR}\,|\psi_{mid}\rangle$ is a basis-phase correspondence form. Let's first without loss of generality consider the 0-branch: note that initially $r_1^{(in)}$ is randomly sampled from $\{0,1\}^\kappa$; then note for each $d^{(inpad)} \neq 0$ (which corresponds to the Π_{pass} space), for each $\theta_1 \in \{0,1\}$, the number of $r_1^{(in)} \in \{0,1\}^\kappa$ such that the value of $\boldsymbol{\theta}_1$ equals θ_1 is $2^{\kappa-1}$ (see equation (33)). Thus once we uncompute $\boldsymbol{r}_1^{(in)}$ with $(\boldsymbol{d}^{(inpad)}, \boldsymbol{R}_{\boldsymbol{\theta}_1})$ the 0-branch will be independent to the value of $\boldsymbol{\theta}_1$. The case for the 1-branch is the same. In summary we could construct AlignR as follows:

1. Uncompute $\boldsymbol{r}_1^{(in)}$ with with $\boldsymbol{d}^{(inpad)}, \boldsymbol{R}_{\boldsymbol{\theta}_1}$.

2. Uncompute $\boldsymbol{r}_0^{(in)}$ with with $\boldsymbol{d}^{(inpad)}, \boldsymbol{R}_{\boldsymbol{\theta}_0}$.

Then $\Pi_{\mathsf{pass}}(\Pi_0 + \Pi_1)\mathsf{AlignR}\,|\psi_{mid}\rangle$ is a basis-phase correspondence form; thus defining $|\tilde{\psi}\rangle = \mathsf{AlignR}\,|\psi_{mid}\rangle$ proves the lemma. □

The proof of Lemma 6.5 is as follows. Note that the overall ideas of the proof details are related to the discussion in Section 2.3.2. We add a discussion on the proof details after the proof.

Proof of Lemma 6.5. Suppose $|\psi\rangle$ is $\mathsf{poly}(\delta)$-close to a state $|\tilde{\psi}\rangle$ in the basis-phase correspondence form. Then by the soundness of $\mathsf{succTest}$ we know there exists a server-side unitary operation Ext that extracts strings to $\boldsymbol{S}_{dstep3}$ such that

$$|\Pi^{\boldsymbol{S}_{dstep3}}_{d^{(sub)},d^{(outpad)},d^{(out)}:\text{LHS of (35)}\equiv\theta_0-\theta_1 \mod 2}\mathsf{Ext}\,|\tilde{\psi}\rangle\,|^2 \geq 1 - \mathsf{poly}(\delta) \tag{41}$$

Recall that the LHS of (35) is $d^{(sub)} + d^{(outpad)} \cdot (r_0^{(out)} + r_1^{(out)}) + d^{(out)} \cdot (f(x_0) + f(x_1))$.

Introduce the following notation to simplify the expression:

- Use $\Pi_\equiv$ to denote the projection onto the space that the value in $\boldsymbol{S}_{dstep3}$ satisfies LHS of (35) $\equiv \theta_0 - \theta_1 \mod 2$; use $\Pi_{\not\equiv}$ to denote the complement of $\Pi_\equiv$.

- Use $|\tilde{\psi}_0\rangle$ to denote the 0-branch of $|\tilde{\psi}\rangle$ and use $|\tilde{\psi}_1\rangle$ to denote the 1-branch of $|\tilde{\psi}\rangle$. Then recall equation (38), and defines $|\tilde{\psi}_{0,\theta_0}\rangle$ for each $\theta_0 \in \{0,1\}$ and $|\tilde{\psi}_{1,\theta_1}\rangle$ for each $\theta_1 \in \{0,1\}$ as (38).

Since $|\tilde{\psi}\rangle$ is in the basis-phase correspondence form, for each branch of this state there is:

$$\forall b \in \{0,1\}, |\Pi_{\equiv}\mathsf{Ext}\,|\tilde{\psi}_b\rangle\,| \leq \frac{1}{\sqrt{2}}|\,|\tilde{\psi}_b\rangle\,| \tag{42}$$

Combining (41)(42) we have

$$\Pi_{\equiv}\mathsf{Ext}\,|\tilde{\psi}_0\rangle \approx_{\mathsf{poly}(\delta)} \Pi_{\equiv}\mathsf{Ext}\,|\tilde{\psi}_1\rangle \tag{43}$$

$$\Pi_{\not\equiv}\mathsf{Ext}\,|\tilde{\psi}_0\rangle \approx_{\mathsf{poly}(\delta)} -\Pi_{\not\equiv}\mathsf{Ext}\,|\tilde{\psi}_1\rangle \tag{44}$$

Expanding $|\tilde{\psi}_b\rangle$ to $|\tilde{\psi}_{b,\theta_b}\rangle$ and making use of the form of $|\tilde{\psi}\rangle$, (43)(44) further imply:

$$|\tilde{\psi}_{0,0}\rangle \approx_{\mathsf{poly}(\delta)} -|\tilde{\psi}_{0,1}\rangle \tag{45}$$

$$|\tilde{\psi}_{1,0}\rangle \approx_{\mathsf{poly}(\delta)} -|\tilde{\psi}_{1,1}\rangle \tag{46}$$

$$U_{f(\boldsymbol{x}),\boldsymbol{r}^{(out)}}\,|\tilde{\psi}_{0,0}\rangle \approx_{\mathsf{poly}(\delta)} |\tilde{\psi}_{1,0}\rangle \tag{47}$$

where $U_{f(\boldsymbol{x}),\boldsymbol{r}^{(out)}}$ is defined to be the following operation:

1. Apply Ext;
2. Add a (-1) phase on the space of $\Pi_{\not\equiv}$. Note that this step requires the information of $f(x_0), f(x_1), r_0^{(out)}, r_1^{(out)}$ so we name U as above.
3. Apply Ext^{-1} (note that we chose Ext to be a unitary above).

Now we construct the Sim as required in the statement of this lemma. Below we give the construction and analyze each step during the construction.

Sim works as follows:

1. First prepare the state $|\psi\rangle$ from $|\varphi_{\text{init}}\rangle$ on its own register; here all the client-side registers, including $\boldsymbol{x}, \boldsymbol{r}^{(out)}, \boldsymbol{\theta}$, are simulated within the simulator's memory; the server-side parts like $\boldsymbol{Q}^{(sub)}, \boldsymbol{Q}^{(outpad)}, \boldsymbol{Q}^{(out)}$ are also simulated in new corresponding registers. Denote these new registers used for simulation as $\boldsymbol{x}'$, $\boldsymbol{r}^{(out)\prime}$, $\boldsymbol{\theta}'$, $\boldsymbol{Q}^{(sub)\prime}$, etc.

   Suppose the state created in this step is $|\psi'\rangle$. The difference between $|\psi\rangle$ and $|\psi'\rangle$ is simply the location of registers (in more detail, $\boldsymbol{x}'$, $\boldsymbol{r}^{(out)\prime}$, $\boldsymbol{\theta}'$, $\boldsymbol{Q}^{(sub)'}$, $\boldsymbol{Q}^{(outpad)\prime}$, $\boldsymbol{Q}^{(out)\prime}$). Then we could define $|\tilde{\psi}'\rangle$, $|\tilde{\psi}'_{b,\theta}\rangle$ etc similarly. By the previous proof $|\psi'\rangle$ is also $\mathsf{poly}(\delta)$-close to $|\tilde{\psi}'\rangle$ and (45)(46)(47) all hold after replacing all the $\tilde{\psi}_{\ldots}$ by the corresponding $\tilde{\psi}'_{\ldots}$.

2. Do an xor operation between $\boldsymbol{Q}^{(sub)}$ and $\boldsymbol{Q}^{(sub)\prime}$. Discuss by cases on the values of $\boldsymbol{Q}^{(sub)}$ and the xor result:

   - Suppose the xor is 1:

– On the 0-branch of $|\varphi_{\text{tar step 2}}\rangle$ (in other words, $\boldsymbol{Q}^{(sub)}$ has value 0): now the 0-branch could be written as (omitting the client-side registers)

$$\underbrace{|0\rangle\,|r_0^{(out)}\rangle\,|f(x_0)\rangle}_{\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}}\,|\tilde{\psi}_1'\rangle \tag{48}$$

We would like to transform this state to $|\tilde{\psi}_0\rangle$. The differences are as follows: (1) the obvious difference is that $|\tilde{\psi}_1'\rangle$ is the 1-branch while we want a 0-branch; (2) the less obvious difference is $|\tilde{\psi}_1'\rangle$ corresponds to the "simulated client-side registers" like $\boldsymbol{x}'$, $\boldsymbol{\theta}'$, $\boldsymbol{r}^{(out)\prime}$ etc while in what we want ($|\tilde{\psi}_0\rangle$) the server-side state should be related to the real client-side regsiters $\boldsymbol{x}$, $\boldsymbol{\theta}$, $\boldsymbol{r}^{(out)}$.
We notice that the following operations allow us to achieve this transformation.

(a) First by (45)(46) we could remove the additional phase in $|\tilde{\psi}_{1,1}'\rangle$ (that is, apply a (-1) phase controlled by $\boldsymbol{\theta}_1'$).

(b) Then by (47) apply $U_{f(\boldsymbol{x}_0),f(\boldsymbol{x}_1'),\boldsymbol{r}_0^{(out)},\boldsymbol{r}_1^{(out)\prime}}$ to map this branch of simulated state to $|\tilde{\psi}_0\rangle$ (omitting temporary registers like $\boldsymbol{x}',\boldsymbol{\theta}',\boldsymbol{r}^{(out)\prime}$, etc; they are now in a fixed product state so we could ignore them).
In more detail, we make use of values in the following registers to implement U: (1) the values in registers $\boldsymbol{Q}^{(outpad)},\boldsymbol{Q}^{(out)}$ (on the server-side of $|\varphi_{\text{tar step 2}}\rangle$, corresponding to $f(\boldsymbol{x}_0)$ and $\boldsymbol{r}_0^{(out)}$ in the subscripts of U); (2) the values in the "simulated client-side registers" $\boldsymbol{x}_1',\boldsymbol{r}_1^{(out)\prime}$.

(c) Now the values of $\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}$ is the same as $\boldsymbol{Q}^{(sub)\prime}\boldsymbol{Q}^{(outpad)\prime}\boldsymbol{Q}^{(out)\prime}$ so $\boldsymbol{Q}^{(sub)\prime}\boldsymbol{Q}^{(outpad)\prime}\boldsymbol{Q}^{(out)\prime}$ could be uncomputed.

– The 1-branch is similar. Note that we could control on the $\boldsymbol{Q}^{(sub)}$ to map the 0-branch first and 1-branch next.

• Suppose the xor is 0: now for example, the 0-branch could be written as (omitting the client-side registers)

$$|0\rangle\,|r_0^{(out)}\rangle\,|f(x_0)\rangle\,|\tilde{\psi}_0'\rangle \tag{49}$$

We note that in the two differences discussed above, although there is only one difference remaining here, applying $U_{f(\boldsymbol{x}_0),f(\boldsymbol{x}_1'),\boldsymbol{r}_0^{(out)},\boldsymbol{r}_1^{(out)\prime}}$ as above does not solve the problem. Here our solution is, in the step (b) above, first apply $U_{f(\boldsymbol{x}_0'),f(\boldsymbol{x}_1'),\boldsymbol{r}_0^{(out)\prime},\boldsymbol{r}_1^{(out)\prime}}$ to map it to the simulated 1-branch, and then apply $U_{f(\boldsymbol{x}_0),f(\boldsymbol{x}_1'),\boldsymbol{r}_0^{(out)},\boldsymbol{r}_1^{(out)\prime}}$ as above.

The 1-branch is similar.

3. Finally uncompute the intermediate registers (like $\boldsymbol{x}',\boldsymbol{\theta}',\boldsymbol{r}^{(out)\prime}$) completes the simulation.

□

Note on the intuition of the proof We note that one way to understand the details of this proof is to return to the discussion in Section 2.3.2. Note that in the discussion in 2.3.2 the client-side Hadamard measurement is assumed to be honest (which introduces phases), while in the condition of this lemma we assume the output state from step 2 of SFSTest is in a basis-phase correspondence form. Although the conditions are not the same, the techniques (like the construction of U) share

similar ideas. One additional difficulty is that in this proof we want the final result to have good (sequential) composability so we need to take the initial state $|\varphi_{\text{init}}\rangle$ into consideration. To solve this problem, we construct Sim as follows: we first solely do the simulation from $|\varphi_{\text{init}}\rangle$, then xor it with the target state $|\varphi_{\text{tar step 2}}\rangle$; finally we make use of U (or in more detail, (45)(46)(47)) to transform the state to the state we want.

Proof of Lemma 6.6. Notice that $r_{1-b}^{(out)}$ is unpredictable from the b-branch. This implies that the left hand side of (39) is negligibly close to the following state on the passing space:

1. Starting from the state $|\varphi_{\text{tar step 2}}\rangle \otimes |\varphi_{\text{init}}\rangle$, do a projection on $\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}$ and store the corresponding x on the client-side register $\boldsymbol{x}^{(out)}$. Keep a copy of the value in $\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}$ in a temporary register $\boldsymbol{aux}$.

2. Apply Adv as given in the lemma statement.

3. Project onto the space that the value of $\boldsymbol{Q}^{(sub)}\boldsymbol{Q}^{(outpad)}\boldsymbol{Q}^{(out)}$ is the same as $\boldsymbol{aux}$; mark $\boldsymbol{flag}$ as pass if the projection succeeds.

   Disgard $\boldsymbol{aux}$.

Notice that the initial state in $(\boldsymbol{x}^{(out)}, \boldsymbol{Q}^{(out)})$ is exactly ρ_{tar}. Thus the operation above gives the Sim we want, which completes the proof. □

Proof of Theorem 6.1. First for any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$ we could always purify it to $|\varphi_{\text{init}}\rangle \in \mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}} \otimes \mathcal{H}_{\boldsymbol{E}'}$; thus we could only consider pure states as in Set-up 8. Then if an adversary Adv could pass with probability $\geq 1 - \delta$ in $\mathsf{SFSTest}$ it should pass with probability $\geq 1 - \mathsf{poly}(\delta)$ in each step of $\mathsf{SFSTest}$. Chaining Lemma 6.3, 6.4, 6.5 and 6.6 completes the proof. □

7 Protocols for General Two-party Computations

In this section we give the generalizations and improvements.

- In Section 7.1 we construct the SFS protocol for randomized functions.
- In Section 7.2 we construct the SFVP protocol, where the function input x is provided by the client.
- In Section 7.3 we construct the protocol for secure two-party computations in succinct communication.
- In Section 7.4 we construct the protocol for classical-channel secure two-party computations in succinct communication.

We refer to Section 2.7 for an overview; in this section we formalize the protocols and security proofs.

7.1 SFS for Randomized Functions

The set-up of our protocol is as follows.

Set-up 9. The set-up is similar to Set-up 5 with the following differences. The function f is an efficient classical randomized function $f : \{0,1\}^n \times \{0,1\}^L \to \{0,1\}^m$ (as described in Fact 2; the second input is the random coins). 1^L is given as a public parameter.

The completeness and soundness are in Definition 7.1, 7.2 below.

Definition 7.1. We say a protocol π under Set-up 9 is complete if when the server is honest, the protocol passes with probability $\geq 1 - \mathsf{negl}(\kappa)$ and furthermore, denote the output state as $\rho_{out} \in \mathrm{D}(\mathcal{H}_{\boldsymbol{x}^{(out)}} \otimes \mathcal{H}_{\boldsymbol{Q}^{(out)}})$, there is

$$\rho_{out} \approx^{\mathrm{ind}} \rho_{tar}$$

where ρ_{tar} is defined to be the following state: the register $\boldsymbol{x}^{(out)}$ holds a random classical string $x \leftarrow_r \{0,1\}^n$, the register $\boldsymbol{Q}^{(out)}$ holds a sampling of $f(x;r)$ where $r \leftarrow_r \{0,1\}^L$.

Note that the completeness here is also defined using the indistinguishability notion.

The soundness is defined in the same way as Definition 4.2 with the difference that the function called by the ideal functionality is now randomized.

Definition 7.2. To define the soundness, first define SFSIdeal as follows.

SFSIdeal receives a bit $b \in \{0,1\}$ from the server, then:

- If $b = 0$, SFSIdeal sets $\boldsymbol{flag}$ to be pass and prepares ρ_{tar} in $\boldsymbol{x}^{(out)}$, $\boldsymbol{Q}^{(out)}$.
- If $b = 1$, SFSIdeal sets $\boldsymbol{flag}$ to be fail.

We say a protocol π under Set-up 9 is ϵ-sound if:

For any efficient adversary Adv, there exist efficient quantum operations $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any polynomial-size inputs $1^n, 1^m, 1^L, 1^{1/\epsilon}, f$ and any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

$$\pi^{\mathsf{Adv}}(\rho_0) \approx^{\mathrm{ind}}_{\epsilon} \underbrace{\mathsf{Sim}_1}_{\text{on } \boldsymbol{S},\boldsymbol{Q}^{(out)}} (\mathrm{SFSIdeal}(\underbrace{\mathsf{Sim}_0}_{\text{on } \boldsymbol{S} \text{ and outputs } b} (\rho_0)))$$

Our SFS protocol for randomized functions is as follows. The protocol uses the pseudorandom generator (PRG) primitive as discussed in Section 3.3.1.

Protocol 9 (SFS for randomized functions)**.** This protocol works under Set-up 9. The protocol inputs are: public parameters $1^n, 1^m, 1^L, 1^\kappa, 1^{1/\epsilon}$, public function $f : \{0,1\}^n \times \{0,1\}^L \to \{0,1\}^m$.

The protocol makes use of a pseudorandom generator $\mathrm{PRG}(1^\kappa, 1^L) : \{0,1\}^\kappa \to \{0,1\}^L$. Below we omit the parameters.

1. The client and the server execute protocol SFS (Protocol 8) with error tolerance ϵ for function $g : \{0,1\}^{n+\kappa} \to \{0,1\}^m$ defined as follows:

   $$g(x,s) := f(x; \mathrm{PRG}(s)) \text{ where } (x,s) \in \{0,1\}^n \times \{0,1\}^\kappa$$

   When the server is honest the client should get a random $x, s \leftarrow_r \{0,1\}^n \times \{0,1\}^\kappa$ and

the server gets the corresponding $g(x,s)$. The server stores $g(x,s)$ in $\boldsymbol{Q}^{(out)}$ as the final outputs; the client disgards s and stores x in $\boldsymbol{x}^{(out)}$ as the final outputs.

The communication is succinct by the property of SFS and the fact that the size of (x,s) is only $n+\kappa$. The assumption is the collapsing hash functions, used in the SFS protocol; note that PRG is implied by the existence of collapsing hash functions.

Theorem 7.1. *Protocol 9 is complete.*

Proof. By the completeness of SFS (Protocol 8) and the property of PRG (Definition 3.3). □

The soundness is below.

Theorem 7.2. *Protocol 9 is ϵ-sound.*

Proof. For any efficient quantum adversary Adv, by Theorem 6.2 there exist efficient quantum operations Sim such that $\mathsf{SFS}^{\mathsf{Adv}}$ is ϵ-indistinguishable to $\mathsf{Disgard}(\boldsymbol{s}) \circ \mathsf{Sim} \circ \mathrm{SFSIdeal}(g)$, where $\mathsf{Disgard}(\boldsymbol{s})$ denotes the client-side operation that disgards the s value (in the last step of Protocol 9). Then by the property of PRG, $\mathsf{Disgard}(\boldsymbol{s}) \circ \mathrm{SFSIdeal}(g)$ is indistinguishable to $\mathrm{SFSIdeal}(f)$ (that is, we replace the pseudorandom coins by the real random coins). Combining them completes the proof. □

7.2 Secure Function Value Preparation

In this section we construct the secure function value preparation (SFVP) protocol, where the function input is provided by the client as its private input. The set-up is formalized below.

Set-up 10. The set-up for SFVP is similar to Set-up 5 with the following differences. Besides the inputs in Set-up 5, the protocol also takes a private input $x \in \{0,1\}^n$ from the client. Then the client-side output register is only $\boldsymbol{flag}$.

The completeness and soundness are in Definition 7.3, 7.4 below.

Definition 7.3. We say a protocol under Set-up 10 is complete if when the server is honest, for any client-side input x, the output state of the protocol is negligibly close to the following state: the register $\boldsymbol{flag}$ holds the value pass, and the register $\boldsymbol{Q}^{(out)}$ holds the classical string $f(x)$.

Definition 7.4. To define the soundness, we first define SFVPIdeal as follows.

SFVPIdeal receives a bit $b \in \{0,1\}$ from the server, then:

- If $b=0$, SFVPIdeal sets $\boldsymbol{flag}$ to be pass and stores $f(x)$ in $\boldsymbol{Q}^{(out)}$.
- If $b=1$, SFVPIdeal sets $\boldsymbol{flag}$ to be fail.

Then we define the soundness as follows. We say a protocol π under Set-up 10 is ϵ-sound if:

For any efficient quantum adversary Adv, there exist efficient quantum operations $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any polynomial-size inputs $1^n, 1^m, 1^{1/\epsilon}, f, x$ and any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

$$\pi^{\mathsf{Adv}}(\rho_0) \approx^{\mathrm{ind}}_{\epsilon} \underbrace{\mathsf{Sim}_1}_{\text{on } \boldsymbol{S},\boldsymbol{Q}^{(out)}} (\mathrm{SFVPIdeal}(\underbrace{\mathsf{Sim}_0}_{\text{on } \boldsymbol{S} \text{ and outputs } b}(\rho_0))) \tag{50}$$

The protocol is as follows. The protocol uses a decomposable garbling scheme as discussed in Section 3.3.6.

Protocol 10 (SFVP)**.** This protocol works under Set-up 10. The protocol inputs are: public parameters $1^n, 1^m, 1^\kappa, 1^{1/\epsilon}$, public function f, client-side input $x \in \{0,1\}^n$.

The protocol makes use of a decomposable randomized encoding scheme ($\mathsf{GarbleC}, \mathsf{GarbleI}, \mathsf{Dec}$).

1. Use C_f to denote the circuit representation of f.

   The client and the server execute the SFS protocol (Protocol 9)[a] for function g: $\{0,1\}^{n\kappa} \to \{0,1\}^{\mathsf{poly}(|C_f|,\kappa)}$:

   $$g(r) := \mathsf{GarbleC}(C_f, r)$$

   with error tolerance ϵ.

   If the server is honest the client should get a random $r \leftarrow_r \{0,1\}^{n\kappa}$ and the server gets a sample of $\mathsf{GarbleC}(C_f, r)$.

2. The client computes and sends $\mathsf{GarbleI}(x, r)$ to the server.

   The honest server could compute $\mathsf{Dec}(\mathsf{GarbleC}(C_f, r), \mathsf{GarbleI}(x, r))$ to get $f(x)$.

[a] Note that g is a randomized function since $\mathsf{GarbleC}$ could have internal random coins; see Section 3.3.6 for details.

The communication is succinct by the property of Protocol 9 and the fact that (x, r) is succinct. The assumption is the collapsing hash functions by the constructions of Protocol 9 and Yao's garbled circuits. The completeness is from the completeness of Protocol 9 and completeness of Yao's garbled circuits; note that since the final output of the protocol is determinisitic, the indistinguishability in the completeness definition of Protocol 9 (Definition 7.1) does not appear here.

Theorem 7.3. *Protocol 10 is ϵ-sound.*

Proof. For any efficient quantum adversary Adv, we would like to construct efficient quantum operations Sim that satisfies Definition 7.4.

First by the soundness of the SFS protocol (Theorem 7.2) we know that, there exist efficient quantum operations $\mathsf{Sim}_{\mathrm{SFS}} = (\mathsf{Sim}_{\mathrm{SFS},0}, \mathsf{Sim}_{\mathrm{SFS},1})$ working on the server side such that for any polynomial-size inputs and any initial state ρ_0,

$$\mathsf{SFVP}^{\mathsf{Adv}}(\rho_0) \approx_\epsilon^{\mathrm{ind}} \mathsf{Hybrid}(\rho_0) \tag{51}$$

where the operation Hybrid, operated on ρ_0, is defined as follows:

1. Apply $\mathsf{Sim}_{\mathrm{SFS},0}$ to ρ_0, which generates $b \in \{0,1\}$ as the server-side input to SFSIdeal.
2. Execute SFSIdeal for function g. (Recall the definition of SFSIdeal in Definition 7.2.)
3. Apply $\mathsf{Sim}_{\mathrm{SFS},1}$. Then apply Adv_1, which denotes the adversary's operation after the first step.
4. The client computes and sends $\mathsf{GarbleI}(x, r)$ to the server.
5. Apply Adv_2, which denotes the adversary's operation after the second step.

Then by the soundness of Yao's garbled circuits (Definition 3.13) there exists an efficient quantum operation $\mathsf{Sim}_{\mathrm{GC}}$ such that

$$\mathsf{Sim}_{\mathrm{GC}}(C_f(x)) \approx^{\mathrm{ind}} \mathrm{Dist}\begin{bmatrix} r \leftarrow_r \{0,1\}^{n\kappa} \\ (\mathsf{GarbleC}(C_f, r), \mathsf{GarbleI}(x, r)) \end{bmatrix} \tag{52}$$

Now we are ready to construct the simulator $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ as follows:

- Sim_0:
  1. Apply $\mathsf{Sim}_{\mathrm{SFS},0}$ to ρ_0, which generates $b \in \{0,1\}$ as the server-side input to SFVPIdeal.
- Sim_1:
  - If $b = 0$:
    1. Use the $C_f(x)$ that it receives from SFVPIdeal to prepare $\mathsf{Sim}_{\mathrm{GC}}(C_f(x))$; call its outputs "simulated circuit encoding" and "simulated input encoding".
    2. Apply $\mathsf{Adv}_1 \circ \mathsf{Sim}_{\mathrm{SFS},1}$, but use the simulated circuit encoding to replace the $\mathsf{GarbleC}(C_f, r)$ (outputted by the end of step 1 of SFVP protocol or from SFSIdeal in Hybrid).
    3. Apply Adv_2, but use the simulated input encoding to replace the $\mathsf{GarbleI}(x, r)$ (transmitted in step 2 of SFVP protocol).
  - If $b = 1$: simply apply $\mathsf{Adv}_2 \circ \mathsf{Adv}_1 \circ \mathsf{Sim}_{\mathrm{SFS},1}$ on the server-side state.

Comparing Hybrid and $\mathsf{Sim}_1 \circ \mathrm{SFVPIdeal} \circ \mathsf{Sim}_0$ and use (52) we could see

$$\mathsf{Hybrid}(\rho_0) \approx^{\mathrm{ind}} \mathsf{Sim}_1(\mathrm{SFVPIdeal}(\mathsf{Sim}_0(\rho_0))) \tag{53}$$

Combining (51)(53) we have

$$\mathsf{SFVP}^{\mathsf{Adv}}(\rho_0) \approx_{\epsilon}^{\mathrm{ind}} \mathsf{Sim}_1(\mathrm{SFVPIdeal}(\mathsf{Sim}_0(\rho_0)))$$

which completes the proof. □

7.3 Secure Two-party Computations Protocol

In this section we give our secure two-party computations (2PC) protocol in succinct communication. As discussed in Section 2.7.3, this is achieved in two steps:

- In Section 7.3.1 we construct the secure function value preparation (SFVP) protocol with certain security against malicious clients.
- In Section 7.3.2 we construct the protocol for secure two-party computations.

7.3.1 Secure function value preparation with soundness against malicious clients

In this section we construct the SFVP protocol with certain security against malicious clients. As discussed in Section 2.7.3, we consider a special set-up where the client-side input x has been committed to the server. The set-up is formalized below.

Set-up 11. This set-up is similar to Set-up 10 (the set-up for ordinary SFVP) with the following differences.

For the client-side inputs, besides the function input x, there is an additional input $r \in \{0,1\}^{n\kappa}$ used for the opening of the commitment; for the public inputs, there are additionally $pp \in \{0,1\}^{3n\kappa}$ and the commitment $com = \mathsf{Commit}(x, r, pp)$ (see Section 3.3.2 for definitions of Commit; note that Commit is statistically binding); here r, pp should be considered to be randomly sampled (in both the completeness and soundness definitions below).

For the output registers, the server also holds a flag register $\boldsymbol{flag}^{(S)}$ (in addition to the output register $\boldsymbol{Q}^{(out)}$). The client-side flag register is denoted as $\boldsymbol{flag}^{(C)}$.

Either the client or the server could be malicious. For modeling the initial states in the malicious server setting, suppose the overall server-side registers are denoted by $\boldsymbol{S}$ and the environment is denoted by $\boldsymbol{E}$; for modeling the initial states in the malicious client setting, suppose the overall client-side registers are denoted by $\boldsymbol{C}$ and the environment is denoted by $\boldsymbol{E}$.

The completeness, soundness against malicious server and soundness against malicious client are defined in Definition 7.5, 7.6 and 7.7 below.

We still want the property that the communication complexity is $\mathsf{poly}(n, \kappa, 1/\epsilon)$.

Definition 7.5. We say a protocol under Set-up 11 is complete if when both the client and the server are honest, the output state of the protocol is negligibly close to the following state: $\boldsymbol{flag}^{(C)}, \boldsymbol{flag}^{(S)}$ both hold the value pass, $\boldsymbol{Q}^{(out)}$ holds the classical string $f(x)$.

The soundness against malicious servers is as follows.

Definition 7.6. To define the soundness against malicious servers, we first define SFVP2IdealS as follows.

SFVP2IdealS receives a bit $b \in \{0, 1\}$ from the server, then:

- If $b = 0$, SFVP2IdealS sets $\boldsymbol{flag}^{(C)}$ to be pass and stores $f(x)$ in $\boldsymbol{Q}^{(out)}$.
- If $b = 1$, SFVP2IdealS sets $\boldsymbol{flag}^{(C)}$ to be fail.

Then we define the soundness as follows. We say a protocol π under Set-up 11 is ϵ-sound against malicious servers if:

For any efficient quantum adversary Adv playing the role of the server, there exist efficient quantum operations $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any polynomial-size inputs that satisfy the conditions in the set-up and any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

$$\pi^{\mathsf{Adv}}(\rho_0) \approx_\epsilon^{\mathrm{ind}} \underbrace{\mathsf{Sim}_1}_{\text{on } \boldsymbol{S}, \boldsymbol{Q}^{(out)}} (\mathrm{SFVP2IdealS}(\underbrace{\mathsf{Sim}_0}_{\text{on } \boldsymbol{S} \text{ and outputs } b} (\rho_0))) \tag{54}$$

The soundness against malicious client is as follows.

Definition 7.7. To define the soundness against malicious clients, we first define SFVP2IdealC as follows.

SFVP2IdealC receives a bit $b \in \{0, 1\}$ from the client, then:

- If $b = 0$, SFVP2IdealC sets $\boldsymbol{flag}^{(S)}$ to be pass and stores $f(x)$ in $\boldsymbol{Q}^{(out)}$.
- If $b = 1$, SFVP2IdealC sets $\boldsymbol{flag}^{(S)}$ to be fail.

Then we define the soundness as follows. We say a protocol π under Set-up 11 is ϵ-sound against malicious clients if:

For any efficient quantum adversary Adv playing the role of the client, there exist efficient quantum operations $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any polynomial-size inputs that satisfy the conditions in the set-up and any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{C}} \otimes \mathcal{H}_{\boldsymbol{E}})$:

$$\pi^{\mathsf{Adv}}(\rho_0) \approx_\epsilon^{\mathrm{ind}} \underbrace{\mathsf{Sim}_1}_{\text{on } \boldsymbol{C}} (\mathrm{SFVP2IdealC}(\underbrace{\mathsf{Sim}_0}_{\text{on } \boldsymbol{C} \text{ and outputs } b} (\rho_0))) \tag{55}$$

The protocol is as follows. The protocol uses a family of collapsing hash functions and a succinct-ZK-AoK protocol (see Section 3.3.3, 3.3.5).

Protocol 11 (SFVP2)**.** This protocol works under Set-up 11. The protocol inputs are: public parameters $1^n, 1^m, 1^\kappa, 1^{1/\epsilon}$, public inputs f, pp, com, client-side inputs x, r, and it should be satisfied that $com = \mathsf{Commit}(x, r, pp)$.

The protocol makes use of a family of collapsing hash functions Hash and a succinct-ZK-AoK protocol $\mathsf{succZKAoK}$ that is[a] $(\delta, \mathsf{poly}_{\mathrm{AoK}}(\delta))$ for any $\delta > 0$.

1. Both parties execute SFVP (Protocol 10) for function f, client-side input x and error tolerance ϵ.

   Denote the server's outcome as y. If both parties are honest there should be $y = f(x)$.

2. The server samples $h \leftarrow \mathsf{Hash}(1^m, 1^\kappa)$, computes $h(y) = c$ and sends (h, c) to the client.

3. Define $L = 16/(\epsilon^2 \cdot \mathsf{poly}^{-1}_{\mathrm{AoK}}(\frac{\epsilon}{4}))$.

   For each $i \in [L]$:

   (a) Using $\mathsf{succZKAoK}$, the client proves the following statement to the server:

   $$\exists (x, r) : h(f(x)) = c \wedge \mathsf{Commit}(x, r, pp) = com \tag{56}$$

   If all the tests pass successfully the server stores y as its output.

[a] Here $\mathsf{poly}_{\mathrm{AoK}}(\delta) = O(\delta)$ as discussed in Theorem 3.8. We use $\mathsf{poly}_{\mathrm{AoK}}$ to allow us to refer to this polynomial explicitly.

The completeness and the succinct communication property are from the protocol description. The primitives used here are all implied by collapsing hash functions.

Below we state the soundness against malicious servers and malicious clients.

Theorem 7.4. *Protocol 11 is ϵ-sound against malcious servers.*

Theorem 7.5. *Protocol 11 is ϵ-sound against malicious clients.*

The proof of soundness against malicious servers is relatively easier.

Proof of Theorem 7.4. For any efficient quantum adversary Adv playing the role of the malicious server, apply the soundness of SFVP protocol (Theorem 7.3) and ZK-property of the $\mathsf{succZKAoK}$ protocol (Definition 3.12)[11] we know there exist efficient quantum operations $\mathsf{Sim}_{\mathrm{SFVP}} = (\mathsf{Sim}_{\mathrm{SFVP},0}, \mathsf{Sim}_{\mathrm{SFVP},1})$, $(\mathsf{Sim}_{\mathrm{ZK},i})_{i\in[L]}$ working on the server side such that for any polynomial-size inputs that satisfy the set-up and any initial states $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}} \otimes \mathcal{H}_{\boldsymbol{E}})$,

$$\mathsf{SFVP2}^{\mathsf{Adv}}(\rho_0) \approx^{\mathrm{ind}}_{\epsilon} \mathsf{Hybrid}(\rho_0)$$

where Hybrid operated on ρ_0 is defined as follows:

[11] Note that here the client proves statements to the server so the role of client and server are swapped compared to Definition 3.12.

1. Apply the server-side operation $\mathsf{Sim}_{\mathrm{SFVP},0}$ to ρ_0, which generates $b \in \{0,1\}$ as the server-side input to SFVPIdeal.
2. Execute SFVPIdeal.
3. Apply the server-side operation $\mathsf{Sim}_{\mathrm{SFVP},1}$. Then apply Adv_1, which denotes the adversary's operation after the first step.
4. The adversary sends a message to the client.
5. For each $i \in [L]$, apply the server-side operation $\mathsf{Sim}_{\mathrm{ZK},i}$.

This could be translated to a simulator interacting with SFVP2IdealS directly, which completes the proof. □

The proof of soundness against malicious clients requires some technical works. We will analyze the third step and construct a simulator that works in a cut-and-choose manner to find a round with high passing probability, which implies that the client indeed holds a $\tilde{x}$ that satisfies (56). Then this together with the statistically-binding of Commit and collision-resistance of h implies that the server either indeed holds $f(x)$ or catches the client cheating, which completes the proof.

Proof of Theorem 7.5. For any efficient adversary Adv playing the role of the client and any initial state, let's use $\sigma_i \in \mathrm{D}(\mathcal{H}_{\boldsymbol{C}} \otimes \mathcal{H}_{\boldsymbol{E}} \otimes \mathcal{H}_{\boldsymbol{S}})$ to denote the joint states by the end of the i-th round of the third step of $\mathsf{SFVP2}$. Then use $\Pi_{\mathsf{pass}}^{(\leq i)}(\sigma_i)$ to denote the projection of σ_i onto the space that all the rounds so far have passed. Use $\Pi_{\mathsf{pass}}^{(i)}$ to denote the projection onto the space that the i-th round passes; $\Pi_{\mathsf{fail}}^{(i)}$ is defined similarly.

Use S to denote the set of round index $i \in [L]$ that satisfies $\mathrm{tr}(\Pi_{\mathsf{fail}}^{(i)}(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1}))) \geq \frac{\epsilon}{4}\mathsf{poly}^{-1}(\frac{\epsilon}{4})$. Then we have $|S|/L \leq \frac{\epsilon}{4}$. For each $i \notin S$, if $\mathrm{tr}(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1})) \geq \frac{\epsilon}{4}$, we could apply the $(\mathsf{poly}^{-1}(\frac{\epsilon}{4}), \frac{\epsilon}{4})$-soundness of the PoK; if $\mathrm{tr}(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1})) \leq \frac{\epsilon}{4}$ the trace of any state $\mathcal{E}(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1}))$ is bounded directly. Combining both cases into one expression we have: there exists an efficient quantum operation Ext_i working on the client side that extracts to register $\boldsymbol{C}_{\mathrm{temp}}$ such that

$$\Pi_{\tilde{x},\tilde{r}:h(f(\tilde{x}))=c\wedge\mathsf{Commit}(\tilde{x},\tilde{r},pp)=com}^{\boldsymbol{C}_{\mathrm{temp}}}\mathsf{Ext}_i(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1})) \approx_{\max\{\frac{\epsilon}{4},2\times\frac{\epsilon}{4}\}} \mathsf{Ext}_i(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1})). \tag{57}$$

(Note that we write $\max\{\frac{\epsilon}{4}, 2\times\frac{\epsilon}{4}\}$ to suggest the fact that we are combining two cases in one expression.)

Note that (57) holds for $i \notin S$. Recall that $|S|/L \leq \frac{\epsilon}{4}$. Thus we get the following result, for a randomly selected $i \in [L]$:

$$\mathrm{Den}\left[\begin{array}{c} i \leftarrow_r [L] \\ \Pi_{\tilde{x},\tilde{r}:h(f(\tilde{x}))=c\wedge\mathsf{Commit}(\tilde{x},\tilde{r},pp)=com}^{\boldsymbol{C}_{\mathrm{temp}}}(\mathsf{Ext}_i(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1}))) \end{array}\right] \approx_\epsilon \mathrm{Den}\left[\begin{array}{c} i \leftarrow_r [L] \\ \mathsf{Ext}_i(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1})) \end{array}\right] \tag{58}$$

where we use Den to denote the density operator of the last row when the variables are sampled according to the first row.

Now by the statistically binding of Commit we know:

$$\mathrm{Den}\left[\begin{array}{c} i \leftarrow_r [L] \\ \Pi_{\tilde{x},\tilde{r}:h(f(\tilde{x}))=c\wedge\mathsf{Commit}(\tilde{x},\tilde{r},pp)=com}^{\boldsymbol{C}_{\mathrm{temp}}}\mathsf{Ext}_i(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1})) \end{array}\right] \approx_{\mathsf{negl}(\kappa)} \mathrm{Den}\left[\begin{array}{c} i \leftarrow_r [L] \\ \Pi_{\tilde{x}:\tilde{x}=x\wedge h(f(\tilde{x}))=c}^{\boldsymbol{C}_{\mathrm{temp}}}(\mathsf{Ext}_i(\Pi_{\mathsf{pass}}^{(\leq i-1)}(\sigma_{i-1}))) \end{array}\right] \tag{59}$$

(Note that $\tilde{x}$ is the value of $\boldsymbol{C}_{\mathrm{temp}}$ and x is the actual input of the protocol.)

Note that intuitively on the right hand side of (59) the client already holds x that satisfies $h(f(x)) = c$; by the collision-resistance of h the value y in the server-side should satisfy $y = f(x)$. Formally, we have

$$\mathrm{Den}\left[\begin{array}{c} i \leftarrow_r [L] \\ \Pi^{\boldsymbol{C}_{\mathrm{temp}}}_{\tilde{x}:\tilde{x}=x\wedge h(f(\tilde{x}))=c}(\mathsf{Ext}_i(\Pi^{(\le i-1)}_{\mathsf{pass}}(\sigma_{i-1}))) \end{array}\right] \approx_{\mathsf{negl}(\kappa)} \mathrm{Den}\left[\begin{array}{c} i \leftarrow_r [L] \\ \Pi^{\boldsymbol{Q}^{(out)}}_{y:y=f(x)}\Pi^{\boldsymbol{C}_{\mathrm{temp}}}_{\tilde{x}:\tilde{x}=x}(\mathsf{Ext}_i(\Pi^{(\le i-1)}_{\mathsf{pass}}(\sigma_{i-1}))) \end{array}\right] \tag{60}$$

Combining (58)(59)(60) we get a good control on the server-side information in the passing space.

Thus the overall simulator Sim working on the client side could go as follows:

1. Simulate the first two steps of the $\mathsf{SFVP2}$ protocol on its own memory.
2. Pick a random $i \leftarrow [L]$.
3. For each $i' < i$, simulate these rounds on its own memory and record the passing/failing outcome.
4. For the i-th round, simulate but do not record the passing/failing outcome.
5. For each $i' > i$, simulate these rounds on its own memory and record the passing/failing outcome.
6. Send the joint passing/failing outcome as the input b in SFVP2IdealC.

□

7.3.2 Construction of the secure two-party computations protocol

In this section we construct the two-party computations (2PC) protocol in succinct communication, which proves Theorem 1.3. The set-up is as follows.

Set-up 12. The set-up for 2PC with approximate security is as follows. The set-up is similar to Set-up 1 with the following differences: the protocol takes an additional approximation error tolerance parameter $1^{1/\epsilon}$.

The soundness is formalized as the approximate variant of Definition 3.8.

We want the communication complexity to be $\mathsf{poly}(n_A + n_B, \kappa, 1/\epsilon)$.

We would like to design a protocol for this problem in succinct communication.

The protocol is as follows. The protocol makes use of several primitives formalized in Section 3.3.

Protocol 12. The protocol works under Set-up 12. The protocol inputs are: public parameters $1^{n_A}, 1^{m_A}, 1^{n_B}, 1^{m_B}, 1^{\kappa}, 1^{1/\epsilon}$, public function f_A, f_B, Alice-side input $x^{(A)} \in \{0,1\}^{n_A}$, Bob-side input $x^{(B)} \in \{0,1\}^{n_B}$.

The protocol uses the following primitive:

- The oneway-based-2PC protocol discussed in Section 3.3.4.
- The statistically-binding commitment scheme Commit as formalized in Section 3.3.2.

- The decomposable randomized encoding scheme $(\mathsf{GarbleC}, \mathsf{GarbleI}, \mathsf{Dec})$ as formalized in Section 3.3.6. Recall that the inputs to a deterministic description of $\mathsf{GarbleC}$ are the circuit, the shared random coins used by $\mathsf{GarbleC}$ and $\mathsf{GarbleI}$, and the internal random coins of $\mathsf{GarbleC}$.
- A pseudorandom generator $\mathrm{PRG}(1^\kappa, 1^{\mathsf{poly}((n_A+n_B),\kappa)})$.

1. Alice and Bob perform 2PC for the following functions:
   (a) (Sampling public randomness for commitments):
   Sample $pp^{(A)}, pp^{(B)} \leftarrow_r \{0,1\}^{3(n_A+n_B+1)\kappa^2}$ as the public outputs.
   (b) (Sample sender-side randomness for commitments):
   Sample $r_{com}^{(A)} \leftarrow_r \{0,1\}^{(n_A+n_B+1)\kappa^2}$ as the Bob-side output; sample $r_{com}^{(B)} \leftarrow_r \{0,1\}^{(n_A+n_B+1)\kappa^2}$ as the Alice-side output.
   (c) (Sample randomness for garbling scheme; r denotes the shared randomness used in $\mathsf{GarbleC}$ and $\mathsf{GarbleI}$ and s is used to generate the internal randomness of $\mathsf{GarbleC}$):
   Sample $r^{(A)} \leftarrow_r \{0,1\}^{(n_A+n_B)\kappa}$, $s^{(A)} \leftarrow_r \{0,1\}^\kappa$ as the Bob-side outputs; sample $r^{(A)} \leftarrow_r \{0,1\}^{(n_A+n_B)\kappa}$, $s^{(B)} \leftarrow_r \{0,1\}^\kappa$ as the Alice-side outputs.
   (d) (Commitments):
   Compute $com^{(A)} = \mathsf{Commit}((r^{(A)}, s^{(A)}), r_{com}^{(A)}, pp^{(A)})$ as the public output; compute $com^{(B)} = \mathsf{Commit}((r^{(B)}, s^{(B)}), r_{com}^{(B)}, pp^{(B)})$ as the public output.
   (e) (Input encoding):
   Compute $\mathsf{GarbleI}((x^{(A)}, x^{(B)}), r^{(A)})$ as the Alice-side output; compute $\mathsf{GarbleI}((x^{(A)}, x^{(B)}), r^{(B)})$ as the Bob-side output.

   Here the superscript (A) is used to denote the intermediate variables used for evaluating f_A and superscript (B) is used to denote the intermediate variables used for evaluating f_B; who receives these variables could depend on the protocol design.
2. Note that the commitments are as required in Set-up 11. Bob uses the $\mathsf{SFVP2}$ protocol to send $g_A(r^{(A)}, s^{(A)})$ to Alice with soundness error tolerance $\epsilon/2$, where:
   $$g_A(r^{(A)}, s^{(A)}) := \mathsf{GarbleC}(C_{f_A}, r^{(A)}; \mathrm{PRG}(s^{(A)}))$$
   where C_{f_A} is the circuit description of f_A.
   Alice could compute
   $$\mathsf{Dec}(\mathsf{GarbleC}(C_{f_A}, r^{(A)}; \mathrm{PRG}(s^{(A)})), \mathsf{GarbleI}((x^{(A)}, x^{(B)}), r^{(A)}))$$
   to get $f_A(x^{(A)}, x^{(B)})$.
3. Alice uses the $\mathsf{SFVP2}$ protocol to send $g_B(r^{(B)}, s^{(B)})$ to Bob with soundness error tolerance $\epsilon/2$, where:
   $$g_B(r^{(B)}, s^{(B)}) := \mathsf{GarbleC}(C_{f_B}, r^{(B)}; \mathrm{PRG}(s^{(B)}))$$

where C_{f_B} is the circuit description of f_B.

Bob could compute

$$\mathsf{Dec}(\mathsf{GarbleC}(C_{f_B}, r^{(B)}; \mathrm{PRG}(s^{(B)})), \mathsf{GarbleI}((x^{(A)}, x^{(B)}), r^{(B)}))$$

to get $f_B(x^{(A)}, x^{(B)})$.

The communication is succinct by the succinctness of SFVP2 protocol and the fact that the two-party computations in step 1 are all performed on inputs and functions with size $\mathsf{poly}(n_A + n_B, \kappa)$. The assumptions and primitives used in this protocol are all implied by the existence of the collapsing hash functions. The completeness is by the protocol description.

The soundness is stated below.

Theorem 7.6. *Protocol 12 is ϵ-sound.*

Proof. We prove the soundness against malicious Bob and the proof of the soundness against malicious Alice is similar.

For any efficient quantum adversary Adv playing the role of Bob, we would like to construct efficient quantum operations Sim playing the role of Bob that interacts with $\text{2PCIdeal}^{(B)}$ and simulates the real protocol.

First apply the soundness of 2PC (the first step of Protocol 12), SFVP2 against malicious clients (the second step of Protocol 12), and SFVP2 against malicious servers (the third step of Protocol 12) we know that, there exist simulators (which interacts with the corresponding ideal functionality) $\mathsf{Sim}_{\text{step 1}}$, $\mathsf{Sim}_{\text{step 2}}$, $\mathsf{Sim}_{\text{step 3}}$ working on the Bob side such that for any polynomial-size inputs and any initial state $\rho_0 \in \mathrm{D}(\mathcal{H}_{\boldsymbol{S}^{(B)}} \otimes \mathcal{H}_{\boldsymbol{E}})$, the output state of the protocol is ϵ-indistinguishable to $\mathsf{Hybrid}_1(\rho_0)$, where Hybrid_1 is as follows:

1. Apply $(\mathsf{Sim}_{\text{step 1}} \circ \text{2PCIdeal}^{(B)}_{\text{step 1}})$. Here $\text{2PCIdeal}^{(B)}_{\text{step 1}}$ is the ideal functionality of the 2PC protocol used in the first step of Protocol 12 (note that this is different from 2PCIdeal, which denotes the ideal functionality of the whole protocol).

2. Apply $(\mathsf{Sim}_{\text{step 2}} \circ \mathrm{SFVP2IdealC}(g_A))$.

3. Apply $(\mathsf{Sim}_{\text{step 3}} \circ \mathrm{SFVP2IdealS}(g_B))$.

Further apply the definition of PRG and the hiding property of Commit we know $\mathsf{Hybrid}_1(\rho_0)$ is indistinguishable to $\mathsf{Hybrid}_2(\rho_0)$ where Hybrid_2 is as follows:

1. Apply $(\mathsf{Sim}_{\text{step 1}} \circ \text{2PCIdeal}'^{(B)}_{\text{step 1}})$, where $\text{2PCIdeal}'^{(B)}_{\text{step 1}}$ is similar to $\text{2PCIdeal}^{(B)}_{\text{step 1}}$ with the following difference: the commitment to $r^{(B)}$ (that is, $com^{(B)}$) is replaced by a commitment to a random string.

2. Apply $(\mathsf{Sim}_{\text{step 2}} \circ \mathrm{SFVP2IdealC}(g'_A))$, where g'_A is a randomized function as follows: $g'_A(r^{(A)}) = \mathsf{GarbleC}(C_{f_A}, r^{(A)})$. Note the definition of SFVP2IdealC (Definition 7.7) could be adapted to randomized function in the natural way.

3. Apply $(\mathsf{Sim}_{\text{step 3}} \circ \mathrm{SFVP2IdealS}(g'_B))$, where g'_B is a randomized function as follows: $g'_B(r^{(B)}) = \mathsf{GarbleC}(C_{f_B}, r^{(B)})$. Note the definition of SFVP2IdealS (Definition 7.6) could be adapted to randomized function in the natural way.

Then what remains to be analyzed is the garbled circuits. By the soundness of the garbled circuits scheme (Definition 3.14, Theorem 3.10) there exist efficient simulator $\mathsf{Sim}_{\mathrm{ie}}, \mathsf{Sim}_{\mathrm{ce}}$ such that

$$\mathrm{Dist}\begin{bmatrix} \tilde{x} \leftarrow \mathsf{Sim}_{\mathrm{ie}}() \\ (\tilde{x}, \mathsf{Sim}_{\mathrm{ce}}(\tilde{x}, C_{f_B}(x^{(A)}, x^{(B)}))) \end{bmatrix} \approx^{\mathrm{ind}} \mathrm{Dist}\begin{bmatrix} r^{(B)} \leftarrow_r \{0,1\}^{(n_A+n_B)\kappa} \\ \hat{x} := \mathsf{Garble\mathsf{I}}((x^{(A)}, x^{(B)}), r^{(B)}) \\ (\hat{x}, \mathsf{GarbleC}(C_{f_B}, r^{(B)})) \end{bmatrix} \tag{61}$$

Now we are ready to construct the overall simulator Sim that interacts with $\mathrm{2PCIdeal}^{(B)}$ as follows.

1. Sim first works as $\mathsf{Sim}_{\mathrm{step\ 1}}$ as in the first step of Hybrid_2; if $\mathsf{Sim}_{\mathrm{step\ 1}}$ early-aborts then Sim early aborts. Otherwise Sim could extract the input $x^{(B)}$ from $\mathsf{Sim}_{\mathrm{step\ 1}}$'s preparation of inputs to $\mathrm{2PCIdeal}'^{(B)}_{\mathrm{step\ 1}}$.

   For the simulation of the output string from $\mathrm{2PCIdeal}'^{(B)}_{\mathrm{step\ 1}}$, Sim uses $\tilde{x} \leftarrow \mathsf{Sim}_{\mathrm{ie}}()$ to simulate the input encoding $\mathsf{Garble\mathsf{I}}((x^{(A)}, x^{(B)}), r^{(B)})$.

2. Sim then works as $\mathsf{Sim}_{\mathrm{step\ 2}}$ and stores $b_{\mathrm{step\ 2}} \in \{0,1\}$ which determines whether the server should abort in SFVP2IdealC.

3. Sim then works as $\mathsf{Sim}_{\mathrm{step\ 3}}$ and aborts if $\mathsf{Sim}_{\mathrm{step\ 3}}$ aborts; otherwise it queries $\mathrm{2PCIdeal}^{(B)}$ with $x^{(B)}$ and gets $f_B(x^{(A)}, x^{(B)})$. Then it computes $\mathsf{Sim}_{\mathrm{ce}}(\tilde{x}, f_B(x^{(A)}, x^{(B)}))$ to simulate the outputs of SFVP2IdealS.

4. Finally Sim determines whether Alice should receives its outputs $f_A(x^{(A)}, x^{(B)})$ based on $b_{\mathrm{step\ 2}}$ (that is, use $b_{\mathrm{step\ 2}}$ as the input to $\mathrm{2PCIdeal}^{(B)}$; recall in Definition 3.8 $\mathrm{2PCIdeal}^{(B)}$ receives a bit from Bob that determines whether Alice should receive its output in the final step).

Comparing each case we know $\mathsf{Hybrid}_2(\rho_0)$ is indistinguishable to $\mathsf{Sim}(\rho_0)$.

Combining this chain of hybrids completes the proof. □

7.4 Classical-channel Secure Two-party Computations Protocol

In this section we construct our classical-channel 2PC protocol in succinct communication.

- In Section 7.4.1 we give a result on classical-channel 2PC (without considering the succinctness of communication). We construct this protocol as a preparation for the final construction.
- In Section 7.4.2 we give our protocol for classical-channel 2PC in succinct communication.

7.4.1 On classical-channel 2PC without the succinct communication requirement

We first construct a classical-channel 2PC protocol with approximate security assuming NTCF. This is by applying the RSPV-for-BB84 states (see [12, 57] and Theorem 3.14) to the oneway-based-2PC in [7].

Protocol 13. This protocol works under Set-up 12 but we do not put restriction on the communication complexity.

Suppose the 2PC protocol in [7] contains L rounds of quantum communication in the form of "One party sends random N-qubits BB84 states to the other party". Our protocol replaces all these quantum communication by RSPV protocols for N-qubits BB84 states with error

tolerance ϵ/L.

The assumption is the existence of NTCF; the completeness and efficiency are from the protocol description.

Theorem 7.7. *Protocol 13 is ϵ-sound.*

Proof. First the original protocol in [7] is $\mathsf{negl}(\kappa)$-sound.

Then suppose we are working on a protocol that is ϵ_0-sound against malicious party B and we are going to replace a quantum communication step in the form of "One party sends random N-qubits BB84 states to the other party" by an RSPV for this state family with error tolerance ϵ/L. Then:

- If party B is the receiver: by the soundness of RSPV the new protocol is $\epsilon_0 + \epsilon/L$-sound against malicious party B.
- If party B is the sender: note that RSPV protocol does not take private inputs; B's any malicious operation in the new protocol could always be simulated in the original protocol as follows: the party B prepares the output states of the RSPV protocol on its own and sends A's part to A. Since the original protocol is ϵ_0-sound against malicious B the new protocol is also ϵ_0-sound against malicious B.

□

7.4.2 Classical-channel 2PC in succinct communication

Finally we are ready to construct our classical-channel 2PC protocol in succinct communication. This is achieved by using Protocol 13 to replace the first step of Protocol 12 and use the RSPV for $|0\rangle|\tilde{x}_0\rangle + |1\rangle|\tilde{x}_1\rangle$ to replace the quantum communication in the second and third step of Protocol 12.

Protocol 14. This protocol works under Set-up 12.

1. Use Protocol 13 to perform the 2PC in the step 1 of Protocol 12, with error tolerance $\frac{\epsilon}{3}$.
2. Consider the step 2 of Protocol 12 executed with error tolerance $\frac{\epsilon}{6}$. Suppose in this step there are L rounds of quantum communication in the form of "Bob samples keys $\tilde{x}_0, \tilde{x}_1$ and sends the states $\frac{1}{\sqrt{2}}(|0\rangle|\tilde{x}_0\rangle + |1\rangle|\tilde{x}_1\rangle)$" to Alice. (Explicitly, $\tilde{x}_1, \tilde{x}_1$ are $x_0||r_0^{(in)}||r_0^{(out)}$ and $x_1||r_1^{(in)}||r_1^{(out)}$ in the first step of Protocol 6.) Use the RSPV for this type of states (Theorem 3.13) to replace this step; the error tolerance for each such step is chosen to be $\frac{\epsilon}{6L}$.
3. Same as the step 2 above with the following differences: we compile the step 3 of Protocol 12 and the roles of Alice and Bob are swapped.

The assumptions are the existence of NTCF and collapsing hash functions; the completeness, efficiency and succinct communication property are from the protocol description and the properties of Protocol 12.

The protocol is ϵ-sound based on a similar proof to the proof of Theorem 7.7.

A On Succinct Arguments for QMA

We note that our work on SFS also leads to a new approach for constructing succinct arguments for QMA. In this section we roughly describe our approach. We do not formally work on this part so we put it in the appendix.

A.1 Background

Let's first review the notion of succinct argument in Section 3.3.5. Suppose the server claims that "I know a w such that $f(w) = 0$", where f is a public function. How efficiently could the client verify the server's claim? When f is an efficient classical function, Kilian's protocol [34] allows us to achieve this task with succinct client-side computation, that is, independent of both the function evaluation time and the witness size (size of w).

So what if the server claims a QMA statement? That is, the server claims that it holds a quantum state w such that $f(w) = 0$, where f denotes a (family of) quantum circuit. The *succinct arguments for QMA* problem seeks for a protocol for this problem where the client side computation is succinct. A closely related problem is the classical verification of quantum computations (CVQC) problem, which seeks for a verification protocol where the client-side computation and the communication should be completely classical.

The first and perhaps the most famous construction for CVQC is given by Mahadev [37]; later a series of new constructions are proposed [19, 26, 54, 40]. For the construction of succinct arguments for QMA, there are currently several existing constructions [19, 24, 8, 38, 29] and two of them are from standard assumptions [38, 29]. These succinct arguments constructions are based on (or highly related to) CVQC constructions and inherit the desirable property that the client is completely classical, which are thus called classical succinct arguments for QMA. In terms of the assumptions, [29] is based on NTCF with distributional adaptive hardcore bit property, and [38] is based on FHE and collapsing hash functions. In more detail:

- [29] constructs their succinct arguments for QMA protocol via a protocol called classical commitments to quantum states. This is a strengthening of the measurement protocol in Mahadev's CVQC work [37]. Their results are based on an assumption called NTCF with distributional strong adaptive hardcore bit property (defined in their work).
- [38] constructs their succinct arguments for QMA protocol based on the KLVY compiler [32], which is based on homomorphic encryption and nonlocal games. [40] constructs a CVQC protocol following the approach of [32]; [38] could be seen as a further step in this approach. [38] assume a mild version of quantum homomorphic encryption [38, 36, 14] and collapsing hash functions [48].

The Mahadev's CVQC construction [37] is under an assumption called the extended NTCF with the adaptive hardcore bit property. This assumption seems incomparable to both the assumptions in [29] (although both could be seen as variants of NTCF) and the assumptions in [38].

A.2 Our Approach For Classical Succinct Arguments for QMA

Here we propose a different approach for constructing classical succinct arguments for QMA. We assume the existence of extended NTCF with the adaptive hardcore bit property and collapsing hash functions, which is different from previous works [8, 38, 29] and is closer to Mahadev's CVQC

protocol. Our approach is to use our results and existing results to compile Mahadev's protocol. The compilation is relatively simple given our results and existing results.

We first give a brief review for the structure of Mahadev's protocol. The Mahadev's protocol roughly goes as follows:

1. For each $i \in [L]$, the client samples pk_i according to certain distribution and sends it to the server.
2. The server commits to its witness state using these keys and sends back the commitment (which is a classical string).
3. For each $i \in [L]$, the client randomly samples a challenge bit $c_i \in \{0, 1\}$ and sends it to the server.
4. The server does certain measurements on the committed state depending on the challenge bits and sends back the results; the client checks the results with certain relations.

Then the compilation is as follows:

1. The client could use the SFS protocol for sampling and sending $(pk_i)_{i\in[L]}$ in the first step. A PRG is used for generating the randomness.

   The calling to the SFS protocol introduces quantum communication; we could further use an RSPV [57] to compile this step to a classical-channel protocol.
2. The communication in steps 2-4 could be made succinct using the compiler in [8].

Note that existing works often consider the first step as the main obstacle for getting a succinct arguments protocol for QMA. [8, 38] In our proposal this is relatively simple given our SFS protocol and existing RSPV protocol.

References

[1] Navid Alamati, Giulio Malavolta, and Ahmadreza Rahimi. Candidate trapdoor claw-free functions from group actions with applications to quantum protocols. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part I*, page 266–293, Berlin, Heidelberg, 2022. Springer-Verlag.

[2] Alexander Poremba Alexandru Gheorghiu, Tony Merger. Quantum cryptography with classical communication: parallel remote state preparation for copy-protection, verification, and more. 2022.

[3] Anurag Anshu, Dave Touchette, Penghui Yao, and Nengkun Yu. Exponential separation of quantum communication and classical information. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 277–288, New York, NY, USA, 2017. Association for Computing Machinery.

[4] Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. *Electron. Colloquium Comput. Complex.*, TR17, 2017.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

[6] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. *IACR Cryptol. ePrint Arch.*, 2011:613, 2012.

[7] James Bartusek, Andrea Coladangelo, Dakshita Khurana, and Fermi Ma. One-way functions imply secure computation in a quantum world. *Lect. Notes Comput. Sci.*, 12825:467, 2021.

[8] James Bartusek, Yael Tauman Kalai, Alex Lombardi, Fermi Ma, Giulio Malavolta, Vinod Vaikuntanathan, Thomas Vidick, and Lisa Yang. Succinct classical verification of quantum computation. In *IACR Cryptology ePrint Archive*, 2022.

[9] James Bartusek and Dakshita Khurana. Cryptography with certified deletion. 2022. `https://eprint.iacr.org/2022/1178`.

[10] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 41–69, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[11] Zvika Brakerski, Paul Christiano, Urmila Mahadev, Umesh V. Vazirani, and Thomas Vidick. 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*, pages 320–331, 2018.

[12] Zvika Brakerski, Alexandru Gheorghiu, Gregory D. Kahanamoku-Meyer, Eitan Porat, and Thomas Vidick. Simple tests of quantumness also certify qubits. 2023.

[13] Zvika Brakerski, Venkata Koppula, Umesh Vazirani, and Thomas Vidick. Simpler proofs of quantumness, 05 2020.

[14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 97–106, Washington, DC, USA, 2011. IEEE Computer Society.

[15] Sergey Bravyi, David Gosset, and Robert König. Quantum advantage with shallow circuits. *Science*, 362:308 – 311, 2017.

[16] Anne Broadbent and Rabib Islam. Quantum encryption with certified deletion. *Lect. Notes Comput. Sci.*, 12552:92, 2020.

[17] Harry Buhrman, Nishanth Chandran, Serge Fehr, Ran Gelles, Vipul Goyal, Rafail M. Ostrovsky, and Christian Schaffner. Position-based quantum cryptography: Impossibility and constructions. In *IACR Cryptology ePrint Archive*, 2010.

[18] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004.

[19] Nai-Hui Chia, Kai-Min Chung, and Takashi Yamakawa. Classical verification of quantum computations with efficient verifier. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography*, pages 181–206, Cham, 2020. Springer International Publishing.

[20] Alessandro Chiesa, Fermi Ma, Nicholas Spooner, and Mark Zhandry. Post-quantum succinct arguments: Breaking the quantum rewinding barrier. *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 49–58, 2021.

[21] Alexandru Cojocaru, Léo Colisson, Elham Kashefi, and Petros Wallden. Qfactory: Classically-instructed remote secret qubits preparation. In Steven D. Galbraith and Shiho Moriai, editors, *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*, volume 11921 of *Lecture Notes in Computer Science*, pages 615–645. Springer, 2019.

[22] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *IACR Cryptology ePrint Archive*, 2012.

[23] Edward Eaton and Fang Song. *A Note on the Instantiability of the Quantum Random Oracle*, pages 503–523. 04 2020.

[24] Islam Faisal. Interactive oracle arguments in the qrom and applications to succinct verification of quantum computation. Cryptology ePrint Archive, Paper 2023/421, 2023. `https://eprint.iacr.org/2023/421`.

[25] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[26] Alexandru Gheorghiu and Thomas Vidick. Computationally-secure and composable remote state preparation. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1024–1033. IEEE Computer Society, 2019.

[27] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. 2001.

[28] Alex B. Grilo, Huijia Lin, Fang Song, and Vinod Vaikuntanathan. Oblivious transfer is in miniqcrypt. In *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part II*, page 531–561, Berlin, Heidelberg, 2021. Springer-Verlag.

[29] Sam Gunn, Yael Tauman Kalai, Anand Natarajan, and Agi Villanyi. Classical Commitments to Quantum States. 4 2024.

[30] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, page 163–172, New York, NY, USA, 2015. Association for Computing Machinery.

[31] Zahra Jafargholi and Daniel Wichs. Adaptive security of yao's garbled circuits. In *Theory of Cryptography Conference*, 2016.

[32] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Lisa Yang. Quantum advantage from any non-local game. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 1617–1628, New York, NY, USA, 2023. Association for Computing Machinery.

[33] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* Chapman & Hall/CRC, 2nd edition, 2014.

[34] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 723–732. ACM, 1992.

[35] Huijia Lin and Rafael Pass. Succinct garbling schemes and applications. *IACR Cryptol. ePrint Arch.*, 2014:766, 2014.

[36] Urmila Mahadev. Classical homomorphic encryption for quantum circuits. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 332–338. IEEE Computer Society, 2018.

[37] Urmila Mahadev. Classical verification of quantum computations. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 259–267. IEEE Computer Society, 2018.

[38] Tony Metger, Anand Natarajan, and Tina Zhang. Succinct Arguments for QMA from Standard Assumptions via Compiled Nonlocal Games . In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1193–1201, Los Alamitos, CA, USA, October 2024. IEEE Computer Society.

[39] Moni Naor. Bit commitment using pseudo-randomness. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, page 128–136, Berlin, Heidelberg, 1989. Springer-Verlag.

[40] Anand Natarajan and Tina Zhang. Bounding the quantum value of compiled nonlocal games: from chsh to bqp verification. 2023.

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

[42] Christopher Portmann and Renato Renner. Security in quantum cryptography. *Rev. Mod. Phys.*, 94:025008, Jun 2022.

[43] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 859–870, 2018.

[44] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009.

[45] Or Sattath. Uncloneable cryptography. *Communications of the ACM*, 66:78 – 86, 2022.

[46] Dominique Unruh. Revocable quantum timed-release encryption. *J. ACM*, 62(6), December 2015.

[47] Dominique Unruh. Collapse-binding quantum commitments without random oracles. In *Proceedings, Part II, of the 22nd International Conference on Advances in Cryptology — ASIACRYPT 2016 - Volume 10032*, page 166–195, Berlin, Heidelberg, 2016. Springer-Verlag.

[48] Dominique Unruh. Computationally binding quantum commitments. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*, page 497–527, Berlin, Heidelberg, 2016. Springer-Verlag.

[49] John Watrous. The theory of quantum information. 2018.

[50] Takashi Yamakawa and Mark Zhandry. Verifiable quantum advantage without structure. *J. ACM*, 71(3), June 2024.

[51] A. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, Oct 1986.

[52] Mark Zhandry. Quantum lightning never strikes the same state twice. or: Quantum money from cryptographic assumptions. *Journal of Cryptology*, 34, 2017.

[53] Mark Zhandry. How to construct quantum random functions. *J. ACM*, 68(5), August 2021.

[54] J. Zhang. Classical verification of quantum computations in linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, Los Alamitos, CA, USA, nov 2022. IEEE Computer Society.

[55] Jiayu Zhang. *Succinct Blind Quantum Computation Using a Random Oracle*, page 1370–1383. Association for Computing Machinery, New York, NY, USA, 2021.

[56] Jiayu Zhang. A Quantum Approach for Reducing Communications in Classical Cryptographic Primitives, 10 2023. `https://arxiv.org/abs/2310.05213v2`.

[57] Jiayu Zhang. Formulations and constructions of remote state preparation with verifiability, with applications. 2023.