# Quantum One-Time Programs, Revisited

Aparna Gupte MIT Jiahui Liu\* MIT Justin Raizes CMU

Bhaskar Roberts UC Berkeley

Vinod Vaikuntanathan MIT

November 28, 2024

#### **Abstract**

One-time programs (Goldwasser, Kalai and Rothblum, CRYPTO 2008) are programs that can be run on any single input of a user's choice, but not on a second input. Classically, they are unachievable without trusted hardware, but the destructive nature of quantum measurements seems to provide an alternate path to constructing them. Unfortunately, Broadbent, Gutoski and Stebila (CRYPTO 2013) showed that even with quantum techniques, a strong notion of one-time programs, similar to ideal obfuscation, cannot be achieved for any non-trivial quantum function. On the positive side, Ben-David and Sattath (Quantum, 2023) showed how to construct a quantum one-time program for a certain (probabilistic) digital signature scheme, under a weaker notion of one-time program security. There is a vast gap between achievable and provably impossible notions of one-time program security, and it is unclear what functionalities are one-time programmable and which are not, under the achievable notions of security.

In this work, we present new, meaningful, yet achievable definitions of one-time program security for *probabilistic* classical functions. We show how to construct one time programs satisfying these definitions for all functions in the classical oracle model and for constrained pseudorandom functions in the plain model. Finally, we examine the limits of these notions: we show a class of functions which cannot be one-time programmed in the plain model, as well as a class of functions which appears to be highly random given a single query, but whose quantum one-time program leaks the entire function even in the oracle model.

<sup>\*</sup>Part of this work done while at Fujitsu Research.

# Contents

1	Introduction 1.1 Our Results	1 2
2	Technical Overview  2.1 Definitional Works  2.2 Positive Results  2.3 Impossibility Results  2.4 Concurrent Work and Related Works	9 12
3	Preliminaries3.1 Quantum Information and Computation3.2 Quantum Query Model3.3 Compressed Random Oracles3.4 Subspace States and Direct Product Hardness3.5 Tokenized Signature Definitions	15 15 17
4	Definitions of One-Time Sampling Programs 4.1 Simulation-based Security Definitions for One-Time Sampling Programs	21 23 25
5	Single-Effective-Query Construction in the Classical Oracle Model5.1Construction	30
6	Construction in the Plain Model 6.1 Preliminaries	
7	<ul> <li>Impossibility Results in the Plain Model and the Oracle Model</li> <li>7.1 Preliminaries</li></ul>	51
8	Applications  8.1 Signature Tokens  8.2 One-Time NIZK Proofs  8.3 Future Work: One-Time MPC  8.4 One-time programs for Verifiable Functions Imply Quantum Money	56 56 58 62 62

A	Missing Proofs for Families of Single-Query Unlearnable Functions	67
	A.1 Pairwise Independent and Highly Random Functions	67
В	Security Proof for Definition 4.26 with Measure-and-Reprogram	79
	B.1 Preliminaries	79
	B.2 Security Proof	81
C	More on the Compressed Oracle	82
	C.1 Alternative Representations of the Decompression Function	82
	C.2 Compressed Oracle Chaining	82
	C.3 Knowledge of Preimage for Expanding Random Oracles	86
D	Additional Prelims	87
	D.1 NIZK	87

## 1 Introduction

The notion of one-time programs, first proposed by Goldwasser, Kalai and Rothblum [GKR08a], allows us to compile a program into one that can be run on a single input of a user's choice, but only one. If realizable, one-time programs would have wide-ranging applications in software protection, digital rights management, electronic tokens and electronic cash. Unfortunately, one-time programs immediately run into a fundamental barrier: software can be copied multiple times at will, and therefore, if it can be run on a single input of a user's choice, it can also be run on as many inputs as desired.

To circumvent this barrier, [GKR08a] designed a one-time program with the assistance of a specialized stateful hardware device that they called a *one-time memory*. A one-time memory is a device instantiated with two strings  $(s_0, s_1)$ ; it takes as input a choice bit  $b \in \{0, 1\}$ , outputs  $s_b$  and then *self-destructs*. Using one-time memory devices, Goldwasser et al. showed how to compile any program into a one-time program, assuming one-way functions exist. Goyal et al. [GIS+10] extended these results by achieving unconditional security against malicious parties and using a weaker type of one-time memories that store single bits. Notwithstanding these developments, the security of these schemes rests on shaky grounds: security relies on how much one is willing to trust the impenetrability of these hardware devices in the hands of a motivated and resourceful adversary who may be willing to mount sophisticated side-channel attacks. Which brings up the motivating question of our paper: *is there any other way to construct one-time programs*?

One might hope that the quantum no-cloning theorem [WZ82] might give us a solution. The no-cloning theorem states that quantum information cannot be generically copied, so if one can encode the given program into an appropriate quantum state, one might expect to circumvent the barrier. However, there is a simple impossibility result by Broadbent, Gutoski and Stebila [BGS13] that rules out quantum one-time versions of any *deterministic* program. Indeed, given a candidate quantum one-time program state  $|\psi_f\rangle$ , an adversary can evaluate f many times on different inputs as follows: it first evaluates the program on some input x, measures the output register to obtain f(x). Since f is deterministic, the measurement does not disturb the state of the program at all (if the computation is perfectly correct). The adversary then uncomputes the first evaluation, restoring the initial program state. She can repeat this process on as many inputs as she wishes.

While this impossibility result rules out one-time programs for deterministic functionalities, it raises the following natural question:

Can we obtain one-time programs for randomized functionalities?

More concretely, can we construct quantum one-time programs for randomized functions  $f: \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$  that lets the user choose the input  $x \in \mathcal{X}$  but not the randomness  $r \in \mathcal{R}$ ? One might hope that by forcing the evaluation procedure to utilize the *inherent randomness* of quantum information in sampling  $r \leftarrow \mathcal{R}$ , measuring the output would collapse the program state in a way that does not allow further evaluations. However, once again, [BGS13] showed that it is impossible to compile *any quantum channel* into a one-time program, unless it is (essentially) learnable with just one query. This is a much more general impossibility; in fact, it rules out non-trivial one-time programs for classical randomized functions. (We refer the reader to Section 2 for a description of this impossibility result.)

On the other hand, more recently, Ben-David and Sattath [BDS23] demonstrated the first instance of a one-time program for a certain randomized function. In particular, they construct a

digital signature scheme where the (randomized) signing procedure can be compiled into a onetime program that the user can use to generate a single signature for a message of her choice.

At a first glance, this positive result might seem like a contradiction to the [BGS13] impossibility; however, that is not so, and the difference lies in which definition of one-time programs one achieves. Ben-David and Sattath [BDS23] achieve a much weaker notion of one-time security than what was proven to be impossible by [BGS13]. On the one hand, [BGS13] demanded that an adversarial user should not be able to do *anything* other than evaluate the one-time program on a single input, an ideal obfuscation-like guarantee [Had00, BGI+01]. On the other hand, the positive result of [BDS23] only claimed security in the sense that an adversarial user cannot output two different valid signatures.

The starting point of this paper is that there is a vast gap between these two security notions. Within the gap, one could imagine several meaningful and useful intermediate notions of quantum one-time programs for classical randomized functions. For example, strengthening the [BDS23] definition, one could imagine requiring that the user should not even be able to verify the correctness of two input-output pairs (and not just be unable to produce them). Such a definition is a meaningful strengthening in the context of indistinguishability games (such as in pseudorandom functions) rather than unpredictability games (such as in digital signatures). One could also imagine realizing one-time programs for a wider class of functions than the signature tokens of [BDS23].

In this work, we revisit notions of quantum one-time programs and make progress on these questions. We propose a number of security notions of quantum one-time programs for randomized functions; give constructions both in the plain model and a classical oracle model; and examine the limits of these notions by showing negative results. We next describe our contributions in more detail.

#### 1.1 Our Results

**Definitions.** Our first contribution is definitional. We give correctness and security definitions for one-time programs of classical randomized circuits, which we call one-time sampling programs.

For correctness of a one-time sampling program for a classical f, any honest user can choose its own input x and the evaluation gives f(x;r) for some random r. For security, we lay out a list of different notions of security that we might desire from the one-time sampling program.

We make a few attempts on a simulation-based definition: the desired one-time sampling program functionality should be indistinguishable from an idealized functionality, where we are allowed to make a single quantum query to a "randomized oracle" for the target functionality f. However, these definitions run into several strong impossibility results, unless assuming hardware assumptions.

We therefore explore a possible weakening on the single quantum query access we allow in the ideal world. Inspired by the compressed oracle technique in [Zha19a] used to record queries for quantum random oracles, we re-define the single-query access oracle in the ideal world. Very informally, the randomized oracle would record queries so that it allows only one "informative" query to be made, but potentially many more dummy queries. We then give a new simulation-based definition based on this oracle we call single-*effective*-query oracle<sup>1</sup>, which allows us to by-

<sup>&</sup>lt;sup>1</sup>We refer to the traditional single query oracle which allows literally one query as single-*physical*-query oracle.

pass the above impossibility results.

We additionally introduce a weaker but highly useful security definition called operational definition, in which the adversary cannot "evaluate" twice given a one-time program<sup>2</sup>.

Constructions and Positive Results. We give a very generic construction for one-time sampling programs in the classical oracle model<sup>3</sup>, inspired by the one-time signature scheme in [BDS23]. We allow an honest user to choose its own input and then generate a random string by measuring a "signature token" state. The evaluation is on the user's input together with this freshly generated randomness.

In particular, an honest evaluator does not need to run a classical circuit coherently on a quantum input, but only needs quantum memory and one measurement to evaluate the program in our construction. But an adversary will likely need the power of evaluating large-depth classical circuits coherently on quantum states.

We prove its security under the single-effective-query simulation-based definition.

**Theorem 1.1.** (Informal) There exists a secure one-time sampling program for all functions (with sufficiently long randomness) in the classical oracle model, with respect to our simulation-based, single-effective-query model one-time sampling security.

We also instantiate the classical oracle using indistinguishability obfuscation, to get a compiler in the plain model, and prove its security for the class of pseudorandom functions under an operational security definition for cryptographic functionalities.

**Theorem 1.2** (Informal). Assuming post-quantum iO and LWE (or alternatively subexponentially secure iO and OWFs), there exists one-time sampling programs for constrained PRFs.

**Impossibilities.** To complement our constructions in the classical oracle model and the plain model, we also give two new negative results. The first negative result shows we cannot hope to one-time program *all randomized* functionalities in the plain model, even under the weakest possible operational security definitions. This impossibility is inspired by the work of [AP21, ABDS20]. We tweak the idea to work with randomized circuits that can only be evaluated once.

**Theorem 1.3** (Informal). Assuming LWE and quantum FHE, there exists a family of circuits with high min-entropy outputs but no secure one-time sampling programs exist for them.

We also show that having high min-entropy outputs is not a sufficient condition to have a secure one-time programs. Our second impossibility result shows that there exists a family of randomized functions with high min-entropy and is unlearnable under a single physical query. But it cannot be one-time programmed even in the classical oracle model, even under the weakest possible operational security definitions<sup>4</sup>.

We demonstrate the definitions presented in this work and their corresponding impossibilities and/or constructions in Figure 1. We recommend the readers to come back to this figure after going through the technical overview.

<sup>&</sup>lt;sup>2</sup>Throughout the work, we may use the terms "one-time programs" and "one-time sampling programs" interchangeably. But they both refer to one-time sampling programs unless otherwise specified.

<sup>&</sup>lt;sup>3</sup>A classical oracle is a classical circuit that can be accessed coherently by quantum users in a black-box manner.

<sup>&</sup>lt;sup>4</sup>This function is securely one-time programmable under the single-effective-query simulation-based definition, but in a "meaningless" sense since both the simulator and the real-world adversary can fully learn the functionality. This demonstrates the separations and relationships between several of our definitions.

Definition	Impossibilities	Construction
Single physical query,	Strong impossibility	For single physical
quantum-output, simulation-based	in oracle model	query learnable
(Definition 4.4)	(Section 2.1)	(trivial) functions only
		[BGS13]
Single physical query,	Impossibility for generic	N/A
classical-output, simulation-based	construction in oracle model	
(Definition 4.6)	(Section 2.3, Section 7.3)	
Single effective query	Impossibility for generic	For all functions
quantum-output,	constructions in plain	(with proper randomness
simulation-based (Definition 4.8)	model (Section 2.3, Section 7)	length) in classical oracle model
Operational definitions	Impossibility for generic	For random functions
(Section 4.4)	constructions in plain	in classical oracle model;
	model (Section 2.3, Section 7)	For constrained PRF
		in plain model

**Figure 1:** Definitions with Impossibilities and Constructions. The exact impossibility results and positive results for operational definitions depend on which definition of single-query model we work with. See Section 4.4, and Section 7 for details.

**Applications.** Using the techniques we developed for one-time programs, we construct the following one-time cryptographic primitives:

- One-Time Signatures. We compile a wide class of existing signature schemes to add signature tokens, which allow a delegated party to sign exactly one message of their choice. Notably, our construction only changes the signing process while leaving the verification almost unmodified, unlike [BDS23]'s construction. Thus, it enables signature tokens for *existing* schemes with keys which are already distributed.
- **One-Time NIZK Proofs.** We show how a proving authority can delegate to a subsidiary the ability to non-interactively prove a single (true) statement in zero-knowledge.
- **Public-Key Quantum Money.** We show that one-time programs satisfying a mild notion of security imply public-key quantum money.

**Future Work** For future work, some possible directions and applications may be one-time MPC, one-time encryptions with advanced security. As discussed in a concurrent work [GM24], another interesting application is one-time tokens for large language models.

#### 2 Technical Overview

#### 2.1 Definitional Works

**First Attempt at Defining One-Time Sampling Programs.** As we discussed in the introduction, we cannot achieve one-time security for deterministic classical functions without hardware assumptions, even after encoding them into quantum states: by applying the gentle measurement

lemma [Aar04, Win99], any adversary can repair the program state after a measurement on the program's output that gives a deterministic outcome.

We therefore resort to considering classical randomized computation, which we model as the following procedure: the user (adversary) can pick its own input x; the program samples a random string r for the user and outputs the evaluation f(x,r) for the user, for some deterministic function f. Note that it's essential that the user does not get to pick their own randomness r – otherwise the evaluation is deterministic again and is subject to the above attack.

For correctness, we need to guarantee that after an honest evaluation, the user gets the outcome f(x,r) for its own choice of x and a uniformly random r. For security, the hope is that when the output of f looks "random enough" (e.g. f is a hash function or a pseudorandom function), the adversary should not be able to do more than evaluating the program honestly once. We discuss several candidate definitions, the corresponding impossibility (no-go) results as well as our solutions that circumvent the impossibilities.

Ideally, we would establish a *simulation-based* security definition. This might require the existence of a QPT algorithm Sim which can produce the adversary's real-world view given a single query to f:

$$\mathsf{OTP}(f) \approx \mathsf{Sim}^{f_1}$$

where  $f_1$  denotes that Sim may query f a single time. Indeed, such a definition is formalized, and subsequently ruled out, by Broadbent, Gutoski, and Stebila [BGS13].

This definition can be adapted to sampling programs by considering sampling f from a function family  $\mathcal{F}$  at the start of the experiment. Additionally, to prevent a trivial definition which can be satisfied by Sim choosing its own f, the distinguisher gets access to the sampled f:

$$\{f, \mathsf{OTP}(f)\}_{f \leftarrow \mathcal{F}} \approx \{f, \mathsf{Sim}^{f_1}\}_{f \leftarrow \mathcal{F}}$$

Unfortunately, this candidate definition suffers from impossibility results of its own.

Impossibility Results for the "Traditional" Simulation-Based Definitions. As often applicable to simulation-based definitions, the first impossibility results from the separation between the simulated oracle world and a plain model where the one-time program can be accessed in a possibly non-black-box way. Our one-time program we give to  $\mathcal{A}$  consists of plain-model circuits (actual code) and quantum states, instead of oracle circuits. Our simulator is given only oracle access to f. Once  $\mathcal{A}$  has non-black-box access to the given one-time program,  $\mathcal{A}$  may be able to perform various attacks that the simulator cannot do: for instance, homomorphic evaluation on the one-time program. Then one can show that there exists a family of circuits, even though "unlearnable" when only given query access, can always be fully learned (i.e. the adversary can fully recover the functionality) once given non-black-box access. As demonstrated in [ABDS20, AP21], this type of non-black-box attacks are applicable even if the obfuscation program is a quantum state.

One may wonder if we can simply use the above impossibility result above for quantum VBB directly as an impossibility result for the one-time program in the plain model. However, there are subtleties we need to deal with: the circuit in the quantum obfuscated program in the above results ([ABDS20, AP21]) is deterministic, which will give a trivial impossibility result in the one-time program setting, irrelevant to non-black-box access. Moreover, the adversary receiving the one-time program is only able to evaluate once and the program may get destroyed. In our case, we need a sampling program with high min-entropy outputs where one can still apply a non-black-box attack with one single evaluation. We design a slightly contrived secret-key encryption

circuit that leads to the impossibility result in the plain model – we will elaborate its details in a later paragraph 2.3 and formally in Section 7. For now, let us proceed with the discussion on the definitions.

**Barriers for Stateless One-Time Programs.** Even more problematic, the above definition encounters impossibilities even in the *oracle* model, where we ensure that the program received by  $\mathcal{A}$  consists of oracle-aided circuits, preventing the non-black-box attack described earlier from applying.

This limitation primarily arises from the fact that Sim is given a stateful oracle, while  $\mathcal{A}$  is provided with a stateless one-time program (which includes a stateless oracle). To illustrate, consider the following  $\mathcal{A}$  and distinguisher  $\mathcal{D}$ :  $\mathcal{A}$  receives a possibly oracle-aided program and simply passes the program itself to  $\mathcal{D}$ . Let  $\mathcal{O}_f$  be a stateless oracle for f that outputs g = f(x, r) on any input f(x, r) and not restricted in the number of queries that it can answer. f(x, r) is given arbitrary oracle access to f(x, r) or f(x, r) and perform the following attack using gentle measurement (Lemma 3.2) and un-computation:

- 1. Evaluate the program given by  $\mathcal{A}$  on a  $|x_1\rangle_{\mathsf{inp}}|0\rangle_{\mathsf{out}}|0\rangle_{\mathsf{check}}$  where the input register inp contains  $x_1, r_1$ , some arbitrary  $x_1$  of  $\mathcal{D}$ 's choice and some randomness  $r_1$  sampled by the program. out is an output register and check is an additional register in  $\mathcal{D}$ 's memory.
- 2. Get outcome  $|x_1\rangle_{\text{inp}}|r_1,y_1=f(x_1,r_1)\rangle_{\text{out}}|0\rangle_{\text{check}}$ .
- 3. But  $\mathcal{D}$  does not proceed to measure the register out. Instead it performs a gentle measurement by checking if the  $y_1$  value in out is equal to the correct  $y_1 = f(x_1, r_1)$ , writing the outcome in register check. It can do so because it has access to  $\mathcal{O}_f$ . Then it measures the bit in register check.
- 4. Since the above measurement gives outcome 1 with probability 1,  $\mathcal{D}$  can uncompute the above results and make sure that the program state is undisturbed. Then, it can evaluate the program again on some different  $x_2$  of its choice.
- 5. But when given a simulator's output, Sim cannot produce a program that contains more information than what's given in a single quantum oracle access to f. Therefore, unless Sim can "learn" f in a single quantum query and produce a program that performs very closely to a real program on most inputs,  $\mathcal{D}$  may easily detect the difference between two worlds.

The above argument is formalized in [BGS13], which rules out stateless one-time programs for quantum channels even in the oracle model unless the function can be learned in a single query (for example, a constant function). The above simulation-based definition discussed can be viewed as a subcase of [BGS13]'s definition. Only in this trivial case, Sim can fully recover the functionality of f and make up a program that looks like a real world program, since both Sim and  $\mathcal{A}$  can learn everything about f.

To get around the above oracle-model attack, we first consider the following weakening: what if we limit both the adversary and simulator to output only *classical information*? Intuitively, this requires both  $\mathcal{A}$  and Sim to dequantize and "compress" what they can learn from the program/oracle into a piece of classical information, so that  $\mathcal{A}$  cannot output the entire functionality unless it has "learned" the classical description of the functionality. However, we will show in 2.3 that there even exists a function with high min-entropy output such that its classical description can be "learned" given any stateless, oracle-based one-time sampling program, but is unlearnable given only a single query. Thus we will need to explore other avenues

These impossibility results appear to stem more from a definitional limitation than a fundamental obstacle. The adversary is always given a stateless program, but the oracle given to the simulator is by definition strongly stateful: it shuts down after answering any single query (we call such an oracle single *physical* query oracle). Therefore, Sim is more restricted than the  $\mathcal A$  in real world.

The Single-Effective-Query Model. To avoid the above issue, we consider a weakening on the restriction of the "single query" which Sim can make. In the traditional one-time security, Sim can merely make one physical query, but  $\mathcal{A}$  and  $\mathcal{D}$  can actually make many queries, as long as the measurements on those queries they make are "gentle" (for example, a query where the outcome f(x,r) is unmeasured and later uncomputed) or repeated (for example, two classical queries on the same (x,r)).

In the single-*effective*-query model, we relax Sim's single-physical-query restriction to also allow multiple queries, as long as they are "gentle" or repeated. We will define a stateful oracle  $f_{\sf SEQ}$  which tracks at all times which evaluations f(x;r) the adversary has knowledge about. If  $f_{\sf SEQ}$  receives a query to some x' while it knows the adversary has knowledge about an evaluation on  $x \neq x'$ , it will refuse to answer. Using this oracle, we may define single-effective query simulation-security in the same manner as our previous attempt by giving the simulator access to the single-effective-query oracle  $f_{\sf SEQ}$  instead of the single-physical-query oracle  $f_1$ :

$$\mathsf{OTP}(f) \approx \mathsf{Sim}^{f_{\mathsf{SEQ}}}$$

The reader may be concerned that since f is not sampled from any distribution here, this definition is subject to the previously discussed impossibility for deterministic functions. As we will see shortly, the randomization of f is directly baked in to the definition of f<sub>SEQ</sub>.

**Defining the Single-Effective-Query Oracle.** To define the single-effective query oracle  $f_{SEQ}$ , we use techniques from compressed random oracles, which were introduced by Zhandry to analyze analyze security in the quantum-accessible random oracle model (QROM) [Zha19a]. Very roughly speaking, a compressed oracle gives an efficient method of simulating quantum query access to a random oracle on the fly by lazily sampling responses in superposition which can be "forgotten" as necessary.

The first main idea in [Zha19a] compressed oracle technique is to take a purified view on the joint view of the adversary's query register and the oracle: evaluating a random function in the adversary's view is equivalent to evaluating on some function H from a uniform superposition over all functions (of corresponding input and output length)  $\sum_{H} |H\rangle_{\mathcal{H}}$ . When adversary makes a query of the form  $\sum_{x,u} \alpha_{x,u} |x,u\rangle$ , the oracle applies the operation

$$\sum_{x,u} \alpha_{x,u} |x,u\rangle \otimes \sum_{H} |H\rangle \Rightarrow \sum_{x,u,H} \alpha_{x,u} |x,u+H(x)\rangle \otimes |H\rangle$$

Zhandry's second contribution is a method to "compress" the exponentially large superposition into a small database. It will be instructive to first consider the uncompressed version, so we defer details about the compressed version to later.

To define  $f_{SEQ}$ , we allow it to maintain a purified version of H, which it represents as a truth table. In other words, it maintains a register  $\mathcal{H} = (\mathcal{H}_x)_{x \in \mathcal{X}}$  which is initialized to

$$|H_{\emptyset}\rangle_{\mathcal{H}} \coloneqq \sum_{H:\mathcal{X} \to \mathcal{R}} \bigotimes_{x \in X} |H(x)\rangle_{\mathcal{H}_x} = \bigotimes_{x \in X} \sum_{r \in \mathcal{R}} |r\rangle_{\mathcal{H}_x}$$

When  $f_{\mathsf{SEQ}}$  decides to answer a query  $|x,u\rangle$ , it computes  $|x,u\oplus f(x;H(x))\rangle$  by reading register  $\mathcal{H}_x$  in the computational basis. The first query made results in the joint state

$$|x^*,u\rangle_{\mathcal{Q}}\otimes|H_{\emptyset}\rangle\stackrel{U_{f_{\S}}}{\longrightarrow}\sum_{r}|x^*,u\oplus f(x^*,r)\rangle_{\mathcal{Q}}\otimes|r\rangle_{\mathcal{H}_{x^*}}\otimes\sum_{H:H(x^*)=r}\bigotimes_{x\in\mathcal{X},x\neq x^*}|H(x)\rangle_{\mathcal{H}_{x}}$$

If f(x;r) were to uniquely determine r, then measuring  $f(x^*,H(x^*))$  would fully collapse register  $\mathcal{H}_{x^*}$  while leaving the others untouched. Afterwards, the single-effective-query oracle  $f_{\mathsf{SEQ}}$  could detect which input was evaluated and measured by comparing each register  $\mathcal{H}_x$  to the uniform superposition  $\sum_{r\in\mathcal{R}}|r\rangle$ . It could then use this information to decide whether to answer further queries. On the other hand, if there were many collisions  $f(x^*;r^*)=f(x^*;r_2^*)$  or the adversary erased its knowledge of  $f(x^*;r^*)$  by querying on the same register again, then  $\mathcal{H}_{x^*}$  might not be fully collapsed. In this case, it is actually beneficial that  $f_{\mathsf{SEQ}}$  does not completely consider  $x^*$  to have been queried, since this represents a "gentle" query which would allow the adversary to continue evaluating a real one-time program.

When we switch to the compressed version of H, collapsing  $\mathcal{H}_{x^*}$  to  $|r^*\rangle$  corresponds to recording  $(x^*, r^*)$  in a database D. Since the adversary's queries may be in superposition, the database register  $\mathcal{H}$  may become entangled with the adversary. In other words, the general state of the system is  $\sum_{a,D} \alpha_{a,D} |a\rangle_{\mathcal{A}} \otimes |D\rangle_{\mathcal{H}}$  where  $\mathcal{A}$  belongs to the adversary and  $\mathcal{H}$  belongs to  $f_{\text{SEQ}}$ . Using this view,  $f_{\text{SEQ}}$  may directly read the currently recorded query off of its database register to decide whether to answer a new query. The entanglement between the adversary's register and the database register enables  $f_{\text{SEQ}}$  to answer or reject new queries x precisely when the adversary does not have another outstanding query x'. As a result, the database register will always contain databases with at most one entry.

**Functions for which SEQ Access is Meaningful.** The single-effective-query simulation definition captures all functions, including those that are trivially one-time programmable. Similarly to the notion of ideal or virtual black-box obfuscation, any "unlearnability" properties depend on the interaction of the function with the obfuscation definition. For example, deterministic functions can be fully learned given access to an SEQ oracle, since measuring evaluations will never restrict further queries.

Intuitively, a function must satisfy two loose properties in order to have any notion of unlearnability with SEQ access:

- **High Randomness.** To restrict further queries, learning (via measuring) f(x;r) must collapse the SEQ oracle's internal state, causing (x,r) to be recorded in the purified oracle H.
- **Unforgeability.** To have any hope that f has properties that cannot be learned with SEQ access, f cannot be learnable given, say, a single evaluation f(x; r).

As an example, truly random functions exemplify both of these properties. A truly random function has maximal randomness on every input and f(x;r) is independent of f(x';r'). We formally explore SEQ access to truly random functions and a few other function families in Section 5.3.

#### 2.2 Positive Results

**Construction with Classical Oracles.** We now give an overview on our construction using classical oracles. We show security with respect to the simulation-based definition where the simulator queries a single-effective-query oracle.

Our construction in the oracle model is inspired by the use of the "hidden subspace states" in the literatures of quantum money [AC12], signature tokens and quantum copy protection [BDS23, CLLZ21a]. A subspace state  $|A\rangle$  is a uniform superposition over all vectors in some randomly chosen, secret subspace  $A \subset \mathbb{F}_2^{\lambda}$ . Specifically,  $|A\rangle \propto \sum_{v \in A} |v\rangle$ , where dimension of A is  $\lambda/2$  and  $\lambda$  is the security parameter. These parameters ensure that A has exponentially many elements but is still exponentially small compared to the entire space.

At a high-level, our one-time scheme requires an authorized user to query an oracle on subspace vectors of A or its dual subspace  $A^{\perp}$ . Let f be the function we want to one-time program. Consider the simple case where x is a single bit in  $\{0,1\}$ . Let G be a PRG or extractor (which can be modeled as a random oracle since we already work in oracle model). The one-time program consists of a copy of the subspace state  $|A\rangle$  along with access to the following classical oracle:

$$\mathcal{O}(x,v) = \begin{cases} f(x,G(v)) & \text{if } x = 0, v \in A \\ f(x,G(v)) & \text{if } x = 1, v \in A^{\perp} \\ \bot & \text{otherwise} \end{cases}$$

To evaluate on input x, an honest user will measure the state  $|A\rangle$  to obtain a uniform random vector in subspace A, if x=0; or apply a binary QFT to  $|A\rangle$  and measure to obtain a uniform random vector in the dual subspace  $A^{\perp}$ , if x=1. It then inputs (x,v) into the oracle  $\mathcal O$  and will obtain the evaluation  $\mathcal O(x,G(v))$  where the randomness G(v) is uniformly random after putting the subspace vector into the random oracle.

For security, we leverage an "unclonability" property of the state  $|A\rangle$  ([BDS23, BKNY23]) called "direct-product hardness": an adversary, given one copy of  $|A\rangle$ , polynomially bounded in query to the above oracle should not be able to produce two vectors v, v' which satisfy either of the following: (1)  $v \in A, v' \in A^{\perp}$ ; (2)  $v, v' \in A$  or  $v, v' \in A^{\perp}$  but  $v \neq v'$ .

First, we consider a simpler scenario: this evaluation is destructive to the subspace state if the user has obtained the outcome f(x,G(v)) and the function f behaves random enough so that measuring the output f(x,G(v)) is (computationally) equivalent to having measured the subspace state v. Now, it will be hard for the user to make a second query into the oracle  $\mathcal O$  on a different input (x',v') so that either  $x\neq x'$  or  $v\neq v'$  because it would lead to breaking the direct-product hardness property mentioned above.

More generally, however, f may be only somewhat random, or the adversary may perform superposition queries. In these cases,  $|A\rangle$  will be only partially collapsed in the real program, potentially allowing further queries. This partial collapse also corresponds to a partial collapse of the single-effective-query oracle's database register, similarly restricting further queries.

To establish security, the main gap that the simulator needs to bridge is the usage of a subspace state  $|A\rangle$  versus a purified random oracle to control query access. Additionally, the real world evaluates f(x,G(v)), where v is a subspace vector corresponding to x, while the ideal world evaluates f(x,H(x)) directly.<sup>5</sup> If we were to purify G as a compressed oracle, then  $|A\rangle$  collapsing corresponds to G recording some subspace vector v in its database. At a high level, this allows the

 $<sup>^{5}</sup>$ Although H and G are both random oracles, we differentiate them to emphasize that they act on different domains.

simulator to bridge the aforementioned gap by using a careful caching routine to ensure that  $|A\rangle$  collapses/v is recorded in the cache if and only if x is recorded in H. Using the direct product hardness property, we can be confident that at most one v and corresponding x are recorded in the simulator. Thus, to show that the simulator is indistinguishable from the real one-time program, we can simply swap the role of x and v in the oracle, changing between x and y in the oracle, changing between x and y in the oracle, changing between x and y in the oracle, changing between y and y in the oracle, y is the oracle of y and y in the oracle or y in

**Operational Security for Cryptographic Functionalities.** While the classical oracle construction is clean, implementing the oracle itself with concrete code will bump into the plain model versus black-box obfuscation barrier again. We cannot hope to make one-time sampling programs for all functions (not even for all high min-entropy output functions) in the plain model, due to the counter-example we provide in the first paragraph of 2.3 we will soon come to. Moreover, the simulation definition we achieve above captures functions all the way from those that can be meaningfully one-time programmed, like a random function, to those "meaningless" one-time programs of for example a constant function, which can be learned in a single query.

One may wonder, what are some meaningful functionalities we can implement a one-time program with and what are some possible security notions we can realize for them in the plain model?

We consider a series of relaxed security notions we call "operational one-time security definition", which can be implied by the simulation-based definition. The intuition of these security definitions is to characterize "no QPT adversary can evaluate the program twice".

Consider a cryptographic functionality f, we define the security game as follows: the QPT adversary  $\mathcal{A}$  receives a one-time program for f and will output its own choice of two input-randomness pairs  $(x_1, r_1), (x_2, r_2)$ . For each  $(x_i, r_i)$ ,  $\mathcal{A}$  needs to answer some challenges from the challenger with respect to the cryptographic functionality f. The security guarantees that  $\mathcal{A}$ 's probability of winning both challenges for  $i \in \{1, 2\}$  is upper bounded by its advantage in a cryptographic security game of winning a single such challenge, but without having acess to the one-time program.

For some cryptographic functionalities, this cryptographic challenge is simply to compute  $y_i = f(x_i, r_i)$ . A good example is a one-time signature scheme:  $\mathcal{A}$  produces two messages of its own choice, but it should not be able to produce valid signatures for both of them with non-negligible probability.

More generically,  $\mathcal{A}$  produces two message-randomness pairs and gives them to the challenger. Then the challenger prepares some challenges independently for  $i \in \{1,2\}$  and  $\mathcal{A}$  has to provide answers so that  $\mathcal{A}$ 's inputs, the challenges and final answer need to satisfy a predicate. In the above signature example, the predicate is simply verifying if the signature is a valid one for the message. Another slightly more contrived predicate is answering challenges for a pseudorandomness game of a PRF: receiving an OTP for a PRF, no QPT adversary can produce two input-randomness pairs  $(x_1, r_1), (x_2, r_2)$  of its own choice such that it can win the pseudorandomness game with respect to both of these inputs. That is, the challenger will flip two independent uniform bits for each  $i \in \{1,2\}$  to decide whether let  $y_i = \mathsf{PRF}(x_i, r_i)$  or let  $y_i$  be real random. The security says that  $\mathcal{A}$ 's overall advantage should not be noticeably larger than 1/2;  $\mathcal{A}$  can always evaluate once and get to answer one of the challenges with probability 1, but for the other challenge it can only make a random guess.

One-Time Program for PRFs in the Plain Model. However, one cannot hope to achieve a one-time program construction that is secure for all functions in the *plain model*, even if we consider the most restrict ourselves to the weakest operational definition and high min-entropy output functions. As aforementioned, we give the counter-example of such a circuit (assuming some mild computational assumptions) in the next paragraph 2.3.

We therefore turn to considering constructions for specific functionalities in the plain model and a give a secure construction for a family of PRFs, with respect to the aforentioned security guarantee: no QPT adversary can produce two input-randomness pairs  $(x_1, r_1), (x_2, r_2)$  of its own choice such that it can win the pseudorandomness game with respect to both of these inputs.

To replace the oracle in the above construction, we use iO (which stands for indistinguishability obfuscation, [BGI+01]) which guarantees that the obfuscations of two functionally-equivalent circuits should be indistinguishable.

The construction in the plain model bears similarities to the one in the oracle model. Let  $\mathsf{PRF}_k(\cdot)$  be the PRF as our major functionality. In our actual construction, we use another PRF G on the subspace vector v to extract randomness, but we omit it here for clarity of presentation. We give out a subspace state  $|A\rangle$  and an iO of a program.

The following program is a simplification of the actual program we put into iO:

$$\mathsf{PRF}_{k,A}(x,v) = \begin{cases} \mathsf{PRF}_k(x,v) & \text{if } x = 0, v \in A \\ \mathsf{PRF}_k(x,v) & \text{if } x = 1, v \in A^\perp \ . \\ \bot & \text{otherwise} \end{cases}.$$

To show security, we utilize a constrained PRF ([BKW17]): a constrained PRF key  $k_C$  constrained to a circuit C will allow us to evaluate on inputs x that satisfy C(x) = 1 and output  $\bot$  on the inputs that don't satisfy. The constrained pseudorandomness security guarantees that the adversary should not be able distinguish between  $(x, \mathsf{PRF}_k(x))$  and  $(x, y \leftarrow \mathsf{random})$ , where  $\mathcal A$  can choose x such that C(x) = 0 and x is the un-constrained key.

In our proof, we can use hybrid arguments to show that we can use the adversary to violate the constrained pseudorandomness security. Let us denote  $A^0=A, A^1=A^\perp$ . First we invoke the security of iO: we change the above program to one using a constrained key  $k_C$  to evaluate PRF, where C(x,v)=1 if and only if  $v\in A^x$ . The circuit has the equivalent functionality as the original one. Next, we invoke a computational version of subspace state direct-product hardness property: we change the one-time security game to rejecting all adversary's chosen inputs  $(x_1,v_1),(x_2,v_2)$  such that both  $v_1\in A^{x_1}$  and  $v_2\in A^{x_2}$  hold. Such a rejection only happens with negligible probability due to the direct-product hardness property. Finally, the adversary must be able to produce some (x,v) where C(x,v)=0 and distinguish PRF $_k(x,v)$  from a random value. We therefore use it to break the constrained pseudorandomness security.

More Applications and Implications: Generic Signature Tokens, NIZK, Quantum Money We show a way to lift signature schemes that satisfy a property called blind unforgeability to possess one-time security. Unlike the [BDS23] signature token scheme where the verification key has to be updated once we delegate a one-time signing token to some delegatee, our signature token scheme can use an existing public verification procedure.

Apart from the above one-time PRF in the plain model, we also instantiate one-time NIZK from iO and LWE in the plain model, using a similar construction as in the one-time PRF and the

NIZK from iO in [SW14]. The proof requires more careful handling because the NIZK proof also has publicly-verifiable property.

Finally, we show that one-time program for publicly verifiable programs (e.g. signature, NIZK) implies public-key quantum money. Despite the destructible nature of the one-time program, we can design a public verification procedure that gently tests a program's capability of computing a function and use a one-time program token state as the banknote.

#### 2.3 Impossibility Results

Impossibility Result in the Plain Model. In this paragraph, we come back to the discussions about impossibility results in the paragraph "Barriers for stateless one-time programs" (2.1). We describe the high level idea on constructing the program used to show an impossibility result in the plain model, inspired by the approach in [AP21, ABDS20]. This impossibility holds *even for the weakest definition* we consider:  $\mathcal{A}$  is not able to produce two input-output pairs after getting one copy of the OTP.

We have provided a table in Figure 1 in Section 1.1 that demonstrates the several security definitions we discuss in this work, their corresponding impossibility results and positive results. Please refer to the table so that the relationships between the several definitions proposed in this work are clearer.

We design an encryption circuit C with a random "hidden point" such that when having non-black-box access to the circuit, one can "extract" this hidden point using a quantum fully homomorphic encryption on the one-time program. However, when having only oracle access, one cannot find this hidden point within any polynomial queries.

Let SKE be a secret key encryption scheme. Let a, b be two randomly chosen values in  $\{0, 1\}^n$ .

```
Input: (x \in \{0,1\}^n, r \in \{0,1\}^\ell)
Hardcoded: (a,b, \mathsf{Enc.sk})
if x = a: output SKE.Enc(Enc.sk, b; r)
else: output SKE.Enc(Enc.sk, x; r).
```

The above circuit also comes with some classical auxiliary information, given directly to  $\mathcal A$  in the real world (and Sim in the simulated world, apart from giving oracle access). Let QHE be a quantum fully homomorphic encryption with semantic security. We also need a compute-and-compare obfuscation program (which can be built from LWE). A compute-and-compare obfuscation program is obfuscation of a circuit  $\mathsf{CC}[f,m,y]$  that does the following: on input x, checks if f(x)=y; if so, output secret message m, else output  $\bot$ . The obfuscation security guarantees that when y has a high entropy in the view of the adversary, the program is indistinguishable from a dummy program always outputing  $\bot$  (a distributional generalization of a point function obfuscation).

In the auxiliary information, we give out  $\operatorname{ct}_a = \operatorname{QHE.Enc}(a)$  and the encryption key, evaluation key QHE.pk for the QFHE scheme. We also give the compute-and-compare obfuscation for the following program:

$$\mathsf{CC}[f, (\mathsf{SKE.sk}, \mathsf{QHE.sk}), b] = \begin{cases} (\mathsf{SKE.sk}, \mathsf{QHE.sk}) & \text{if } f(x) = b \\ \bot & \text{otherwise} \end{cases}$$

where  $f(x) = \mathsf{SKE.Dec}(\mathsf{SKE.sk}, \mathsf{QHE.Dec}(\mathsf{QHE.sk}, x))$ . Any adversary with non-black-box access to the program can homomorphically encrypt the program and then evaluate to get a QFHE encryption of a ciphertext  $\mathsf{SKE.Enc}(b)$  (doubly encrypted by  $\mathsf{SKE}$  and then  $\mathsf{QHE}$ ). This ciphertext,

once put into the above CC obfuscation program, will give out all the secrets we need to recover the functionality of the circuit C.

However, when only given oracle access, we can invoke the obfuscation security of the computeand-compare program and the semantic security of QFHE, finally removing the information of bcompletely from the oracle, rendering the functionality as a regular SKE encryption sheme (which behaves like a random oracle in the classical oracle model). The actual proof is more intricate, involving a combination of hybrid arguments, quantum query lower bounds and induction, since the secret information on a, b is scattered around in the auxiliary input. We direct readers to Section 7 for details.

**Impossibility Result in the Oracle Model.** In this section, we show a circuit family with highentropy output which cannot be one-time programmed in the oracle model, with respect to the classical-output simulator definition discussed in Section 2.1. It can be one-time programmed with respect to our single-effective-query simulator definition, but only in a "meaningless" sense, since both the simulator and the real-world adversary can fully learn the functionality.

In short, it demonstrates several separations: (1) It separates a single-physical-query unlearnable function from single-effective-query unlearnable: one can fully recover the functionality once having a single effective query oracle, but one cannot output two input-output pairs when having only one physical query. (2) It is a single-physical-query unlearnable function that cannot be securely one-time programmed with respect to the operational definition where we require the adversary to output two different correct evaluations. (3) Having high min-entropy output distributions is not sufficient to prevent the adversary from evaluating twice (i.e. "meaningfully" one-time programmed). We would also like to make a note that this result does not contradict our result on the single-effective-query unlearnable families of functions in Appendix A because it is not truly random or pairwise-independent.

Now consider the following circuit. An adversary receives an oracle-based one-time program and a simulator gets only one (physical) query to the functionality's oracle. Both are required to output a piece of classical information to a distinguisher.

Let a be a uniformly random string in  $\{0,1\}^n$ . Let k be a random PRF key. The following  $\mathsf{PRF}_k(\cdot)$  maps  $\{0,1\}^{2n} \to \{0,1\}^{2n}$ . Let our circuit be the following:

$$f_{a,k}(x;r) = \begin{cases} (a, \mathsf{PRF}_k(0||r)) & \text{if } x = 0, \\ (k, \mathsf{PRF}_k(a||r)) & \text{if } x = a, \\ (0, \mathsf{PRF}_k(x||r)) & \text{otherwise.} \end{cases}$$
 (1)

When  $\mathcal{A}$  is given an actual program, even using an oracle-aided circuit,  $\mathcal{A}$  can do the following: evaluate the program on input 0 (and some randomness r it cannot control) to get output  $(a, \mathsf{PRF}_k(0||r))$ ; but instead of measuring the entire output, only measure the first n-bits to get a with probability 1; evaluate the program again on a and some r' to obtain  $(k, \mathsf{PRF}_k(a|r'))$ . Now it can reconstruct the classical description of the entire circuit using a and b.

But when given only a single *physical* query, we can remove the information of k using [BBBV97a] argument since for a random k, no adversary should have non-negligible query weight on k with just one single query.

#### 2.4 Concurrent Work and Related Works

Concurrent Work. A concurrent and independent work [GM24] presents a construction of quantum one-time programs for randomized classical functions. While our main constructions are very similar, there are some differences between our works which we outline next. (1) We undertake a more comprehensive study of security definitions for quantum one-time programs and come up with both simulation definitions in the oracle model as well as operational definitions in the oracle and plain model. [GM24] focuses on the oracle model. (2) We show simulation-based security for our construction in the oracle model, which is likely stronger than the security definition used in [GM24]; meanwhile, [GM24]'s security definition is likely stronger than the weaker security definition we consider in the oracle model, the operational definition; (3) We show an impossibility result for generic one-time randomized programs in the plain model; (4) We give constructions for PRFs and NIZKs in the plain model whereas all constructions in [GM24] are in the oracle model; (5) We also show a generic way to lift a plain signature schemes satisfying a security notion called blind unforgeability to one-time signature tokens. (5) On the other hand, the oracle construction in [GM24] is more generic by using any signature token state as the quantum part of the one-time program, whereas we use the subspace state (namely, the signature token state in [BDS23]).

One-Time Programs. One-time programs were first proposed by Goldwasser, Kalai, and Rothblum [GKR08b] and further studied in a number of followup works [GIS+10, GG17, ACE+22]. Although these are impossible in the plain model, a number of alternative models have been proposed to enable them, ranging from hardware assumptions to protein sequencing. Broadbent, Gutoski, and Stebila [BGS13] asked the question of whether quantum mechanics can act as a stand-in for hardware assumptions. However, they found that quantum one-time programs are only possible for "trivial" functions, which can be learned in a single query, and are generally impossible for deterministic classical functions. A pair of later works circumvented the impossibility by allowing the program to output an incorrect answer with some probability [RKB+18, RKFW21]. Although their results are quite interesting, they do not give formal security definitions for their scheme, and seem to assume a weaker adversarial model where the evaluator must make many intermediate physical measurements in an online manner. In contrast, we present a formal treatment with an adversary who may perform arbitrary quantum computations on the one time program as a whole.

[CGLZ19] develops a first quantum one-time program for classical message-authentication codes, assuming stateless classical hardware tokens.

Besides, [LSZ20] studied security of classical one-time memory under quantum superposition attacks. [Liu23] builds quantum one-time memory with quantum random oracle in the depth-bounded adversary model, where the honest party only needs a short-term quantum memory but the adversary, which attempts to maintain a quantum memory for a longer term cannot perform attacks due to bounded quantum depth.

**Signature Tokens.** Signature tokens are a special case of one-time programs that allow the evaluator to sign a single message, and no more. They were proposed by Ben-David and Sattath [BDS23] in the oracle model and subsequently generalized to the plain model using indistinguishability obfuscation [CLLZ21b]. Both of these works consider a very specific form of one-time security: an adversarial evaluator should not be able to output two (whole) valid signatures.

## 3 Preliminaries

#### 3.1 Quantum Information and Computation

We provide some basics frequently used in this work and refer to [NC02] for comprehensive details.

A projection is a linear operator P on a Hilbert space that satisfies the property:  $P^2 = P$  and is Hermitian  $P^{\dagger} = P$ .

A projective-valued measurement (PVM) a generalization of this idea to an entire set of measurement outcomes. A PVM is a collection of projection operators  $\{P_i\}$  that are associated with the outcomes of a measurement, and they satisfy two important properties: (1) orthogonality:  $P_iP_j=0, \forall i\neq j$ ; (2) completeness:  $\sum_i P_i=\mathbf{I}$  where  $\mathbf{I}$  is the identity operator.

**Definition 3.1** (Trace distance). Let  $\rho, \sigma \in \mathbb{C}^{2^n \times 2^n}$  be the density matrices of two quantum states. The trace distance between  $\rho$  and  $\sigma$  is

$$\|\rho - \sigma\|_{\mathrm{tr}} := \frac{1}{2} \sqrt{\mathsf{Tr}[(\rho - \sigma)^{\dagger}(\rho - \sigma)]},$$

Here, we only state a key lemma for our construction: the Gentle Measurement Lemma proposed by Aaronson [Aar04], which gives a way to perform measurements without destroying the state.

**Lemma 3.2** (Gentle Measurement Lemma [Aar04]). Suppose a measurement on a mixed state  $\rho$  yields a particular outcome with probability  $1 - \epsilon$ . Then after the measurement, one can recover a state  $\tilde{\rho}$  such that  $\|\tilde{\rho} - \rho\|_{\mathrm{tr}} \leq \sqrt{\epsilon}$ . Here  $\|\cdot\|_{\mathrm{tr}}$  is the trace distance (defined in Definition 3.1).

## 3.2 Quantum Query Model

We consider the quantum query model in this work, which gives quantum circuits access to some oracles.

**Definition 3.3** (Classical Oracle). A classical oracle  $\mathcal{O}$  is a unitary transformation of the form  $U_f | x, y, z \rangle \to | x, y + f(x), z \rangle$  for classical function  $f : \{0, 1\}^n \to \{0, 1\}^m$ . Note that a classical oracle can be queried in quantum superposition.

In the rest of the paper, unless specified otherwise, we refer to the word "oracle" as "classical oracle". A quantum oracle algorithm with oracle access to  $\mathcal{O}$  is a sequence of local unitaries  $U_i$  and oracle queries  $U_f$ . Thus, the query complexity of a quantum oracle algorithm is defined as the number of oracle calls to  $\mathcal{O}$ .

#### 3.3 Compressed Random Oracles

Zhandry [Zha19a] gives an efficient method of simulating a random oracle on the fly. The method maintains a database register  $\mathcal{D}$  which enables it to answer queries. At the start of time, the database is initialized to  $|\emptyset\rangle$ , representing the even superposition over all possible functions  $H:\mathcal{X}\to\mathcal{Y}$ . As the random oracle is queried, the database is updated. A database D takes the form of a set containing pairs (x,y). The state  $|D\rangle$  represents the even superposition over all functions  $H:\mathcal{X}\to\mathcal{Y}$  which are consistent with H(x)=y for all  $(x,y)\in D$ . At any given time, the database

register  $\mathcal{D}$  is in superposition over one or more databases  $|D\rangle$ . We write D(x) = y if  $(x, y) \in D$ . If no such entry exists, then we write  $D(x) = \bot$ .

In detail, the compressed oracle is defined using the following procedures:

• Decomp is defined by  $|x,u\rangle \otimes |D\rangle \mapsto |x,u\rangle \otimes \mathsf{Decomp}_x |D\rangle$ , where  $\mathsf{Decomp}_x$  is defined as follows for any D such that  $D(x) = \bot$ :

$$\mathsf{Decomp}_x |D\rangle \coloneqq \frac{1}{\sqrt{|\mathcal{Y}|}} \sum_{y \in \mathcal{Y}} |D \cup \{(x,y)\}\rangle \tag{2}$$

$$\mathsf{Decomp}_x \left( \frac{1}{\sqrt{|\mathcal{Y}|}} \sum_{y \in \mathcal{Y}} |D \cup \{(x,y)\}\rangle \right) \coloneqq |D\rangle \tag{3}$$

$$\mathsf{Decomp}_{x}\left(\frac{1}{\sqrt{|\mathcal{Y}|}}\sum_{y\in\mathcal{Y}}(-1)^{y\cdot u}\left|D\cup\{(x,y)\}\right\rangle\right) \coloneqq \left(\frac{1}{\sqrt{|\mathcal{Y}|}}\sum_{y\in\mathcal{Y}}(-1)^{y\cdot u}\left|D\cup\{(x,y)\}\right\rangle\right) \text{ for } u\neq 0$$

$$\tag{4}$$

• CO' maps  $|x, u, D\rangle \mapsto |x, u \oplus D(x), D\rangle$ 

On query in register Q, the compressed oracle applies

$$CO = Decomp \circ CO' \circ Decomp$$

to registers  $(Q, \mathcal{D})$ .

In more descriptive words, if D already is specified on x, and moreover if the corresponding y registers are in a state orthogonal to the uniform superposition (i.e. if we apply a QFT to the register, the resulting state contains no  $|0\rangle$ ), then there is no need to decompress and Decomp is the identity. On the other hand, if D is specified at x and the corresponding y registers are in the state of the uniform superposition, Decomp will remove x and the y register superposition from D.

We mention a few other results about compressed oracles. First, any adversary who successfully finds valid input/output pairs from the random oracle must cause the corresponding input/output pairs to be recorded in the compressed version of the oracle.

**Lemma 3.4** ([Zha19a], Lemma 5). Consider a quantum algorithm  $\mathcal{A}$  making queries to a random oracle H and outputting tuples  $(x_1, \dots, x_k, y_1, \dots, y_k)$ . Let R be a collection of such tuples. Suppose with probability p,  $\mathcal{A}$  outputs a tuple such that (1) the tuple is in R and (2)  $H(x_i) = y_i$  for all i. Now consider running  $\mathcal{A}$  with the compressed oracle defined in Section 3.3, and suppose the database D is measured after  $\mathcal{A}$  produces its output. Let p' be the probability that (1) the tuple is in R, and (2)  $D(x_i) = y_i$  for all i (and in particular  $D(x_i) \neq \bot$ ). Then:

$$\sqrt{p} \leq \sqrt{p'} + \sqrt{k/|\mathcal{Y}|}$$

Second, the probability of the compressed oracle's database containing a collision after q queries is bounded in terms of q and the size of the range.

**Lemma 3.5** ([Zha19a]). For any adversary with query access to a compressed oracle  $G: \mathcal{X} \to \mathcal{Y}$ , if the database register is measured after q queries to obtain D, then D contains a collision with probability at most  $O(q^3/|\mathcal{Y}|)$ .

As a corollary of this and Lemma 3.4, any adversary making q queries to a random oracle finds a collision with probability at most  $O(q^3/|\mathcal{Y}|)$ .

We mention a few facts about the compressed oracle that are not explicitly mentioned in the original work. First, if some x has never been queried to the oracle, i.e. has 0 amplitude in all prior queries, then  $D(x) = \bot$ . This is evident from the definition of Decomp. Second, the compressed oracle acts identically on all inputs, up to their names. In other words, querying  $x_1$  is the same as renaming  $x_1$  to  $x_2$  in both the query register and the database register, then querying  $x_2$ , then undoing the renaming.

**Claim 3.6.** Let SWITCH $_{x_1,x_2}$  be the unitary which maps any database D to the unique D' defined by  $D'(x_1) = D(x_2)$ ,  $D'(x_2) = D(x_1)$ , and  $D'(x_3) = D(x_3)$  for all  $x_3 \notin \{x_1, x_2\}$  and acts as the identity on all orthogonal states. Let  $U_{x_1,x_2}$  be the unitary which maps  $|x_1\rangle \mapsto |x_2\rangle$ ,  $|x_2\rangle \mapsto |x_1\rangle$  and acts as the identity on all orthogonal states. Then

$$\mathsf{CO} = (U_{x_1,x_2} \otimes I \otimes \mathsf{SWITCH}_{x_1,x_2}) \mathsf{CO}(U_{x_1,x_2} \otimes I \otimes \mathsf{SWITCH}_{x_1,x_2})$$

*Proof.* This follows from the observation that  $(U_{x_1,x_2} \otimes I \otimes \mathsf{SWITCH}_{x_1,x_2})$  commutes with both Decomp and  $\mathsf{CO}'$  and it is self-inverse.

### 3.4 Subspace States and Direct Product Hardness

In this subsection, we provide the basic definitions and properties of subspace states.

For any subspace A, its complement is  $A^{\perp} = \{b \in \mathbb{F}^n \mid \langle a, b \rangle \mod 2 = 0, \forall a \in A\}$ . It satisfies  $\dim(A) + \dim(A^{\perp}) = n$ . We also let  $|A| = 2^{\dim(A)}$  denote the size of the subspace A.

**Definition 3.7** (Subspace States). For any subspace  $A \subseteq \mathbb{F}_2^n$ , the subspace state  $|A\rangle$  is defined as

$$|A\rangle = \frac{1}{\sqrt{|A|}} \sum_{a \in A} |a\rangle$$
.

Note that given A, the subspace state  $|A\rangle$  can be constructed efficiently.

**Definition 3.8** (Subspace Membership Oracles). A subspace membership oracle  $\mathcal{O}_A$  for subspace A is a classical oracle that outputs 1 on input vector v if and only if  $v \in A$ .

**Query Lower Bounds for Direct-Product Hardness** We now present a query lower bound result for "cloning" the subspace states. These are useful for our security proof in the classical oracle model.

The theorem states that when given one copy of a random subspace state, it requires exponentially many queries to the membership oracles to produce two vectors either: one is in the primal subspace, the other in the dual subspace; or both are in the same subspace but are different.

**Theorem 3.9** (Direct-Product Hardness, [BDS23, BKNY23]). Let  $A \subseteq \mathbb{F}_2^n$  be a uniformly random subspace of dimension n/2. Let  $\epsilon > 0$  be such that  $1/\epsilon = o(2^{n/2})$ . Given one copy of  $|A\rangle$ , and quantum access to membership oracles for A and  $A^{\perp}$ , an adversary needs  $\Omega(\sqrt{\epsilon}2^{n/2})$  queries to output with probability at least  $\epsilon$  either of the following: (1) a pair (v, w) such that  $v \in A$  and  $w \in A^{\perp}$ ; (2) v, w are both in A or  $A^{\perp}$ ,  $v \neq w$ .

### 3.5 Tokenized Signature Definitions

A tokenized signature scheme is a signature scheme (Gen, Sign, Verify) equipped with two additional algorithms GenTok and TokSign. GenTok takes in a signing key sk and outputs a quantum token  $|T\rangle$ . We overload it to also take in an integer n, in which case it outputs n signing tokens. TokSign takes in a quantum token  $|T\rangle$  and a message m, then outputs a signature  $\sigma$  on m.

A tokenized signature scheme must satisfy correctness and tokenized unforgeability. Correctness requires that a signature token can be used to generate a valid signature on any m:

$$\Pr\left[ \begin{array}{c} (\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{Verify}(\mathsf{vk},m,\sigma) = \mathsf{Reject} : & |T\rangle \leftarrow \mathsf{GenTok}(\mathsf{sk}) \\ & \sigma \leftarrow \mathsf{TokSign}(m,|T\rangle) \end{array} \right] = \mathsf{negl}$$

**Definition 3.10.** A tokenized signature scheme (Gen, Sign, Verify, GenTok, TokSign) has tokenized unforgeability if for every QPT adversary A,

$$\Pr\left[ \begin{aligned} \operatorname{Verify}(\mathsf{vk}, m_i, \sigma_i) &= \operatorname{Accept} \, \forall i \in [n+1] : & (\mathsf{sk}, \mathsf{vk}) \leftarrow \operatorname{Gen}(1^\lambda) \\ & (m_1, \sigma_1) : & (m_{i+1}, \sigma_{i+1}) \leftarrow \operatorname{GenTok}(\mathsf{sk}) \\ & ((m_1, \sigma_1), \dots, (m_{n+1}, \sigma_{n+1}) \leftarrow \mathcal{A}(\mathsf{vk}, \bigotimes_{i=1}^n |T_i\rangle) \end{aligned} \right] = \mathsf{neglos}(1^\lambda)$$

## 4 Definitions of One-Time Sampling Programs

A one-time sampling program compiler OTP compiles a randomized function f into a quantum state  $\mathsf{OTP}(f)$  that can then be evaluated on any input x to learn f(x,R) for a uniformly random string R. The syntax and correctness of OTP are defined below. Then in the rest of the section, we will then present various notions of security and discuss their feasibility as well as their relative strengths. Also, we will sometimes refer to the one-time sampling programs as "one-time programs" for short.

Let  $\mathcal{F}$  be a family of randomized classical functions, such that any  $f \in \mathcal{F}$  takes as input  $x \in \mathcal{X}$ , then samples  $R \xleftarrow{\$} \mathcal{R}$ , and outputs f(x,R). For simplicity, let  $\mathcal{X} = \{0,1\}^m$  for some parameter  $m \in \mathbb{N}$ , and let  $1^m$  be implicit in the description of f.

**Definition 4.1** (Syntax of OTP). Let  $\mathcal{F}$  be a family of randomized classical functions. For any given  $f \in \mathcal{F}$ , define  $\mathcal{X}, \mathcal{R}, \mathcal{Y}$  to be the sets for which  $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ , and let m be the bit-length of inputs to f; i.e. let  $\mathcal{X} = \{0,1\}^m$ .

*Next*, OTP *is the following set of quantum polynomial-time algorithms:* 

Generate( $1^{\lambda}$ , f): Takes as input the security parameter  $1^{\lambda}$  and a description of the function  $f \in \mathcal{F}$ , and outputs a quantum state  $\mathsf{OTP}(f)$ . We assume that  $1^m$  is implicit in  $\mathsf{OTP}(f)$ .

Eval(OTP(f), x): Takes as input a quantum state OTP(f) and classical input  $x \in \{0,1\}^m$ , and outputs a classical value  $y \in \mathcal{Y}$ .

**Definition 4.2** (Correctness). OTP satisfies correctness for a given function family  $\mathcal{F}$  if Generate and Eval run in polynomial time, and for every  $f \in \mathcal{F}$  and  $x \in \{0,1\}^m$ , there exists a negligible function  $\operatorname{negl}(\cdot)$ , such that for all  $\lambda \in \mathbb{N}$ , the distributions

$$\{\operatorname{Eval}(\operatorname{Generate}(1^{\lambda},f),x)\} \quad \textit{ and } \quad \{f(x,r)\}$$

are  $negl(\lambda)$ -close in statistical distance, where the randomness in the second distribution  $\{f(x,r)\}$  is over the choice of  $r \stackrel{\$}{\leftarrow} \mathcal{R}$ .

**Remark 4.3.** *In this work, we consider sampling uniform randomness as the input. Such a distribution suffices for many applications we will discuss (one time signatures, one-time encryptions).* 

### 4.1 Simulation-based Security Definitions for One-Time Sampling Programs

Ideally we want to achieve a *simulation-based* security definition where we define the desired one-time program functionality and insist that the real scheme be indistinguishable from this idealized functionality. A natural simulation-based security definition along these is as follows. In the real world, the challenger samples  $f \leftarrow \mathcal{F}$ , and gives the adversary the output  $\mathsf{OTP}(f)$  of the one-time program obfuscator. On the other hand, in the ideal world, the challenger samples the function  $f \leftarrow \mathcal{F}$ , and the simulator is allowed to make one (quantum) query to an oracle  $O_{f(\cdot,\$)}^{(1)}$  that implements the randomized function f: on query input x, the oracle samples  $r \leftarrow \mathcal{R}$  and outputs f(x,r). This single-query oracle is modeled as a *stateful* oracle that shuts down after being queried once (regardless of the query being quantum/classical).

In the end, the distinguisher is given the output of the adversary (in the real world), or that of the simulator (in the ideal world) and is tasked with telling apart these two worlds. In both worlds, the distinguisher also receives an oracle  $O_f$  that implements the deterministic functionality of f: given query (x, r), it outputs f(x, r).

**Definition 4.4** (Single physical query simulation-based one-time security). Fix input length n and security parameter  $\lambda$ . For all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$ , there is a (non-uniform) quantum polynomial-time simulator Sim that is given single (quantum) query access to f and classical auxiliary information  $\mathtt{aux}_f$  such that for any QPT distinguisher  $\mathcal{D}$ , and for any  $f \in \mathcal{F}$ , there exists a negligible  $\mathtt{negl}(\cdot)$  such that for all  $\lambda \in \mathbb{N}$ ,

$$\left|\Pr[1\leftarrow \mathcal{D}^{O_f}(\mathcal{A}(1^{\lambda},\mathsf{OTP}(f),\mathsf{aux}_f),\mathsf{aux}_f)] - \Pr[1\leftarrow \mathcal{D}^{O_f}(\mathsf{Sim}^{O_{f(\cdot,\$)}^{(1)}}(\mathsf{aux}_f),\mathsf{aux}_f)]\right| \leq \mathsf{negl}(\lambda).$$

where both A and Sim are allowed to output oracle-aided circuits using oracles given to them (respectively).  $O_f$  is a stateless oracle for f that outputs y = f(x, r) ion any input (x, r) and not restricted in the number of queries that it can answer.

**Remark 4.5** (Auxiliary Input). The auxiliary information  $aux_f$  is a piece of classical, public information sampled together when sampling/choosing f. Therefore, all parties, A, D, Sim get to see  $aux_f$ .

For instance, when f is a signing function or a publicly-verifiable proof algorithm, this  $\mathsf{aux}_f$  can be the public verification key.

The above definition is a worst-case definition for all f, but we will see in the following discussions that even if we relax the definition to an average case f sampled from the function family, a strong impossibility result still holds.

**Impossibilities for Definition 4.4** As often applicable to simulation-based definitions, such as Virtual-Black-Box Obfuscation ([BGI+01]) the above definition suffers from several impossibilities.

The first impossibility results from the separation between the simulated oracle world and a plain model where the one-time program can be accessed in a possibly non-black-box way. While

our simulator is given only oracle access to f, the program we give to  $\mathcal{A}$  consists of actual code (and quantum states). Once  $\mathcal{A}$  has non-black-box access to the given one-time program,  $\mathcal{A}$  may be able to perform various attacks where the simulator cannot do: for instance, evaluating homomorphically on the one-time program. As demonstrated in [ABDS20, AP21], this type of non-black-box attacks are applicable even if the program is a quantum state.

Formalizing the actual non-black-box attack takes some effort due to the randomized evaluation on f in our setting. Nevertheless, we show that even for one-time programs with a sampling circuit, there exists circuits that can never be securely one-time programmed when given non-black-box access in Section 7.

**Barriers for Stateless One-Time Programs** Even worse, the above definition suffers from impossibilities even in the *oracle* model, where we make sure that the program  $\mathcal{A}$  receives also consists of oracle-aided circuits, so that the above non-black-box attack does not apply.

This barrier mainly results from the fact that we give Sim a stateful oracle but A a stateless one-time program (which contains a stateless oracle).

Consider the following  $\mathcal{A}$  and distinguisher  $\mathcal{D}$ :  $\mathcal{A}$  receives a possibly oracle-aided program and simply outputs the program itself to  $\mathcal{D}$ .  $\mathcal{D}$  is given arbitrary oracle access to  $\mathcal{O}_f$  so  $\mathcal{D}$  can perform the following attack using gentle measurement (Lemma 3.2) and un-computation, discussed in Section 2.1, paragraph "Overcoming barriers for stateless one-time programs" (2.1).

Conclusively, unless the function f itself is "learnable" through a single oracle query (for exaple, a constant function), one cannot achieve the above definition. Only in this trivial case, the simulator can fully recover the functionality of f and make up a program that looks like a real world program, since both Sim and  $\mathcal{A}$  can learn everything about f. This argument is formalized in [BGS13], which rules out stateless one-time programs even in the oracle model. Note that this impossibility holds even if we consider a randomized f sampled from the function family and or give a verification oracle that verifies whether a computation regarding f is correct, instead of full access to f.

To get around the above oracle-model attack, we first consider the following weakening: what if we limit both the adversary and simulator to output only *classical information*? Intuitively, this requires both  $\mathcal A$  and Sim to dequantize and "compress" what they can learn from the program/oracle into a piece of classical information.

**Definition 4.6** (Single physical query classical-output simulation-based one-time security). Let  $\lambda$  be the security parameter. For all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}'$  with classical output, there is a (non-uniform) quantum polynomial-time simulator Sim that is given single (quantum) query access to f such that for any QPT distinguisher  $\mathcal{D}$ , for any  $f \in \mathcal{F}_{\lambda}$ , there exists a negligible  $negl(\cdot)$  such that:

$$\left|\Pr[1\leftarrow \mathcal{D}^{O_f}(\mathcal{A}'(1^{\lambda},\mathsf{OTP}(f),\mathsf{aux}_f),\mathsf{aux}_f)] - \Pr[1\leftarrow \mathcal{D}^{O_f}(\mathsf{Sim}^{O_{f(\cdot,\$)}^{(1)}}(\mathsf{aux}_f),\mathsf{aux}_f)]\right| \leq \mathsf{negl}(\lambda).$$

where both A and Sim are allowed to output oracle-aided circuits, but only classical information.

 $\mathcal{O}_f$  is a stateless oracle for f that outputs y = f(x,r) on any input (x,r) and not restricted in the number of queries that it can answer.

However, we will demonstrate in Section 7 that even for f with high min-entropy output, there exists a family of functions such that: no single physical query simulator can "learn" much about f

when only given oracle access to stateful oracle  $\mathcal{O}_{f(\cdot),\$}^{(1)}$ , but there exists an efficient  $\mathcal{A}$  that can fully recover the functionality of f when given a stateless oracle-aided one-time sampling program. Therefore,  $\mathcal{A}$  can output a classical output that separates itself from Sim.

**Re-defining the stateful Single-Query Oracle** These barriers inspire us to look into the definition of "single query oracle" we give to Sim and consider a weakening on the restriction of the "single-query" we allow Sim to make: Sim can merely make one physical query, but  $\mathcal{A}$  and  $\mathcal{D}$  can actually make many queries, as long as the measurements on those queries they make are "gentle".

Is it possible to allow Sim to make "gentle" queries to the oracle for f, just as the adversary and distinguisher can do with their stateless oracles? We hope that for certain functionalities with sampled random inputs, we will be able to detect whether Sim makes a destructive (but meaningful) measurement on its query or a gentle (but likely uninformative) query, so that the stateful oracle can turn off once it has made a "destructive" query, but stay on when it has only made gentle queries.

The second weaker security definition we propose examines the above "gentle measurement attack" by the distinguisher more closely, and redefines what a single query means in the quantum query model. We develop this notion in the following subsection.

## 4.2 Single effective query (SEQ) simulation-based one-time security

In this section, we describe an alternative model of access which we call the single-effective-query (SEQ) model. This model intuitively allows the adversary to make multiple queries, but it can only have information about one evaluation at a time.

For example, the adversary can make a query, then uncompute it, and then make a query on a different input. After any query, the adversary only has information about at most one point of the function, so they never make more than a single effective query. We will use the compressed oracle technique to record queries the adversary makes and enforce that they cannot make more than a single effective query.

Let  $f: \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$  be a randomized function, where the user specifies x and r is chosen randomly<sup>6</sup>. Let  $f_{\$}$  be an implementation of f that uses a random oracle H to compute the randomness. On receiving input x,  $f_{\$}$  computes r = H(x) and outputs

$$f_{\mathfrak{R}}(x) = f(x, H(x))$$

The single effective query (SEQ) oracle  $f_{\$,1}$  augments  $f_{\$}$  with the ability to recognize how many distinct (effective) queries the user has made to H by implementing H as a compressed oracle that records the user's queries. If answering a given query would increase the number of recorded queries to 2 or more, then  $f_{\$,1}$  does not answer it. Also, when  $f_{\$,1}$  does answer a query, it will flip a bit b to indicate that it has done so. The formal definition  $f_{\$,1}$  is given below.

**Compressed Single-Effective-Query Oracle**  $f_{\$,1}$  We define  $f_{\$,1}$  to implement the random oracle H using the compressed oracle technique described in Section 3.3. In this case, the oracle's  $\mathcal{H}$ 

<sup>&</sup>lt;sup>6</sup>In general, this function may or may not output its randomness r.

register contains a superposition over databases D, which are sets of input/output pairs (x, r). When  $f_{\$,1}$  is initialized,  $\mathcal{H}$  is initialized to  $|\emptyset\rangle$ .

 $f_{\$,1}$  responds to queries by an isometry implementing the following procedure coherently on basis states  $|x,u,b\rangle_O\otimes |D\rangle_{\mathcal{H}}$ :

- 1. If  $(x', r) \in D$  for some  $x' \neq x$  and some  $r \in \mathcal{R}$ , skip the following steps.
- 2. Otherwise, prepare  $|x,0\rangle$  in a register Q' and query it to the compressed random oracle H.
- 3. Apply the isometry

$$|x,u,b\rangle_{\mathcal{O}}\otimes|x,r\rangle_{\mathcal{O}'}\mapsto|x,u\oplus f(x;r),b\oplus 1\rangle_{\mathcal{O}}\otimes|x,r\rangle_{\mathcal{O}'}$$

to registers (Q, Q')

4. Uncompute step 2 by querying Q' to the compressed oracle H.

It is not hard to see that  $\mathcal{H}$  records at most one query at a time.

**Claim 4.7.** Let A be an oracle algorithm interacting with  $f_{\$,1}$ . After every query in this interaction,  $\mathcal{H}$  is supported on databases D with at most one entry.

*Proof.* This is clearly true after 0 queries. We proceed by induction. Consider the projector onto the space spanned by states of the form

$$|x,u\rangle_{\mathcal{Q}}\otimes|\emptyset\rangle$$
$$|x,u\rangle_{\mathcal{Q}}\otimes|\{(x,r)\}\rangle$$

for some  $x \in \mathcal{X}$ ,  $u \in \mathcal{Y}$ , and  $r \in \mathcal{R}$ .  $f_{\$,1}$  acts as the identity on all states outside of this space. Furthermore, this space is invariant under queries to H. This follows from the fact that the compressed oracle operation Decomp maps  $|x,u\rangle \otimes |D\rangle \mapsto |x,u\rangle \otimes |D'\rangle$  where D and D' are the same, except for the possibility that  $D(x) \neq D'(x)$ , and the fact that the compressed oracle operation CO' does not modify  $|D\rangle$ . Finally,  $f_{\$,1}$  only operates on  $\mathcal H$  by querying H on states in this space, so it is also invariant on the space.

**Definition 4.8** (Single-Effective-Query Simulation-based One-Time Program Security). Let  $\lambda$  be the security parameter. Let f be a function in a function family  $\mathcal{F}$  that possibly comes with a public, classical auxiliary input  $\mathsf{aux}_f$ . Let D be a distinguisher that is bounded to make a polynomial number of oracle queries but is otherwise computationally unbounded. D is given a one-time program (real or simulated), a classical plaintext description of f, as well as  $\mathsf{aux}_f$ .

 $\mathsf{OTP}(f)$  is a **secure OTP** for  $\mathcal{F}$  if there exists a QPT simulator  $\mathsf{Sim}$  such that for every  $f \in \mathcal{F}$  and all distinguishers D, there exists a negligible function  $\mathsf{negl}(\cdot)$  such that for all  $\lambda \in \mathbb{N}$ :

$$\left| \Pr\left[ 1 \leftarrow D((\mathsf{OTP}(f), f, \mathsf{aux}_f) \right] - \Pr\left[ 1 \leftarrow D\left(\mathsf{Sim}^{f_{\$,1}}(\mathsf{aux}_f), f, \mathsf{aux}_f \right) \right] \right| \leq \mathsf{negl}(\lambda) \tag{5}$$

Note that this is in the oracle model, so both OTP and Sim can output oracle-aided programs. This definition helps us circumvent the strong impossibility result aforementioned: Sim can also make up a program that uses the single effective query oracle  $f_{\$,1}$ . If the distinguisher tries to gently query the oracle by only checking the correctness of evaluations, Sim can also gently query  $f_{\$,1}$  oracle as well, which does not prevent further queries.

## 4.3 Single-Query Learning Game and Learnability

In this section, we will give some further characterization and generalization of what types of functions can be made into one-time sampling programs with a meaningful security notion.

Eventually, we want the adversary not to evaluate the program twice. Clearly, this is only possible if the function itself cannot be learned with a single oracle query or at least it should be hard to learn two input-output pairs given one oracle query. We formalize such unlearnability through several definitions in this section.

The first definition is the most natural, which requires the adversary to be able to evaluate on two different inputs of its own choice correctly.

**Definition 4.9** (Single-Query Learning Game). A learning game for a function family  $\mathcal{F} = \{\mathcal{F}_{\lambda} : [N] \to [M]\}$ , a distribution family  $\mathcal{D} = \{D_f\}$ , and an adversary  $\mathcal{A}$  is denoted as LearningGame $_{\mathcal{F},\mathcal{D}}^{\mathcal{A}}(1^{\lambda})$  which consists of the following steps:

- 1. Sampling Phase: At the beginning of the game, the challenger takes a security parameter  $\lambda$  and samples a function  $(f, \mathsf{aux}_f) \leftarrow \mathcal{F}_{\lambda}$  at random according to distribution  $\mathcal{D}_f$ , where  $\mathsf{aux}_f$  is some classical auxiliary information.
- 2. Query Phase: A then gets a single oracle access to  $f^7$  and some classical auxiliary information  $aux_f$ ;
- 3. Challenge Phase:
  - (a) A outputs two input-output tuples  $(x_1, r_1; y_1), (x_2, r_2; y_2)$  where  $(x_1, r_1) \neq (x_2, r_2)$ .
  - (b) Challenger checks if  $f(x_1, r_1) = y_1$  and  $f(x_2, r_2) = y_2$ .

Challenger outputs 1 if and only if both the above checks pass.

**Definition 4.10** (Single-Query  $\gamma$ -Unlearnability). Let  $\lambda$  be the security parameter and let  $\gamma = \gamma(\lambda)$  be a function. A function family  $\mathcal{F} = \{\mathcal{F}_{\lambda}\}_{{\lambda} \in \mathbb{N}}$  is single-query  $\gamma$ -unlearnable if for all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$ ,

$$\Pr_{f \leftarrow \mathcal{F}_n}[\mathsf{LearningGame}_{\mathcal{F},\mathcal{D}}^{\mathcal{A}}(1^{\lambda}) = 1] \leq \gamma(\lambda).$$

The most often used unlearnability definition referred to in this work is when  $\gamma = \mathsf{negl}(\lambda)$ . We will often refer to the following definition as "Single-Query unlearnable" for short.

**Definition 4.11** (Single-query negl( $\lambda$ )-unlearnable functions). Let  $\lambda$  be the security parameter. A function family  $\mathcal{F} = \{\mathcal{F}_{\lambda}\}_{\lambda \in \mathbb{N}}$  is single-query unlearnable if for all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$ ,

$$\Pr_{f \leftarrow \mathcal{F}_n}[\mathsf{LearningGame}_{\mathcal{F},\mathcal{D}}^{\mathcal{A}}(1^{\lambda}) = 1] \leq \mathsf{negl}(\lambda).$$

**Remark 4.12** (Examples of Single-query  $negl(\lambda)$ -Unlearnable Functions). For example, a random function  $\mathcal{F}: \mathcal{X} \to \mathcal{Y}$ , when the range size  $\mathcal{Y}$  of f is superpolynomially large in the security parameter, it is easy to show that it is single-physical-query  $negl(\lambda)$ -unlearnable via existing quantum random oracle techniques, such as by applying Theorem 4.2 in [YZ21] or Lemma 5 in [Zha19a]. We refer to Appendix A for more details.

<sup>&</sup>lt;sup>7</sup>If we consider a single physical (effective, resp.) query to the oracle, then the learnability property will correspond to single physical (effective, resp.) query learnability. Unless otherwise specified (by emphasizing whether the query is physical/effective), the remaining definitions in the rest of this work refer to both cases.

**Remark 4.13** (Examples of Single-Query  $\gamma$ -Unlearnability). For some functionalities, A may be able to learn two input-outputs with a larger probability, (e.g. for a binary outcome random function, a random guess would succeed with probability at least 1/2), but not non-negligibly larger than some threshold  $\gamma(\lambda)$ .

**Generalized Unlearnability** Note that the above single-query learnability is not strong enough when we want f to be a cryptographic functionality: the adversary may learn something important about  $(x_1, r_1, y_1), (x_2, r_2, y_2)$  without outputting the entire input-output pair.

For example, when f is a pseudorandom function: it may not suffice to guarantee that  $\mathcal{A}$  cannot compute correctly two pairs of input-outputs. We should also rule out  $\mathcal{A}$ 's ability to win an indistinguishability-based pseudorandomness game for *both inputs*  $(x_1, r_1, y_1), (x_2, r_2, y_2)$ .

Before going into this more generic definition, we first define a notion of "predicate" important to our generic definition.

**Definition 4.14** (Predicate). A predicate P(f, x, r, z, ans) is a binary outcome function that runs a program f on a some input (x, r) to get output y, and outputs 0/1 depending on whether the tuple (x, r, y, z, ans) satisfies a binary relation  $R_f$  corresponding to  $f: (x, r, y, z, ans) \in R_f$ . z is some auxiliary input that specifies the relation.

Note that the above predicate definition implicates that with the capability to evaluate f (even if using only oracle) access, one has the ability to verify whether the predicate  $P(f,x,r,z,\mathsf{ans})$  is satisfied.

#### **Remark 4.15.** We provide two concrete examples:

- 1. A first concrete example for the above predicate is a secret-key encryption scheme: f is an encryption function. The predicate is encrypting a message x using r and and is an alleged valid ciphertext on message x using randomness r. Then the predicate P is simply encrypting x using r to check if and is the corresponding ciphertext.
- 2. Another example is when f is a signing function. The predicate signs message x using randomness r and checks if ans is a valid signature for message x with randomness r.
- 3. When f is a PRF, a possible predicate is to check if an alleged evaluation y is indeed the evaluation PRF(k, x|r).

**Definition 4.16** (Generalized Single-Query Learning Game). *learning game for a sampler* Samp (which samples a function in  $\mathcal{F}_{\lambda}$ ), a predicate  $P = \{P_{\lambda}\}$ , and an adversary  $\mathcal{A}$  is denoted as GenLearningGame $_{\mathsf{Samp},P}^{\mathcal{A}}(1^{\lambda})$ , which consists the following steps:

- 1. Sampling Phase: At the beginning of the game, the challenger samples  $(f, \mathsf{aux}_f) \leftarrow \mathsf{Samp}(1^\lambda)$ , where  $\mathsf{aux}_f$  is some classical auxiliary information.
  - **Query Phase:** A then gets a single oracle access to f and also gets  $aux_f$ ;
- 2. Challenge Phase:
  - (a) A outputs two input-randomness pairs  $(x_1, r_1), (x_2, r_2)$  where  $(x_1, r_1) \neq (x_2, r_2)$ .
  - (b) Challenger prepares challenges  $\ell_1, \ell_2$  i.i.d using  $(x_1, r_1)$  and  $(x_2, r_2)$  respectively and sends them to A.
  - (c) A outputs answers ans<sub>1</sub>, ans<sub>2</sub> for challenges  $\ell_1, \ell_2$ .

The game outputs 1 if and only if the answers satisfy the predicate  $P_{\lambda}(f, x_1, r_1, \ell_1, \mathsf{ans}_1)$  and  $P_{\lambda}(f, x_2, r_2, \ell_2, \mathsf{ans}_2)$  are both satisfied.

**Definition 4.17** (Generalized Single-query  $\gamma$ -Unlearnable functions). Let  $\lambda$  be the security parameter. A function family  $\mathcal{F} = \{\mathcal{F}_{\lambda}\}_{{\lambda} \in \mathbb{N}}$  is generalized single-query  $\gamma$  unlearnable for some  $\gamma = \gamma(\lambda)$  if for all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$ ,

$$\Pr_{(f,\mathsf{aux}_f)\leftarrow\mathsf{Samp}(1^\lambda)}[\mathsf{GenLearningGame}_{\mathsf{Samp},P}^{\mathcal{A}}(1^\lambda)=1] \leq \gamma.$$

**Remark 4.18** (Example). To demonstrate how the above definition works on a concrete level, we give an example of PRF: in the challenge phase, the adversary will provide  $(x_1, r_1), (x_2, r_2)$ . The challenger then samples two independent, uniform random bits  $b_1, b_2$ : if  $b_1 = 0$ , let  $y_1$  be the PRF evaluation on  $(x_1, r_1)$ ; else let  $y_1$  be a random value. We assign  $y_2$  correspondingly using  $b_2$  similarly. The security guarantees that A should not be able to output correct guesses for both  $b_1, b_2$  with overall probability non-negligibly larger than 1/2: A supposedly always has the power to compute one of them correctly with probability 1; but for the other challenge, A should not be able to win with probability larger than it can do in a regular pseudorandomness game, when it was not given the power to evaluate the PRF.

All the above definitions are well-defined for single-physical-query oracles or single effective query oracles. We also make the following simple observation:

**Claim 4.19.** Single-effective-query  $\gamma$ -unlearnability implies single-physical-query  $\gamma$ -unlearnability.

The single-physical-query oracle is a strictly stronger oracle and therefore anything unlearnable with a single-effective query is unlearnable with a single physical query.

**Remark 4.20.** However, the other way of the above implication is not true. We will give a counter example in Section 7.3.

## 4.4 Operational security definitions

In this section, inspired by the above unlearnability definitions, we consider a further relaxation of the above simulation based definition, which we can realize security for certain functionalities in the *plain model* without oracles in Section 6.

In these definitions, we can partially characterize the operations an adversary will do.

**Strong Operational One-Time Security** We first give a one-time security that relaxes the simulation based definitions Definition 4.8 and Definition 4.6.

The following definition says that given a one time program for f, any QPT (or polynomial quantum query in the oracle model) adversary should not be able to learn two input-output pairs with a noticeably larger probability than a simulator with single query (physical/effective, resp.) to the oracle.

**Definition 4.21** (Strong Operational Security). A one-time (sampling) program for a function family  $\mathcal{F}_{\lambda}$  satisfies strong operational security if: for all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$  receiving one copy of  $\mathsf{OTP}(f)$ ,  $\mathsf{aux}_f$ ,  $(f, \mathsf{aux}_f) \leftarrow \mathcal{F}_{\lambda}$ , there is a (non-uniform) quantum polynomial-time simulator Sim that is given single (physical/effective, resp.) quantum query access to f, there exists a negligible function  $\mathsf{negl}(\lambda)$ , the following holds for all  $\lambda \in \mathbb{N}$ :

$$\begin{split} |\Pr_{f,\mathsf{aux}_f \leftarrow \mathcal{F}_{\lambda}}[f(x_1,r_1) = y_1 \wedge f(x_2,r_2) = y_2 : ((x_1,r_1,y_1),(x_2,r_2,y_2)) \leftarrow \mathcal{A}(\mathsf{OTP}(f),\mathsf{aux}_f)] - \\ &\Pr_{f,\mathsf{aux}_f \leftarrow \mathcal{F}_{\lambda}}[\mathsf{LearningGame}_{f,\mathcal{D}}^{\mathsf{Sim}}(1^{\lambda}) = 1]| \leq \mathsf{negl}(\lambda). \end{split}$$

where LearningGame $^{\text{Sim}}_{\mathcal{F},\mathcal{D}}(1^{\lambda})$  is the single query learnability game defined in Definition 4.10 and  $(x_1,r_1)\neq (x_2,r_2)$ .

**Remark 4.22.** We will call the above  $\gamma$ -strong operational one-time security if we have  $\Pr[\mathsf{LearningGame}_{f,\mathcal{D}}^{\mathsf{Sim}}(1^{\lambda})] = \gamma$ .

**Generalized One-time Security** Similar to the discussions in Section 4.3 the above security is not sufficient when we want f to be a cryptographic functionality: the adversary may learn something important about  $(x_1, r_1, y_1), (x_2, r_2, y_2)$  without outputting the entire input-output pair.

Corresponding to the above generalized learning game in Definition 4.16, we give the following definition:

**Definition 4.23** (Generalized Operational One-Time Security Game). A Generalized Operational One-Time Security game for a sampler Samp (which samples a function in  $\mathcal{F}_{\lambda}$ ), a predicate  $P = \{P_{\lambda}\}$ , and an adversary  $\mathcal{A}$  is denoted as  $\mathsf{GenOTP}^{\mathcal{A}}_{\mathsf{Samp},P}(1^{\lambda})$ , which consists the following steps:

1. Sampling Phase: At the beginning of the game, the challenger samples  $(f, \mathsf{aux}_f) \leftarrow \mathsf{Samp}(1^\lambda)$ , where  $\mathsf{aux}$  is some classical auxiliary information.

**Query Phase**: A then gets a single copy OTP(f) and classical auxiliary information  $aux_f$ ;

- 2. Challenge Phase:
  - (a) A outputs two input-randomness pairs  $(x_1, r_1), (x_2, r_2)$  where  $(x_1, r_1) \neq (x_2, r_2)$ .
  - (b) Challenger prepares challenges  $\ell_1, \ell_2$  i.i.d using  $(x_1, r_1)$  and  $(x_2, r_2)$  respectively and sends them to A.
  - (c) A outputs answers ans<sub>1</sub>, ans<sub>2</sub> for challenges  $\ell_1, \ell_2$ .

The game outputs 1 if and only if the predicate  $P_{\lambda}(f, x_1, r_1, \ell_1, \mathsf{ans}_1)$  and  $P_{\lambda}(f, x_2, r_2, \ell_2, \mathsf{ans}_2)$  are both satisfied.

**Definition 4.24** (Generalized Strong Operational Security). A one-time (sampling) program for a function family  $\mathcal{F}_{\lambda}$  satisfies generalized strong operational security if: for all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$  there is a (non-uniform) quantum polynomial-time simulator Sim that is given single (physical/effective, resp.) quantum query access to f, there exists a negligible function  $\operatorname{negl}(\lambda)$ , the following holds for all  $\lambda \in \mathbb{N}$ :

$$|\Pr_{f,\mathsf{aux}_f\leftarrow\mathcal{F}_\lambda}[\mathsf{GenOTP}_{P,\mathsf{Samp}}^{\mathcal{A}}=1] - \Pr_{f,\mathsf{aux}_f\leftarrow\mathcal{F}_\lambda}[\mathsf{GenLearningGame}_{\mathcal{F},\mathcal{D}}^{\mathsf{Sim}}(1^\lambda)=1]| \leq \mathsf{negI}(\lambda).$$

where  $\mathsf{GenLearning}\mathsf{Game}^{\mathsf{Sim}}_{\mathsf{Samp},P}(1^{\lambda})$  is the single query learnability game defined in Definition 4.16.

**Remark 4.25** (Example). We will give a formal concrete example for the above definition for PRFs for the above in Definition 6.14.

**Weak Operational One-time Security** We finally present a relatively limited but intuitive definition: no efficient quantum adversary should be able to output two distinct samples, i.e. tuples of the form (x, r, f(x, r)) with non-negligible probability, given the one-time program  $\mathsf{OTP}(f)$  for f.

We can observe that this definition is only applicable to single-query  $negl(\lambda)$ -unlearnable functions defined in Definition 4.9. But this function class already covers many cryptographic applications that have search-based security, such as one-time signatures, encryptions and proofs.

**Definition 4.26** (Weak operational one-time security). A one-time (sampling) program for a function family  $\mathcal{F}_{\lambda}$  satisfies weak operational one-time security if: for all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function  $\operatorname{negl}(\lambda)$ , the following holds for all  $\lambda \in \mathbb{N}$ :

$$\Pr_{f,\mathsf{aux}_f \leftarrow \mathcal{F}_{\lambda}}[f(x_1,r_1) = y_1 \land f(x_2,r_2) = y_2 : ((x_1,r_1,y_1),(x_2,r_2,y_2)) \leftarrow \mathcal{A}(\mathsf{OTP}(f),\mathsf{aux}_f)] \leq \mathsf{negl}(\lambda).$$
 where  $(x_1,r_1) \neq (x_2,r_2)$ .

**Remark 4.27.** It is to observe that for a family of functions  $\mathcal{F}_{\lambda}$  which are  $\gamma$ -unlearnable for any inverse polynomial  $\gamma$  (i.e.  $\text{negl}(\lambda)$ -unlearnable), Definition 4.26 and Definition 4.21 are equivalent.

First, it is easy to observe that Definition 4.21 implies Definition 4.26. When  $\mathcal{F}_{\lambda}$  is  $\operatorname{negl}(\lambda)$ -unlearnable, the winning probability of the learning game for  $\operatorname{Sim}$  is  $\operatorname{negl}(\lambda)$ , which makes  $\operatorname{Pr}_{f,\operatorname{\mathsf{aux}}_f\leftarrow\mathcal{F}_{\lambda}}[f(x_1,r_1)=y_1\wedge f(x_2,r_2)=y_2:((x_1,r_1,y_1),(x_2,r_2,y_2))\leftarrow\mathcal{A}(\operatorname{OTP}(f),\operatorname{\mathsf{aux}}_f)]$  negligible.

**Operational one-time security for verifiable functions** Another natural definition could require security against adversaries to produce two input-output (x, f(x, r)) pairs without necessarily providing the randomness r used to generate the output. This notion would make most sense when there is an (not necessarily efficient) verification algorithm  $\mathsf{Verify}_f$  that takes pairs of the form (x, y) and either accepts or rejects. Call such function families verifiable.

**Definition 4.28.** A randomized function family  $\mathcal{F} = \{f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}\}$  is **verifiable** if there is an efficient procedure to sample a function along with an associated verification key  $(f, vk_f) \leftarrow \mathcal{F}$ , and there exists an efficient verification procedure Ver such that for all  $x \in \mathcal{X}$ ,

$$\Pr_{r \leftarrow \mathcal{R}}[1 \leftarrow \mathsf{Ver}_{\mathcal{F}}(vk_f, x, f(x, r))] \ge 1 - \mathsf{negl}(\lambda).$$

**Definition 4.29** (Operational one-time security for verifiable functions). A one-time (sampling) program for a function family  $\mathcal{F}_{\lambda}$  satisfies verifiable operational one-time security if: for all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function  $\operatorname{negl}(\lambda)$ , the following holds for all  $\lambda \in \mathbb{N}$ :

$$\Pr_{(f,vk_f)\leftarrow\mathcal{F}_{\lambda}}[\mathsf{Ver}_{\mathcal{F}}(vk_f,x_1,y_1)=\mathsf{Ver}_{\mathcal{F}}(vk_f,x_2,y_2)=1:((x_1,y_1),(x_2,y_2))\leftarrow\mathcal{A}(\mathsf{OTP}(f),vk_f)]\leq\mathsf{negl}(\lambda).$$

where  $x_1 \neq x_2$ .

### 4.5 Relationships among the definitions

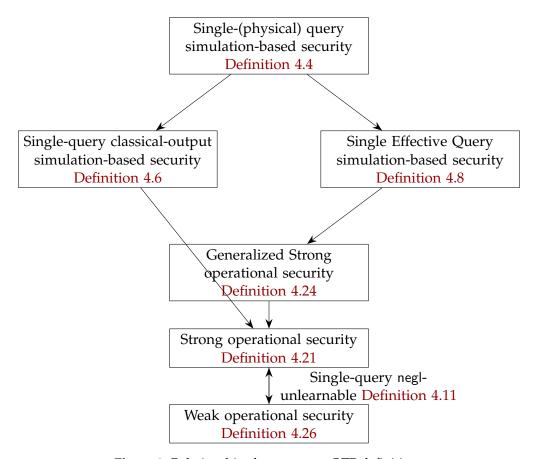


Figure 2: Relationships between our OTP definitions

We sort out the relations among the definitions we discussed in Figure 2.

Note that as we discussed in Remark 4.27: the weak operational security also implies strong operational security when the function family is single (physical/effective)-query  $negl(\lambda)$ -unlearnable. The implication from generalized strong operational security to strong operational security is also easy to see and we omit the proof.

The next two lemmas are true simply because the the adversary in the hypothesized security definition (Definition 4.4) is stronger than the adversary in the implied security definitions (Definition 4.6 and Definition 4.8).

**Lemma 4.30.** Suppose OTP is a one-time program compiler that satisfies the single-query simulation-based security definition (Definition 4.4) for a function family  $\mathcal{F}$ . Then, it also satisfies the single-query classical-output simulation-based security definition (Definition 4.6) for  $\mathcal{F}$ .

**Lemma 4.31.** Suppose OTP is a one-time program compiler that satisfies the single-query simulation-based security definition (Definition 4.4) for a function family  $\mathcal{F}$ . Then, it also satisfies the single-effective-query simulation-based security definition (Definition 4.8) for  $\mathcal{F}$ .

**Lemma 4.32.** Suppose OTP is a one-time program compiler that satisfies the single-query classical-output

simulation-based security (Definition 4.6) for a function family  $\mathcal{F}$ . Then, it satisfies strong operational security (Definition 4.21)  $\mathcal{F}$ .

*Proof.* We can observe by looking into the definition 4.21 that the adversary and simulator in this definition are simply a special case of the classical-output adversary and simulator, by outputting two correctly evaluated input-output pairs.

**Lemma 4.33.** Suppose OTP is a one-time program compiler that satisfies the single-effective-query simulation-based security (Definition 4.8) for a function family  $\mathcal{F}$ . Then, it satisfies generalized strong operational security (Definition 4.24)  $\mathcal{F}$ .

*Proof.* The difference between definition 4.24 (resp. Definition 4.16) and Definition 4.21 (respectively Definition 4.9) is that we can view the adversary/simulator as outputting a potentially quantum state together with its own choice of  $(x_1, r_1)$  and  $(x_2, r_2)$  in the challenge phase; then the quantum state is going to answer the challenger's challenges. Therefore, it is a special case of the single-physical/effective-query (depending which oracle we give to Sim) simulation definition with quantum outputs, but not necessarily a special case of the simulation definition with classical outputs.

# 5 Single-Effective-Query Construction in the Classical Oracle Model

#### 5.1 Construction

For a function  $f: \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$  with  $\mathcal{X} = \{0,1\}^m$ , and security parameter  $\lambda$ , we construct a one-time program  $\mathsf{OTP}(f)$  for f as described in Figure 3.

**Theorem 5.1.** For any function  $f \in \mathcal{F}$  that maps  $\mathcal{X} \times \mathcal{R} \to \mathcal{Y}$  where  $\mathcal{X} = \{0,1\}^m$ , the OTP construction given in Figure 3 satisfies correctness (Definition 4.2).

*Proof.* For every  $i \in [m]$ ,

$$|\psi_i\rangle = (H^{\otimes n})^{x_i} |A_i\rangle = \begin{cases} |A_i\rangle, & x_i = 0\\ |A_i^{\perp}\rangle, & x_i = 1 \end{cases}$$

Then the state  $|\psi_1\rangle \otimes \cdots \otimes |\psi_m\rangle$  gives an overwhelming fraction of its amplitude to values  $\mathbf{v}=(v_1,\ldots,v_m)$  for which  $\mathcal{O}_{A_i^{x_i}}(v_i)=1$  for all  $i\in[m]$ .

Then after applying  $\mathcal{O}_{f,G,\mathbf{A}}$  to  $\mathcal{Q}$ , the state of the  $\mathcal{Q}$  register gives an overwhelming fraction of its amplitude to values  $(x, \mathbf{v}, y)$  such that  $y = f(x, G(\mathbf{v}))$ .

Note that  $G(\mathbf{v})$  is uniformly random over  $\mathcal{R}$  due to the randomness of G. Then the output of  $\text{Eval}(1^{\lambda}, \text{OTP}(f), x)$  is negligibly close in statistical distance to f(x, R), where R is uniformly random over  $\mathcal{R}$ .

**Theorem 5.2.** Let  $\lambda$  be the security parameter. Let  $m, \ell \in \mathbb{N}$ . For any function  $f \in \mathcal{F} : \mathcal{X} \times \mathcal{R} \to \{0,1\}^{\ell}$  for which we let  $\mathcal{X} = \{0,1\}^m$ , the OTP construction given in Figure 3 satisfies the single-effective query simulation-based OTP security notion of Definition 4.8.

In general, f(x, r) may or may not output its randomness r. This will not modify our proof.

Our construction does not have any requirements on f's input lengths, for either the message or randomness. However, we remark that the SEQ simulation definition may not be very meaningful when the randomness is small. For example, if  $|\mathcal{R}|$  is only polynomially large, then any

Generate $(1^{\lambda}, f)$ :

- 1. Let  $n = \lambda$ . Then sample a random oracle  $G : \{0, 1\}^{m \cdot n} \to \mathcal{R}$ .
- 2. For each  $i \in [m]$ : sample a random subspace  $A_i \subseteq \mathbb{F}_2^n$  of dimension n/2, and create the membership oracles  $\mathcal{O}_{A_i^0}$ ,  $\mathcal{O}_{A_i^1}$  for  $A_i$  and  $A_i^{\perp}$  respectively.

$$\mathcal{O}_{A_i^0}(v) = \begin{cases} 1 & \text{if } v \in A_i \backslash \{0\}, \\ 0 & \text{otherwise}, \end{cases} \quad \text{and} \quad \mathcal{O}_{A_i^1}(v) = \begin{cases} 1 & \text{if } v \in A_i^{\perp} \backslash \{0\}, \\ 0 & \text{otherwise}. \end{cases}$$

Also let  $\mathbf{A} = (A_i)_{i \in [m]}$ .

3. Create oracle  $\mathcal{O}_{f,G,\mathbf{A}}$  that takes as input  $x \in \{0,1\}^m$ ,  $\mathbf{v} = (v_1,\ldots,v_m)$  where  $v_i \in \mathbb{F}_2^n$ , and  $u \in \mathcal{Y}$ , and outputs

$$\mathcal{O}_{f,G,\mathbf{A}}(x,\mathbf{v},u) = \begin{cases} (x,\mathbf{v},u \oplus f(x,G(\mathbf{v}))) & \text{if } \mathcal{O}_{A_i^{x_i}}(v_i) = 1 \text{ for all } i \in [m], \\ (x,\mathbf{v},u) & \text{otherwise} \end{cases}$$

4. Output  $OTP(f) = ((|A_i\rangle)_{i \in [m]}, \mathcal{O}_{f,G,\mathbf{A}}).$ 

Eval(OTP(f), x):

1. For each  $i \in [m]$ , compute:

$$|\psi_i\rangle = (H^{\otimes n})^{x_i} |A_i\rangle$$

2. Prepare the following state on the query register Q:

$$|x\rangle_{\mathcal{Q}_{\mathcal{X}}}\otimes|\psi_{1}\rangle\otimes\cdots\otimes|\psi_{m}\rangle\otimes|0\rangle_{\mathcal{Q}_{\mathcal{Y}}}$$

where  $0 \in \mathcal{Y}$ . Apply  $\mathcal{O}_{f,G,\mathbf{A}}$  to  $\mathcal{Q}$ .

3. Measure the  $Q_{\mathcal{Y}}$  register to obtain a value  $y \in \mathcal{Y}$  and output y.

**Figure 3:** Construction of OTP(f)

measurement made to f can at most disturb the program state by 1 - 1/poly. Thus, the adversary may be able to perform a second query with a 1/poly success rate.

The above theorem combined with the relationships between security definitions in Section 4.5 directly gives the following corollary:

**Corollary 5.3.** For function families satisfying the unlearnability definitions in Definition 4.17 (or Definition 4.10, Definition 4.11 respectively), there exists secure one-time sampling programs for them in the classical oracle model with respect to security definition Definition 4.24 (or Definition 4.21, Definition 4.26) resp.).

## 5.2 Proof of Security (Theorem 5.2)

The simulator is defined in Figure 4.

**Intuition.** To gain intuition about the simulator, we first recall the differences between the real and ideal worlds. In the real world, the OTP uses randomness G(v) to evaluate f, where v is a measurement of  $|A\rangle$  in a basis corresponding to the chosen input x. If f is sufficiently random, then measuring f(x, G(v)) may collapse  $|A\rangle$ , preventing further queries. On the other hand, in the ideal world, the SEQ oracle uses randomness H(x) to evaluate f. If f is sufficiently random, then measuring f(x; H(x)) may collapse the SEQ oracle's internal state, preventing further queries.

The main gaps that the simulator needs to bridge between these worlds are the usage of  $|A\rangle$ versus the usage of an internal state to control query access, as well as the usage of G(v) versus H(x). Since G and H are internal to the OTP oracle and SEQ oracle, respectively, the latter is not an issue even if f outputs its randomness directly. The simulator addresses the former by maintaining a cache for subspace vectors. If it detects that the SEQ oracle will not permit other queries, it stores the most recent subspace vector locally. This collapses  $|A\rangle$  in the view of the adversary, ensuring that a successful f evaluation in the ideal world looks similar to if f were evaluated using  $|A\rangle$  in the real world.

```
1. For each i \in [m], sample a subspace A_i \subseteq \mathbb{F}_2^{\lambda} of dimension \lambda/2 uniformly at random.
```

- 2. Initialize vector cache register  $\mathcal{V} = \mathcal{V}_1 \times \cdots \times \mathcal{V}_m$  to  $|0^{\lambda}\rangle_{\mathcal{V}_1} \otimes \cdots \otimes |0^{\lambda}\rangle_{\mathcal{V}_m}$ .
- 3. Prepare an oracle  $\mathcal{O}_{\mathsf{Sim}}$  as follows.
  - (a)  $\mathcal{O}_{\mathsf{Sim}}$  acts on a query register  $\mathcal{Q} = (\mathcal{Q}_x, \mathcal{Q}_\mathbf{v}, \mathcal{Q}_u)$ , which contains superpositions over states of the form  $|x, \mathbf{v}, u\rangle$ , where  $x \in \mathcal{X}$ ,  $\mathbf{v} = (v_1, \dots, v_m) \in \{0, 1\}^{n \cdot m}$ ,  $u \in \mathcal{Y}$ .
  - (b) If  $\mathcal{V}$  contains  $0^{n \cdot m}$  or  $\mathbf{v}$ , and if  $v_i \in A_i^{x_i} \setminus \{0\}$  for all  $i \in [m]$ , then  $\mathcal{O}_{\mathsf{Sim}}$  does the following:
    - i. Prepare  $|x\oplus 1,0,0\rangle$  in a register  $\mathcal{Q}'=(\mathcal{Q}'_x,\mathcal{Q}'_u,\mathcal{B}')$ , then query  $f_{\$,1}$  on register
    - ii. If  $\mathcal{B}'$  has value 0, apply a CNOT from register  $\mathcal{Q}_{\mathbf{v}}$  to register  $\mathcal{V}$ .
    - iii. Uncompute step 3(b)i

    - iv. Query  $f_{\S,1}$  on  $|x,u,0\rangle_{\mathcal{Q}_x,\mathcal{Q}_u,\mathcal{B}}$ . v. Prepare  $|x\oplus 1,0,0\rangle$  in a register  $\mathcal{Q}'=(\mathcal{Q}'_x,\mathcal{Q}'_u,\mathcal{B}')$ , then query  $f_{\S,1}$  on register 9.
    - vi. If  $\mathcal{B}'$  has value 0, apply a CNOT from register  $\mathcal{Q}_{\mathbf{v}}$  to register  $\mathcal{V}$ .
    - vii. Uncompute step 3(b)v.
- 4. Output  $((|A_i\rangle)_{i\in[m]}, \mathcal{O}_{\mathsf{Sim}})$ .

Figure 4: Simulator  $Sim^{f_{\$,1}}$ 

#### **Analysis of the Simulator.** Consider the following hybrid experiments:

- Hyb<sub>0</sub>: The real distribution OTP(f). Recall that OTP(f) outputs an oracle  $\mathcal{O}_{f,G,A}$  which acts as follows on input (x, v, u):
  - 1. **Vector Check:** It checks that  $v_i \in A_i^{x_i} \setminus \{0\}$  for all  $i \in [\lambda]$ . If not, it immediately outputs (x,v,u).
  - 2. **Evaluation:** Compute  $u \oplus f(x; G(v))$ .
  - 3. Output  $(x, v, u \oplus f(x; H(v)))$ .

•  $\underline{\mathsf{Hyb}_1}$ : The only difference from  $\mathsf{Hyb}_0$  is that G is implemented as a compressed oracle.  $\mathcal{O}_{f,G,A}$  maintains the compressed oracle's database register  $\mathcal{D}$  internally. To evaluate f in step 2, it queries  $|v,0\rangle_{\mathcal{O}'}$  to G in register  $\mathcal{Q}'$  to obtain  $|v,G(v)\rangle_{\mathcal{O}'}$ , then applies the isometry

$$|x,v,u\rangle_{\mathcal{O}}\otimes|v,r\rangle_{\mathcal{O}'}\mapsto|x,v,u\oplus f(x;r)\rangle_{\mathcal{O}}\otimes|v,r\rangle_{\mathcal{O}'}$$

to registers Q and Q', and finally queries G on register Q' again to reset it to  $|v,0\rangle_{Q'}$ .

- $\underline{\mathsf{Hyb}_2}$ : The only difference from  $\underline{\mathsf{Hyb}_1}$  is in step 1 of  $\mathcal{O}_{f,G,A}$ . Instead of checking that  $v_i \in \overline{A_i^{x_i}} \setminus \{0\}$ , it checks that  $v_i \in A_i^{x_i} \setminus (A_i \cap A_i^{\perp})$ .
- $\underline{\mathsf{Hyb}_3}$ : The only difference from  $\mathsf{Hyb}_2$  is we add a single-effective-query check to  $\mathcal{O}_{f,G,A}$ . It now answers basis state queries  $|x,v,u\rangle$  as follows:
  - 1. **Vector Check:** Check that  $v_i \in A_i^{x_i} \setminus (A_i \cap A_i^{\perp})$  for all  $i \in [\lambda]$ . If not, immediately output (x, v, u).
  - 2. **SEQ Check (New):** Look inside G's compressed database register  $\mathcal{D}$  to see if there is an entry of the form (v', r) for some r and  $v' \notin \{0, v\}$ . If so, then immediately output register  $\mathcal{Q}$ .
  - 3. **Evaluation:** Compute  $|x, v, v\rangle \mapsto |x, v, u \oplus f(x; G(v))\rangle$  on register  $\mathcal{Q}$ . This involves querying the compressed oracle G twice, as described in Hyb<sub>1</sub>.
  - 4. Output register Q.
- $\underline{\mathsf{Hyb}_4}$ : The only difference from  $\mathsf{Hyb}_2$  is we add a caching routine to  $\mathcal{O}_{f,G,A}$ . The oracle maintains a register  $\mathcal{R}_x$  which is initialized to  $|0\rangle$ . On receiving query (x,v,u), the oracle  $\mathcal{O}_{f,G,A}$  does the following:
  - 1. **Vector Check:** Check that  $v_i \in A_i^{x_i} \setminus (A_i \cap A_i^{\perp})$  for all  $i \in [\lambda]$ . If not, immediately output (x, v, u).
  - 2. **SEQ Check:** Do the single-effective query check that was added in Hyb<sub>3</sub>.
  - 3. Cache 1 (New): Look inside G's compressed database register  $\mathcal{D}$ . If it contains a nonempty database  $D \neq \emptyset$ , then perform a CNOT from register  $\mathcal{Q}_x$  to register  $\mathcal{R}_x$ .
  - 4. **Evaluation:** Apply the isometry  $|x,v,u\rangle_{\mathcal{Q}} \mapsto |x,v,u\oplus f(x;G(v))\rangle_{\mathcal{Q}}$  to register  $\mathcal{Q}$ . This involves querying the compressed oracle G twice, as described in  $\mathsf{Hyb}_1$ .
  - 5. Cache 2 (New): Look inside G's compressed database register  $\mathcal{D}$ . If there is an entry of the form (v', r) for some r and v', then perform a CNOT from register  $\mathcal{Q}_x$  to register  $\mathcal{R}_x$ .
  - 6. It outputs register Q.
- Hyb<sub>5</sub>: In this hybrid,  $\mathcal{O}_{f,G,A}$  swaps the role of v and x in the cache and compressed oracle. The oracle, which we rename to  $\mathcal{O}_{f,H,A}$ , maintains a cache register  $\mathcal{V}$  and a compressed oracle  $H: \mathcal{X} \to \mathcal{R}$  instead of  $G: \mathbb{F}_2^n \to \mathcal{R}$ . On query (x, v, u), it does the following:
  - 1. **Vector Check:** Check that  $v_i \in A_i^{x_i} \setminus (A_i \cap A_i^{\perp})$  for all  $i \in [\lambda]$ . If not, immediately output (x, v, u).
  - 2. **SEQ Check (Modified):** Look inside V to see if it contains some  $v' \notin \{0, v\}$ . If so, immediately output Q.

<sup>&</sup>lt;sup>8</sup>Intuitively, this hybrid will ensure that whenever some v corresponding to input x is recorded, x is also recorded. Thus, the internal oracle state will look like  $|0,\emptyset\rangle$  or  $|x,\{(v,r)\}\rangle$  in Hyb<sub>4</sub>. This intuition is made formal in Claim 5.6.

<sup>&</sup>lt;sup>9</sup>Intuitively, this modifies the internal oracle state from  $|x, \{(v, r)\}\rangle$  to  $|v, \{(x, r)\}\rangle$ , without modifying the case where the oracle state would be  $|0, \emptyset\rangle$ .

- 3. Cache 1 (Modified): Look inside H's compressed database register  $\mathcal{D}$ . If it contains a nonempty database  $D \neq \emptyset$ , then perform a CNOT from register  $Q_x$  to register  $R_x$ .
- 4. **Evaluation (Modified):** Applies the isometry  $|x, v, u\rangle_{\mathcal{Q}} \mapsto |x, v, u \oplus f(x; H(x))\rangle_{\mathcal{Q}}$  to register Q. This involves querying the compressed oracle H twice, analogously to the procedure in Hyb<sub>1</sub>.
- 5. Cache 2 (Modified): It looks inside H's compressed database register  $\mathcal{D}$ . If it contains a nonempty database  $D \neq \emptyset$ , then perform a CNOT from register  $Q_x$  to register  $R_x$ .
- 6. Output register Q.
- Hyb<sub>6</sub>: The only difference from Hyb<sub>4</sub> is a change to the SEQ check in step 2. Instead of looking inside  $\mathcal{V}$  to see if it contains some  $v' \notin \{0, v\}$ , the oracle  $\mathcal{O}_{f,H,A}$  instead looks inside H's compressed database register  $\mathcal{D}$  to see if there is an entry of the form (x', r) for  $x \neq x'$ .
- Hyb<sub>7</sub> = Sim: The only differences from Hyb<sub>5</sub> are in the caching routine in steps 3 and 5. It replaces each of these steps with the following procedure:
  - 1. Prepare a  $|0\rangle$  state in register  $\mathcal{B}$ . Controlled on H's database register having an entry of the form (x', r) for  $x' \neq x \oplus 1$ , apply a NOT operation to register  $\mathcal{B}$ .
  - 2. If register  $\mathcal{B}$  contains  $|1\rangle$ , apply a CNOT operation from register  $\mathcal{Q}_v$  to register  $\mathcal{V}$ .
  - 3. Uncompute step 1.

Hyb<sub>0</sub> is perfectly indistinguishable from Hyb<sub>1</sub> by the properties of a compressed random oracle. We now show indistinguishability for each of the other sequential pairs of hybrid experiments.

### **Claim 5.4.** Hyb<sub>1</sub> is computationally indistinguishable from Hyb<sub>2</sub>.

*Proof.* The only difference between these is that Hyb<sub>2</sub> additionally returns early whenever some  $v_i \in (A_i \cap A_i^{\perp}) \setminus \{0\}$ . This condition can only occur with negligible probability; otherwise, we can break direct product hardness for subspace states (Theorem 3.9) by embedding a challenge subspace state in a random index  $i^*$ , using polynomially many queries to the challenge membership oracles to run Hyb<sub>1</sub>, and after every query, if  $v_i \in (A_i \cap A_i^{\perp}) \setminus \{0\}$  for some  $i \in \lambda$ , measuring the query register to obtain v. Since  $i^*$  is independent of the adversary's view,  $i = i^*$  with probability 1/m whenever this occurs. Note that m is polynomial in  $\lambda$ .

#### **Claim 5.5.** Hyb<sub>2</sub> is computationally indistinguishable from Hyb<sub>3</sub>.

*Proof.* It is sufficient to show that the single-effective-query check added in step 2 causes an early output only with probability  $negl(\lambda)$ . We reduce this fact to the direct product hardness of subspace states (Theorem 3.9). Say that some adversary  $\mathcal{A}_{OTP}$  caused this event to occur with noticeable probability. We construct an adversary  $A_{DP}$  to break direct product hardness as follows:

- 1.  $\mathcal{A}_{DP}$  receives from the challenger  $\left(\ket{A_*},\mathcal{O}_{A_*},\mathcal{O}_{A_*^{\perp}}\right)$  for some subspace  $A_*\subseteq\mathbb{F}_2^n$  of dimension n/2 sampled uniformly at random.
- 2.  $\mathcal{A}_{DP}$  samples an index  $i^* \stackrel{\$}{\leftarrow} [m]$  in which to embed  $A_*$ , and they set  $\left( |A_i\rangle, \mathcal{O}_{A_i}, \mathcal{O}_{A_i^{\perp}} \right) =$  $\left(\ket{A_*},\mathcal{O}_{A_*},\mathcal{O}_{A_*^\perp}\right)$ . 3. For each  $i\in[m]\backslash\{i^*\}$ ,  $\mathcal{A}_{DP}$  samples a subspace  $A_i\subseteq\mathbb{F}_2^n$  of dimension n/2 uniformly at
- random. Then they prepare  $(|A_i\rangle, \mathcal{O}_{A_i}, \mathcal{O}_{A_i^{\perp}})$ .

- 4.  $\mathcal{A}_{DP}$  uses  $\left(\left|A_{i}\right\rangle, \mathcal{O}_{A_{i}}, \mathcal{O}_{A_{i}^{\perp}}\right)_{i \in [m]}$  to construct  $\mathsf{OTP}(f)$ , as described in  $\mathcal{H}_{3}$ . Then they run  $\mathcal{A}_{\mathsf{OTP}}$  on this construction, with the following modification: in 2, it *measures* the early return condition, instead of checking it coherently.
- 5.  $\mathcal{A}_{DP}$  terminates  $\mathsf{Hyb}_3$  as soon as the measurement result indicates to return early. Then, it measures registers  $\mathcal{Q}_v$  and the oracle database  $\mathcal{D}$  to obtain v and (v', r). It outputs  $v_{i^*}$  and  $v'_{i^*}$ .

Let  $\nu(\lambda)$  be the probability that  $\mathcal{A}_{DP}$  finds and outputs two vectors  $v_{i^*}$  and  $v'_{i^*}$ .  $\nu(\lambda)$  is non-negligible because otherwise,  $\mathcal{A}_{OTP}$  would trigger the early return condition with only negligible probability.

By definition of the early return condition, there is at least one index where  $v_i \neq v_i'$ . With probability  $\geq \frac{1}{m}$ , this index is  $i^*$  because the value of  $i^*$  is independent of  $\mathcal{A}_{\mathsf{OTP}}$ 's view. Furthermore, by definition of  $\mathsf{Hyb}_2$ , the only vectors w that are queried to G are those that satisfy  $w_i \in (A \cup A^{\perp}) \setminus (A \cap A^{\perp})$ , so at all times, the entries (w, r) in G's database satisfy this form. Therefore  $v_i' \in (A \cup A^{\perp}) \setminus (A \cap A^{\perp})$ . Similarly, if step 2 is reached, then  $v_i \in (A \cup A^{\perp}) \setminus (A \cap A^{\perp})$ .

Therefore whenever  $v_{i^*} \neq v'_{i^*}$ ,  $\mathcal{A}_{DP}$  wins the direct product hardness game. This occurs with probability  $\geq \frac{\nu(\lambda)}{m}$ . By Theorem 3.9,  $\nu(\lambda)$  must be negligible.

To aid in showing that  $\mathsf{Hyb}_3$  is indistinguishable from  $\mathsf{Hyb}_4$ , we prove that the cache introduced in  $\mathsf{Hyb}_4$  maintains the invariant that x is cached if and only if some v uniquely corresponding to x is recorded in the compressed oracle G.

**Claim 5.6.** After every query in Hyb<sub>4</sub>, the internal states of  $\mathcal{O}_{f,G,A}$ , consisting of the cache register  $\mathcal{R}_x$  and G's database register  $\mathcal{D}$ , lies entirely within the space spanned by states of the form

$$|0\rangle_{\mathcal{R}_x} \otimes |\emptyset\rangle_{\mathcal{D}}$$
$$|x\rangle_{\mathcal{R}_x} \otimes |\{(v,r)\}\rangle_{\mathcal{D}}$$

for some  $x \in \mathcal{X}$ ,  $r \in \mathcal{R}$ , and  $v \in \{0,1\}^{m \cdot n}$  such that  $v_i \in A_i^{x_i} \setminus (A_i \cap A_i^{\perp})$  for all  $i \in [\lambda]$ . In other words, if the state at time t is  $\rho_{\mathcal{R}_x,\mathcal{D}}^t$  and  $\Pi_{\mathsf{Hyb}_4}$  projects onto this space, then

$$\operatorname{Tr}[\Pi_{\mathsf{Hyb}_4} \rho^t_{\mathcal{R}_x,\mathcal{D}}] = 1$$

*Proof.* This is true at when  $f_{\$,1}$  is initialized, since registers  $\mathcal{R}_x$ ,  $\mathcal{D}$  are initialized to  $|0,\emptyset\rangle$ . We now show that  $\mathcal{O}_{f,G,A}$  is invariant on the space determined by  $\Pi_{\mathsf{Hyb}_4}$ . It suffices to consider the action of  $\mathcal{O}_{f,G,A}$  on these basis states.

We go through the operations of  $\mathcal{O}_{f,G,A}$  step-by-step. If step 1 (vector check) does not return early, then  $v_i \in A_i^{x_i} \setminus (A_i \cap A_i^{\perp})$  for all  $i \in [\lambda]$ . In step 2 (SEQ check), there are two cases. If D has an entry (v',r), then the SEQ check causes  $\mathcal{O}_{f,G,A}$  to return register  $\mathcal Q$  without modifying its internal state. On the other hand, if v = v' or if  $D = \emptyset$ , then  $\mathcal{O}_{f,G,A}$  proceeds to step 3.

We claim that at the end of step 3 (cache 1),  $\mathcal{R}_x$  contains  $|0\rangle$ . If  $D=\emptyset$ , step 3 leaves  $\mathcal{R}_x$  as  $|0\rangle$ . If v=v',  $\mathcal{O}_{f,G,A}$  applies a CNOT operation from  $\mathcal{Q}_x$  to  $\mathcal{R}_x$ . Before this operation,  $\mathcal{Q}_x$  contained a value x' such that  $v'_i=v_i\in A_i^{x_i}$  for all  $i\in [\lambda]$ . Since  $v_i\notin (A\cap A^\perp)$  for all  $i\in [\lambda]$ , v uniquely determines x, so x'=x. Therefore after applying the CNOT, register  $\mathcal{Q}_x$  contains  $|x\oplus x\rangle=|0\rangle$ .

In step 4 (evaluation), the oracle queries G on v. During this, the compressed oracle modifies its database register  $\mathcal{D}$ , but the new databases D' in the support of the state satisfy D'(w) = D(w)

for all  $w \neq v$ . Since  $D(w) = \bot$  for all  $w \neq v$ , register  $\mathcal{D}$  is supported on  $|\emptyset\rangle$  and  $|\{(v,r)\}\rangle$  for some  $r \in \mathcal{R}$  at the end of this step.

Applying step 5 (cache 2) produces  $|x\rangle_{\mathcal{R}_x}\otimes|\{(v,r)\}\rangle_{\mathcal{D}}$  when  $D=\{(v,r)\}$  and produces  $|0\rangle_{\mathcal{R}_x}\otimes|\emptyset\rangle_{\mathcal{D}}$  when  $D=\emptyset$ . Step 6 does not modify these registers further.

**Claim 5.7.** Hyb<sub>3</sub> and Hyb<sub>4</sub> are perfectly indistinguishable.

*Proof.* By Claim 5.6, the state of registers  $(\mathcal{X}, \mathcal{D})$  in Hyb<sub>3</sub> is always supported on states of the form  $|0,\emptyset\rangle$  or  $|x,(v,r)\rangle$  where  $x\in\mathcal{X}$ ,  $r\in\mathcal{R}$ , and  $v\in\{0,1\}^{m\cdot n}$  such that  $v_i\in A_i^{x_i}\setminus (A_i\cap A_i^\perp)$  for all  $i\in[\lambda]$ . This condition implies that v uniquely determines x, which we now denote by  $x_v$ . Therefore there is an isometry mapping

$$|0,(v,r)\rangle_{\mathcal{X},\mathcal{D}} \mapsto |x_v,(v,r)\rangle_{\mathcal{X},\mathcal{D}}$$

Let U be a unitary which implements this isometry. The state of  $\mathsf{Hyb}_4$  at any time t can be generated by running  $\mathsf{Hyb}_3$  until time t with the following modification: before each query, apply  $U^\dagger$  and after answering it, apply U.

**Claim 5.8.** Hyb<sub>4</sub> and Hyb<sub>5</sub> are perfectly indistinguishable.

*Proof.* Let  $|\psi_t^{\mathsf{Hyb}_4}\rangle_{\mathcal{A},\mathcal{Q},\mathcal{V},\mathcal{D}}$  be the joint state of the adversary and  $\mathcal{O}_{f,G,A}$  in  $\mathsf{Hyb}_4$  when query t is submitted and denote the state for  $\mathsf{Hyb}_5$  similarly. Let  $\mathcal{O}_{f,G,A}^{\mathsf{Hyb}_4}$  and  $\mathcal{O}_{f,G,A}^{\mathsf{Hyb}_5}$  denote the OTP oracle's unitary operations in  $\mathsf{Hyb}_4$  and  $\mathsf{Hyb}_5$ , respectively. Let U be the unitary mapping

$$|x,(v,r)\rangle \mapsto |v,(x,r)\rangle$$

and acting as the identity on all orthogonal states. We show by induction over the time t that

$$(I_{A,\mathcal{Q}} \otimes U_{\mathcal{V},\mathcal{D}}) |\psi_t^{\mathsf{Hyb}_4}\rangle_{\mathcal{A},\mathcal{Q},\mathcal{V},\mathcal{D}} = |\psi_t^{\mathsf{Hyb}_5}\rangle_{\mathcal{A},\mathcal{Q},\mathcal{V},\mathcal{D}}$$

This is clearly true for t=0, since both hybrids initialize  $\mathcal{V},\mathcal{D}$  to  $|0,\emptyset\rangle$ , which U acts as the identity on. Now consider some time t. By the inductive hypothesis and linearity of quantum computation, it suffices to consider the actions of  $\mathcal{O}_{f,G,A}^{\mathsf{Hyb}_4}$  and  $\mathcal{O}_{f,G,A}^{\mathsf{Hyb}_5}$  on the *same* basis state  $|x,v,u\rangle_{\mathcal{Q}}\otimes|x',D\rangle_{\mathcal{R}_x,\mathcal{D}}$ . Furthermore, by Claim 5.6, we may restrict ourselves to basis states of the form

$$|x, v, u\rangle_{\mathcal{Q}} \otimes |0, \emptyset\rangle_{\mathcal{R}_x, \mathcal{D}}$$
  
 $|x, v, u\rangle_{\mathcal{Q}} \otimes |x', (v', r)\rangle_{\mathcal{R}_x, \mathcal{D}}$ 

where  $v_i' \in A_i^{x_i'} \setminus (A_i \cap A_i^{\perp})$ .

Observe that step 1 (vector check) of  $\mathcal{O}_{f,G,A}^{\mathsf{Hyb}_4}$  and  $\mathcal{O}_{f,G,A}^{\mathsf{Hyb}_5}$  are identical, and that step 2 (SEQ check) is also identical after applying U to registers  $\mathcal{R}_x = \mathcal{V}$  and  $\mathcal{D}$ . If either step prompts an early return, then the states are identical. Otherwise, the input state is now constrained to (1)  $D = \emptyset$  and x' = 0 or (2)  $D \in \{\{(v,r)\}\}_{r \in \mathcal{R}}$  and x = x'. In either case, after step 3, the state of  $\mathsf{Hyb}_3$  and  $\mathsf{Hyb}_4$  are, respectively:

$$|x, v, u\rangle_{\mathcal{Q}} \otimes |0, D_v\rangle_{\mathcal{R}_x, \mathcal{D}}$$
  
 $|x, v, u\rangle_{\mathcal{Q}} \otimes |0, D_x\rangle_{\mathcal{V}, \mathcal{D}}$ 

where either (1)  $D_v = D_x = 0$  or  $D_v = \{(v, r)\}$  and  $D_x = \{(x, r)\}$  for some  $r \in \mathcal{R}$ . In either case,  $D_v$  is related to  $D_x$  by  $D_v(v) = D_x(x)$ ,  $D_v(x) = \bot = D_x(v)$  and  $D_v(w) = D_v(w)$  for all  $w \notin \{x, v\}$ .

By Claim 3.6, the states after each query to G (respectively H) in step 4 (evaluation) are identical up to renaming v to x in the compressed database. Thus, at the end of step 4, the states are, respectively:<sup>10</sup>

$$\alpha_{\emptyset} | x, v, \psi_{\emptyset} \rangle_{\mathcal{Q}} \otimes | 0, \emptyset \rangle_{\mathcal{R}_{x}, \mathcal{D}} + \sum_{r} \alpha_{r} | x, v, u \oplus f(x; r) \rangle_{\mathcal{Q}} \otimes | 0, \{(v, r)\} \rangle_{\mathcal{R}_{x}, \mathcal{D}}$$

$$\alpha_{\emptyset} | x, v, \psi_{\emptyset} \rangle_{\mathcal{Q}} \otimes | 0, \emptyset \rangle_{\mathcal{V}, \mathcal{D}} + \sum_{r} \alpha_{r} | x, v, u \oplus f(x; r) \rangle_{\mathcal{Q}} \otimes | 0, \{(x, r)\} \rangle_{\mathcal{V}, \mathcal{D}}$$

After applying step 5, the states become

$$\alpha_{\emptyset} | x, v, \psi_{\emptyset} \rangle_{\mathcal{Q}} \otimes | 0, \emptyset \rangle_{\mathcal{R}_{x}, \mathcal{D}} + \sum_{r} \alpha_{r} | x, v, u \oplus f(x; r) \rangle_{\mathcal{Q}} \otimes | x, \{(v, r)\} \rangle_{\mathcal{R}_{x}, \mathcal{D}}$$

$$\alpha_{\emptyset} | x, v, \psi_{\emptyset} \rangle_{\mathcal{Q}} \otimes | 0, \emptyset \rangle_{\mathcal{V}, \mathcal{D}} + \sum_{r} \alpha_{r} | x, v, u \oplus f(x; r) \rangle_{\mathcal{Q}} \otimes | v, \{(x, r)\} \rangle_{\mathcal{V}, \mathcal{D}}$$

Finally,  $U^{\dagger}$  maps the latter to the former.

To aid in the proof that  $\mathsf{Hyb}_5$  and  $\mathsf{Hyb}_6$  are indistinguishable, we show that a cache invariant holds for  $\mathsf{Hyb}_5$  and  $\mathsf{Hyb}_6$  which is similar to the one for  $\mathsf{Hyb}_4$ .

**Claim 5.9.** After every query in Hyb<sub>5</sub>, the internal states of  $\mathcal{O}_{f,G,A}$ , consisting of the cache register  $\mathcal{R}_x$  and G's database register  $\mathcal{D}$ , lies entirely within the space spanned by states of the form

$$|0\rangle_{\mathcal{R}_x} \otimes |\emptyset\rangle_{\mathcal{D}}$$
$$|x\rangle_{\mathcal{R}_x} \otimes |\{(v,r)\}\rangle_{\mathcal{D}}$$

for some  $x \in \mathcal{X}$ ,  $r \in \mathcal{R}$ , and  $v \in \{0,1\}^{m \cdot n}$  such that  $v_i \in A_i^{x_i} \setminus (A_i \cap A_i^{\perp})$  for all  $i \in [\lambda]$ .

*Proof.* As shown in the proof of Claim 5.8, the internal states of  $\mathcal{O}_{f,G,A}$  in Hyb<sub>4</sub> and Hyb<sub>5</sub> are related by the register renaming  $\mathcal{R}_x = \mathcal{V}$  and a unitary U mapping

$$|x\rangle_{\mathcal{R}_x} \otimes |\{(v,r)\}\rangle_{\mathcal{D}} \mapsto |v\rangle_{\mathcal{V}} \otimes |\{(x,r)\}\rangle_{\mathcal{D}}$$

and acting as the identity on all other states. By Claim 5.6, the internal state in  $\mathsf{Hyb}_4$  is supported on  $|0,\emptyset\rangle$ , which U acts trivially on, or  $|x,\{(v,r)\}\rangle$  where  $v_i\in A_i^{x_i}\setminus (A_i\cap A_i^\perp)$ , which U maps to  $|v,\{(x,r)\}\rangle$ .

**Claim 5.10.** Hyb<sub>5</sub> and Hyb<sub>6</sub> are perfectly indistinguishable.

*Proof.* The only change between these two hybrids is the single-effective-query check in step 2. Hyb<sub>5</sub> returns early if and only if  $\mathcal V$  contains some  $v'\notin\{0,v\}$ , whereas Hyb<sub>6</sub> returns early if and only if  $\mathcal D$  contains a database with an entry (x',r) for  $x\neq x'$ . Neither condition can occur for the first query, since the internal state of  $\mathcal O_{f,H,A}$  is initialized to  $|0,\emptyset\rangle$ . Thus, the internal states of the two hybrids after query 0 are identical. Inducting over the number of queries, the invariant show in Claim 5.9 applies to both Hyb<sub>5</sub> and Hyb<sub>6</sub> when query t+1 is submitted (but not yet answered). The invariant shows that at this point,  $v'\notin\{0,v\}$  if and only if  $\mathcal D$  contains an entry (x',r) for  $x\neq x'$ . Therefore Hyb<sub>5</sub> will return early when answering query t+1 if and only if Hyb<sub>6</sub> will.  $\square$ 

 $<sup>^{10}|\</sup>psi_{\emptyset}\rangle$  is not necessarily associated with a single r value, due to the potential of collisions  $f(x;r_1)=f(x;r_2)$  which might lead to partial compression.

#### **Claim 5.11.** Hyb<sub>6</sub> and Hyb<sub>7</sub> are perfectly indistinguishable.

*Proof.* Observe that the new caching procedure in Hyb<sub>7</sub> for steps 3 and 5 applies a CNOT operation from register  $\mathcal{Q}_v$  to register  $\mathcal{V}$  if and only if H's database register contains an entry (x',r) for  $x' \neq x \oplus 1$ . The single-effective-query check from step 2 ensures that H's database register is in the span of  $|D\rangle$  where  $D = \emptyset$  or D contains exactly one entry of the form (x,r). Since  $x \neq x \oplus 1$ , the new caching procedure applies a CNOT if and only if H's database register contains an entry of the form x.

On the other hand, steps 3 and 5 in  $Hyb_6$  apply the same CNOT operation if H's database register is non-empty. The single-effective-query check in step 2 ensures that this occurs only when H contains an entry of the form x. This is identical to the new caching procedure in  $Hyb_7$ .

# 5.3 Which Functions is SEQ Access Meaningful For?

Although it is possible to achieve SEQ simulation security for every function, not all functions are meaningfully restricted by SEQ access. For example, deterministic functions clearly can be fully learned with access to an SEQ oracle. In this section, we explore what properties imply that a function is unlearnable given SEQ access, culminating in a general criteria for achieving Definition 4.26.

Intuitively, a function must satisfy two loose properties in order to have any notion of unlearnability with SEQ access:

- **High Min-Entropy.** If f(x;r) is not sufficiently dependent on the randomness r, then measuring f(x;r) may only gently measure r. In this case, the SEQ oracle will allow additional queries with some lower, but still inverse polynomial, amplitude.
- **Unforgeability.** If it is possible to compute some f(x';r') given only f(x;r), then the adversary could learn two function evaluations using one query.

We emphasize that any reasonable notion of unlearnability must be *average-case* over the choice of f from some family. Otherwise, an adversary could trivially learn everything about f by receiving it as auxiliary input.

**Truly Random Functions.** As a concrete example, a truly random function exemplifies both of the above properties; it has maximal entropy on every input and f(x;r) is completely independent of f(x';r'). Indeed, we are able to show that any adversary with SEQ access to a truly random function cannot output two input/output pairs, except with negligible probability.

**Proposition 5.12.** Random functions with superpolynomial range size are single-effective-query  $\operatorname{negl}(\lambda)$ -unlearnable Definition 4.11. More formally, for functions  $\mathcal{F}: \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ , where  $|\mathcal{R}| = 2^{\lambda}$  and  $|\mathcal{Y}|$  is superpolynomial in  $\lambda$ , and for all (non-uniform) quantum polynomial-time adversaries  $\mathcal{A}$ , there exists a negligible function  $\operatorname{negl}(\cdot)$  such that:

$$\Pr_{f \leftarrow \mathcal{F}_n}[\mathcal{A}^{f_{\$,1}}(1^{\lambda}) \to (x_1, r_1, y_1 = f(x_1, r_1)), (x_2, r_2, y_2 = f(x_2, r_2))] \leq \mathsf{negl}(\lambda).$$

f is sampled uniformly at random from  $\mathcal{F}$  and  $f_{\$,1}$  is the single-effective-query oracle for f defined in Section 4.2.

To prove this claim, we introduce the following technical lemma about compositions  $f \circ H$  of random functions where f and H are implemented as compressed oracles. It shows that if f records a query  $x \| y$ , then H must record a corresponding query x where f(x) = y. Intuitively, this implies that f can only record a single query, since that is the restriction on H. Any input/output pairs that the adversary learns will be recorded by f, so they can only learn a single one. We prove the technical lemma in Appendix C.2. <sup>11</sup>

**Lemma 5.13.** Let  $G: \mathcal{X} \to \mathcal{Y}$  and  $H: \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$  be random oracles implemented by the compressed oracle technique. Define the function  $F: \mathcal{X} \to \mathcal{Z}$  by F(x) = H(x, G(x)). Consider running an interaction of an oracle algorithm with F until query t, then measuring the internal state of G and H to obtain  $D_G$  and  $D_H$ .

Let  $E_t$  be the event that after the measurement at time t, for all  $(x||y,z) \in D_H$ , there exists a entry  $(x,y) \in D_G$ . Then

$$\Pr[E_t] \ge 1 - 4t^2 \left( \frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^2} \right)$$

*Proof of Proposition 5.12.* We modify our view on the use of the function f to be the compressed oracle defined in Section 3.3. Recall that in our implementation of the SEQ oracle  $f_{\$,1}$  definition in Section 4.2, we already maintain a compressed oracle database for a random oracle  $H: \mathcal{X} \to \mathcal{R}$  which computes a fresh randomness r = H(x). Let us call this database for compressed oracle of H as  $D_H$ . Now we additionally have a database  $D_f$  for the compressed oracle of the function  $\mathcal{F}: \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ .

By Claim 4.7,  $D_H$  is a state in superposition of basis states where each represents a classical database that records only one entry (x, r), where r = h(x) for some random  $h \leftarrow H$ . These basis states are orthogonal to each other since each state records a different value x by our implementation of the SEQ oracle.

We take a further low-level view on our SEQ oracle implementation for a random function f: we can consider the adversary and the implementation of the SEQ oracle together as a "semi-honest" adversary  $\mathcal{A}'$  that queries the (compressed) random oracle f if and only if there is no entry in the database  $D_H$ . Note that the (compressed) random oracle f is a regular compressed oracle without any query restrictions.  $\mathcal{A}'$  receives  $\mathcal{A}'$ s query and simulates the SEQ oracle for  $\mathcal{A}$  using its access to oracle f. In the end,  $\mathcal{A}'$  outputs  $\mathcal{A}'$ s output. It is easy to observe that their advantage is the same since  $\mathcal{A}'$  perfectly simulates the SEQ oracle for  $\mathcal{A}$ .

Let us denote  $p:=\mathcal{A}^{f_{\$,1}}(1^{\lambda}) \to (x_1,r_1,y_1=f(x_1,r_1)), (x_2,r_2,y_2=f(x_2,r_2)).$  By the above observation, we can also denote p as  $\mathcal{A}'$ 's advantage of outputting  $(x_1,r_1,y_1=f(x_1,r_1)), (x_2,r_2,y_2=f(x_2,r_2)).$  By Lemma 3.4, we have  $\sqrt{p} \leq \sqrt{p'} + \sqrt{2/|\mathcal{Y}|}$ , where p' is the probability that  $D_f(x_1,r_1)=y_1, D_f(x_2,r_2)=y_2$  after a computational basis measurement on  $D_f$ . Lemma C.4 shows that whenever  $D_f(x_1,r_1)=y_1$  and  $D_f(x_2,r_2)=y_2$ , there are corresponding entries  $D_H(x_1)=r_1$  and  $D_H(x_2)=r_2$  in H's compressed database, except with probability  $4q^2\left(2/|\mathcal{R}|-1/|\mathcal{R}|^2\right)$ , where q is the number of queries the adversary has made. Since  $D_H$  contains at most one entry at a time (Claim 4.7) and  $\mathcal{R}$  has superpolynomial size, p' must be negligible in  $\lambda$ . Since  $\mathcal{Y}$  also has superpolynomial size, p must be negligible as well.

<sup>&</sup>lt;sup>11</sup>Appendix C.2 also contains a related technical lemma which shows that if an adversary has access to  $H \circ G$  and H records an entry (y, z), then G records an entry (x, y) for some x. The difference from the lemma mentioned here is that H does not also take G's input x as part of its input.

As an immediate corollary of Proposition 5.12, psuedorandom functions are SEQ-unlearnable under Definition 4.11.

**Pairwise Independent Functions.** We are also able to relax the requirement that f is a truly random function to just require that it is pairwise independent and has high entropy. Intuitively, the pairwise independence plays the role of unforgeability by ensuring the adversary cannot use one evaluation f(x;r) to learn anything about other evaluations f(x';r'). We prove the following statement in Appendix A.1.

**Proposition 5.14.** *Let*  $\mathcal{F}$  *be a family of functions mapping*  $\mathcal{X} \times \mathcal{R} \to \mathcal{Y}$  *that satisfies:* 

- 1. Pairwise independence: For any  $(x,r,y), (x',r',y') \in \mathcal{X} \times \mathcal{R} \times \mathcal{Y}$  such that  $(x,r) \neq (x',r'), \Pr_f[f(x,r) = y \land f(x',r') = y'] = \Pr_f[f(x,r) = y] \cdot \Pr_f[f(x',r') = y'].$
- 2. High Randomness: There is a negligible function  $\nu(\lambda)$  such that for any  $(x, r, y) \in \mathcal{X} \times \mathcal{R} \times \mathcal{Y}$ ,

$$\Pr_f[f(x,r) = y] \le \nu(n)$$

3.  $\frac{1}{|\mathcal{R}|} = \text{negl}(\lambda)$ 

*Then*  $\mathcal{F}$  *is* SEQ-negl( $\lambda$ )-unlearnable.

**Computational Unforgeability.** So far, we have considered functions which are unforgeable in a very strong, information-theoretic sense. We can also consider functions which satisfy a computational notion of unforgeability. It is important that this notion of unforgeability consider *quantum* query access. We introduce the following generalization of quantum blind unforgeability for signatures [AMRS20].

**Definition 5.15.** Let  $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}_f$  be a function family associated with a distribution  $\mathsf{Distr}_{\mathcal{F}}$  over function and auxiliary input pairs  $(f, \mathsf{aux}_f)$  and let P be a predicate on f,  $\mathsf{aux}_f$ , and a pair of strings (x, s).  $\mathcal{F}$  and  $\mathsf{Distr}_{\mathcal{F}}$  are quantum blind unforgeable with respect to P if for every QTP adversary  $\mathcal{A}$  and blinding set  $B \subset \mathcal{X} \times \mathcal{R}$ ,

$$\Pr\left[x \in B \land P(f, \mathsf{aux}_f, x, s) = \mathsf{Accept}: \begin{array}{c} (f, \mathsf{aux}_f) \leftarrow \mathsf{Distr}_{\mathcal{F}} \\ (x, s) \leftarrow \mathcal{A}^{f_B}(\mathsf{aux}_f) \end{array}\right]$$

where  $f_B$  denotes a (quantumly-accessible) oracle that takes as input x then outputs f(x) if  $x \notin B$ , and otherwise outputs  $\bot$ .

We will show that if a randomized function f is quantum blind unforgeable and uses the sampled randomness in a particular way, then it is hard to come up with two input/output pairs of f when given SEQ access to it.

**Proposition 5.16.** Let  $\mathcal{F} = \{f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}\}_f$  be a function family associated with distribution D. Let  $G : \mathcal{X} \times \mathcal{R}' \to \mathcal{R}$  be a random function where  $|\mathcal{R}'|^2/|\mathcal{R}| = \text{negl}(\lambda)$  and define  $f_G : \mathcal{X} \times \mathcal{R}' \to \mathcal{Y}$  by  $f_G(x;r) = f(x;G(x,r))$ . Define  $\text{Distr}_G$  as the distribution which samples  $f \leftarrow D$ , then outputs  $f_G$ .

If  $(\mathcal{F}, \text{Distr})$  is blind-unforgeable with respect to the predicate that outputs  $f_G$  on input  $f_G(x,r)$  such that f(x,r) = y, then  $f_G(x,r)$  is single-effective-query unlearnable under  $f_G(x,r)$  before  $f_G(x,r)$  is single-effective-query unlearnable under  $f_G(x,r)$ .

Intuitively, because the blind unforgeability property is also being applied to r, it also ensures that f must be highly randomized. For example, if f ignored its randomness r, then it would be trivially forgeable simply by outputting the same f(x;r) with two different r and r'.

To prove this claim, we use another technical lemma which shows that adversaries who can find images of G must know corresponding preimages. Intuitively, it will allow us to show that an adversary who finds an input/output pair ((x,r),y) must know an r' such that G(x,r')=r. We prove the technical lemma in Appendix C.3.

**Lemma 5.17.** Let  $G: \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{Y}$  be a random function where  $|\mathcal{X}_2| < |\mathcal{Y}|$ . Consider an oracle algorithm A makes q of queries to G, then outputs two vector of k values  $\mathbf{x}^{(1)} = \left(x_1^{(1)}, \dots, x_k^{(1)}\right)$  and  $\mathbf{y} = (y_1, \dots, y_k)$ . Let p be the probability that for every i, there exists an  $x_i^{(2)} \in \mathcal{X}$  such that  $G\left(x_i^{(1)}, x_i^{(2)}\right) = y_i$ .

Now consider running the same experiment where G is instead implemented as a compressed oracle, and measuring its database register after A outputs to obtain D. Let p' be the probability that for every i, there exists an  $x_i^{(2)} \in \mathcal{X}_2$  such that  $D\left(x_i^{(1)} \| x_i^{(2)}\right) = y_i$ . If k and q are  $poly(\lambda)$  and  $|\mathcal{X}_2|^k/|\mathcal{Y}| = negl(\lambda)$ , then  $|\mathcal{X}_2|^k$ 

$$p \le p' + \mathsf{negl}(\lambda)$$

*Proof of Proposition 5.16.* We first claim that no adversary can output a tuple ((x, r), y) such that f(x, r) = y and r is not in the image of  $G(x, \cdot)$ , except with negligible probability. This follows from quantum blind unforgeability along with the observation that the SEQ oracle for  $f_G$  can be evaluated using a list of all evaluations f(x, G(x, r')) for  $r' \in \mathcal{R}'$ .

Second, we claim that if an adversary outputs two valid input/output tuples  $((x_1,r_1),y_1)$  and  $((x_2,r_2),y_2)$  where  $x_1\neq x_2$ , then with overwhelming probability  $r_1\neq r_2$ . Say the adversary did so with probability p. By the previous claim, whenever the adversary succeeds, there exist  $r'_1$  and  $r'_2$  such that  $r_1=G(x_1,r'_1)$  and  $r_2=G(x_2,r'_2)$ , except with negligible probability. If  $r_1=r_2$ , then  $G(x_1,r'_1)=G(x_2,r'_2)$ . Thus, we could find a collision in G with probability  $p/|\mathcal{R}'|^2-\text{negl}(\lambda)$  by guessing  $r'_1$  and  $r'_2$ . Lemma 3.5 shows that after q queries, the probability of finding a collision is  $O(q^3/|\mathcal{R}|)$ . Since  $|\mathcal{R}'|^2/|\mathcal{R}|=\text{negl}(\lambda)$ , we have  $p\leq O(q^3|\mathcal{R}^2|/|\mathcal{R}|)=\text{negl}(\lambda)$ .

Lemma 5.17 shows that if we were to implement G as a compressed oracle, then whenever the adversary finds two valid input/output tuples where  $r_1 \neq r_2$ , with overwhelming probability G's compressed database contains entries  $(x_1'\|r_1',r_1)$  and  $(x_2'\|r_2',r_2)$ . By Lemma 5.13, whenever this occurs, H also contains two entries, except with negligible probability. However, H never contains more than one entry (Claim 4.7). Combining all of these facts together, any QPT adversary given SEQ access to  $f_G \leftarrow \operatorname{Distr}_G$  cannot output two distinct tuples  $((x_1,r_1),y_1)$  and  $((x_2,r_2),y_2)$  such that  $f(x_1,r_1)=y_1$  and  $f(x_2,r_2)=y_2$ , except with negligible probability.

In Section 8, we use the techniques developed in this section to compile any signature scheme satisfying quantum blind unforgeability to enable signature tokens, almost without modifying the verification process.

<sup>&</sup>lt;sup>12</sup>We remark that the reliance on the number of queries is unlikely to be tight. However, this bound is sufficient for our purposes since we will anyway combine it with results that require a query-bounded adversary.

# 6 Construction in the Plain Model

In this section, we give a construction of a one-time sampling program in the plain model for constrained PRFs. We prove the following theorem:

**Theorem 6.1.** Assuming the security of post-quantum indistinguishability obfuscation, and LWE (or alternatively, assuming sub-exponentially secure iO and OWFs), then there exists secure one-time sampling programs for constrained PRFs PRF:  $\{0,1\}^k \times \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$  with respect to the weak operational security Definition 4.26. Here we let  $\lambda, k, m \in \mathbb{N}$  and  $\ell \geq n \cdot \lambda$ .  $\{0,1\}^k$  is the key space for the PRF,  $\{0,1\}^n$  the input space and  $\{0,1\}^\ell$  the randomness space.

#### 6.1 Preliminaries

#### 6.1.1 Indistinguishability Obfuscation

**Definition 6.2** (Indistinguishability Obfuscator (iO) [BGI+01, GGH+16, SW14]). A uniform PPT machine iO is an indistinguishability obfuscator for a circuit class  $\{C_{\lambda}\}_{{\lambda}\in\mathbb{N}}$  if the following conditions are satisfied:

• For all  $\lambda$ , all  $C \in \mathcal{C}_{\lambda}$ , all inputs x, we have

$$\Pr\left[\widehat{C}(x) = C(x) \,|\, \widehat{C} \leftarrow \mathsf{iO}(1^{\lambda}, C)\right] = 1$$

• (Post-quantum security): For all (not necessarily uniform) QPT adversaries (Samp, D), the following holds: if  $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^{\lambda})] > 1 - \alpha(\lambda)$  for some negligible function  $\alpha$ , then there exists a negligible function  $\beta$  such that:

$$\begin{split} & \left| \Pr \left[ D(\sigma, \mathsf{iO}(1^\lambda, C_0)) = 1 \, : \, (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda) \right] \right. \\ & \left. - \Pr \left[ D(\sigma, \mathsf{iO}(1^\lambda, C_1)) = 1 \, : \, (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda) \right] \right| \leq \beta(\lambda) \end{split}$$

Whenever we assume the existence of iO in the rest of the paper, we refer to iO for the class of polynomial-size circuits, i.e. when  $C_{\lambda}$  is the collection of all circuits of size at most  $\lambda$ .

# 6.1.2 Subspace Hiding Obfuscation

Subspace-hiding obfuscation was introduced by Zhandry [Zha19b] as a key component in constructing public-key quantum money. This notion requires that the obfuscation of a circuit that computes membership in a subspace A is indistinguishable from the obfuscation of a circuit that computes membership in a uniformly random superspace of A (of dimension sufficiently far from the full dimension). The formal definition is as follows.

**Definition 6.3** ([Zha19b]). A subspace hiding obfuscator (shO) for a field  $\mathbb{F}$  and dimensions  $d_0, d_1$  is a PPT algorithm shO such that:

• *Input.* shO takes as input the description of a linear subspace  $S \subseteq \mathbb{F}^n$  of dimension  $d \in \{d_0, d_1\}$ . For concreteness, we will assume S is given as a matrix whose rows form a basis for S.

• Output. shO outputs a circuit  $\hat{S}$  that computes membership in S. Precisely, let S(x) be the function that decides membership in S. Then there exists a negligible function negl,

$$\Pr[\hat{S}(x) = S(x) \ \forall x : \hat{S} \leftarrow \mathsf{shO}(S)] \ge 1 - \mathsf{negl}(n)$$

- Security. For security, consider the following game between an adversary and a challenger.
  - The adversary submits to the challenger a subspace  $S_0$  of dimension  $d_0$ .
  - The challenger samples a uniformly random subspace  $S_1 \subseteq \mathbb{F}^n$  of dimension  $d_1$  such that  $S_0 \subseteq S_1$ .
    - It then runs  $\hat{S} \leftarrow \mathsf{shO}(S_b)$ , and gives  $\hat{S}$  to the adversary.
  - The adversary makes a guess b' for b.

shO is secure if all QPT adversaries have negligible advantage in this game.

Zhandry [Zha19b] gives a construction of a subspace hiding obfuscator based on one-way functions and iO.

**Theorem 6.4** (Theorem 6.3 in [Zha19b]). If injective one-way functions exist, then any indistinguishability obfuscator, appropriately padded, is also a subspace hiding obfuscator for field  $\mathbb{F}$  and dimensions  $d_0, d_1$ , as long as  $|\mathbb{F}|^{n-d_1}$  is exponential.

### 6.1.3 Subspace Coset States and Computational Direct-Product Hardness

In this section, we provide the computational version for the direct product hardness property given in Section 3.4.

In order to use the direct product hardness property assuming only the security of iO and injective OWFs, we use a variant of the subspace states Section 3.4 called "subspace coset states" as follows, often referred to as "coset states" for short.

**Definition 6.5** (Subspace Coset States, [CLLZ21a, VZ21]). For any subspace  $A \subseteq \mathbb{F}_2^n$  and vectors  $s, s' \in \mathbb{F}_2^n$ , the coset state  $|A_{s,s'}\rangle$  is defined as:

$$|A_{s,s'}\rangle = \frac{1}{\sqrt{|A|}} \sum_{a \in A} (-1)^{\langle s',a\rangle} |a+s\rangle.$$

Note that by applying  $H^{\otimes n}$ , which is QFT for  $\mathbb{F}_2^n$ , to the state  $|A_{s,s'}\rangle$ , one obtains exactly  $|A_{s',s}^{\perp}\rangle$ . Additionally, note that given  $|A\rangle$  and s,s', one can efficiently construct  $|A_{s,s'}\rangle$  as follows:

$$\sum_{a} |a\rangle \xrightarrow{\operatorname{add} s} \sum_{a} |a+s\rangle \xrightarrow{H^{\otimes n}} \sum_{a' \in A^{\perp}} (-1)^{\langle a', s \rangle} |a'\rangle$$

$$\xrightarrow{\operatorname{adding} s'} \sum_{a' \in A^{\perp}} (-1)^{\langle a', s \rangle} |a' + s'\rangle \xrightarrow{H^{\otimes n}} \sum_{a \in A} (-1)^{\langle a, s' \rangle} |a+s\rangle$$

For a subspace A and vectors s, s', we define  $A+s=\{v+s:v\in A\}$ , and  $A^{\perp}+s'=\{v+s':v\in A^{\perp}\}$ .

We then present the computational version of the direct product hardness property of Theorem 3.9, by combining two theorems from [CLLZ21a] and [CHV23].

We have defined shO above. For our construction, we will need the following variant of shO.

**Definition 6.6** (Coset Subspace Obfuscation Programs). We denote shO(A + s) for the following program:  $iO(shO_A(\cdot - s))$ , where  $shO_A()$  denotes the subspace-hiding program shO(A), and shO is the subspace hiding obfuscator defined in Section 6.1.2. Therefore,  $shO_A(\cdot - s)$  is the program that on input x, runs program shO(A) on input x - s.  $iO(shO_A(\cdot - s))$  is an indistinguishability obfuscation of  $shO_A(\cdot - s)$ .

**Theorem 6.7** (Computational Direct Product Hardness, [CLLZ21a, CHV23]). Assume the existence of post-quantum iO and injective one-way function. Let  $A \subseteq \mathbb{F}_2^n$  be a uniformly random subspace of dimension n/2, and s, s' be uniformly random in  $\mathbb{F}_2^n$ . Given one copy of  $|A_{s,s'}\rangle$ , shO(A+s) and  $shO(A^{\perp}+s')$ , any polynomial time adversary outputs a pair (v,w) with only negligible probability such that either of the following is satisfied: (1)  $v \in A + s$  and  $w \in A^{\perp} + s'$ ; (2)  $v, w \in A + s$  or  $v, w \in A^{\perp} + s'$  and  $v \neq w$ .

#### 6.1.4 Puncturable, Constrained and Invertible PRFs

**Puncturable PRFs** A *puncturable* PRF is a PRF augmented with a procedure that allows to "puncture" a PRF key K at a set of points S, in such a way that the adversary with the punctured key can evaluate the PRF at all points except the points in S. Moreover, even given the punctured key, an adversary cannot distinguish between a uniformly random value and the evaluation of the PRF at a point S with respect to the original unpunctured key. Formally:

**Definition 6.8** ((Post-quantum) Puncturable PRF). A PRF family  $F: \{0,1\}^{k(\lambda)} \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$  with key generation procedure KeyGen<sub>F</sub> is said to be puncturable if there exists an algorithm Puncture<sub>F</sub>, satisfying the following conditions:

• Functionality preserved under puncturing: Let  $S \subseteq \{0,1\}^{n(\lambda)}$ . For all  $x \in \{0,1\}^{n(\lambda)}$  where  $x \notin S$ , we have that:

$$\Pr[F(K,x) = F(K_S,x) : K \leftarrow \mathsf{KeyGen}(1^{\lambda}), K_S \leftarrow \mathsf{Puncture}_F(K,S)] = 1$$

• **Pseudorandom at punctured points:** For every QPT adversary  $(A_1, A_2)$ , there exists a negligible function negl such that the following holds. Consider an experiment where  $K \leftarrow \mathsf{KeyGen}_F(1^\lambda)$ ,  $(S, \sigma) \leftarrow A_1(1^\lambda)$ , and  $K_S \leftarrow \mathsf{Puncture}_F(K, S)$ . Then, for all  $x \in S$ ,

$$\left|\Pr[A_2(\sigma,K_S,S,F(K,x))=1] - \Pr_{r \leftarrow \{0,1\}^{m(\lambda)}}[A_2(\sigma,K_S,S,r)=1]\right| \leq \mathsf{negl}(\lambda)$$

**Constrained PRFs** A PRF  $F: \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$  has an additional key space  $\mathcal{K}_{\mathsf{Constrain}}$  and two additional algorithms  $F.\mathsf{Constrain}$  and  $F.\mathsf{ConstrainEval}$  as follows. A constrained key key  $K_C$  with respect to a circuit C enables the evaluation of F(K,x) for all x such that C(x)=1 and no other x.

KeyGen $(1^{\lambda}, 1^n) \to msk$ . On input the security parameter  $\lambda$ , outputs the master secret key msk. Eval $(msk, x) \to y \in \mathcal{Y}$ : On master secret key msk and value  $x \in \mathcal{X}$ , outputs the evaluation  $y \in \mathcal{Y}$ . Constrain $(msk, C) \to sk_C$ : takes as input a PRF key  $K \in \mathcal{K}$  and the description of a circuit C (so that domain of  $C \subseteq \mathcal{X}$ ); outputs a constrained key  $sk_C$ .

ConstrainEval( $\operatorname{sk}_C, x$ )  $\to y/\bot$ : On input a secret key  $\operatorname{sk}_C$ , and an input  $x \in \{0,1\}^n$ , the constrained evaluation algorithm PRF.ConstrainEval outputs an element  $y \in \{0,1\}^m$ .

**Definition 6.9** (Constrained PRF Correctness). A constrained PRF is correct for a circuit class C if msk  $\leftarrow$  PRF.KeyGen $(1^{\lambda})$ , for every circuit  $C \in C$  and input  $x \in \{0,1\}^n$  such that C(x) = 1, it is the case that:

$$F.\mathsf{ConstrainEval}(F.\mathsf{Constrain}(\mathsf{msk},C),x) = F.\mathsf{Eval}(\mathsf{msk},x).$$

**Definition 6.10** (Adaptive single-key constrained pseudorandomness). [Constrained PRF Security] We say that a cPRF satisfies adaptive single-key constrained pseudorandomness security if for any stateful admissible QPT adversary  $\mathcal{A}$  there exists a negligible function  $\operatorname{negl}(\cdot)$ , such that for all  $\lambda \in \mathbb{N}$ , the following holds:

$$\Pr\left[ \begin{matrix} \mathcal{A}^{\mathsf{Eval}(\mathsf{msk},\cdot),\mathsf{Constrain}(\mathsf{msk},\cdot)}(r_b) = b: & \underset{r_0}{\mathsf{msk}} \leftarrow \mathsf{KeyGen}(1^\lambda), \ b \leftarrow \{0,1\} \\ x \leftarrow \mathcal{A}^{\mathsf{Eval}(\mathsf{msk},\cdot),\mathsf{Constrain}(\mathsf{msk},\cdot)}(1^\lambda) \end{matrix} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Here the adversary A is said to be admissible as long as it satisfies the following conditions — (1) it makes at most one query to the constrain oracle Constrain(msk, ·), and its queried circuit C must be such that C(x) = 0, (2) it must send x as one of its evaluation queries to Eval(msk, ·).

We only need single-key security of the above definition for our use.

**Remark 6.11** (Double-challenge security). We will make a simple remark here on the following variant of the above security game: the adversary submits two arbitrarily chosen  $x_1, x_2$ ; the challenger chooses  $r_{1,0}, r_{1,1}$ , and  $r_{2,0}, r_{2,1}$  independently as in the above security game. A receives both  $r_{1,b_1}$  and  $r_{2,b_2}$  and has to guess both  $b_1, b_2$  correctly. The winning probability of any A in this "double-challenge" version of the game is upper bounded by the probability of it winning the single challenge game. We will make use of this fact later.

The type of constrained PRF we use in this work can be built from standard lattice assumptions ([BV15]) or alternatively from subexponentially-secure iO and OWFs [BLW17].

**Invertible PRFs** An invertible pseudorandom function (IPF) is an injective PRF whose inverse function can be computed efficiently (given the secret key).

Therefore, it has the following additional algorithm apart from the PRF KeyGen and Eval:

Invert(sk, y)  $\rightarrow x$ : on key sk and value  $y \in \mathcal{Y}$ , output a value  $x \in \mathcal{X} \cup \{\bot\}$ .

**Definition 6.12** (Correctness for Injective IPR). A invertible PRF is correct if for all msk  $\leftarrow$  PRF.KeyGen(1 $^{\lambda}$ ), it is the case that:

$$F$$
.Invert(sk,  $F$ .Eval(sk,  $x$ )) =  $x$ .

and F.Invert(sk, y) =  $\perp$  where y is not an image of any  $x \in \mathcal{X}$ .

In this paper, we only need the "regular" pseudorandomness property of the IPF and therefore we do not provide additional security definitions as in [BKW17]. We also do not need the IPF in our construction to be puncturable/constrainable.

In addition, we would like the IPF we use to act as extractors on their inputs:

**Definition 6.13** (Extracting PRF). An extracting PRF with error  $\epsilon(\cdot)$  for min-entropy  $k(\cdot)$  is a (puncturable) PRF F mapping  $n(\lambda)$  bits to  $\ell(\lambda)$  bits such that for all  $\lambda$ , if X is any distribution over  $n(\lambda)$  bits with min-entropy greater than  $k(\lambda)$ , then the statistical distance between  $(\mathsf{sk}, F(K, X))$  and  $(\mathsf{sk}, r \leftarrow \{0, 1\}^{\ell(\lambda)})$  is at most  $\epsilon(\cdot)$ , where  $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^{\lambda})$ .

The constrained PRFs and invertible PRFs used in this work can all be obtained from LWE or from subexponentially-secure iO plus OWFs. [SW14, BKW17, BV15, ?].

#### 6.1.5 One-Time Sampling Program for PRF: Indistinguishability Operational Definition

In this section, we present the strengthened version of the operational one-time sampling program security for PRFs, which is a specific case of Definition 4.21.

**Definition 6.14** (Indistinguishability Operational Security Definition for PRF One-Time Sampling Programs). *We define security through the following game:* 

- 1. The challenger samples  $sk \leftarrow \mathsf{PRF}.\mathsf{KeyGen}(1^\lambda)$  and prepares the program  $\mathsf{OTP}(\mathsf{PRF}.\mathsf{Eval}(sk,\cdot))$ .  $\mathcal{A}$  gets a copy of the one-time program for  $\mathsf{OTP}(\mathsf{PRF}.\mathsf{Eval}(sk,\cdot))$ .
- 2. A outputs two (input, randomness) pairs  $(x_1, r_1), (x_2, r_2)$ , which satisfies  $x_1 \neq x_2$  or  $r_1 \neq r_2$ , otherwise A loses.
- 3. Challenger samples two independent, uniform random bits  $b_1 \leftarrow \{0,1\}, b_2 \leftarrow \{0,1\}$ . If  $b_1 = 0$ , then let  $y_1 = \mathsf{PRF.Eval}(\mathsf{sk}, x_1, r_1)$ ; else let  $y_1 \leftarrow \{0,1\}^m$ . If  $b_2 = 0$ , then let  $y_2 = \mathsf{PRF.Eval}(\mathsf{sk}, x_2, r_2)$ ; else let  $y_2 \leftarrow \{0,1\}^m$ . Challenger sends  $(y_1, y_2)$  to  $\mathcal{A}$ .
- 4. A outputs guesses  $(b'_1, b'_2)$  for  $(b_1, b_2)$  respectively. A wins if and only if  $b'_1 = b_1$  and  $b'_2 = b_2$ .

We say that a one-time sampling program for PRF satisfies indistinguishability operational security if for any QPT adversary A, there exists a negligible function  $negl(\cdot)$ , such that for all  $\lambda \in \mathbb{N}$ , the following holds:

$$\Pr\left[A \text{ wins the above game }\right] \leq \frac{1}{2} + \operatorname{negl}(\lambda).$$

# 6.2 Construction and Security

We first give the following building blocks for our construction.

- 1. A constrained PRF  $F_1: \{0,1\}^{k_1} \times \{0,1\}^{n+\ell} \to \{0,1\}^m$ , where  $\{0,1\}^{k_1}$  is the key space and  $\{0,1\}^{n+\ell}$  is the input space. Let  $\mathsf{sk}_1 \leftarrow F_1$ . Key $\mathsf{Gen}(1^\lambda)$ .
- 2. An extracting invertible PRF  $F_2: \{0,1\}^{k_2} \times \{0,1\}^{n \cdot \lambda} \to \{0,1\}^{\ell}$ , where  $\{0,1\}^{k_2}$  is the key space and  $\{0,1\}^{\lambda}$  is the input space.  $\ell \geq n \cdot \lambda$  and  $\lambda$  is the security parameter. The PRF is extracting with negligible error for inputs for min-entropy  $n \cdot \lambda/2$ . Let  $\mathsf{sk}_2 \leftarrow F_2$ . KeyGen $(1^{\lambda})$ .
- 3. Sample n random subspaces  $A_1, \dots, A_n$  independently from  $\mathbb{F}_2^{\lambda}$ , where each  $\dim(A_i) = \lambda/2$ . Sample 2n random strings,  $s_1, s_1', \dots s_n, s_n'$  each uniformly random from  $\{0,1\}^{\lambda}$ . Prepare the coset subspace-hiding obfuscation programs  $\{(\mathsf{shO}(A_1+s_1), \mathsf{shO}(A_1^{\perp}+s_1'), \dots, (\mathsf{shO}(A_n+s_n), \mathsf{shO}(A_n^{\perp}+s_n'))\}$  as defined in Definition 6.6. For convenience, we will use the notation  $\mathsf{shO}_i^0$  for  $\mathsf{shO}(A_i+s_i)$ , and  $\mathsf{shO}_i^1$  for  $\mathsf{shO}(A_i+s_i')^{\perp}$  for the rest of this section.

The OTP(cPRF( $\mathsf{sk}_1, \cdot$ )) consists of the subspace states  $(|A_1\rangle, \cdots, |A_n\rangle)$  and an iO of the following program Figure 5:

```
Hardcoded: \mathsf{sk}_1, \mathsf{sk}_2, \{\mathsf{shO}_i^0, \mathsf{shO}_i^1\}_{i \in [n]}. On input (x \in \{0,1\}^n, u = u_1 || u_2 || \cdots || u_n \in \{0,1\}^{n \cdot \lambda}) (where each u_i \in \mathbb{F}_2^{\lambda}):

1. If for all i \in [n], \mathsf{shO}_i^{x_i}(u_i) = 1, where x_i is the i-th bit of x:

Let r \leftarrow F_2. Eval(\mathsf{sk}_2, u).

Output (r, F_1. Eval(\mathsf{sk}_1, x || r)).

2. Else:

Output \bot
```

Figure 5: Program cPRF<sub>OTP</sub>

**Correctness** The correctness follows from the extracting property of the PRF  $F_2$ : in any honest evaluation with  $u \in \{0,1\}^{\lambda \cdot n}$  satisfies  $u_i \in \mathsf{shO}_{A_i}^{x_i}$  for each  $i \in [n]$ . Therefore u has min-entropy  $n \cdot \lambda/2$ . By the extracting property of  $F_2$  and the evaluation correctness of  $F_1$ , the above scheme satisfies correctness Definition 4.2.

#### **6.2.1** Security Proof

To prove security, we consider the following hybrids:

 $H_0$ : In this hybrid, the challenger plays the original game defined in Definition 6.14 using the above construction:

- 1. The challenger prepares the program  $\mathsf{OTP}(F_1.\mathsf{Eval}(\mathsf{sk}_1,\cdot))$  as in Section 6.2.  $\mathcal{A}$  gets a copy of the one-time program for  $\mathsf{OTP}(F_1.\mathsf{Eval}(\mathsf{sk}_1,\cdot))$ .
- 2. A outputs two (input, randomness) pairs  $(x_1, r_1), (x_2, r_2)$  such that  $x_1 \neq x_2$  or  $r_1 \neq r_2$ .
- 3. Challenger samples two independent, uniform random bits  $b_1 \leftarrow \{0,1\}, b_2 \leftarrow \{0,1\}$ . If  $b_1 = 0$ , then let  $y_1 = F_1$ . Eval(sk<sub>1</sub>,  $x_1, r_1$ ); else let  $y_1 \leftarrow \{0,1\}^m$ . If  $b_2 = 0$ , then let  $y_2 = F_1$ . Eval(sk<sub>1</sub>,  $x_2, r_2$ ); else let  $y_2 \leftarrow \{0,1\}^m$ . Challenger sends  $(y_1, y_2)$  to  $\mathcal{A}$ .
- 4. A outputs guesses  $(b'_1, b'_2)$  for  $(b_1, b_2)$  respectively. A wins if and only if  $b'_1 = b_1$  and  $b'_2 = b_2$ .

 $H_1$ : In this hybrid, the challenger modifies the original OTP program in Figure 5 and the game as follows:

- 1. The challenger prepares the program  $OTP(F_1.Eval(sk_1, \cdot))$  as follows:
  - (a) Sample msk  $\leftarrow F_1$ .KeyGen $(1^{\lambda})$ , sk $_2 \leftarrow F_2$ .KeyGen $(1^{\lambda})$ . Sample the subspaces and prepares the  $\{\operatorname{shO}_{A_i}^b\}_{i\in[n],b\in\{0,1\}}$  programs.
  - (b) Compute constrained key  $\mathsf{sk}_A \leftarrow F_1.\mathsf{Constrain}(\mathsf{msk}, C_A)$  for the following circuit  $C_A$ :
  - (c) Prepare the iO-ed classical program in the  $OTP(F_1(\cdot))$  scheme as in Figure 7, using the constrained key  $sk_A$  for  $F_1$ :

```
Hardcoded: \operatorname{sk}_2, \{\operatorname{shO}_i^0,\operatorname{shO}_i^1\}_{i\in[n]}.

On input (x\in\{0,1\}^n,r\in\{0,1\}^\ell):

i. Compute (u=u_1||u_2||\cdots||u_n\in\{0,1\}^{n\cdot\lambda})\leftarrow F_2. Invert(\operatorname{sk}_2,r), where each u_i\in\mathbb{F}_2^\lambda.

If the inversion result is \bot, output 0.

ii. If for all i\in[n], \operatorname{shO}_i^{x_i}(u_i)=1, where x_i is the i-th bit of x:

Output 1.

iii. Else:

Output 0
```

**Figure 6:** Constraint Circuit  $C_A$ 

```
Hardcoded: \mathsf{sk}_A, \mathsf{sk}_2, C_A.

On input (x \in \{0,1\}^n, u = u_1 || u_2 || \cdots || u_n \in \{0,1\}^{n \cdot \lambda})

(where each u_i \in \mathbb{F}_2^{\lambda}):

i. Let r \leftarrow F_2. Eval(\mathsf{sk}_2, u).

ii. Output (r, F_1. Constrain Eval(\mathsf{sk}_A, x || r)).
```

**Figure 7:** Program cPRF<sub>OTP</sub> in  $H_1$ 

- 2. A outputs two (input, randomness) pairs  $(x_1, r_1), (x_2, r_2)$  such that  $x_1 \neq x_2$  or  $r_1 \neq r_2$ .
- 3. Challenger samples two independent, uniform random bits  $b_1 \leftarrow \{0,1\}, b_2 \leftarrow \{0,1\}$ . If  $b_1 = 0$ , then let  $y_1 = F_1$ . Eval(sk<sub>1</sub> = msk,  $x_1, r_1$ ); else let  $y_1 \leftarrow \{0,1\}^m$ . If  $b_2 = 0$ , then let  $y_2 = F_1$ . Eval(sk<sub>1</sub> = msk,  $x_2, r_2$ ); else let  $y_2 \leftarrow \{0,1\}^m$ . Challenger sends  $(y_1, y_2)$  to  $\mathcal{A}$ .
- 4.  $\mathcal{A}$  outputs guesses  $(b'_1, b'_2)$  for  $(b_1, b_2)$  respectively.  $\mathcal{A}$  wins if and only if  $b'_1 = b_1$  and  $b'_2 = b_2$ .

 $H_2$ : In this hybrid, all steps are the same except in step 2, the challenger additionally checks if  $F_2$ .Invert(sk<sub>2</sub>,  $r_1$ ) and  $F_2$ .Invert(sk<sub>2</sub>,  $r_2$ ) are in the subspaces with respect to  $x_1, x_2$ : if so, abort the game and  $\mathcal A$  loses.

- 1. The challenger prepares the program as in Hybrid 1.
- 2.  $\mathcal{A}$  outputs two (input, randomness) pairs  $(x_1, r_1), (x_2, r_2)$  such that  $x_1 \neq x_2$  or  $r_1 \neq r_2$ . Challenger makes the following check on  $(x_1, r_1), (x_2, r_2)$ :
  - (a) Compute  $(u_1 = u_{1,1}||u_{1,2}||\cdots||u_{1,n} \in \{0,1\}^{n\cdot\lambda}) \leftarrow F_2.\mathsf{Invert}(\mathsf{sk}_2,r_1)$ , where each  $u_{1,i} \in \mathbb{F}_2^\lambda$ . Similarly compute  $u_2 \leftarrow F_2.\mathsf{Invert}(\mathsf{sk}_2,r_2)$ . If one of the inversion results is  $\bot$ , continue to step 3.
  - (b) If for both  $j \in \{1, 2\}$ , for all  $i \in [n]$ ,  $\mathsf{shO}_i^{x_{j,i}}(u_{j,i}) = 1$ , where  $x_{j,i}$  is the i-th bit of  $x_j$ : abort and output 0.
  - (c) Else if there exists one of  $j \in \{1, 2\}$ ,  $i \in [n]$  such that  $\mathsf{shO}_i^{x_{j,i}}(u_{j,i}) = 0$ , then continue to step 3.

- 3. Challenger samples two independent, uniform random bits  $b_1 \leftarrow \{0,1\}, b_2 \leftarrow \{0,1\}$ . If  $b_1 = 0$ , then let  $y_1 = F_1$ . Eval( $\mathsf{sk}_1, x_1, r_1$ ); else let  $y_1 \leftarrow \{0,1\}^m$ . If  $b_2 = 0$ , then let  $y_2 = F_1$ . Eval( $\mathsf{sk}_1, x_2, r_2$ ); else let  $y_2 \leftarrow \{0,1\}^m$ . Challenger sends  $(y_1, y_2)$  to  $\mathcal{A}$ .
- 4.  $\mathcal{A}$  outputs guesses  $(b'_1, b'_2)$  for  $(b_1, b_2)$  respectively.  $\mathcal{A}$  wins if and only if  $b'_1 = b_1$  and  $b'_2 = b_2$ .

**Claim 6.15.** Assuming the post-quantum security of iO, The difference between A's advantage in  $H_0$  and  $H_1$  is negligible.

*Proof.* The program cPRF<sub>OTP</sub> in  $H_0$  and  $H_1$  have the same functionality: we only change the time of when we check the input vectors u are in the corresponding subspaces. Therefore, by the security of iO, the above claim holds.

**Claim 6.16.** By the security of subspace-hiding obfuscation (*Definition 6.3*), the difference between A's advantage in  $H_1$  and  $H_2$  is negligible.

*Proof.* We invoke the computational direct product hardness Theorem 6.7: when giving a subspace state, and subspace-hiding obfuscations for the corresponding primal and dual subspaces, it is hard to produce two different vectors in the subspaces. Therefore, the event that challenger aborts on the event in Step 2 of Hybrid 1 is negligible.

Otherwise, if all the preimages of  $r_1, r_2$  are valid subspace vectors, since we require  $x_1 \neq x_2$  or  $r_1 \neq r_2$ , there must exist at least an index  $i^* \in [n]$  such that  $u_{1,i^*} \neq u_{2,i^*}$ . Therefore, we can build a reduction to break the computational direct product hardness property: the reduction can sample its own msk, sk<sub>2</sub> and constrain the key msk on the circuit  $C_A$  since it is given the programs  $\operatorname{shO}_{A_i}$ 's. When receiving the adversary's output of  $r_1, r_2$ , it can invert them to find the vectors  $u_{1,i^*}, u_{2,i^*}$  that help it break Theorem 6.7.

**Claim 6.17.** Assuming adaptive single-key constrained pseudorandomness of cPRF $F_1$ , then A's advantage in  $H_2$  is negligible.

*Proof.* If there exists an A that wins the game in  $H_2$  with probability non-negligibly larger than 1/2, then we can build a reduction B to break the adaptive single-key constrained pseudorandomness in Definition 6.10.

Note that in this game, we have ruled out all  $\mathcal{A}$  that outputs  $(x_1, r_1), (x_2, r_2)$  that satisfies  $C_A(x_1, r_1) = C_A(x_2, r_2) = 1$ . That is, at least one of the above evaluations is 0. Therefore, the reduction, which makes a single key query on the circuit  $C_A$ , can use this input as the challenge input to the Definition 6.10 security game. If both inputs satisfy that  $C_A(x_1, r_1) = C_A(x_2, r_2) = 0$ , then we use the variant game in Remark 6.11. In either case, the security of the constrained PRF guarantees that  $\mathcal{A}$ 's winning probability in the game of  $H_2$  is  $1/2 + \text{negl}(\lambda)$ .

# 7 Impossibility Results in the Plain Model and the Oracle Model

In this section, we provide two different infeasibility results for one-time sampling programs that complement our positive results from two perspectives:

1. Assuming LWE, there exists a family of one-query unlearnable, high min-entropy output functions where there are no insecure OTP for it *in the plain model*, even with respect to the weak operational definition Definition 4.26.

2. There exists a family of single *physical* query unlearnable, high average min-entropy output, but partially deterministic functions where there are no insecure OTP for it, even in the oracle model, with respect to the weakest operational definition Definition 4.26.

The first result in the non-black-box model is inspired by the non-black-box impossibility result of quantum obfuscation and copy protection in [ABDS20, AP21]. However, the circuit family they use is a deterministic one. In order to show a nontrivial result for the one-time program, we design a family of randomized circuits which have almost full entropy output, but can nevertheless be "learned" through a single non-black-box evaluation.

# 7.1 Preliminaries

#### 7.1.1 Quantum Fully Homomorphic Encryption Scheme

We give the definition of the type of QFHE we need for the construction in this section.

**Definition 7.1** (Quantum Fully Homomorphic Encryption). Let  $\mathcal{M}$  be the Hilbert space associated with the message space (plaintexts),  $\mathcal{C}$  be the Hilbert space associated with the ciphertexts, and  $\mathcal{R}_{ek}$  be the Hilbert space associated with the evaluation key. A quantum fully homomorphic encryption scheme is a tuple of QPT algorithms QHE = (KeyGen, Enc, Dec, Eval) satisfying:

KeyGen( $1^{\lambda}$ ): a classical probabilistic algorithm that outputs a public key, a secret key as well as an evaluation key, (pk, sk, ek).

 $\mathsf{Enc}(\mathsf{pk}, \rho_{\mathcal{M}})$ : takes as input a state  $\rho_{\mathcal{M}}$  in the space  $L(\mathcal{M})$  and outputs a ciphertext  $\sigma$  in  $L(\mathcal{C})$ .

 $\mathsf{Dec}(\mathsf{sk}, \sigma)$ : takes a quantum ciphertext  $\sigma$ , and outputs a state  $\rho_{\mathcal{M}}$  in the message space  $L(\mathcal{M})$ .

Eval(ek,  $U, \sigma_1, \dots, \sigma_k$ ) takes input of a quantum citcuit U with k-qubits input and k'-qubits of outputs. Its output is a sequence of k' quantum ciphertexts.

The semantic security is analogous to the classical semantic security of FHE. We refer to [BJ15].

Classical Ciphertexts for Classical Plaintexts For the impossibility result, we require a QFHE scheme where ciphertexts of classical plaintexts are also classical. Given any  $x \in \{0,1\}$ , we want QHE.Enc(pk,  $|x\rangle\langle x|$  to be a computational basis state  $|z\rangle\langle z|$  for some  $z\in\{0,1\}^l$  (here, l is the length of ciphertexts for 1-bit messages). In this case, we write QHE.Enc(pk,x). We also want the same to be true for evaluated ciphertexts: if  $U|x\rangle\langle x|=|y\rangle\langle y|$  for some basis state  $x\in\{0,1\}^n,y\in\{0,1\}^l$ , then we have: QHE.Eval(ek, U, QHE.Enc(pk,  $|x\rangle\langle x|$ ))  $\to$  QHE.Enc(pk,  $|y\rangle\langle y|$ ) where the result is a classical ciphertext.

The QFHE schemes in [Bra18, Mah20] satisfy the above requirement.

Note that we also need to evaluate on a possibly arbitrary polynomial depth circuit. The QFHE schemes in [Bra18, Mah20] still require circular security to go beyond leveled FHE.

#### 7.1.2 Compute-and-Compare Obfuscation

**Definition 7.2** (Compute-and-Compare Program). Given a function  $f: \{0,1\}^{\ell_{\text{in}}} \to \{0,1\}^{\ell_{\text{out}}}$  along with a target value  $y \in \{0,1\}^{\ell_{\text{out}}}$  and a message  $z \in \{0,1\}^{\ell_{\text{msg}}}$ , we define the compute-and-compare program:

$$\mathsf{CC}[f,y,z](x) = \begin{cases} z & \textit{if } f(x) = y \\ \bot & \textit{otherwise} \end{cases}$$

We define the following class of *unpredictable distributions* over pairs of the form (CC[f, y, z], aux), where aux is auxiliary quantum information. These distributions are such that y is computationally unpredictable given f and aux.

**Definition 7.3** (Unpredictable Distributions). We say that a family of distributions  $D = \{D_{\lambda}\}$  where  $D_{\lambda}$  is a distribution over pairs of the form (CC[f, y, z], aux) where aux is a quantum state, belongs to the class of unpredictable distributions if the following holds. There exists a negligible function negl such that, for all QPT algorithms A,

$$\Pr_{(\mathsf{CC}[f,y,z],\mathsf{aux}) \leftarrow D_{\lambda}} \left[ A(1^{\lambda},f,\mathsf{aux}) = y \right] \leq \mathsf{negl}(\lambda).$$

We assume that a program *P* has an associated set of parameters *P*.param (e.g input size, output size, circuit size, etc.), which we are not required to hide.

**Definition 7.4** (Compute-and-Compare Obfuscation). A PPT algorithm CC.Obf is an obfuscator for the class of unpredictable distributions (or sub-exponentially unpredictable distributions) if for any family of distributions  $D = \{D_{\lambda}\}$  belonging to the class, the following holds:

• Functionality Preserving: there exists a negligible function negl such that for all  $\lambda$ , every program P in the support of  $D_{\lambda}$ ,

$$\Pr[\forall x, \, \widetilde{P}(x) = P(x), \, \widetilde{P} \leftarrow \mathsf{CC.Obf}(1^{\lambda}, P)] \geq 1 - \mathsf{negl}(\lambda)$$

• Distributional Indistinguishability: there exists an efficient simulator Sim such that:

$$(\mathsf{CC.Obf}(1^{\lambda}, P), \mathsf{aux}) \approx_c (\mathsf{Sim}(1^{\lambda}, P.\mathsf{param}), \mathsf{aux})$$

where 
$$(P, \mathsf{aux}) \leftarrow D_{\lambda}$$
.

Combining the results of [WZ17, GKW17] with those of [Zha16], we have the following two theorems.

**Theorem 7.5.** Assuming the existence of the quantum hardness of LWE, there exist obfuscators for unpredictable distributions, as in Definition 7.4.

### 7.1.3 Quantum Query Lower Bounds

In the analysis of the unlearnability of circuits (Section 7), we will use the following theorem from [BBBV97a] to bound the change in adversary's state when we change the oracle's input-output behavior at places where the adversary hardly ever queries on.

Let  $|\phi_i\rangle$  be the state of the adversary after the *i*-th query to the oracle  $\mathcal{O}$ , i.e.  $|\phi_i\rangle = U_i\mathcal{O}\cdots\mathcal{O}U_2\mathcal{O}U_1 |\phi_0\rangle$ , where  $|\phi_0\rangle$  is the initial adversary state.

**Theorem 7.6** ([BBBV97a]). We have defined the quantum query model for classical oracles in Section 3.2. We give some further preliminaries here.

Let  $|\phi_i\rangle$  be the superposition of oracle quantum algorithms  $\mathcal{M}$  with oracle  $\mathcal{O}$  on input x at time i. Define  $W_y(|\phi_i\rangle)$  to be the sum of squared magnitudes in  $|\phi_i\rangle$  of configurations of  $\mathcal{M}$  which are querying the oracle on string y. For  $\epsilon>0$ , let  $F\subseteq [0,T-1]\times \Sigma^*$  be the set of time-string pairs such that  $\sum_{(i,y)\in F}W_y(|\phi_i\rangle)\leq \epsilon^2/T$ .

Now suppose the answer to each query  $(i,y) \in F$  is modified to some arbitrary fixed  $a_{i,y}$  (these answers need not be consistent with an oracle). Let  $|\phi_i'\rangle$  be the superposition of  $\mathcal M$  on input x at time i with oracle  $\mathcal O$  modified as stated above. Then  $|||\phi_T\rangle - |\phi_T'\rangle||_{\mathrm{tr}} \leq \epsilon$ .

# 7.2 Impossibility Result for Single-Query Security in the Plain Model: for fully randomized functions

In this section, we present a lower bound/impossibility result for a generic one-time program in the plain model (i.e. without using black-box oracles). The result states that there is no way to construct a generic one-time sampling program for *all randomized functionalities* if the programs allow *non-black-box* access.

Our result is inspired by the non-black-box impossibility result of quantum obfuscation and copy protection in [ABDS20, AP21]. However, the circuit family they use is a deterministic one. In order to show a nontrivial result for the one-time program, we design a family of randomized circuits which have almost full entropy output, but can nevertheless be "learned" through a single non-black-box evaluation.

**Theorem 7.7.** Assuming the post-quantum security of LWE and QFHE, there exists a family of randomized circuits which are single-query  $negl(\lambda)$ -unlearnable Definition 4.11, but not one-time program secure with respect to the the weak operational one-time security Definition 4.26.

We construct the following circuit which has high-entropy outputs and is 1-query unlearnable with only (quantum) single-query access to the function and access to a piece of classical information, but once put into any OTP in the plain model, is insecure.

First we give a few building blocks for the following circuit *C*.

- 1. Let  $n = n(\lambda), \ell = \ell(\lambda)$  be polynomials in the security parameter  $\lambda$ .
- 2. Let SKE = (SKE.KeyGen, SKE.Enc, SKE.Dec) be any secret key encryption scheme. The SKE scheme only needs to satisfy relatively weak security notion to be single-query unlearnable. For the sake of convenience, we use the textbook construction of post-quantum IND-CPA secure SKE from PRFs ([Zha12a]):

```
\begin{aligned} \mathsf{SKE}.\mathsf{KeyGen}(1^\lambda) : \mathsf{sk} \leftarrow \mathsf{PRF}.\mathsf{KeyGen}(\lambda) \\ \mathsf{SKE}.\mathsf{Enc}(\mathsf{sk}, m \in \{0,1\}^n) \rightarrow \mathsf{ct} : \mathsf{samples} \ r \leftarrow \{0,1\}^\ell; \mathsf{output} \ \mathsf{ct} \leftarrow (r, \mathsf{PRF}.\mathsf{Eval}(\mathsf{sk}, r) \oplus m). \\ \mathsf{SKE}.\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) : \mathsf{parse} \ \mathsf{ct} := (r, \mathsf{ct}'); \mathsf{compute} \ m := \mathsf{PRF}.\mathsf{Eval}(\mathsf{sk}, r) \oplus \mathsf{ct}'. \end{aligned}
```

Let  $SKE.sk \leftarrow SKE.KeyGen(1^{\lambda})$  be the key we use in the following circuit construction.

- 3. Let QHE = (QHE.KeyGen, QHE.Enc, QHE.Dec, QHE.Eval) be a quantum fully homomorphic encryption scheme. Let the keys be (QHE.pk, QHE.sk)  $\leftarrow$  QHE.KeyGen( $1^{\lambda}$ ). Without loss of generality, we consider the evaluation key to be part of QHE.pk.
- 4. Let CC = CC.Obf be a compute-and-compare obfuscation scheme in Section 7.1.2.
- 5. Let  $a \leftarrow \{0,1\}^n$ ,  $b \leftarrow \{0,1\}^n$  be two uniformly random strings.

We design the following with auxiliary information  $\mathsf{aux} = (\mathsf{ct}_a = \mathsf{QHE}.\mathsf{Enc}(\mathsf{QHE}.\mathsf{pk}, a); \tilde{P} = \mathsf{CC.Obf}(\mathsf{SKE}.\mathsf{Dec}(\mathsf{SKE}.\mathsf{sk}, \mathsf{QHE}.\mathsf{Dec}(\mathsf{QHE}.\mathsf{sk}, \cdot)), b, (\mathsf{SKE}.\mathsf{sk}, \mathsf{QHE}.\mathsf{sk})), \mathsf{QHE}.\mathsf{pk}).$ 

```
\begin{array}{l} \text{Input: } (x \in \{0,1\}^n, r \in \{0,1\}^\ell) \\ \text{Hardcoded: } (a,b,\mathsf{Enc.sk}) \\ \text{if } x = a \text{:} \\ \text{output SKE.Enc}(\mathsf{Enc.sk},b;r) \end{array}
```

else:

output SKE.Enc(Enc.sk, x; r).

Note that the program  $\tilde{P}$  in the auxiliary information aux is a compute-and-compare obfuscation program of the following circuit:

$$\mathsf{CC}[f, (\mathsf{SKE}.\mathsf{sk}, \mathsf{QHE}.\mathsf{sk}), b] = \begin{cases} (\mathsf{SKE}.\mathsf{sk}, \mathsf{QHE}.\mathsf{sk}) & \text{if } f(x) = b \\ \bot & \text{otherwise} \end{cases}$$

where f(x) = SKE.Dec(SKE.sk, QHE.Dec(QHE.sk, x)).

**Claim 7.8.** The above circuit with auxiliary information (C, aux) can be perfectly reconstructed by any QPT adversary given any one-time program with correctness of the above circuit.

*Proof.* Given any OTP  $|\psi_C\rangle$  of the circuit C together with the auxiliary information aux. A QPT adversary can perform the following attack:

- 1. Encrypt the program:  $\operatorname{ct}_{|\psi_C\rangle} \leftarrow \operatorname{QHE.Enc}(\operatorname{QHE.pk}, |\psi_C\rangle)$  using the QFHE public key QHE.pk given in aux.
- 2. Homomorphically evaluate the program on the input  $ct_a = QHE.Enc(QHE.pk, a)$  from aux, with respect to a universal quantum circuit U, to obtain an outcome  $ct_{SKE.Enc(b)}$ :

 $\mathsf{ct}_{\mathsf{SKE}.\mathsf{Enc}(b)} := \mathsf{QHE}.\mathsf{Enc}(\mathsf{QHE.pk},\mathsf{SKE}.\mathsf{Enc}(\mathsf{SKE.sk},b;r_b)) \leftarrow \mathsf{QHE}.\mathsf{Eval}(\mathsf{QHE.pk},U,\mathsf{ct}_{|\psi_C\rangle},\mathsf{ct}_a).$  for some random  $r_b$ .

The above evaluation holds due to the correctness of OTP scheme and the QFHE scheme: when one evaluates  $U(|\psi_C\rangle,a)$  honestly, then one obtains SKE.Enc(SKE.sk,  $b;r_b$ ) for some random classical string  $r_b$ . Therefore, by the correctness of QFHE, we obtain the a QFHE ciphertext QHE.Enc(QHE.pk, SKE.Enc(SKE.sk,  $b;r_b$ )). This evaluation procedure is randomized and the original OTP state  $|\psi_C\rangle$  gets destroyed during the procedure.

Since the message SKE. $\operatorname{Enc}(b; r_b)$  is classical, the ciphertext  $\operatorname{ct}_{\mathsf{SKE.Enc}(b)}$  under QFHE is classical by the property of the QFHE we use.

- 3. Evaluate the compute-and-compare obfuscation program  $\tilde{P}$  on input  $\mathsf{ct}_{\mathsf{SKE}.\mathsf{Enc}(b)}$ . Note that by the correctness of the compute-and-compare obfuscation program, the input  $\mathsf{ct}_{\mathsf{SKE}.\mathsf{Enc}(b)}$  satisfies that  $f(\mathsf{ct}_{\mathsf{SKE}.\mathsf{Enc}(b)}) = \mathsf{SKE}.\mathsf{Dec}(\mathsf{SKE}.\mathsf{sk},\mathsf{QHE}.\mathsf{Dec}(\mathsf{QHE}.\mathsf{sk},\mathsf{ct}_{\mathsf{SKE}.\mathsf{Enc}(b)})) = b$ .
  - Therefore, one will obtain the information (SKE.sk, QHE.sk).
- 4. Now one can first decrypt the QFHE ciphertext  $\operatorname{ct}_a = \operatorname{QHE.Enc}(\operatorname{QHE.pk}, a)$  to obtain a and the doubly-encrypted ciphertext  $\operatorname{ct}_{\mathsf{SKE.Enc}(b)}$  to obtain b, with keys QHE.sk, SKE.sk. Given the above information, one can fully reconstruct the circuit C together with all the auxiliary information in aux perfectly (note that the information in aux are classical and can be copied and kept in the first place).

**Remark 7.9.** Note that the above construction actually shows a stronger statement than an infeasibility result of OTP: it lets a QPT adversary recover the entire circuit perfectly, which obviously allows it to violate the OTP security. But if we only need the adversary to output two input-output pairs, storing SKE.sk as the secret message in the compute-and-compare program suffices.

**Claim 7.10.** Assuming the post-quantum security of LWE, the above circuit with auxiliary information (C, aux) satisfies single-query unlearnability (Definition 4.11, for both physical and effective queries).

*Proof.* We will prove through a sequence of hybrids to show that the oracle is indistinguishable from a regular SKE functionality, which is single-query unlearnable.

The proof is similar to the proof for Claim 47 in [AP21], with some modifications. We directly use the [BBBV97b] argument instead of adversary method.

Let  $|\phi_i\rangle$  be the state of the adversary after the *i*-th query to the oracle  $\mathcal{O}_C$  (quantum black-box access to the circuit C), i.e.  $|\phi_i\rangle = U_i\mathcal{O}_C\cdots\mathcal{O}_CU_2\mathcal{O}_CU_1|\phi_0\rangle$ , where  $|\phi_0\rangle$  is the initial adversary state. We first make the following claim:

**Claim 7.11.** Assuming the post-quantum security of LWE, the sum of squared amplitudes of query on strings starting with  $a: \sum_{i=1}^{T} W_a(|\phi_i\rangle) \leq \text{negl}(\lambda)$ , where T is the total number of steps.

*Proof.* We prove this by induction and the security properties of QFHE and CC.Obf.

**Base case**: before the adversary makes the first query, clearly  $W_a(|\phi_0\rangle)$  is negligible (in fact 0 here), we consider the following hybrids:

- 1.  $H_0$ : this is the original game where we give out auxiliary information aux =  $(\mathsf{ct}_a = \mathsf{QHE}.\mathsf{Enc}(\mathsf{QHE.pk}, a); \tilde{P} = \mathsf{CC.Obf}(\mathsf{SKE.Dec}(\mathsf{SKE.sk}, \mathsf{QHE.Dec}(\mathsf{QHE.sk}, \cdot)), b, (\mathsf{SKE.sk}, \mathsf{QHE.sk})), \mathsf{QHE.pk}).$
- 2.  $H_1$ : reprogram the oracle  $\mathcal{O}_C$  to have the functionality in Figure 8.

**Figure 8:**  $\mathcal{O}'_C$  in  $H_1$ 

```
Input: (x \in \{0,1\}^n, r \in \{0,1\}^\ell)
Hardcoded: Enc.sk
output SKE.Enc(Enc.sk, x; r).
```

- 3.  $H_2$ : replace  $\tilde{P}$  in the above aux with  $Sim(1^{\lambda}, P.param)$ .
- 4.  $H_3$ : replace  $ct_a = QHE.Enc(QHE.pk, a)$  with  $ct_0 = QHE.Enc(QHE.pk, 0^n)$ .

The adversary's state  $|\phi_0\rangle$  should have negligible difference in terms of trace distance by [BBBV97a] (by plugging T=0) in  $H_0$  and  $H_1$ .

Let the projection  $\Pi_a := (|a\rangle \langle a|_x \otimes \mathbf{I}_{r,\mathcal{A}})$ , where x, r are the registers corresponding to the input (x, r) to  $\mathcal{O}_C$  and  $\mathcal{A}$  represents the rest of registers in the adversary's state.

We can measure the adversary's first query by projecting the state  $|\phi_0\rangle$  onto  $\mathcal{O}_c U_1 \Pi_a(\mathcal{O}_c U_1)^{\dagger}$ . The adversary with state  $|\phi_0\rangle$  should have negligible difference in query weight on a for the first query in  $H_1$  and  $H_2$ : since b is sampled uniformly at random and for the adversarial state  $U_1 |\phi_0\rangle$ , b satisfies the unpredictable distribution property in Definition 7.3. Therefore, by the property of Definition 7.4, any measurement in the adversary's behaviors in  $H_1$  and  $H_2$  should result in computationally indistinguishable outcomes.

In more details, the reduction to the compute-and-compare security works as follows: since the oracle  $\mathcal{O}_C$  is now independent of a,b, the reduction can sampleSKE, QHE keys, prepare the oracle  $\mathcal{O}_C$  and sample its own a; it then receives the obfuscated compute-and-compare program  $\tilde{P}$  (or the simulated program  $\mathrm{Sim}(1^{\lambda},1^{|f|})$ ) from the challenger, where b is uniformly random. If

the measurement  $\mathcal{O}_c U_1 \Pi_a (\mathcal{O}_c U_1)^{\dagger}$  on adversary's first query gives outcome 1, then output "real program", else output "simulated program".

The query weights  $H_2$  and  $H_3$  should have negligible difference by the security of the QFHE. Since the program  $\tilde{P}$  has been replaced with a simulated program, the QHE reduction can prepare the programs as well as the oracle  $\mathcal{O}_C$ . It receives  $\mathsf{ct}_a$  or  $\mathsf{ct}_0$  from the challenger. If the measurement on the adversary's first query returns 1, then guess  $\mathsf{ct}_a$ , else guess  $\mathsf{ct}_0$ .

The adversary's first query weight on a in  $H_3$  is negligible since now there is no information about a anywhere in aux and a is only a uniform random string in  $\{0,1\}^n$ . By the above arguments, the adversary's first query's weight on a is negligible in the original game  $H_0$ .

**Induction**: the above argument applies to the k-th query, if the sum of squared amplitudes over the first (k-1) queries,  $\sum_i^{k-1} W_a(|\phi_i\rangle)$ , is negligible, then we can invoke the above arguments and show that  $W_a(|\phi_k\rangle)$  is negligible as well.

Since we have shown that the total (squared) query weight on a,  $\sum_i^T W_a(|\phi_i\rangle)$  is negligible, we can replace the oracle  $\mathcal{O}_C$  for the entire game with the oracle  $\mathcal{O}_C'$  in the above  $H_1$ , i.e. Figure 8 and by [BBBV97a], the trace distance between  $|\phi_T\rangle$  using the original oracle  $\mathcal{O}_C$  and  $|\phi_T\rangle$  using the oracle  $\mathcal{O}_C'$  in Figure 8 is negligible. Now it remains to show that  $\mathcal{O}_C'$  together with aux is single-query unlearnable for any QPT adversary.

By similar hybrids as above, we can replace the information in aux with a dummy program and dummy ciphertext so that aux =  $(\mathsf{ct}_0 = \mathsf{QHE}.\mathsf{Enc}(\mathsf{QHE}.\mathsf{pk},0^n),\mathsf{Sim}(1^\lambda,1^{|f|}),\mathsf{QHE}.\mathsf{pk})$ . The adversary's advantage in the unlearnability game should have negligible difference by similar hybrid arguments.

Now recall that we instantiate SKE from PRF using the textbook construction for IND-CPA SKE. We can then show that if there exists an adversary that violates the single-query unlearnability of a PRF that maps  $2 \cdot |R|$ -length inputs to |m|-length inputs (which is essentially a random oracle when accessed in the oracle model and thus it satisfies single-query unlearnability and high min-entropy outputs): the reduction can simulate the oracle  $\mathcal{O}'_C$  on query x by querying a PRF oracle on some random string  $r_1$  of its own choice; the PRF oracle will return  $(r_2, \mathsf{PRF}(\mathsf{sk}, r_1||r_2))$ , where  $r_2$  is the randomness chosen by the randomized PRF oracle itself; the reduction then replies the adversary with  $(r_1||r_2, \mathsf{PRF}(\mathsf{sk}, r_1||r_2) \oplus m)$ . In the end, if the adversary wins by outputting two pairs  $(m, r, \mathsf{PRF}(\mathsf{sk}, r) \oplus m)$  and  $(m', r', \mathsf{PRF}(\mathsf{sk}, r') \oplus m')$ , then the reduction outputs  $(r, \mathsf{PRF}(\mathsf{sk}, r))$  and  $(r', \mathsf{PRF}(\mathsf{sk}, r'))$  and would break the single-query unlearnability of the PRF (see Remark 4.12 and Section 6).

# 7.3 Impossibility Result for Partially Randomized Functions in the Oracle Model

In this section, we give an example of a function family that cannot be compiled into a one-time program under even the weakest definition of operational security, even in the classical oracle model. It is known that deterministic functions fall into this category, but this counterexample is not only randomized but also has high entropy in a very strong sense. It is however *partially* deterministic: the (high) entropy is restricted to one half of the output and the other half is essentially deterministic, demonstrating that high entropy is not sufficient for a function to be one-time programmable.

Moreover, this example also demonstrates the following

1. It separates a single-physical-query unlearnable function from single-effective-query unlearnable

2. It is a single-physical-query unlearnable function that cannot be securely one-time programmed with respect to the classical-output simulation definition Definition 4.6.

Suppose PRF is a length-preserving pseudorandom function that is secure against adversaries who are allowed to make quantum superposition queries. Let a be a uniformly random string in  $\{0,1\}^n$ . Let k be a random PRF key. Consider the function family  $\mathcal{F}_n = \{f_{a,k}\}_{a,k\in\{0,1\}^n}$ , where  $f_{a,k}:\{0,1\}^n\times\{0,1\}^n\to\{0,1\}^{2n}$  is defined as

$$f_{a,k}(x;r) = \begin{cases} (a, \mathsf{PRF}_k(0||r)) & \text{if } x = 0, \\ (k, \mathsf{PRF}_k(a||r)) & \text{if } x = a, \\ (0, \mathsf{PRF}_k(x||r)) & \text{otherwise.} \end{cases}$$
 (6)

Also define an associated distribution  $\mathcal{D}$  over this function family such that  $a \leftarrow \{0,1\}^n$  is chosen uniformly at random, and the key k is sampled according to the PRF key-generation procedure.

First, we establish that this function family cannot be compiled into a one-time program even under the weak operational security definition.

**Lemma 7.12.**  $\mathcal{F}$  cannot be compiled into a one-time program under the weak operational security definition.

*Proof.* We will construct an adversary  $\mathcal{A}$  that is able to entirely learn the function given its one-time program. First, the  $\mathcal{A}$  runs the one-time program evaluation procedure on input x=0, and measures the first n bits of the output to obtain value a. Since the first n bits of  $f_{a,k}(0,\cdot)$  will always equal a, by gentle measurement lemma, this measurement by the adversary cannot disturb the one-time program state. Now, having learned the value of a, the adversary  $\mathcal{A}$  uncomputes its query on 0 to restore the intial state of the one-time program. Finally, the adversary makes a second query to the one-time program evaluation procedure on input a, and measures the first a0 bits of the output to get the PRF key a2. This reveals the entire function description to the adversary. In particular, the adversary can break the weakest operational security definition by computing two input-output pairs a3 and a4 and a5 and a5 are interesting to entire function description to the adversary.

Now consider the following notion of min-entropy for randomized functions *f* 

$$H_{\min}(f) = \min_{x,y} \log \frac{1}{p(y|x)},$$

where  $p(y|x) = \Pr_{r \leftarrow \mathcal{R}}[f(x,r) = y].$ 

**Claim 7.13** (High entropy). *The randomized function family*  $\mathcal{F}$  *has high entropy for every input.* 

If the PRF has output length m, this counterexample is insidtinguishable from a function  $f^*$  that has high min-entropy for every input  $x \in \{0,1\}^n$ ,

$$\min_{x} H_{\min}(f^*(x,\cdot)) = m$$

**Claim 7.14** (Single physical query unlearnable). The function family  $\mathcal{F}$  defined in Equation (6) is unlearnable given a single physical query under the associated probability distribution  $\mathcal{D}$ .

<sup>&</sup>lt;sup>13</sup>The GGM construction, for instance is secure against quantum superposition queries provided that the underlying PRG is quantum-secure [Zha12b].

*Proof.* We show that for every non-uniform quantum-polynomial-time A,

$$\Pr_{f \leftarrow \mathcal{F}_n}[\mathsf{LearningGame}_{\mathcal{F},\mathcal{D}}^{\mathcal{A}} = 1] \leq \mathsf{negl}(n).$$

We consider a sequence of hybrids. Let the first hybrid  $\mathcal{H}_1$  be the learning game. In the second hybrid  $\mathcal{H}_2$ , the function family is now

$$f_{a,k}(x;r) = \begin{cases} (a, \mathsf{PRF}_k(0||r)) & \text{if } x = 0, \\ (0, \mathsf{PRF}_k(a||r)) & \text{if } x = a, \\ (0, \mathsf{PRF}_k(x||r)) & \text{otherwise.} \end{cases}$$

Since the adversary  $\mathcal{A}$  gets to make a single quantum query to the oracle  $O_{f(\cdot,\$)^{(1)}}$ , and since  $a \leftarrow \{0,1\}^n$  is sampled uniformly at random by the challenger, with overwhelming probability, the weight placed by this query on input x=a must be negligible. Therefore, hybrids  $\mathcal{H}_1$  and  $\mathcal{H}_2$  are indistinguishable by [BBBV97a].

Now consider a third hybrid  $\mathcal{H}_3$  where the PRF is replaced by a random oracle H.

$$f_{a,k}(x;r) = \begin{cases} (a, H(0||r)) & \text{if } x = 0, \\ (0, H(a||r)) & \text{if } x = a, \\ (0, H(x||r)) & \text{otherwise.} \end{cases}$$

By the security of the PRF against quantum superposition query attacks, hybrids  $\mathcal{H}_2$  and  $\mathcal{H}_3$  are indistinguishable. Therefore, since the random oracle family is unlearnable under a single physical query, so is the function family  $\mathcal{F}$ .

Since we know that the function family cannot be compiled into a one-time program that satisfies weak operational security definition, by Lemma 7.12 and Claim 7.14, we now know this function family cannot be compiled into a one-time program that satisfies the single physical query classical output simultion-based definition.

**Corollary 7.15.**  $\mathcal{F}$  cannot be compiled into a one-time program under the single physical query classical-output simulation-based definition.

**Claim 7.16** (SEQ learnable). There is an adversary that, given single effective query access to  $f_{\$,1}$  succeeds in the learning game Learning Game<sub> $\mathcal{F},\mathcal{D}$ </sub> with probability 1.

*Proof.* The proof of Lemma 7.12 also shows that the function is learnable in the SEQ model.  $\Box$ 

This trivially implies that the function family is one-time programmable in the SEQ model.

**Corollary 7.17.** The OTP construction in Section 5.1 gives a one-time compiler for  $\mathcal{F}$  in the classical oracle model, under the single effective query simulation-based definition.

# 8 Applications

#### 8.1 Signature Tokens

**Motivation and Comparison to [BDS23]** In this section we briefly discuss how to generate one-time tokens for the Fiat-Shamir signature schemes, by embedding our construction in Section 5.1 in to the plain signature scheme.

One might wonder why we cannot simply use the signature token in [BDS23], where the signatures are simply measured subspace vectors corresponding to the messages.

One unsatisfactory property of [BDS23] is that it doesn't satisfy the regular existential unforgeability of signatures. If we give a "signing oracle" to the adversary, then it can trivially break the one-time security. A more idealized notion would be allowing the adversary to query a signing oracle, but not enabling it to produce two signatures where neither is in the queried set.

The other unsatisfactory part of the [BDS23] signature scheme is that one has to use subspace vectors as signatures and hard to integrate other properties of a signature scheme we may want (e.g. a short signature scheme). More importantly, a corporation may have been using a plain signature scheme for a long time but when they occasionally need to delegate a one-time signing key to some external third-party, they have to change their entire cooperation's verification scheme into the subspace signature token scheme, which can result in more cost and inconvenience. Therefore, one interesting question is: Can we build a generic way to upgrade an existing signature scheme to be one-time secure such that the verification algorithm?

The advantage of the signature schemes below over [BDS23] is that they preserve the original signature scheme's properties. In particular, the verification algorithm is almost identical to the original verification algorithm of the signature scheme being compiled; the signature tokens produce signatures from the original scheme on messages of the form m||r for some r. Thus, the verifier can use the original verification procedure and ignore the latter half of the signed message.

**Blind Unforgeability** We describe here how to compile signature schemes satisfying a certain notion of unforgeability with quantum query access into signature tokens (see Section 3.5 for definitions). The notion we require is a slight variant on blind unforgeability.

**Definition 8.1** (Quantum Blind Unforgeability [AMRS20]). *A signature scheme* (Gen, Sign, Verify) *for message space*  $\mathcal{M}$  *is* blind-unforgeable *if for every QPT adversary*  $\mathcal{A}$  *and blinding set*  $\mathcal{B} \subset \mathcal{M}$ ,

$$\Pr\left[m \in B \land \mathsf{Verify}(\mathsf{vk}, m, \sigma) = \mathsf{Accept}: \begin{array}{c} (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}_B(\mathsf{sk}, \cdot)}(\mathsf{vk}) \end{array}\right]$$

where  $\mathsf{Sign}_B(\mathsf{sk},\cdot)$  denotes a (quantumly-accessible) signature oracle that signs messages m using  $\mathsf{sk}$  if  $m \notin B$ , and otherwise outputs  $\bot$ .

This definition differs from the original in that the adversary may choose its blinding set B. [AMRS20] show that the hardness of this task is polynomially related to their original definition, which samples B uniformly at random.

We note that any sub-exponentially secure signature scheme is blind-unforgeable, since the adversary could simply query for all signatures in  $\mathcal{M}\backslash B$ , then simulate the blind-unforgeability experiment. [AMRS20] also gives several other signature schemes which are blind-unforgeable (under the original definition).

**Construction.** Given a signing key sk, the signer constructs a signature token by outputting a one-time program for the following functionality:

To sign a message m using a signature token T, the temporary signer evaluates T(m) and measures the output.

```
Hardcoded: A signing key sk, two PRF key k_1 and k_2.
On input m:

1. Sample randomness r \leftarrow \{0,1\}^{\lambda}.
2. Compute r_1 = \mathsf{PRF}(k_1,m\|r) and r_2 = \mathsf{PRF}(k_2,m).
3. Output \mathsf{Sign}(\mathsf{sk},m\|r\|r_1;r_2).
```

Figure 9: Signature Token Functionality

**Theorem 8.2.** If (Gen, Sign, Verify) satisfies blind-unforgeability (Definition 8.1), PRF is a psuedorandom function secure against quantum queries, and the one-time program satisfies SEQ simulation security (Definition 4.8), then the above construction is one-time unforgeable (Definition 3.10).

*Proof.* We first show that any QPT adversary can only sign messages of the form  $m||r|| \mathsf{PRF}(k_1, r)$  for some r. Consider the following hybrid experiments:

- Hyb<sub>0</sub> is the one-time unforgeability game.
- Hyb<sub>1</sub> is the same as Hyb<sub>0</sub>, except that  $r_2$  is replaced by true randomness, instead of being a PRF evaluation.
- Hyb<sub>2</sub> is the same as Hyb<sub>1</sub>, except that the SEQ oracle does not have the signing key sk hard-coded. Instead, it queries  $m||r||r_1$  to an external signing oracle.

 $\mathsf{Hyb}_1$  is computationally indistinguishable from  $\mathsf{Hyb}_1$  by the psuedorandomness of PRF.  $\mathsf{Hyb}_2$  is identical to  $\mathsf{Hyb}_1$  in the view of the adversary. Note that in  $\mathsf{Hyb}_2$ , the SEQ oracle *only* submits queries of the form  $m\|r\|\mathsf{PRF}(k_1,r)$  for some r. By the blind-unforgeability of the signature scheme, no adversary can produce signatures on messages not of this form in  $\mathsf{Hyb}_2$ . Since  $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_0$ , this also holds in the original one-time unforgeability game.

Finally, consider the hybrid experiment where  $r_1$  is replaced by G(r), where G is a random function. This is indistinguishable from the one-time unforgeability game by the pseudorandomness of PRF. By the previous claim, any adversary producing two signatures must have signed  $m_1 ||r_1|| G(m||r_1)$  and  $m_2 ||r_1|| G(m||r_1)$ . If  $m_1 \neq m_2$ , then this contradicts the SEQ-unlearnability of random functions (Proposition 5.12).

#### 8.2 One-Time NIZK Proofs

Our final application of one-time programs is to a notion of one-time non-interactive zero-knowledge (ZK) proofs that we define and construct. Here, a proving authority (say, a government) publishes a verification key vk and delegates to its subsidiaries the ability to certify (prove) a limited number of statements on its behalf by giving them a one-time proving token prk. A prover in possession of prk, an NP statement x and its witness x, can generate a proof x that anyone can verify against vk. Importantly, he can only do so for a single valid statement-witness pair. Thus, we have the following tuple of algorithms:

- Setup: produces a master secret key msk together with a verification key/common reference string CRS.
- Delegate: on input msk, produces a one-time proving token  $\rho$ .

- Prove: on input  $\rho$  and a statement-witness pair (x, w), produces a proof  $\pi$ .
- Verify: on input  $x, \pi$  and vk, outputs accept or reject.

Note that all objects here are classical except for the proving token  $\rho$  which is a quantum state. In addition to the usual properties of completeness, soundness and zero knowledge, we require that the proving token is one-time use only.

We construct a one-time proof token, following constructions of one-time PRFs in the plain model. The proving token consists of a sequence of n=|x| many subspace states corresponding to subspaces  $A_1,\ldots,A_n$  together with the obfuscation of a program that contains a PRF key K together with the  $A_i$ ; takes as input x,w and n vectors  $v_1,\ldots,v_n$ ; checks that  $(x,w)\in R_L$ , and that each  $v_i\in A_i$  if  $x_i=0$  and  $v_i\in A_i^\perp$  if  $x_i=1$ . If all checks pass, output  $\mathsf{PRF}_K(x,v_1,\ldots,v_n)$ ; otherwise output  $\perp^{14}$ .

The security of the one-time proof follows from similar arguments of the security for one-time PRF in the plain model Section 6.2. We give a more formal description of the scheme below.

**One-time Security Definition** The one-time NIZK scheme first needs to satisfy the usual NIZK soundness and zero knowledge property. We omit these standard definitions here and refer to [SW14] Section 5.5 for details.

We then define a very natural one-time security through the following game, as a special case of the Definition 4.23 for NIZK proofs:

- 1. The challenger samples CRS  $\leftarrow$  Setup(1 $^{\lambda}$ ) and prepares the program OTP for the Prove functionality.  $\mathcal{A}$  gets a copy of the one-time program for OTP and CRS.
- 2.  $\mathcal{A}$  outputs two instance-proof pairs (or instance-randomness-proof tuple, for a relaxed notion)  $(x_1, \pi_1), (x_2, \pi_2)$ , which satisfies  $x_1 \neq x_2$  or  $\pi_1 \neq \pi_2$ , otherwise  $\mathcal{A}$  loses.
- 3. Challenger checks if  $Verify(x_1, \pi_1) = 1$  and  $Verify(x_2, \pi_2) = 1$ . output 1 if and only both are satisfied.

We say that a one-time sampling program for NIZK satisfies security if for any QPT adversary A, there exists a negligible function negl(·), such that for all  $\lambda \in \mathbb{N}$ , the following holds:

$$\Pr[A \text{ wins the above game }] \leq \mathsf{negl}(\lambda).$$

We defer the regular soundness and zero-knowledge definition of NIZK tp Appendix D.

**Construction** Let F be a constrainable PRF that takes inputs of  $\ell$  bits and outputs  $\lambda$  bits. Let  $f(\cdot)$  be a PRG. Let L be a language and  $R(\cdot,\cdot)$  be a relation that takes in an instance and a witness. Our system will allow proofs of instances of  $\ell$  bits and witness of  $\ell'$  bits. The values of bounds given by the values  $\ell$  and  $\ell'$  can be specified at setup, although we suppress that notation here.

For simplicity of presentation, we omit the need of a second PRF used to extract randomness in Section 6 since it is only used to extract full entropy and will not affect the security proof.

Setup( $1^{\lambda}$ ): The setup algorithm in our case generates a common reference string CRS along with a one-time proof token.

<sup>&</sup>lt;sup>14</sup>We may also obtain a construction from random oracle based NIZK such as Fiat Shamir, but it would require the use of classical oracles while using iO and PRF gives a plain model result.

- 1. The setup algorithm first chooses a puncturable PRF key *K* for *F*. Next, it creates an obfuscation of the Verify NIZK of Figure 11. The size of the program is padded to be the maximum of itself and the program we define later in the security game.
- 2. It samples n independent subspaces  $\{|A_i\rangle\}_{i\in[n]}$ . It creates an obfuscation of the program Prove NIZK of Figure 10. The size of the program is padded to be the maximum of itself and the program we define later in the security game.

The common reference string CRS consists of the two obfuscated programs and a master secret key  $msk = \{A_i\}_{i \in [n]}$ .

Delegate: takes in a master secret key msk =  $\{A_i\}_{i\in[n]}$  and outputs  $\{|A_i\rangle\}_{i\in[n]}$  as the one-time proof token.

Prove(CRS, (x, w),  $\{|A_i\rangle\}_{i\in[n]}$ ): The NIZK prove algorithm runs the obfuscated program of Prove from CRS on inputs (x, w) and subspace states  $|A_i\rangle$ ,  $i\in[n]$  in the following way: Apply QFT to each  $|A_i\rangle$  if  $x_i=1$ , else apply identity operator. Run the program in Figure 10 on input (x, w) and the modified subspace states. If R(x, w) holds the program returns a proof  $\pi=(r, F(K, x||r))$ .

```
Hardcoded: K, \{\mathsf{shO}_i^0, \mathsf{shO}_i^1\}_{i \in [n]}.

On input ((x, w) \in \{0, 1\}^{n+m}, u = u_1 || u_2 || \cdots || u_n \in \{0, 1\}^{n \cdot \lambda}) (where each u_i \in \mathbb{F}_2^{\lambda}):

1. If (x, w) \in R_L and for all i \in [n], \mathsf{shO}_i^{x_i}(u_i) = 1, where x_i is the i-th bit of x:

Output (r = u, F(k, x || r)).

2. Else:
Output \bot
```

Figure 10: Program NIZK ProveOTP

Verify( $x, \pi$ , CRS): Run the input ( $x, \pi$ ) into the obfuscated program Figure 11 and output the program's output.

```
Hardcoded: K, f, \{\mathsf{shO}_i^0, \mathsf{shO}_i^1\}_{i \in [n]}.
On input (x \in \{0, 1\}^n, \pi):

1. Parse \pi := (r, y)

2. If for all i \in [n], \mathsf{shO}_i^{x_i}(r_i) = 1, where x_i is the i-th bit of x: proceed to step 3; else output 0.

3. Check if f(y) = f(F(K, x || r)): if yes, output 1; if no, output 0.
```

Figure 11: Program NIZK Verify<sub>OTP</sub>

We then prove the following statement:

**Theorem 8.3.** Assuming LWE and subexponentially secure iO, there exists one-time NIZK proofs satisfying the above definition.

**One-Time Security** The correctness and one-time security proof follows relatively straightforward in a similar way as the proof for the PRF construction Section 6.

We first replace the PRF F's kardcoded key K in both programs with a key  $K^*$  that is constrained on a circuit to evaluate only on (x, r) such that r are subspace vectors corresponding to x. By the property of iO, this change is indistinguishable. Then by the computational direct product hardness of the subspace states, we can argue that the adversary can only provide two proofs such that one of  $r_1$  and  $r_2$  are not valid subspace vectors, which will break the constrained pseudorandomness of the constrained PRF.

**Soundness** For the regular soundness of the construction is inspired by the proof for Theorem 9 in [SW14]. but we need to use subexponentially secure iO and slightly more complicated hybrids. In hybrid 1, for some instance  $x^* \notin L$ , we constrain the PRF F key K used in the program Figure 10 to a constrained key  $K^*$  that evaluates on inputs  $(x\|r)$  such that  $x \neq x^*$ . Since  $x^* \notin L$ , the functionality of the program Prove<sub>OTP</sub> is unchanged and we can invoke iO. The next few steps deviate from [SW14].

We design hybrids (2.j.1) for  $i=1,2\cdots,t,t=2^{\lambda\cdot n/2}$ : for all vectors  $u_1\in A^{x_1},\cdots,u_n\in A^{x_n}$ , we make a lexigraphical order on them and call the j-th vector  $\mathbf{u}_j$ . In hybrid (2.j.1), we modify the program Verify<sub>OTP</sub> into the following.

Note that the key  $K_{x^*,r_i}$  is a punctured/constrained key that does not evaluate at  $(x^*|r_i)$ .

```
Hardcoded: K_{x^*,r_j}, f, \{\mathsf{shO}_i^0,\mathsf{shO}_i^1\}_{i\in[n]}, y^* = F(k,x^*\|r_j).

On input (x\in\{0,1\}^n,\pi):

1. Parse \pi:=(r,y)
2. If for all i\in[n], \mathsf{shO}_i^{x_i}(r_i)=1, where x_i is the i-th bit of x: proceed to step 3; else output 0.
3. If x=x^* and x< r_j: output 0.
4. Else if x=x^* and r=r_j: check if f(y)=f(y^*): if yes, output 1; if no, output 0.
5. Else if x\neq x^* or x>r_j: Check if f(y)=f(F(K_{x^*,r_j},x\|r)): if yes, output 1; if no, output 0.
```

**Figure 12:** Program NIZK Verify<sub>OTP</sub> in hybrid 2.j.1. Note that the key  $K_{x^*,r_j}$  is a punctured/constrained key that does not evaluate at  $(x^*|r_j)$ 

Note that the functionality of the program is essentially the same as in the original program in Figure 11 and therefore we can invoke the security of iO. Aftern hybrid (2.j.1), we introduce the next hybrid (2.j.2). In the next hybrid (2.j.2), we will replace  $y^* = F(K, x^* || r_j)$  hard-coded in the program with a random value. This follows from the pseudorandomness at punctured/constrained values. In the following hybrid (2.j.3), we replace the value  $f(y^*)$  with a random value from the range of f. This is indistinguishable by the property of PRG f. Now with overwhelming probability, when the input is  $(x^*, \pi = (r_j, y))$ , there exists no value g that will make the program output 1.

After hybrid (2.j.3), we move to the next hybrid (2.j + 1.1), where Verify<sub>OTP</sub> is the same program as in Figure 12 but we increment the counter j to j+1. By the above observations, the hybrid

(2.j.3) is statistically indistinguishable from the next hybrid 2.j+1.1, and now for all inputs where  $x=x^*, r < r_{j+1}$ , the program outputs 0.

Finally, after hybrid 2.t.3, we obtain a Verify program that outputs all 0 for inputs  $(x, \pi = (r, y))$  where  $x = x^*, r \in A^{x_1}, \cdots, A^{x_n}$ . The adversary's advantage in getting an acceptance on instance  $x^*$  is 0.

**Zero Knowledge** The zero-knowledge proof follows exactly from [SW14]. A simulator S that on input x, runs the setup algorithm and outputs the corresponding CRS along with  $\pi = F(K, x || r)$  for some r of its own choice since it samples the subspaces on its own. The simulator has the exact same distribution as any real prover's algorithm for any  $x \in L$  and witness w where R(x, w) = 1.

#### 8.3 Future Work: One-Time MPC

Our one-time sampling program has a definition in between a (deterministic) one-time memory and a fully random one-time memory, where the latter says that the receiver's input is uniformly random and it does not get to choose which message it receives (similar discussions in [BKS23]). In this light, we hope that our one-time program may be extended to a one-time 2PC/MPC where each party has some choice over their inputs, but the rest of the inputs are random. We leave this question for future work.

## 8.4 One-time programs for Verifiable Functions Imply Quantum Money

In this section, we show that our definition of one-time (sampling) program for publicly verifiable functions implies public -key quantum money.

**Definition 8.4** (Public-key quantum money). *A quantum money scheme consists of a pair of quantum polynomial-time algorithms* (Gen, Ver) *with the following syntax, correctness and security specifications.* 

- Syntax: The generation procedure  $\mathsf{Gen}(1^\lambda)$  takes as input a security parameter  $\lambda$  and outputs a classical serial number  $\sigma$  and a quantum state  $|\psi\rangle$ . The verification algorithm  $\mathsf{Ver}(\sigma, |\psi\rangle)$  outputs an accept/reject bit, 0 or 1.
- Correctness: There exists a negligible function negl such that

$$\Pr[1 \leftarrow \mathsf{Ver}(\mathsf{Gen}(1^{\lambda}))] \ge 1 - \mathsf{negl}(\lambda).$$

- Security: For all quantum polynomial-time algorithms A, there exists a negligible functions negl such that A wins the following security game with probability  $negl(\lambda)$ :
  - The challenger runs  $(\sigma, |\psi\rangle) \leftarrow \text{Gen}(1^{\lambda})$ , and give  $\sigma, |\psi\rangle$  to A.
  - A produces a (potentially entangled) join state  $\rho_{1,2}$  over two registers  $\rho_1$  and  $\rho_2$ . A sends  $\rho_{1,2}$  to the challenger.
  - The challenger runs  $b_1 \leftarrow \mathsf{Ver}(\sigma, \rho_1)$  and  $b_2 \leftarrow \mathsf{Ver}(\sigma, \rho_2)$ . The adversary A wins if and only if  $b_1 = b_2 = 1$ .

**Theorem 8.5.** Let  $\mathcal{F}$  be a verifiable randomized function family. Suppose there exists a one-time program scheme that satisfies the operational one-time security notion for verifiable functions (Definition 4.29). Then, a public-key quantum money scheme exists.

*Proof.* Let  $\mathcal{F}$  be a verifiable randomized function family, and let OTP = (Generate<sub>OTP</sub>, Evaluate<sub>OTP</sub>) be a one-time program scheme that satisfies the operational one-time security notion for verifiable functions. Then, we can construct a quantum money scheme QMoney = (Gen<sub>QM</sub>, Ver<sub>QM</sub>) as follows:

- $\mathsf{Gen}_{\mathsf{QM}}(1^{\lambda})$ : Sample a function along with its verification key  $f, vk_f \leftarrow \mathcal{F}_{\lambda}$ , and generate its one-time program  $|\psi\rangle \leftarrow \mathsf{OTP}(f)$ . Output classical serial number  $\sigma = vk_f$  and the money state  $|\psi\rangle = \mathsf{OTP}(f)$ .
- Ver<sub>QM</sub> $(\sigma, \rho)$ : Sample a random  $x \leftarrow \mathcal{X}$  and initialize an input register X to be  $|x\rangle$ . Coherently run (without performing any measurement) Evaluate<sub>OTP</sub> $(\rho, x)$  to get the output in register Y. Coherently run Ver<sub> $\mathcal{F}$ </sub> $(vk_f, X, Y)$  on the X and Y registers and measure the resulting accept/reject bit. Accept if the verification passes and reject otherwise.

The correctness of this quantum money scheme follows from the correctness of the one-time program OTP(f), the verification procedure  $Ver_{\mathcal{F}}$  as well as the gentle measurement lemma Lemma 3.2.

We now prove security. Suppose for contradiction there exists an adversary A that wins the quantum money security game with non-negligible probability, so that it outputs a joint state  $\rho_{1,2}$  on two registers  $\rho_1$  and  $\rho_2$  such that both verification checks pass. Then, there exists an adversary B that breaks the one-time program security of  $\mathsf{OTP}(f)$ : given  $\mathsf{OTP}(f)$  and  $vk_f$  as auxiliary information, B runs A on the same input,  $A(\mathsf{OTP}(f), vk_f)$ . Suppose that A outputs (a joint state on) two registers  $\rho_1, \rho_2$ . Then, since the verification procedure verifies by sampling a random input to test the functionality on each state, running the verification procedure on both these registers results in two samples  $(x_1, y_1)$  and  $(x_2, y_2)$  with non-negligible probability, breaking the one-time program security of  $\mathsf{OTP}$ .

#### 9 References

- [Aar04] Scott Aaronson. Limitations of quantum advice and one-way communication. In *Proceedings*. 19th IEEE Annual Conference on Computational Complexity, 2004., pages 320–332. IEEE, 2004. 5, 15
- [ABDS20] Gorjan Alagic, Zvika Brakerski, Yfke Dulek, and Christian Schaffner. Impossibility of quantum virtual black-box obfuscation of classical circuits, 2020. 3, 5, 12, 20, 49, 51
- [AC12] Scott Aaronson and Paul Christiano. Quantum money from hidden subspaces. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 41–60. ACM, 2012. 9
- [ACE<sup>+</sup>22] Ghada Almashaqbeh, Ran Canetti, Yaniv Erlich, Jonathan Gershoni, Tal Malkin, Itsik Pe'er, Anna Roitburd-Berman, and Eran Tromer. Unclonable polymers and their cryptographic applications. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 759–789. Springer, Cham, May / June 2022. 14
- [AMRS20] Gorjan Alagic, Christian Majenz, Alexander Russell, and Fang Song. Quantum-access-secure message authentication via blind-unforgeability. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 788–817. Springer, Cham, May 2020. 39, 57

- [AP21] Prabhanjan Ananth and Rolando L. La Placa. Secure software leasing. Springer-Verlag, 2021. 3, 5, 12, 20, 49, 51, 53
- [BBBV97a] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997. 13, 50, 53, 54, 56
- [BBBV97b] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, Oct 1997. 53
- [BDS23] Shalev Ben-David and Or Sattath. Quantum tokens for digital signatures. *Quantum*, 7:901, 2023. 1, 2, 3, 4, 9, 11, 14, 17, 56, 57
- [BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual International Cryptology Conference*, pages 1–18. Springer, 2001. 2, 11, 19, 41
- [BGS13] Anne Broadbent, Gus Gutoski, and Douglas Stebila. Quantum one-time programs. In *Annual Cryptology Conference*, pages 344–360. Springer, 2013. 1, 2, 4, 5, 6, 14, 20
- [BJ15] Anne Broadbent and Stacey Jeffery. Quantum homomorphic encryption for circuits of low t-gate complexity. In *Annual Cryptology Conference*, pages 609–629. Springer, 2015.
- [BKNY23] James Bartusek, Fuyuki Kitagawa, Ryo Nishimaki, and Takashi Yamakawa. Obfuscation of pseudo-deterministic quantum circuits. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1567–1578, 2023. 9, 17
- [BKS23] James Bartusek, Dakshita Khurana, and Akshayaram Srinivasan. Secure computation with shared epr pairs (or: How to teleport in zero-knowledge). In *Annual International Cryptology Conference*, pages 224–257. Springer, 2023. 62
- [BKW17] Dan Boneh, Sam Kim, and David J Wu. Constrained keys for invertible pseudorandom functions. In *Theory of Cryptography Conference*, pages 237–263. Springer, 2017. 11, 44, 45
- [BLW17] Dan Boneh, Kevin Lewi, and David J Wu. Constraining pseudorandom functions privately. In *IACR International Workshop on Public Key Cryptography*, pages 494–524. Springer, 2017. 44
- [Bra18] Zvika Brakerski. Quantum fhe (almost) as secure as classical. In *Annual International Cryptology Conference*, pages 67–95. Springer, 2018. 49
- [BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions: Or: How to secretly embed a circuit in your prf. In *Theory of Cryptography: 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II 12*, pages 1–30. Springer, 2015. 44, 45
- [CGLZ19] Kai-Min Chung, Marios Georgiou, Ching-Yi Lai, and Vassilis Zikas. Cryptography with disposable backdoors. *Cryptography*, 3(3):22, 2019. 14

- [CHV23] Céline Chevalier, Paul Hermouet, and Quoc-Huy Vu. Semi-quantum copy-protection and more. In *Theory of Cryptography Conference*, pages 155–182. Springer, 2023. 42, 43
- [CLLZ21a] Andrea Coladangelo, Jiahui Liu, Qipeng Liu, and Mark Zhandry. Hidden cosets and applications to unclonable cryptography. In *Advances in Cryptology*—CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41, pages 556–584. Springer, 2021. 9, 42, 43
- [CLLZ21b] Andrea Coladangelo, Jiahui Liu, Qipeng Liu, and Mark Zhandry. Hidden cosets and applications to unclonable cryptography. In Tal Malkin and Chris Peikert, editors, CRYPTO 2021, Part I, volume 12825 of LNCS, pages 556–584, Virtual Event, August 2021. Springer, Cham. 14
- [DFM20] Jelle Don, Serge Fehr, and Christian Majenz. The measure-andreprogram technique 2.0: multi-round fiat-shamir and more. In *Annual International Cryptology Conference*, pages 602–631, 2020. 79
- [DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the fiat-shamir transformation in the quantum random-oracle model. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II 39*, pages 356–383. Springer, 2019. 79
- [GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017*, *Part I*, volume 10677 of *LNCS*, pages 529–561. Springer, Cham, November 2017. 14
- [GGH<sup>+</sup>16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016. 41
- [GIS<sup>+</sup>10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2010. 1, 14
- [GKR08a] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. One-time programs. In *Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28*, pages 39–56. Springer, 2008.
- [GKR08b] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Berlin, Heidelberg, August 2008. 14
- [GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), pages 612–621. IEEE, 2017. 50

- [GM24] Sam Gunn and Ramis Movassagh. Quantum one-time protection of any randomized algorithm. *private communication*, 2024. 4, 14
- [Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In Tatsuaki Okamoto, editor, ASIACRYPT 2000, volume 1976 of LNCS, pages 443–457. Springer, Berlin, Heidelberg, December 2000. 2
- [Liu23] Qipeng Liu. Depth-bounded quantum cryptography with applications to one-time memory and more. In 14th Innovations in Theoretical Computer Science Conference (ITCS 2023), 2023. 14
- [LSZ20] Qipeng Liu, Amit Sahai, and Mark Zhandry. Quantum immune one-time memories. *Cryptology ePrint Archive*, 2020. 14
- [Mah20] Urmila Mahadev. Classical homomorphic encryption for quantum circuits. *SIAM Journal on Computing*, 52(6):FOCS18–189, 2020. 49
- [NC02] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002. 15
- [RKB<sup>+</sup>18] Marie-Christine Roehsner, Joshua A. Kettlewell, Tiago B. Batalhão, Joseph F. Fitzsimons, and Philip Walther. Quantum advantage for probabilistic one-time programs. *Nature Communications*, 9(1):5225, Dec 2018. 14
- [RKFW21] Marie-Christine Roehsner, Joshua A. Kettlewell, Joseph Fitzsimons, and Philip Walther. Probabilistic one-time programs using quantum entanglement. *npj Quantum Information*, 7(1):98, Jun 2021. 14
- [SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 475–484, 2014. 12, 41, 45, 59, 61, 62
- [VZ21] Thomas Vidick and Tina Zhang. Classical proofs of quantum knowledge. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 630–660. Springer, 2021. 42
- [Win99] Andreas Winter. Coding theorem and strong converse for quantum channels. *IEEE Transactions on Information Theory*, 45(7):2481–2485, 1999. 5
- [WZ82] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, October 1982. 1
- [WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under lwe. In 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), pages 600–611. IEEE, 2017. 50
- [YZ21] Takashi Yamakawa and Mark Zhandry. Classical vs quantum random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 568–597. Springer, 2021. 23, 79, 80, 82

- [Zha12a] Mark Zhandry. How to construct quantum random functions. In *Proceedings of the* 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, FOCS '12, page 679–687, USA, 2012. IEEE Computer Society. 51
- [Zha12b] Mark Zhandry. How to construct quantum random functions. In 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, pages 679–687. IEEE, 2012. 55
- [Zha16] Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, CRYPTO 2016, Part I, volume 9814 of LNCS, pages 479–508. Springer, Berlin, Heidelberg, August 2016. 50
- [Zha19a] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology CRYPTO 2019*, pages 239–268, Cham, 2019. Springer International Publishing. 2, 7, 15, 16, 23
- [Zha19b] Mark Zhandry. Quantum lightning never strikes the same state twice. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 408–438. Springer, 2019. 41, 42

# A Missing Proofs for Families of Single-Query Unlearnable Functions

# A.1 Pairwise Independent and Highly Random Functions

In this section, we present the full proof that pairwise independent and highly random functions are single-effective-query  $negl(\lambda)$ -unlearnable in the single-effective-query oracle model (Section 4.2).

**Lemma A.1.** Let  $\mathcal{F}$  be a family of functions mapping  $\mathcal{X} \times \mathcal{R} \to \mathcal{Y}$  that satisfies:

- 1. Pairwise independence: For any  $(x,r,y), (x',r',y') \in \mathcal{X} \times \mathcal{R} \times \mathcal{Y}$  such that  $(x,r) \neq (x',r'), \Pr_f[f(x,r) = y \land f(x',r') = y'] = \Pr_f[f(x,r) = y] \cdot \Pr_f[f(x',r') = y'].$
- 2. High Randomness: There is a negligible function  $\nu(\lambda)$  such that for any  $(x, r, y) \in \mathcal{X} \times \mathcal{R} \times \mathcal{Y}$ ,

$$\Pr_f[f(x,r) = y] \le \nu(n)$$

3.  $\frac{1}{|\mathcal{R}|} = \text{negl}(\lambda)$ 

*Then*  $\mathcal{F}$  *is* SEQ-negl( $\lambda$ )-unlearnable.

Proof.

1. The SEQ oracle implements a compressed oracle for H. In particular, it maintains a database register  $\mathcal{D}_H$  that stores  $\emptyset$  or a list of values  $(x,r) \in \mathcal{X} \times \mathcal{R}$ . See Section 3.3 for a formal definition of the compressed oracle representation.

The SEQ oracle also stores the function f on register  $\mathcal{F}$ . Let  $\mathcal{F}$  be initialized to the uniform superposition:

$$|F_{\emptyset}\rangle := \frac{1}{\sqrt{|\mathcal{F}|}} \sum_{f \in \mathcal{F}} |f\rangle_{\mathcal{F}}$$

Then the SEQ oracle will answer queries coherently, without measuring the superposition. Also, let us represent f as its truth table, so for every  $(x,r) \in \mathcal{X} \times \mathcal{R}$ , there is a register  $\mathcal{F}_{x,r}$  that holds the value y = f(x,r).

Let  $\mathcal{O} = \mathcal{D}_H \times \mathcal{F}$  be the oracle's internal register, let  $\mathcal{Q}$  be the query register submitted to the oracle, and let  $\mathcal{A}$  be the adversary's private register. We can assume that the state of the system is a pure state over  $\mathcal{A} \times \mathcal{Q} \times \mathcal{O}$  since the adversary and the SEQ oracle act as unitaries over these registers.

2. Let us define the states that  $\mathcal{O}$  is allowed to be in, and let E project onto the allowed states.

Let 
$$\mathcal{F}_{x,r,y} = \{ f \in \mathcal{F} : f(x,r) = y \}$$

$$|F_{x,r,y}\rangle = \frac{1}{\sqrt{|\mathcal{F}_{x,r,y}|}} \cdot \sum_{f \in \mathcal{F}_{x,r,y}} |f\rangle_{\mathcal{F}}$$

$$E_{\mathcal{O}} = |\emptyset\rangle \langle \emptyset|_{\mathcal{D}_{H}} \otimes |F_{\emptyset}\rangle \langle F_{\emptyset}|_{\mathcal{F}} + \sum_{(x,r,y) \in \mathcal{X} \times \mathcal{R} \times \mathcal{Y}} |(x,r)\rangle \langle (x,r)|_{\mathcal{D}_{H}} \otimes |F_{x,r,y}\rangle \langle F_{x,r,y}|_{\mathcal{F}}$$

$$E = \mathbb{I}_{\mathcal{A} \times \mathcal{Q}} \otimes E_{\mathcal{O}}$$

 $E_{\mathcal{O}}$  projects onto all states on  $\mathcal{O}$  in the span of the following basis states:

$$|\emptyset\rangle_{\mathcal{D}_H}\otimes|F_{\emptyset}\rangle_{\mathcal{F}}$$
 or  $\Big(|(x,r)\rangle_{\mathcal{D}_H}\otimes|F_{x,r,y}\rangle_{\mathcal{F}}\Big)_{(x,r,y)\in\mathcal{X}\times\mathcal{R}\times\mathcal{Y}}$ 

Let us also define  $\overline{E}=\mathbb{I}-E$ , and  $\overline{E}_{\mathcal{O}}=\mathbb{I}-E_{\mathcal{O}}.$ 

- 3. After any polynomial number of queries to the SEQ oracle, the state of the system  $|\psi\rangle$  satisfies  $|E \cdot |\psi\rangle| \ge 1 \mathsf{negl}(\lambda)$ . This is proven in Lemma A.2.
- 4. At the end of the SEQ learning game, the following steps are executed. We have added an additional step (step 2, shown in red).
  - (a) The adversary outputs two pairs (x, r, y) and (x', r', y') such that  $(x, r) \neq (x', r')$ .
  - (b) The challenger measures  $\mathcal{D}_H$  in the computational basis to obtain  $(x'', r'') \in \mathcal{X} \times \mathcal{R}$  or  $\emptyset$ . If the outcome is some (x'', r'') (and not  $\emptyset$ ), then the challenger measures the register  $\mathcal{F}_{x'',r''}$  to obtain y'' = f(x'',r'').
  - (c) The challenger measures  $\mathcal{F}$  in the computational basis to get a function f.
  - (d) The challenger checks whether f(x,r) = y and f(x',r') = y'. If so, the adversary wins. If not, the adversary loses.

Adding step 2 does not affect the probability that the adversary wins the learning game because the measurements made in step 2 commute with the measurements made in step 3. Clearly, the measurement on  $\mathcal{D}_H$  commutes with the measurement on  $\mathcal{F}$  in step 3 because they act on disjoint registers. Furthermore, measuring  $\mathcal{F}_{x'',r''}$  is a partial measurement on the  $\mathcal{F}$  register in the computational basis. Therefore, it commutes with step 3's full measurement of  $\mathcal{F}$  in the computational basis.

- 5. We will show that the measurement outcome of f in step 3 is highly random, so the adversary wins the learning game with negligible probability. Let us assume that at the start of step 1, the state of the system  $|\psi\rangle$  satisfies  $E\cdot|\psi\rangle=|\psi\rangle$ . Next, there are two cases to consider:
  - (a) Case 1: The measurement on  $\mathcal{D}_H$  returns  $\emptyset$ . Then at the end of step 2, the state on  $\mathcal{F}$  is  $|F_{\emptyset}\rangle$ . When we measure  $\mathcal{F}$  in step 3, we obtain a uniformly random  $f \stackrel{\$}{\leftarrow} \mathcal{F}$ . The

adversary will lose the learning game with overwhelming probability because

$$\Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x,r) = y \land f(x',r') = y'] = \Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x,r) = y] \cdot \Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x',r') = y']$$

$$\leq \nu(\lambda)^2$$

$$= \mathsf{negl}(\lambda)$$

(b) Case 2: Otherwise, the measurement on  $\mathcal{D}_H$  returns some (x'', r''), and we also measure y''. At the end of step 2, the state on the  $\mathcal{F}$  register is  $|F_{x'',r'',y''}\rangle$ . When we measure  $\mathcal{F}$  in step 3, we obtain a uniformly random  $f \stackrel{\$}{\leftarrow} \mathcal{F}_{x'',r'',y''}$ .

At least one of (x,r) and (x',r') do not equal (x'',r''). Without loss of generality, let us say that  $(x,r) \neq (x'',r'')$ . Then the probability that the adversary wins the learning game is

$$\begin{split} \Pr[\mathcal{A} \text{ wins}] & \leq \Pr_{f \overset{\$}{\leftarrow} \mathcal{F}_{x'',r'',y''}} [f(x,r) = y] \\ & = \frac{|\mathcal{F}_{x,r,y} \cap \mathcal{F}_{x'',r'',y''}|}{|\mathcal{F}_{x'',r'',y''}|} \\ & = \frac{|\mathcal{F}| \cdot \Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x,r) = y \wedge f(x'',r'') = y'']}{|\mathcal{F}| \cdot \Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x'',r'') = y'']} \\ & = \frac{\Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x,r) = y] \cdot \Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x'',r'') = y'']}{\Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x'',r'') = y'']} \\ & = \Pr_{f \overset{\$}{\leftarrow} \mathcal{F}} [f(x,r) = y] \\ & \leq \nu(\lambda) = \operatorname{negl}(\lambda) \end{split}$$

We've shown that in both cases, the adversary wins the learning game with negligible probability. Therefore  $\mathcal{F}$  is SEQ-negl( $\lambda$ )-unlearnable.

**Lemma A.2.** *After any polynomial number of queries to the SEQ oracle, the state of the system*  $|\psi\rangle$  *satisfies*  $|E\cdot|\psi\rangle$   $|\geq 1-\mathsf{negl}(\lambda)$ .

Proof.

- 1. Let us define some useful operations.
  - Let  $V_x$  be a projector acting on  $\mathcal{O}$  that projects onto all states for which  $\mathcal{D}_H$  contains  $\emptyset$  or a single entry of the form (x,r) for some  $r \in \mathcal{R}$ . Furthermore, let V be the following projector acting on  $\mathcal{A} \times \mathcal{Q} \times \mathcal{O}$ :

$$V = \sum_{x \in \mathcal{X}} \mathbb{I}_{\mathcal{A} \times \mathcal{Q}_u \times \mathcal{Q}_b} \otimes |x\rangle \langle x|_{\mathcal{Q}_x} \otimes (V_x)_{\mathcal{O}}$$

In other words, V verifies whether the query x on  $Q_x$ , if answered by the SEQ oracle, would keep the number of queries recorded in  $\mathcal{D}_H \leq 1$ .

Note that *E* and *V* commute with each other and share an eigenbasis.

- Let Decomp and Decomp<sub>x</sub> be defined as they were in section 3.3.
- The SEQ oracle maintains an internal register  $\mathcal{R}$  that is used to store an intermediate r-value. Then  $\mathsf{CO}'_H$  copies the output of H(x) into the  $\mathcal{R}$  register.  $\mathsf{CO}'_H$  computes the following mapping:

$$|x\rangle_{\mathcal{Q}_{\mathcal{X}}}\otimes|r'\rangle_{\mathcal{R}}\otimes|(x,r)\rangle_{\mathcal{D}_{H}}\stackrel{\mathsf{CO}'_{H}}{\longrightarrow}|x\rangle_{\mathcal{Q}_{\mathcal{X}}}\otimes|r'\oplus r\rangle_{\mathcal{R}}\otimes|(x,r)\rangle_{\mathcal{D}_{H}}$$

• Let CO'<sub>F</sub> answer the query to f with the following mapping:

$$|x,u,b\rangle_{\mathcal{Q}}\otimes|r\rangle_{\mathcal{R}}\otimes|f\rangle_{\mathcal{F}}\stackrel{\mathsf{CO}_{\mathcal{F}}'}{\longrightarrow}|x,u\oplus f(x,r),b\oplus 1\rangle_{\mathcal{Q}}\otimes|r\rangle_{\mathcal{R}}\otimes|f\rangle_{\mathcal{F}}$$

2. Let us describe how the SEQ oracle operates.

The SEQ oracle acts as the identity on all states  $|\psi\rangle$  for which  $V \cdot |\psi\rangle = \mathbf{0}$ . And for all states  $|\psi\rangle$  for which  $V \cdot |\psi\rangle = |\psi\rangle$ , the SEQ oracle operates as follows:

- (a) Initialize  $\mathcal{R}$  to  $|0\rangle$ . Then apply Decomp  $\circ \mathsf{CO}'_H \circ \mathsf{Decomp}$  to  $\mathcal{Q}_{\mathcal{X}} \times \mathcal{R} \times \mathcal{D}_H$ .
- (b) Apply  $CO'_F$  to  $\mathcal{Q} \times \mathcal{R} \times \mathcal{F}$ .
- (c) Apply Decomp  $\circ \mathsf{CO}'_H \circ \mathsf{Decomp}$  to  $\mathcal{Q}_{\mathcal{X}} \times \mathcal{R} \times \mathcal{D}_H$ .

Note that Decomp commutes with  $\mathsf{CO}_F'$ . Decomp depends on the computational basis state of  $\mathcal{Q}_{\mathcal{X}}$ , and applies  $\mathsf{Decomp}_x$  to  $\mathcal{D}_H$ .  $\mathsf{CO}_F'$  depends on the computational basis state of  $\mathcal{Q}_{\mathcal{X}}$  and otherwise acts on registers  $(\mathcal{Q}_u \times \mathcal{Q}_b \times \mathcal{R} \times \mathcal{F})$  that are disjoint from the ones acted on by Decomp.

Furthermore,  $Decomp^{-1} = Decomp$ , so we can write the action of the SEQ oracle as follows:

$$\mathsf{Decomp} \circ \mathsf{CO}'_H \circ \mathsf{CO}'_F \circ \mathsf{CO}'_H \circ \mathsf{Decomp}$$

- 3. Let us assume that at the beginning of the query, the state  $|\psi\rangle$  is in the span of E. This is true at the beginning of the first query because the state of  $\mathcal{O}$  is  $|\emptyset\rangle_{\mathcal{D}_H}\otimes|F_\emptyset\rangle_{\mathcal{F}}$ , which is in the span of E. Then we will show that at the end of the query, the state  $|\psi'\rangle$  is negligibly close to a state in the span of E. Then by induction, after any polynomial number of queries to the SEQ oracle, the state of  $\mathcal{O}$  will be negligibly close to a state in the span of E.
- 4. Next, we split  $|\psi\rangle$  into the components in the span of V and perpendicular to V.  $|\psi\rangle = (\mathbb{I} V) \cdot |\psi\rangle + V \cdot |\psi\rangle$  Note that  $\mathbf{s}_{\overline{V}} := (\mathbb{I} V) \cdot |\psi\rangle$  and  $\mathbf{s}_{V} := V \cdot |\psi\rangle$  are still in the span of E because applying  $(\mathbb{I} V)$  or V to  $|\psi\rangle$  just checks whether  $\mathcal{Q}_{\mathcal{X}}$  records a different x-value than  $\mathcal{D}_{H}$ .

The SEQ oracle applies  $\mathbb{I}$  to the first component  $\mathbf{s}_{\overline{V}}$  and applies  $\mathsf{Decomp} \circ \mathsf{CO}'_H \circ \mathsf{CO}'$ 

5. Lemma A.3 says that for any state  $|\phi\rangle$  on  $\mathcal{A}\times\mathcal{Q}\times\mathcal{O}$ , such that  $V\cdot|\phi\rangle=|\phi\rangle$ ,

$$\|E\cdot|\phi\rangle\,\|-\mathsf{negl}(\lambda)\leq \|E\cdot\mathsf{Decomp}\cdot E\cdot|\phi\rangle\,\|$$

This means that if a state starts in the span of E and V, then applying Decomp to it will not move it out of the span of E, except by a negligible amount.

Let us set  $|\phi\rangle=\frac{\mathbf{s}_V}{\|\mathbf{s}_V\|}$ . Note that  $E\,|\phi\rangle=|\phi\rangle$ , and  $V\,|\phi\rangle=|\phi\rangle$ . Then applying lemma A.3 shows that

$$\begin{aligned} 1 - \mathsf{negl}(\lambda) &\leq \frac{\|E \cdot \mathsf{Decomp} \cdot \mathbf{s}_V\|}{\|\mathbf{s}_V\|} \\ \|\mathbf{s}_V\| - \mathsf{negl}'(\lambda) &\leq \|E \cdot \mathsf{Decomp} \cdot \mathbf{s}_V\| \end{aligned}$$

6. Next, the operations  $\mathsf{CO}'_H$  and  $\mathsf{CO}'_F$  commute with E. If a state  $|\phi\rangle$  is in the span of E, then  $\mathsf{CO}'_H \circ \mathsf{CO}'_F \circ \mathsf{CO}'_H |\phi\rangle$  will be in the span of E as well. Therefore,

$$\|\mathbf{s}_V\| - \mathsf{negl}'(\lambda) \le \|E \cdot \mathsf{CO}_H' \circ \mathsf{CO}_F' \circ \mathsf{CO}_H' \circ \mathsf{Decomp} \cdot \mathbf{s}_V\|$$

Furthermore,  $CO'_H \circ CO'_F \circ CO'_H \circ Decomp \cdot s_V$  will be in the span of V since all of the operations  $CO'_H$ ,  $CO'_F$ , Decomp map a state in the span of V to a state in the span of V.

7. After applying  $CO'_H \circ CO'_F \circ CO'_H$ , the SEQ oracle applies Decomp again. We can apply lemma A.3 again to show that

$$\|\mathbf{s}_V\| - \mathsf{negl}(\lambda) \leq \|E \cdot \mathsf{Decomp} \circ \mathsf{CO}_H' \circ \mathsf{CO}_F' \circ \mathsf{CO}_H' \circ \mathsf{Decomp} \cdot \mathbf{s}_V\|$$

8. In summary, after the query to the SEQ oracle, the state is

$$|\psi'\rangle = (\mathbb{I} - V) \cdot |\psi\rangle + \left(\mathsf{Decomp} \circ \mathsf{CO}_H' \circ \mathsf{CO}_F' \circ \mathsf{CO}_H' \circ \mathsf{Decomp} \cdot V\right) \cdot |\psi\rangle$$

Then,

$$\begin{split} \|E\cdot|\psi'\rangle\,\|^2 &= \|E\cdot(\mathbb{I}-V)\cdot|\psi\rangle\,\|^2 + \|E\cdot\left(\mathsf{Decomp}\circ\mathsf{CO}_H'\circ\mathsf{CO}_F'\circ\mathsf{CO}_H'\circ\mathsf{Decomp}\cdot V\right)\cdot|\psi\rangle\,\|^2 \\ &\geq \|(\mathbb{I}-V)\cdot|\psi\rangle\,\|^2 + \|V\cdot|\psi\rangle\,\|^2 - \mathsf{negl}(\lambda) \\ &= \|\,|\psi\rangle\,\|^2 - \mathsf{negl}(\lambda) \\ &= 1 - \mathsf{negl}(\lambda) \end{split}$$

This shows that after any polynomial number of queries to the SEQ oracle, the state of the system  $|\psi\rangle$  satisfies  $|E\cdot|\psi\rangle|| \geq 1 - \mathsf{negl}(\lambda)$ .

**Lemma A.3.** For any state  $|\psi\rangle$  on  $\mathcal{A}\times\mathcal{Q}\times\mathcal{O}$ , such that  $V\cdot|\psi\rangle=|\psi\rangle$ ,  $||E\cdot|\psi\rangle||-\mathsf{negl}(\lambda)\leq ||E\cdot\mathsf{Decomp}\cdot E\cdot|\psi\rangle||$ .

Proof.

1. A generic state over  $\mathcal{A} \times \mathcal{Q} \times \mathcal{O}$  such that  $V \cdot |\psi\rangle = |\psi\rangle$  can be written as follows:

$$|\psi\rangle = \sum_{a,x,u,b} \alpha_{a,x,u,b} \cdot |a\rangle_{\mathcal{A}} \otimes |x,u,b\rangle_{\mathcal{Q}} \otimes |\Psi_{a,x,u,b}\rangle_{\mathcal{O}}$$

where  $1 = \sum_{a,x,u,b} |\alpha_{a,x,u,b}|^2$ , and for each (a,x,u,b),  $V_x \cdot |\Psi_{a,x,u,b}\rangle = |\Psi_{a,x,u,b}\rangle$ . This means the  $\mathcal{D}_H$  register of  $|\Psi_{a,x,u,b}\rangle$  contains either  $\emptyset$  or (x,r) for some  $r \in \mathcal{R}$ .

Next,

$$\begin{split} E \cdot \mathsf{Decomp} \cdot E \cdot |\psi\rangle &= \sum_{a,x,u,b} \alpha_{a,x,u,b} \cdot |a\rangle_{\mathcal{A}} \otimes |x,u,b\rangle_{\mathcal{Q}} \otimes \left(E_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi_{a,x,u,b}\rangle_{\mathcal{O}}\right) \\ \|E \cdot \mathsf{Decomp} \cdot E \cdot |\psi\rangle\|_2^2 &= \sum_{a,x,u,b} |\alpha_{a,x,u,b}|^2 \cdot \left\|E_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi_{a,x,u,b}\rangle_{\mathcal{O}}\right\|_2^2 \end{split}$$

Additionally,

$$E \cdot |\psi\rangle = \sum_{a,x,u,b} \alpha_{a,x,u,b} \cdot |a\rangle_{\mathcal{A}} \otimes |x,u,b\rangle_{\mathcal{Q}} \otimes (E_{\mathcal{O}} \cdot |\Psi_{a,x,u,b}\rangle_{\mathcal{O}})$$
$$||E \cdot |\psi\rangle||_{2}^{2} = \sum_{a,x,u,b} |\alpha_{a,x,u,b}|^{2} \cdot ||E_{\mathcal{O}} \cdot |\Psi_{a,x,u,b}\rangle_{\mathcal{O}}||_{2}^{2}$$

It suffices to prove that for any  $x \in \mathcal{X}$ , and any state  $|\Psi\rangle$  on register  $\mathcal{O}$  such that  $V_x \cdot |\Psi\rangle = |\Psi\rangle$ ,

$$\|E_{\mathcal{O}}\cdot|\Psi\rangle_{\mathcal{O}}\|_2 - \mathsf{negl}(\lambda) \leq \|E_{\mathcal{O}}\cdot\mathsf{Decomp}_x\cdot E_{\mathcal{O}}\cdot|\Psi\rangle_{\mathcal{O}}\|_2$$

because that would imply that

$$\begin{split} \sum_{a,x,u,b} |\alpha_{a,x,u,b}|^2 \cdot \left( \left\| E_{\mathcal{O}} \cdot |\Psi_{a,x,u,b}\rangle_{\mathcal{O}} \right\|_2^2 - \mathsf{negl}(\lambda) \right) &\leq \| E \cdot \mathsf{Decomp} \cdot E \cdot |\psi\rangle \|_2^2 \\ & \| E \cdot |\psi\rangle \|_2^2 - \mathsf{negl}(\lambda) = \\ & \| E \cdot |\psi\rangle \|_2 - \mathsf{negl}'(\lambda) \leq \| E \cdot \mathsf{Decomp} \cdot E \cdot |\psi\rangle \|_2 \end{split}$$

which is what we wanted to prove. From now on, we will focus on proving that for any  $x \in \mathcal{X}$ , and any state  $|\Psi\rangle$  on register  $\mathcal{O}$  such that  $V_x \cdot |\Psi\rangle = |\Psi\rangle$ ,

$$||E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}||_2 - \mathsf{negl}(\lambda) \le ||E_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}||_2$$

2.

Let 
$$E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}} = v_{\emptyset} \cdot |\emptyset\rangle_{\mathcal{D}_{H}} \otimes |F_{\emptyset}\rangle_{\mathcal{F}} + \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot |(x,r)\rangle_{\mathcal{D}_{H}} \otimes |F_{x,r,y}\rangle_{\mathcal{F}}$$

where  $\mathbf{v} = (v_{\emptyset}, (v_{r,y})_{(r,y \in \mathcal{R} \times \mathcal{Y})})$  is a vector of norm  $\|\mathbf{v}\| \leq 1$ . Next, we will show that  $\|\overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle\| = \mathsf{negl}(\lambda)$ .

$$\overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle \quad = \overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot \left( v_{\emptyset} \cdot |\emptyset\rangle_{\mathcal{D}_H} \otimes |F_{\emptyset}\rangle_{\mathcal{F}} + \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot |(x,r)\rangle_{\mathcal{D}_H} \otimes |F_{x,r,y}\rangle_{\mathcal{F}} \right)$$

The first term –  $\mathsf{Decomp}_x \cdot \left( |\emptyset\rangle_{\mathcal{D}_H} \otimes |F_{\emptyset}\rangle_{\mathcal{F}} \right)$  – lies in the span of  $E_{\mathcal{O}}$ :

$$\begin{split} \mathsf{Decomp}_x \left( |\emptyset\rangle_{\mathcal{D}_H} \otimes |F_\emptyset\rangle_{\mathcal{F}} \right) &= \frac{1}{\sqrt{|\mathcal{R}|}} \cdot \sum_{r \in \mathcal{R}} |(x,r)\rangle_{\mathcal{D}_H} \otimes |F_\emptyset\rangle_{\mathcal{F}} \\ &= \frac{1}{\sqrt{|\mathcal{R}| \cdot |\mathcal{F}|}} \cdot \sum_{(r,f) \in \mathcal{R} \times \mathcal{F}} |(x,r)\rangle_{\mathcal{D}_H} \otimes |f\rangle_{\mathcal{F}} \\ &= \frac{1}{\sqrt{|\mathcal{R}| \cdot |\mathcal{F}|}} \cdot \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} \sqrt{|\mathcal{F}_{x,r,y}|} \cdot |(x,r)\rangle_{\mathcal{D}_H} \otimes |F_{x,r,y}\rangle_{\mathcal{F}} \end{split}$$

$$\overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \left( |\emptyset\rangle_{\mathcal{D}_H} \otimes |F_{\emptyset}\rangle_{\mathcal{F}} \right) = \mathbf{0}$$

We used the fact that for any (x, r), the sets  $(\mathcal{F}_{x,r,y})_{y \in \mathcal{Y}}$  partition  $\mathcal{F}$ . Therefore, we only need to focus on the remaining terms.

$$\begin{split} \overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle &= \overline{E}_{\mathcal{O}} \cdot \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot (\mathsf{Decomp}_x \cdot |\langle x,r \rangle\rangle_{\mathcal{D}_H}) \otimes |F_{x,r,y}\rangle_{\mathcal{F}} \\ &= \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot \left(1 - \frac{1}{|\mathcal{R}|}\right) \cdot \overline{E}_{\mathcal{O}} \cdot |\langle x,r \rangle\rangle \otimes |F_{x,r,y}\rangle_{\mathcal{F}} \\ &+ \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot \frac{-1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R} \setminus \{r\}} \overline{E}_{\mathcal{O}} \cdot |\langle x,r' \rangle\rangle \otimes |F_{x,r,y}\rangle_{\mathcal{F}} \\ &+ \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot \frac{1}{\sqrt{|\mathcal{R}|}} \cdot \overline{E}_{\mathcal{O}} \cdot |\emptyset\rangle \otimes |F_{x,r,y}\rangle_{\mathcal{F}} \end{split}$$

We used the fact that

$$\mathsf{Decomp}_x \cdot |(x,r)\rangle_{\mathcal{D}_H} = \left(1 - \frac{1}{|\mathcal{R}|}\right) \cdot |(x,r)\rangle - \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R} \backslash \{r\}} |(x,r')\rangle + \frac{1}{\sqrt{|\mathcal{R}|}} \cdot |\emptyset\rangle$$

from lemma A.4.

3. Next,

$$\begin{split} \overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle &= \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot \left(1 - \frac{1}{|\mathcal{R}|}\right) \cdot \mathbf{0} \\ &+ \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot \frac{-1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R} \setminus \{r\}} |(x,r')\rangle \otimes \left(\mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \left\langle F_{x,r',y'}|\right) \cdot |F_{x,r,y}\rangle_{\mathcal{F}} \\ &+ \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} v_{r,y} \cdot \frac{1}{\sqrt{|\mathcal{R}|}} \cdot |\emptyset\rangle \otimes (\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset}|\right) \cdot |F_{x,r,y}\rangle_{\mathcal{F}} \end{split}$$

Next, we change the order of summation to obtain:

$$\begin{split} \overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle \\ &= \frac{1}{\sqrt{|\mathcal{R}|}} \cdot \sum_{r' \in \mathcal{R}} |(x,r')\rangle \otimes \left(\mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \left\langle F_{x,r',y'} \right| \right) \cdot \left( \sum_{(r,y) \in \mathcal{R} \backslash \{r'\} \times \mathcal{Y}} \frac{-v_{r,y}}{\sqrt{|\mathcal{R}|}} \cdot |F_{x,r,y}\rangle_{\mathcal{F}} \right) \\ &+ |\emptyset\rangle \otimes (\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset} | \right) \cdot \left( \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} \frac{v_{r,y}}{\sqrt{|\mathcal{R}|}} \cdot |F_{x,r,y}\rangle_{\mathcal{F}} \right) \end{split}$$

4. Let  $M_x$  be a matrix whose columns are  $\left(\frac{1}{\sqrt{|\mathcal{R}|}}\cdot|F_{x,r,y}\rangle\right)_{(r,y)\in\mathcal{R}\times\mathcal{Y}}$ . Also, for a given  $r'\in\mathcal{R}$ , let  $\mathbf{v}_{r'}$  be the same as  $\mathbf{v}$  except that for every  $y\in\mathcal{Y}$ , the (r',y)-th entry is set to 0. Note that  $\|\mathbf{v}_{r'}\|\leq \|\mathbf{v}\|\leq 1$ . Then,

$$\begin{split} \overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle &= \frac{1}{\sqrt{|\mathcal{R}|}} \cdot \sum_{r' \in \mathcal{R}} |(x,r')\rangle \otimes \left(\mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \left\langle F_{x,r',y'} \right| \right) \cdot \left( -M_x \cdot \mathbf{v}_{r'} \right) \\ &+ |\emptyset\rangle \otimes \left(\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset} \right| \right) \cdot M_x \cdot \mathbf{v} \\ \left\| \overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle \right\|_2^2 &= \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R}} \left\| \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \left\langle F_{x,r',y'} \right| \right) \cdot M_x \cdot \mathbf{v}_{r'} \right\|_2^2 \\ &+ \left\| \left( \mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset} \right| \right) \cdot M_x \cdot \mathbf{v} \right\|_2^2 \\ &\leq \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R}} \left\| \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \left\langle F_{x,r',y'} \right| \right) \cdot M_x \right\|_2^2 \cdot \left\| \mathbf{v}_{r'} \right\|_2^2 \\ &+ \left\| \left( \mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset} \right| \right) \cdot M_x \right\|_2^2 \cdot \left\| \mathbf{v} \right\|_2^2 \end{split}$$

5. We know that

$$\left\| \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} \left| F_{x,r',y'} \right\rangle \left\langle F_{x,r',y'} \right| \right) \cdot M_x \right\|_2 \le \left\| \left( \mathbb{I} - \left| F_{\emptyset} \right\rangle \left\langle F_{\emptyset} \right| \right) \cdot M_x \right\|_2 \le \frac{1}{\sqrt{|\mathcal{R}|}}$$

by Lemmas A.5 and A.6. Therefore,

$$\begin{split} \left\| \overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot \left| \Psi \right\rangle \right\|_2^2 &\leq \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R}} \frac{1}{|\mathcal{R}|} \cdot \left\| \mathbf{v}_{r'} \right\|_2^2 \\ &+ \frac{1}{|\mathcal{R}|} \cdot \left\| \mathbf{v} \right\|_2^2 \\ &\leq \frac{1}{|\mathcal{R}|} \cdot \left( \left\| \mathbf{v}_{r'} \right\|_2^2 + \left\| \mathbf{v} \right\|_2^2 \right) \leq \frac{2}{|\mathcal{R}|} \\ \left\| \overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot \left| \Psi \right\rangle \right\|_2 &\leq \sqrt{\frac{2}{|\mathcal{R}|}} = \mathsf{negl}(\lambda) \end{split}$$

6. Finally,

$$\begin{split} \|E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\|_2^2 &= \|\mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\|_2^2 \\ &= \|E_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\|_2^2 + \left\|\overline{E}_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\right\|_2^2 \\ &\leq \|E_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\|_2^2 + \frac{2}{|\mathcal{R}|} \end{split}$$

We used the fact that  $Decomp_x$  is a unitary, so it preserves norms. Then,

$$\begin{split} \|E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\|_2^2 - \frac{2}{|\mathcal{R}|} &\leq \|E_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\|_2^2 \\ \|E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\|_2 - \mathsf{negl}(\lambda) &\leq \|E_{\mathcal{O}} \cdot \mathsf{Decomp}_x \cdot E_{\mathcal{O}} \cdot |\Psi\rangle_{\mathcal{O}}\|_2 \end{split}$$

which completes the proof.

**Lemma A.4.**  $\operatorname{Decomp}_x \cdot |(x,r)\rangle_{\mathcal{D}_H} = \left(1 - \frac{1}{|\mathcal{R}|}\right) \cdot |(x,r)\rangle - \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R} \setminus \{r\}} |(x,r')\rangle + \frac{1}{\sqrt{|\mathcal{R}|}} \cdot |\emptyset\rangle$  *Proof.* 

$$\begin{split} \text{Let } \mathbf{v_0} &= \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R}} |(x, r')\rangle \\ \mathbf{v_1} &= |(x, r)\rangle - \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R}} |(x, r')\rangle \\ |(x, r)\rangle &= \mathbf{v_0} + \mathbf{v_1} \end{split}$$

Next,  $v_0$  and  $v_1$  are orthogonal:

$$\langle \mathbf{v}_{0} | \mathbf{v}_{1} \rangle = \left( \frac{1}{|\mathcal{R}|} \cdot \sum_{r'' \in \mathcal{R}} \langle (x, r'') | \right) \cdot \left( |(x, r)\rangle - \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R}} |(x, r')\rangle \right)$$

$$= \frac{1}{|\mathcal{R}|} \langle (x, r) | (x, r)\rangle - \frac{1}{|\mathcal{R}|^{2}} \cdot \sum_{r' \in \mathcal{R}} \langle (x, r') | (x, r')\rangle$$

$$= \frac{1}{|\mathcal{R}|} - \frac{1}{|\mathcal{R}|^{2}} \cdot |\mathcal{R}| = 0$$

Next,  $\mathsf{Decomp}_x$  maps  $\mathbf{v_0}$  to  $\frac{1}{\sqrt{|\mathcal{R}|}} \cdot |\emptyset\rangle$  and acts as the identity on  $\mathbf{v_1}$ . Therefore,

$$\begin{split} \mathsf{Decomp}_x \cdot |(x,r)\rangle &= \frac{1}{\sqrt{|\mathcal{R}|}} \cdot |\emptyset\rangle + \mathbf{v_1} \\ &= |(x,r)\rangle - \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R}} |(x,r')\rangle + \frac{1}{\sqrt{|\mathcal{R}|}} \cdot |\emptyset\rangle \\ &= \left(1 - \frac{1}{|\mathcal{R}|}\right) \cdot |(x,r)\rangle - \frac{1}{|\mathcal{R}|} \cdot \sum_{r' \in \mathcal{R} \backslash \{r\}} |(x,r')\rangle + \frac{1}{\sqrt{|\mathcal{R}|}} \cdot |\emptyset\rangle \end{split}$$

**Lemma A.5.** For any matrix M of the appropriate dimensions,

$$\left\| \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} \left| F_{x,r',y'} \right\rangle \left\langle F_{x,r',y'} \right| \right) \cdot M \right\|_{2} \leq \left\| \left( \mathbb{I} - \left| F_{\emptyset} \right\rangle \left\langle F_{\emptyset} \right| \right) \cdot M \right\|_{2}$$

Proof.

1. Note that  $\mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \langle F_{x,r',y'}|$  is a projector because the states  $(|F_{x,r',y'}\rangle)_{y' \in \mathcal{Y}}$  are mutually orthogonal. Therefore,

$$\left\| \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \langle F_{x,r',y'}| \right) \right\|_{2} \le 1$$

2.  $|F_{\emptyset}\rangle$  is orthogonal to  $\mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \langle F_{x,r',y'}|$ .

$$\begin{split} \left(\mathbb{I} - \sum_{y' \in \mathcal{Y}} \cdot |F_{x,r',y'}\rangle \left\langle F_{x,r',y'}|\right) \cdot |F_{\emptyset}\rangle &= |F_{\emptyset}\rangle - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \cdot \left\langle F_{x,r',y'}|F_{\emptyset}\right\rangle \\ &= |F_{\emptyset}\rangle - \sum_{y' \in \mathcal{Y}} \sum_{f \in \mathcal{F}_{x,r',y'}} |f\rangle \cdot \frac{1}{\sqrt{|\mathcal{F}_{x,r',y'}|}} \cdot \left(\sum_{f' \in \mathcal{F}_{x,r',y'}} \frac{1}{\sqrt{|\mathcal{F}_{x,r',y'}|} \cdot |\mathcal{F}|}\right) \\ &= |F_{\emptyset}\rangle - \sum_{y' \in \mathcal{Y}} \sum_{f \in \mathcal{F}_{x,r',y'}} |f\rangle \cdot \left(\frac{|\mathcal{F}_{x,r',y'}|}{|\mathcal{F}_{x,r',y'}|} \cdot \sqrt{|\mathcal{F}|}\right) \\ &= |F_{\emptyset}\rangle - \sum_{y' \in \mathcal{Y}} \sum_{f \in \mathcal{F}_{x,r',y'}} |f\rangle \cdot \frac{1}{\sqrt{|\mathcal{F}|}} \\ &= |F_{\emptyset}\rangle - \sum_{f \in \mathcal{F}} |f\rangle \cdot \frac{1}{\sqrt{|\mathcal{F}|}} = |F_{\emptyset}\rangle - |F_{\emptyset}\rangle \\ &= \mathbf{0} \end{split}$$

3. Next,

$$\begin{split} \left(\mathbb{I} - \sum_{y' \in \mathcal{Y}} \left| F_{x,r',y'} \right\rangle \left\langle F_{x,r',y'} \right| \right) \cdot \left( \mathbb{I} - \left| F_{\emptyset} \right\rangle \left\langle F_{\emptyset} \right| \right) &= \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} \left| F_{x,r',y'} \right\rangle \left\langle F_{x,r',y'} \right| \right) \\ &- \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} \left| F_{x,r',y'} \right\rangle \left\langle F_{x,r',y'} \right| \right) \cdot \left| F_{\emptyset} \right\rangle \left\langle F_{\emptyset} \right| \\ &= \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} \left| F_{x,r',y'} \right\rangle \left\langle F_{x,r',y'} \right| \right) \end{split}$$

4. Finally,

$$\left\| \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \langle F_{x,r',y'}| \right) \cdot M \right\|_{2} = \left\| \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \langle F_{x,r',y'}| \right) \cdot (\mathbb{I} - |F_{\emptyset}\rangle \langle F_{\emptyset}|) \cdot M \right\|_{2}$$

$$\leq \left\| \left( \mathbb{I} - \sum_{y' \in \mathcal{Y}} |F_{x,r',y'}\rangle \langle F_{x,r',y'}| \right) \right\|_{2} \cdot \left\| (\mathbb{I} - |F_{\emptyset}\rangle \langle F_{\emptyset}|) \cdot M \right\|_{2}$$

$$\leq \left\| (\mathbb{I} - |F_{\emptyset}\rangle \langle F_{\emptyset}|) \cdot M \right\|_{2}$$

**Lemma A.6.** Let  $M_x$  be a matrix whose columns are  $\left(\frac{1}{\sqrt{|\mathcal{R}|}} \cdot |F_{x,r,y}\rangle\right)_{(r,y)\in\mathcal{R}\times\mathcal{V}}$ . Then

$$\left\| \left( \mathbb{I} - \left| F_{\emptyset} \right\rangle \left\langle F_{\emptyset} \right| \right) \cdot M_x \right\|_2 \le \frac{1}{\sqrt{|\mathcal{R}|}}$$

Proof.

1. The (r, y), (r', y')-th entry of  $M_x^{\dagger} \cdot M_x$  is

$$(M_x^{\dagger} \cdot M_x)_{(r,y),(r',y')} = \frac{1}{|\mathcal{R}|} \cdot \langle F_{x,r,y} | F_{x,r',y'} \rangle$$

$$= \begin{cases} \frac{1}{|\mathcal{R}|}, & (r,y) = (r',y') \\ 0, & r = r' \land y \neq y' \\ \frac{1}{|\mathcal{R}|} \cdot \sqrt{\Pr_f[f(x,r) = y]} \cdot \sqrt{\Pr_f[f(x,r') = y']}, & r \neq r' \end{cases}$$

by lemma A.7.

2. Let us write  $M_x^{\dagger} \cdot M_x$  as a linear combination of PSD matrices. First, let  $|\Phi_0\rangle$  be a column vector such that the (r,y)-th entry is  $\sqrt{\frac{\Pr_f[f(x,r)=y]}{|\mathcal{R}|}}$ . Additionally, let  $|\Phi_{0,r}\rangle$  be a column vector such that the (r',y)-th entry is 0 if  $r \neq r'$ , and  $\sqrt{\Pr_f[f(x,r)=y]}$  if r=r'.

Note that these vectors have unit norm:

$$\| |\Phi_0\rangle \|_2^2 = \sum_{r,y} \frac{\Pr_f[f(x,r) = y]}{|\mathcal{R}|} = \sum_r \frac{\sum_y \Pr_f[f(x,r) = y]}{|\mathcal{R}|} = \sum_r \frac{1}{|\mathcal{R}|} = 1$$

$$\| |\Phi_{0,r}\rangle \|_2^2 = \sum_{y \in \mathcal{Y}} \Pr_f[f(x,r) = y] = 1$$

Second,

let 
$$N_x = \sum_{r \in \mathcal{R}} \frac{1}{|\mathcal{R}|} \cdot |\Phi_{0,r}\rangle \langle \Phi_{0,r}|$$

Note that  $(|\Phi_{0,r}\rangle)_{r\in\mathcal{R}}$  are mutually orthogonal, so  $N_x$  is positive semi-definite.

Third, we claim that  $M_x^{\dagger} \cdot M_x$  can be written in the following form:

$$M_x^{\dagger} \cdot M_x = \frac{1}{|\mathcal{R}|} \cdot \mathbb{I} + |\Phi_0\rangle \langle \Phi_0| - N_x$$

The (r,y),(r',y')-th entry of  $\left(\frac{1}{|\mathcal{R}|}\cdot\mathbb{I}+|\Phi_0\rangle\langle\Phi_0|-N_x\right)$  is given below. As shorthand, we let  $p_{x,r,y}=\Pr_f[f(x,r)=y]$ .

$$\left(\frac{1}{|\mathcal{R}|} \cdot \mathbb{I} + |\Phi_0\rangle \langle \Phi_0| - N_x\right)_{(r,y),(r',y')} \\
= \begin{cases}
\frac{1}{|\mathcal{R}|} \left(1 + p_{x,r,y} - p_{x,r,y}\right), & (r,y) = (r',y') \\
\frac{1}{|\mathcal{R}|} \left(0 + \sqrt{p_{x,r,y}} \cdot \sqrt{p_{x,r',y'}} - \sqrt{p_{x,r,y}} \cdot \sqrt{p_{x,r',y'}}\right), & r = r' \wedge y \neq y' \\
\frac{1}{|\mathcal{R}|} \left(0 + \sqrt{p_{x,r,y}} \cdot \sqrt{p_{x,r',y'}} - 0\right), & r \neq r'
\end{cases} \\
= \begin{cases}
\frac{1}{|\mathcal{R}|}, & (r,y) = (r',y') \\
0, & r = r' \wedge y \neq y' \\
\frac{1}{|\mathcal{R}|} \cdot \sqrt{p_{x,r,y}} \cdot \sqrt{p_{x,r',y'}}, & r \neq r'
\end{cases} \\
= (M_x^{\dagger} \cdot M_x)_{(r,y),(r',y')}$$

Therefore,  $M_x^\dagger \cdot M_x = \frac{1}{|\mathcal{R}|} \cdot \mathbb{I} + |\Phi_0\rangle \left<\Phi_0| - N_x \right.$ 

3. For any  $|\Phi_1\rangle$  of unit norm that is orthogonal to  $|\Phi_0\rangle$ ,

$$\begin{split} \|M_x\cdot|\Phi_1\rangle\,\|_2^2 &= \langle\Phi_1|\cdot M_x^\dagger\cdot M_x\cdot|\Phi_1\rangle \\ &= \langle\Phi_1|\cdot \frac{1}{|\mathcal{R}|}\cdot \mathbb{I}\cdot |\Phi_1\rangle + \langle\Phi_1|\Phi_0\rangle\cdot \langle\Phi_0|\Phi_1\rangle - \langle\Phi_1|\cdot N_x\cdot |\Phi_1\rangle = \frac{1}{|\mathcal{R}|} - \langle\Phi_1|\cdot N_x\cdot |\Phi_1\rangle \\ &\leq \frac{1}{|\mathcal{R}|} \\ \|M_x\cdot|\Phi_1\rangle\,\|_2 &\leq \frac{1}{\sqrt{|\mathcal{R}|}} \end{split}$$

We used the fact that  $\langle \Phi_1 | \cdot N_x \cdot | \Phi_1 \rangle \ge 0$  because  $N_x$  is PSD.

4. We will show that  $M_x \cdot |\Phi_0\rangle = |F_\emptyset\rangle$ .

$$M_{x} \cdot |\Phi_{0}\rangle = \frac{1}{|\mathcal{R}|} \cdot \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} |F_{x,r,y}\rangle \cdot \sqrt{\Pr_{f}[f(x,r) = y]}$$

$$= \frac{1}{|\mathcal{R}|} \cdot \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} \sum_{f \in \mathcal{F}_{x,r,y}} |f\rangle \cdot \frac{1}{\sqrt{|\mathcal{F}_{x,r,y}|}} \cdot \sqrt{\frac{|\mathcal{F}_{x,r,y}|}{|\mathcal{F}|}} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(r,y) \in \mathcal{R} \times \mathcal{Y}} \sum_{f \in \mathcal{F}_{x,r,y}} |f\rangle \cdot \frac{1}{\sqrt{|\mathcal{F}|}}$$

$$= \frac{1}{|\mathcal{R}|} \cdot \sum_{r \in \mathcal{R}} \sum_{f \in \mathcal{F}} |f\rangle \cdot \frac{1}{\sqrt{|\mathcal{F}|}} = \frac{1}{|\mathcal{R}|} \cdot \sum_{r \in \mathcal{R}} |F_{\emptyset}\rangle$$

$$= |F_{\emptyset}\rangle$$

We used the fact that for any (x,r), the sets  $(\mathcal{F}_{x,r,y})_{y\in\mathcal{Y}}$  partition  $\mathcal{F}$ .

5. Any vector  $|\Phi\rangle$  of unit norm can be written as  $|\Phi\rangle = \alpha \cdot |\Phi_0\rangle + \beta \cdot |\Phi_1\rangle$  for some vector  $|\Phi_1\rangle$ of unit norm that is orthogonal to  $|\Phi_0\rangle$  and for some  $\alpha, \beta$  for which  $|\alpha|^2 + |\beta|^2 = 1$ . Next,

$$\begin{split} \left(\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset}|\right) \cdot M_{x} \cdot |\Phi\rangle &= \left(\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset}|\right) \cdot M_{x} \cdot \left(\alpha \cdot |\Phi_{0}\rangle + \beta \cdot |\Phi_{1}\rangle\right) \\ &= \alpha \cdot \left(\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset}|\right) \cdot |F_{\emptyset}\rangle + \beta \cdot \left(\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset}|\right) \cdot M_{x} \cdot |\Phi_{1}\rangle \\ &= \beta \cdot \left(\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset}|\right) \cdot M_{x} \cdot |\Phi_{1}\rangle \\ \|\left(\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset}|\right) \cdot M_{x} \cdot |\Phi\rangle \right\|_{2} \leq |\beta| \cdot \|\left(\mathbb{I} - |F_{\emptyset}\rangle \left\langle F_{\emptyset}|\right)\|_{2} \cdot \|M_{x} \cdot |\Phi_{1}\rangle \|_{2} \\ &\leq 1 \cdot 1 \cdot \frac{1}{\sqrt{|\mathcal{R}|}} = \frac{1}{\sqrt{|\mathcal{R}|}} \end{split}$$

We've shown that for any vector  $|\Phi\rangle \neq \mathbf{0}$ ,  $\frac{\|(\mathbb{I}-|F_{\emptyset}\rangle\langle F_{\emptyset}|)\cdot M_x\cdot |\Phi\rangle\|_2}{\||\Phi\rangle\|_2} \leq \frac{1}{\sqrt{|\mathcal{R}|}}$ , so

$$\|(\mathbb{I} - |F_{\emptyset}\rangle\langle F_{\emptyset}|) \cdot M_x\|_2 \le \frac{1}{\sqrt{|\mathcal{R}|}}$$

**Lemma A.7.** For any  $(x, r, y), (x', r', y') \in \mathcal{X} \times \mathcal{R} \times \mathcal{Y}$ ,

- $\langle F_{\emptyset}|F_{x,r,y}\rangle = \sqrt{\Pr_f[f(x,r)=y]}$  If (x,r)=(x',r') and  $y\neq y'$ , then  $\langle F_{x,r,y}|F_{x',r',y'}\rangle = 0$ .

• If  $(x,r) \neq (x',r')$ , then  $\langle F_{x,r,y} | F_{x',r',y'} \rangle = \sqrt{\Pr_f[f(x,r) = y] \cdot \Pr_f[f(x',r') = y']}$ .

Proof.

$$\langle F_{\emptyset}|F_{x,r,y}\rangle = \left(\frac{1}{\sqrt{|\mathcal{F}|}} \cdot \sum_{f \in \mathcal{F}} \langle f|\right) \cdot \left(\frac{1}{\sqrt{|\mathcal{F}_{x,r,y}|}} \cdot \sum_{f' \in \mathcal{F}_{x,r,y}} |f'\rangle\right)$$

$$= \frac{1}{\sqrt{|\mathcal{F}| \cdot |\mathcal{F}_{x,r,y}|}} \cdot \sum_{f \in \mathcal{F}_{x,r,y}} \langle f|f\rangle$$

$$= \frac{|\mathcal{F}_{x,r,y}|}{\sqrt{|\mathcal{F}| \cdot |\mathcal{F}_{x,r,y}|}} = \sqrt{\frac{|\mathcal{F}_{x,r,y}|}{|\mathcal{F}|}}$$

$$= \sqrt{\Pr_{f}[f(x,r) = y]}$$

Next, if (x,r)=(x',r'), but  $y\neq y'$ , then  $\mathcal{F}_{x,r,y}\cap\mathcal{F}_{x',r',y'}=\{\}$ , so  $\langle F_{x,r,y}|F_{x',r',y'}\rangle=0$ . Finally, if  $(x,r)\neq (x',r')$ , then

$$\langle F_{x,r,y}|F_{x',r',y'}\rangle = \left(\frac{1}{\sqrt{|\mathcal{F}_{x,r,y}|}} \cdot \sum_{f \in \mathcal{F}_{x,r,y}} \langle f|\right) \cdot \left(\frac{1}{\sqrt{|\mathcal{F}_{x',r',y'}|}} \cdot \sum_{f' \in \mathcal{F}_{x',r',y'}} |f'\rangle\right)$$

$$= \frac{1}{\sqrt{|\mathcal{F}_{x,r,y}| \cdot |\mathcal{F}_{x',r',y'}|}} \cdot \sum_{f \in \mathcal{F}_{x,r,y} \cap \mathcal{F}_{x',r',y'}} \langle f|f\rangle$$

$$= \frac{|\mathcal{F}_{x,r,y} \cap \mathcal{F}_{x',r',y'}|}{\sqrt{|\mathcal{F}_{x,r,y}| \cdot |\mathcal{F}_{x',r',y'}|}} = \frac{|\mathcal{F}| \cdot \Pr_{f}[f(x,r) = y \wedge f(x',r') = y']}{\sqrt{|\mathcal{F}| \cdot \Pr_{f}[f(x,r) = y] \cdot |\mathcal{F}|} \cdot \Pr_{f}[f(x',r') = y']}$$

$$= \frac{\Pr_{f}[f(x,r) = y] \cdot \Pr_{f}[f(x',r') = y']}{\sqrt{\Pr_{f}[f(x,r) = y] \cdot \Pr_{f}[f(x',r') = y']}} = \sqrt{\Pr_{f}[f(x,r) = y] \cdot \Pr_{f}[f(x',r') = y']}$$

# B Security Proof for Definition 4.26 with Measure-and-Reprogram

In this section, we show that if we aim at proving the weak operational one-time security definition Definition 4.26 for random functions with superpolynomial range size, for the construction in Section 5.1, we can have a simple proof using a technique called measure-and-reprogram lemma developed in [DFMS19]. The following proof actually is applicable to a class of functions we call 2-replaceable below, which is more generic than truly random functions.

#### **B.1** Preliminaries

We review the measure-and-reprogram lemma developed in [DFMS19] and [DFM20]. We adopt the formulation presented in in [YZ21, Section 4.2].

**Definition B.1** (Reprogramming Oracle). Let A be a quantum algorithm that is given quantum oracle access to an oracle O, where O is an oracle that is intialized to compute a classical function  $f: \mathcal{X} \to \mathcal{Y}$ 

such that A. At some point in an execution of  $A^{\mathcal{O}}$ , we say that we reprogram  $\mathcal{O}$  to output g(x) on  $x \in \mathcal{X}$ if we update the oracle to compute the function  $f_{x,q}$  defined by

$$f_{x,g}(x') = \begin{cases} g(x') & \text{if } x = x', \\ f(x') & \text{otherwise.} \end{cases}$$

This updated oracle is used in the rest of execution of  $\mathcal{A}^{\mathcal{O}}$ . We denote the above reprogramming procedure  $as \mathcal{O} \leftarrow \mathsf{Reprogram}(\mathcal{O}, x', g).$ 

**Definition B.2** (Measure-and-Reprogram Algorithm). Let  $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$  be a set of classical strings and k be a positive integer. Let A be a q-quantum-query algorithm that is given quantum oracle access to an oracle that computes a classical function  $f: \mathcal{X} \to \mathcal{Y}$ . The algorithm  $\mathcal{A}$ , when given a (possibly quantum) input input, outputs  $\mathbf{x} \in \mathcal{X}^k$  and  $z \in \mathcal{Z}$ . For a function  $g: \mathcal{X} \to \mathcal{Y}$ , we define a measure-and-reprogram algorithm  $\hat{\mathcal{A}}[f,g]$  as follows:

## $\tilde{\mathcal{A}}[f,g](\mathsf{input})$

- 1. For each  $j \in [k]$ , uniformly pick  $(i_j, b_j) \in ([q] \times \{0, 1\}) \cup \{(\bot, \bot)\}$  such that there does not exist  $j \neq j'$  such that  $i_j = i_{j'} \neq \bot$ .
- 2. Run  $\mathcal{A}^{\mathcal{O}}(input)$  where the oracle  $\mathcal{O}$  is initialized to be a quantumly-accessible classical oracle that computes the classical function f. When A makes its ith query, the oracle is simulated as follows:
  - (a) If  $i = i_j$  for some  $j \in [k]$ , measure A's query register to obtain  $x'_j$ , and do either of the following:
    - i. If  $b_i = 0$ , reprogram  $\mathcal{O} \leftarrow \mathsf{Reprogram}(\mathcal{O}, x_i', g)$  and answer  $\mathcal{A}$ 's ith query using the reprogrammed oracle.
    - ii. If  $b_i = 1$ , answer A's ith query by using the oracle before the reprogramming and then reprogram  $\mathcal{O} \leftarrow \mathsf{Reprogram}(\mathcal{O}, x_i', g)$ .
  - (b) Otherise, answer A's ithe query by just using the oracle O without any measurement or reprogramming.
- 3. Let  $(\mathbf{x} = (x_1, \dots, x_k), z)$  be  $\mathcal{A}'s$  output.
- 4. For all  $j \in [k]$  such that  $i_j = \bot$ , set  $x'_j := x_j$ . 5. Output  $(\mathbf{x}', z)$ , where  $\mathbf{x}' := (x'_1, \ldots, x'_k)$ .

**Lemma B.3.** Let  $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ , and  $\mathcal{A}$  be as in Definition B.2. Then, for any input input, functions  $f, g: \mathcal{X} \to \mathcal{X}$  $\mathcal{Y}$ ,  $\mathbf{x}^* \in \mathcal{X}^k$  such that  $x_i^* \neq x_i^*$ , for all  $j \neq j'$ , and relation  $R \subseteq \mathcal{X}^k \times \mathcal{Y}^k \times \mathcal{Z}$ , we have

$$\Pr\left[\begin{array}{c} \mathbf{x}' = \mathbf{x}^* \; \wedge \\ (\mathbf{x}', g(\mathbf{x}'), z) \in R \end{array} : (\mathbf{x}', z) \leftarrow \tilde{\mathcal{A}}[f, g](\mathsf{input}) \right] \geq \frac{1}{2q+1}^{2k} \Pr\left[\begin{array}{c} \mathbf{x} = \mathbf{x}^* \; \wedge \\ (\mathbf{x}, g(\mathbf{x}'), z) \in R \end{array} : (\mathbf{x}, z) \leftarrow \mathcal{A}^{|f_{\mathbf{x}^*, g}\rangle}(\mathsf{input}) \right],$$

where  $\tilde{\mathcal{A}}[f,g]$  is the measure-and-reprogram algorithm as defined in Definition B.2, and  $f_{\mathbf{x}^*,g}$  is defined as

$$f_{\mathbf{x}^*,g}(x') := \begin{cases} g(x') & \text{if } \exists j \in [k] \text{ s.t. } x' = x_j^*, \\ f(x') & \text{otherwise.} \end{cases}$$

**Definition B.4** (k-wise replaceable). A function family  $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$  is k-wise replaceable if the following holds. For all  $\mathbf{x} \in \mathcal{X}^k$ , the distribution over functions  $f'_{\mathbf{x},f(\mathbf{x})}$  obtained by sampling  $f,f' \leftarrow \mathcal{F}$  is the same as the distribution of functions obtained by sampling  $f \leftarrow \mathcal{F}$ .

A random function with superpolynomial range size is k-wise replaceable as shown in [YZ21].

#### **B.2** Security Proof

**Theorem B.5.** Let  $\mathcal{F}$  be a function family that is 1-query unlearnable Definition 4.11 and 2-wise replaceable. Then the scheme in Figure 3 satisfies the one-time security of Definition 4.26.

*Proof.* We first prove one-time secrecy for what we call a "mini-scheme" where the user's input is one bit, i.e.  $\mathcal{X} = \{0, 1\}$ . We will call this user input  $b \in \{0, 1\}$  for "bit".

Suppose for contradiction that there exists a (quantum) polynomial time algorithm  $\mathcal{A}$  that takes as input one-time sampling program  $\mathcal{P}_f = (|A\rangle, \mathcal{O}_A, \mathcal{O}_{A^\perp}, \mathcal{O}_{f,A})$  and breaks one-time secrecy with non-negligible probability. Note that we can consider  $\mathcal{A}$  as having access to oracles  $f_A$ , which compute f restricted on valid inputs:

$$f_A(b,u) = \begin{cases} f(b,H(u)) & \text{if } u \in A^b, \\ \bot & \text{otherwise.} \end{cases}$$

We will use A to construct a (quantum) polynomial time algorithm B that either (1) breaks the one-query unlearnability of f, or (2) violates the direct product hardness for random subspaces.

$$\mathcal{B}(1^{\lambda}, \mathcal{P}_f)$$

- 1. Sample a random function  $g \leftarrow \mathcal{F}$ .
- 2. Run the measure-and-reprogram algorithm  $\tilde{\mathcal{A}}[g, f_A]$  as defined in Definition B.2 with k=2.
- 3. Output the output of  $\tilde{\mathcal{A}}[g, f_A]$ .

Using the shorthand  $\mathbf{x}=(x_1,x_2)\in(\{0,1\}\times\mathcal{R})^2$  where  $x_1=(b_1,r_1)$  and  $x_2=(b_2,r_2)$  and  $\mathbf{y}=(y_1,y_2)\in\mathcal{Y}^2$ , define the relation  $R\subseteq(\{0,1\}\times\mathcal{R})^2\times\mathcal{Y}^2$  as

$$R = \{(\mathbf{x}, \mathbf{y}) \mid f(x_1) = y_1, f(x_2) = y_2\}.$$

By Lemma B.3, we have that for all  $\mathbf{x}^* \in (\{0,1\} \times \mathcal{R})^2$  such that  $x_1^* \neq x_2^*$ ,

$$\begin{split} &\Pr_{f \leftarrow \mathcal{F}} \left[ (\mathbf{x}, \mathbf{y}) \in R \ \land \mathbf{x} = \mathbf{x}^* : (\mathbf{x}, \mathbf{y}) \leftarrow \mathcal{B}(1^{\lambda}, \mathcal{P}_f) \right] \\ \geq &\frac{1}{(2q+1)^4} \Pr_{f, g \leftarrow \mathcal{F}} \left[ (\mathbf{x}, \mathbf{y}) \in R \ \land \mathbf{x} = \mathbf{x}^* : (\mathbf{x}, \mathbf{y}) \leftarrow \mathcal{A}^{|g_{\mathbf{x}^*, f}\rangle}(1^{\lambda}, |A\rangle, \mathcal{O}_A, \mathcal{O}_{A^{\perp}}) \right]. \end{split}$$

Since  $\mathcal{F}$  is 2-wise replaceable, the above probability expression is

$$\begin{split} &= \frac{1}{(2q+1)^4} \Pr_{f \leftarrow \mathcal{F}} \left[ (\mathbf{x}, \mathbf{y}) \in R \ \land \mathbf{x} = \mathbf{x}^* : (\mathbf{x}, \mathbf{y}) \leftarrow \mathcal{A}^{|f\rangle}(1^\lambda, |A\rangle \,, \mathcal{O}_A, \mathcal{O}_{A^\perp}) \right] \\ &= \frac{1}{(2q+1)^4} \Pr_{f \leftarrow \mathcal{F}} \left[ (\mathbf{x}, \mathbf{y}) \in R \ \land \mathbf{x} = \mathbf{x}^* : (\mathbf{x}, \mathbf{y}) \leftarrow \mathcal{A}(1^\lambda, \mathcal{P}_f) \right] \geq \text{non-negl}(\lambda), \end{split}$$

where the last inequality holds because A breaks the one-time security of  $\mathcal{P}_f$ , by assumption. Note that  $\mathcal{B}$  makes at most 2 queries, both of which are classical, to  $\mathcal{O}_{f,A}$ . Consider the following two cases:

**Case 1.**  $\mathcal{B}$  makes two valid and distinct classical queries  $x_1 = (b_1, r_1), x_2 = (b_2, r_2) \in \{0, 1\} \times \mathcal{R}$  to  $\mathcal{O}_{f,A}$  such that  $\mathcal{O}_{f,A}(x_1) \neq \bot$  and  $\mathcal{O}_{f,A}(x_2) \neq \bot$ . This means that  $r_1 \in A^{b_1}$  and  $r_2 \in A^{b_2}$ . Since these are classical queries, they can be recorded, and this immediately gives us an adversary that breaks the direct product hardness of subspace states (Theorem 3.9).

**Case 2.**  $\mathcal{B}$  makes at most one valid classical queries  $x=(b,r)\in\{0,1\}\times\mathcal{R}$  to  $\mathcal{O}_{f,A}$  such that  $\mathcal{O}_{f,A}(x)\neq \bot$ . This is impossible because it would break the 1-query unlearnability of  $\mathcal{F}$ .

**Remark B.6.** We conjecture that the above proof extends to function families larger than uniformly random functions (which are shown to be k-wise replaceable in [YZ21]. We will leave to future works on how to characterize the k-wise replaceable property.

## C More on the Compressed Oracle

#### C.1 Alternative Representations of the Decompression Function

Let D be a database for a random function H such that  $D(x) = \bot$ . If one were to query H and receive result H(x) = y, the state of the oracle before the final decompression operation is  $|D \cup (x,y)\rangle$ . The effect of the x decompression operation D on a state  $|D \cup (x,y)\rangle$  is

$$\begin{split} &\operatorname{Decomp}_x \left| D \cup (x,y) \right\rangle \\ &= \operatorname{Decomp}_x \frac{1}{|\mathcal{Y}|} \sum_{y'} \left| D \cup (x,y') \right\rangle \sum_{r \in \mathcal{Y}} (-1)^{r \cdot (y'-y)} \\ &= \operatorname{Decomp}_x \frac{1}{|\mathcal{Y}|} \left( \sum_{r \neq 0} (-1)^{r \cdot -y} \sum_{y'} (-1)^{r \cdot y'} \left| D \cup (x,y') \right\rangle + \sum_{y'} \left| D \cup (x,y') \right\rangle \right) \\ &= \frac{1}{|\mathcal{Y}|} \left( \sum_{r \neq 0} (-1)^{r \cdot -y} \sum_{y'} (-1)^{r \cdot y'} \left| D \cup (x,y') \right\rangle + \sqrt{|\mathcal{Y}|} \left| D \right\rangle \right) \\ &= \left( \left| D \cup (x,y) \right\rangle - \frac{1}{|\mathcal{Y}|} \sum_{y'} \left| D \cup (x,y') \right\rangle + \frac{1}{\sqrt{|\mathcal{Y}|}} \left| D \right\rangle \right) \\ &= \left( \left( 1 - \frac{1}{|\mathcal{Y}|} \right) \left| D \cup (x,y) \right\rangle - \frac{1}{|\mathcal{Y}|} \sum_{y' \neq y} \left| D \cup (x,y') \right\rangle + \frac{1}{\sqrt{|\mathcal{Y}|}} \left| D \right\rangle \right) \end{split}$$

We note that this only describes the state *in between* queries to x. A second query to x will apply  $\mathsf{Decomp}_x$  to the database register before determining the response, which maps the database register back to  $|D \cup (x,y)\rangle$ . Thus, despite the support of the state on other databases, G(x) cannot actually change in between queries to x. However, this form is relevant when looking at the database register without knowledge of x.

### C.2 Compressed Oracle Chaining

In this section, we show that if an adversary has access to the composition of compressed oracles  $H \circ G$  and H records some input/output pair H(y) = z, then with overwhelming probability G also records a matching input/output pair G(x) = y.

Following almost the same analysis, we will also show that if an adversary has access to a function F(x) = H(x, G(x)) and H records H(x||y) = z, then with overwhelming probability G contains a matching entry G(x) = y. See Lemma C.4 at the end of this subsection for more details.

**Lemma C.1** (Compressed Oracle Chaining). Let  $G: \mathcal{X} \to \mathcal{Y}$  and  $H: \mathcal{Y} \to \mathcal{Z}$  be random oracles implemented by the compressed oracle technique. Consider running an interaction of an oracle algorithm with their composition  $H \circ G$  until query t, then measuring the internal state of G and H to obtain  $D_G$  and  $D_H$ .

Let  $E_t$  be the event that after the measurement at time t, for all  $(y, z) \in D_H$ , there exists an  $x \in \mathcal{X}$  such that  $(x, y) \in D_G$ . Then

$$\Pr[E_t] \ge 1 - 4t^2 \left(\frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^2}\right)$$

*Proof.* We first explicitly describe how  $H \circ G$  works. The internal states of H and G are stored in registers  $\mathcal{D}_H$  and  $\mathcal{D}_G$ , respectively. In general, we can consider a basis query  $|x,u\rangle_{\mathcal{Q}}$  in register  $\mathcal{Q} = \mathcal{Q}_{\mathcal{X}}, \mathcal{Q}_{\mathcal{Z}}$ . For convenience, we insert into  $\mathcal{Q}$  an additional work register  $\mathcal{Q}_{\mathcal{Y}}$  which is initialized to  $|0\rangle$ , resulting in  $|x,0,u\rangle_{\mathcal{Q}}$ . To compute  $H \circ G$  on this query, first query  $(\mathcal{Q}_{\mathcal{X}},\mathcal{Q}_{\mathcal{Y}})$  to G to obtain  $|x,G(x),u\rangle_{\mathcal{Q}}$ , i.e. apply  $\mathsf{Decomp}_G \circ \mathsf{CO}_G' \circ \mathsf{Decomp}_G$  on  $(\mathcal{Q}_{\mathcal{X}},\mathcal{Q}_{\mathcal{Y}},\mathcal{D}_G)$ . Then query  $(\mathcal{Q}_{\mathcal{Y}},\mathcal{Q}_{\mathcal{Z}})$  to H to obtain  $|x,G(x),u\oplus H(G(x))\rangle$ . Finally, query  $(\mathcal{Q}_{\mathcal{X}},\mathcal{Q}_{\mathcal{Y}})$  to G again to obtain  $|x,0,u\oplus H(G(x))\rangle$  and return registers  $(\mathcal{Q}_{\mathcal{X}},\mathcal{Q}_{\mathcal{Z}})$ .

Observe that  $\mathsf{Decomp}_G$  commutes with the query to H, since they operate on disjoint registers  $(\mathcal{Q}_{\mathcal{X}}, \mathcal{D}_G)$  and  $(\mathcal{Q}_{\mathcal{Y}}, \mathcal{Q}_{\mathcal{Z}}, \mathcal{D}_G)$ , respectively. Furthermore,  $\mathsf{Decomp}_G \circ \mathsf{Decomp}_G = I$ . Thus, if we write the operation of H as  $U_H$ , we may write the implementation of a query to  $H \circ G$  as

$$U_{H \circ G} := (\mathsf{Decomp}_G \circ \mathsf{CO}'_G) \circ (U_H) \circ (\mathsf{CO}'_G \circ \mathsf{Decomp}_G)$$

where  $\mathsf{Decomp}_G$  and  $\mathsf{CO}_G'$  act on registers  $(\mathcal{Q}_{\mathcal{X}},\mathcal{Q}_{\mathcal{Y}},\mathcal{D}_G)$ , while  $U_H$  acts on registers  $(\mathcal{Q}_{\mathcal{Y}},\mathcal{Q}_{\mathcal{Z}},\mathcal{D}_H)$ . Now consider the interaction of the algorithm with  $H \circ G$ , where the algorithm maintains an additional internal register  $\mathcal{A}$ . Define the projector E onto states  $|a\rangle_{\mathcal{A}}\otimes|x,0,u\rangle_{\mathcal{Q}}\otimes|D_G,D_H\rangle_{\mathcal{D}}$  where  $D_G$  and  $D_H$  satisfy the requirements of event  $E_t$  and define  $\overline{E}=I-E$ . We will upper bound the norm  $\|\overline{E}U_{H\circ G}|\psi\rangle\|$  from after the query in terms of the norm  $\|\overline{E}|\psi\rangle\|$  from before the query. To do this, we will individually bounding the norm after each step in terms of the norm before that step, e.g. bound  $\|\overline{E}\mathsf{Decomp}_G|\psi'\rangle\|$  from after the query in terms of the norm  $\|\overline{E}|\psi'\rangle\|$ .

#### Claim C.2.

$$\left\|\overline{E}(\mathsf{CO}_G')\cdot (U_H)\cdot (\mathsf{CO}_G'\cdot \mathsf{Decomp}_G)\left|\psi\right>\right\| = \left\|\overline{E}(\mathsf{Decomp}_G)\left|\psi\right>\right\|$$

We first bound the intermediate operations, in between the two  $Decomp_G$  operations.

*Proof.* First, observe that for all states  $|\psi'\rangle$ ,

$$\left\|\overline{E}\cdot\operatorname{CO}_G'\left|\psi'\right>\right\|=\left\|\overline{E}\left|\psi'\right>\right\|$$

since  $CO'_G$  does not modify the database registers.

Now consider the operation of  $U_H$  on  $(\mathsf{CO}' \cdot \mathsf{Decomp}_G) | \psi \rangle$ . Observe that by the definition of  $\mathsf{CO}'_G$ , the state of the query register and  $H \circ G$  when H is queried is always supported on basis states

$$|x,y,u\rangle_{\mathcal{Q}}\otimes|D_G\cup(x,y)\rangle_{\mathcal{D}_G}\otimes|D_H\rangle_{\mathcal{D}_H}$$

A query of y to H can only result in a new database  $D'_H$  where the only potential difference between  $D_H$  and  $D'_H$  is  $D_H(y) \neq D'_H(y)$ . Since (x,y) is already recorded in the contents of register

<sup>&</sup>lt;sup>15</sup>Since G and H have different domains and ranges, we differentiate their decompression operations, which depend on the domain/range. We also differentiate the  $CO'_G$  operation to clarify that it acts on registers corresponding to a query to G.

 $\mathcal{D}_G$ , applying H always results in a valid database state with respect to the event E. In particular, if  $D_H$  and  $D_G$  already satisfied E before the query to H, they continue to do so afterwards. Thus

$$\left\|\overline{E}(U_H)\cdot \left(\mathsf{CO}_G'\cdot\mathsf{Decomp}_G)\left|\psi\right>\right\| = \left\|\overline{E}(\mathsf{CO}_G'\cdot\mathsf{Decomp}_G)\left|\psi\right>\right\|$$

Putting this together with the prior bound on the effect of  $CO'_G$  yields the claim.

Next, we bound the effect of  $\mathsf{Decomp}_G$  on a general state  $|\psi'\rangle$ . In general, both the first and the last  $\mathsf{Decomp}_G$  operation will operate on a state of the form

$$\sum_{a,x,u,D_G,D_H} \alpha_{a,x,u,D_G,D_H} |a\rangle_{\mathcal{A}} \otimes |x,0,u\rangle_{\mathcal{Q}} \otimes |D_G,D_H\rangle_{\mathcal{D}}$$

where A is the adversary's internal register, Q contains the (expanded) query, and  $\mathcal{D}$  contains the oracle databases. This is clearly true for the first application; it holds for the second because  $\mathsf{CO}'_G$  is applied twice and  $U_H$  does not modify the registers that  $\mathsf{CO}'_G$  acts on.<sup>16</sup>

#### Claim C.3.

$$\left\|\overline{E}\mathrm{Decomp}_G\left|\psi'\right>\right\| \leq \left\|\overline{E}\left|\psi\right>\right\| + \sqrt{\frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^2}}$$

*Proof.* Define the following projectors on states  $|a\rangle \otimes |x,0,u\rangle \otimes |D_G,D_H\rangle$ :

- $E_{Y1,Z}$  projects onto states where (1)  $D_G$  and  $D_H$  satisfy E, (2)  $D_G(x) = y$  for some y, (3)  $D_H(y) = z$  for some z, and (4) x is the unique preimage of y under G, i.e.  $D_G(x') \neq y$  for all  $x \neq x'$ .
- $E_{Y+,Z}$  projects onto states where (1)  $D_G$  and  $D_H$  satisfy E, (2)  $D_G(x) = y$  for some y, (3)  $D_H(y) = z$  for some z, and (4) there exists at least one  $x' \neq x$  such that  $D_G(x') = y$ .
- $E_{Y,\perp}$  projects onto states where (1)  $D_G$  and  $D_H$  satisfy E, (2)  $D_G(x) = y$  for some y, and (3)  $D_H(y) = \bot$ .
- $E_{\perp}$  projects onto states where (1)  $D_G$  and  $D_H$  satisfy E, (2)  $D_G(x) = \bot$ .

Observe that  $E=E_{Y1,Z}+E_{Y+,Z}+E_{Y,\perp}+E_{\perp}$ , so  $I=\overline{E}+E_{Y1,Z}+E_{Y+,Z}+E_{Y,\perp}+E_{\perp}$ . Furthermore, Decomp<sub>G</sub> only modifies register  $\mathcal{D}_G$ , where it maps  $D_G$  to  $D_G'$ , with the only potential difference that  $D_G(x)\neq D_G'(x)$ . In the case where x is not part of an  $(x,y)\in D_G$  and  $(y,z)\in D_H$  pair mandated by E, this modification does not affect the containment of the state in space E. Thus

$$\left\|\overline{E}\cdot\mathsf{Decomp}_G\cdot E_{Y,\perp}\left|\psi'\right>\right\| = \left\|\overline{E}\cdot\mathsf{Decomp}_G\cdot E_{\perp}\left|\psi'\right>\right\| = 0$$

Similarly, because there is a "backup" option  $D_G(x') = y$  for y in the case  $E_{Y+,Z}$ ,

$$\left\| \overline{E} \cdot \mathsf{Decomp}_G \cdot E_{Y+,Z} \left| \psi' \right\rangle \right\| = 0$$

For basis vectors in the support of  $E_{Y1,Z}$ , we can write  $D_G = D'_G \cup (x,y)$  and  $D_H = D'_H \cup (y,z)$ , where  $D'_G(x) = \bot$  and  $D'_H(y) = \bot$ .  $D'_G$  and  $D'_H$  represent  $D_G$  and  $D_H$  with x and y removed, respectively. Furthermore, there does not exist an  $x' \in \mathcal{X}$  such that  $D'_G(x') = y$ , since

<sup>&</sup>lt;sup>16</sup>We note that  $CO'_G$  and  $U_H$  do not commute, despite this, since  $U_H$  does certain actions controlled on the registers that  $CO'_G$  acts on.

x is the unique preimage of y under G. Recall from Appendix C.1 that the effect of  $Decomp_G$  on  $|x,0,D\cup(x,y)\rangle$  is

$$|x,0
angle \otimes \left(\left(1-rac{1}{|\mathcal{Y}|}
ight)|D\cup(x,y)
angle - rac{1}{|\mathcal{Y}|}\sum_{y'
eq y}|D\cup(x,y')
angle + rac{1}{\sqrt{|\mathcal{Y}|}}|D
angle
ight)$$

Since  $D_H(y) \neq \bot$ , the projection  $\overline{E} \cdot \mathsf{Decomp}_G |a\rangle_{\mathcal{A}} \otimes |x,0,u\rangle_{\mathcal{Q}} \otimes |D'_G \cup (x,y), D'_H \cup (y,z)\rangle_{\mathcal{D}}$  is

$$|a, x, 0, D'_H \cup (y, z)\rangle_{\mathcal{A}, \mathcal{Q}, \mathcal{D}_H} \otimes \left(-\frac{1}{|\mathcal{Y}|} \sum_{y' \neq y} |D'_G \cup (x, y')\rangle + \frac{1}{\sqrt{|\mathcal{Y}|}} |D'_G\rangle\right)_{\mathcal{D}_G}$$

We can write  $E_{Y1,Z} | \psi' \rangle$  in general as

$$E_{Y1,Z} |\psi'\rangle = \sum_{a,x,u,y} \sum_{D'_G \not\ni u, D_H \ni u} \alpha_{a,x,u,y,D'_G,D_H} |a\rangle \otimes |x,0,u\rangle \otimes |D'_G \cup (x,y), D_H\rangle$$

Then we can compute

$$\begin{split} \left\| \overline{E} \cdot \mathsf{Decomp}_G \cdot E_{Y1,Z} \left| \psi' \right\rangle \right\| &= \sum_{\substack{a,x,u,y \\ D'_G \not\ni y, D_H \ni y}} \left\| \alpha_{a,x,u,D'_G,D_H} \left( -\frac{1}{|\mathcal{Y}|} \sum_{y' \neq y} |D'_G \cup (x,y') \rangle + \frac{1}{\sqrt{|\mathcal{Y}|}} |D'_G \rangle \right) \right\| \\ &= \sum_{\substack{a,x,u,y \\ D'_G \not\ni y, D_H \ni y}} \left\| \alpha_{a,x,u,D'_G,D_H} \right\| \sqrt{\frac{|\mathcal{Y}| - 1}{|\mathcal{Y}|^2} + \frac{1}{|\mathcal{Y}|}} \\ &= \left\| E_{Y1,Z} \left| \psi' \right\rangle \right\| \sqrt{\frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^2}} \end{split}$$

Finally,  $\|\overline{E}\cdot \mathsf{Decomp}_G\cdot\overline{E}\,|\psi\rangle\| \leq \|\overline{E}\,|\psi\rangle\|$ . Putting these bounds together with the decomposition  $|\psi'\rangle = \overline{E}\,|\psi'\rangle + E_{Y,Z}\,|\psi'\rangle + E_{Y,Z}\,|\psi'\rangle + E_{Y,L}\,|\psi'\rangle + E_{L}\,|\psi'\rangle$ , we have

$$\begin{split} \left\| \overline{E} \cdot \mathsf{Decomp}_{G} \left| \psi \right\rangle \right\| &\leq \left\| \overline{E} \left| \psi \right\rangle \right\| + \left\| E_{Y1,Z} \left| \psi' \right\rangle \right\| \sqrt{\frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^{2}}} + 0 + 0 + 0 \\ &\leq \left\| \overline{E} \left| \psi \right\rangle \right\| + \sqrt{\frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^{2}}} \end{split}$$

Putting together Claim C.2 and Claim C.3, the norm after any single query to  $H \circ G$  is bounded as

$$\begin{split} \left\| \overline{E} \cdot U_{H \circ G} \left| \psi \right\rangle \right\| &\leq \left\| \overline{E} \cdot (\mathsf{CO}_G') \cdot (U_H) \cdot (\mathsf{CO}' \cdot \mathsf{Decomp}_G) \left| \psi \right\rangle \right\| + \sqrt{\frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^2}} \\ &= \left\| \overline{E} \mathsf{Decomp}_G \left| \psi \right\rangle \right\| + \sqrt{\frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^2}} \\ &\leq \left\| \overline{E} \left| \psi \right\rangle \right\| + 2\sqrt{\frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^2}} \end{split}$$

The norm starts at 0, so after t queries to  $H \circ G$  it is at most  $2t\sqrt{\frac{2}{|\mathcal{Y}|}-\frac{1}{|\mathcal{Y}|^2}}$ . The probability of seeing the event corresponding to  $\overline{E}$  when we measure  $\mathcal{D}$  is the square of the norm, which is at most  $4t^2\left(\frac{2}{|\mathcal{Y}|}-\frac{1}{|\mathcal{Y}|^2}\right)$ . Therefore the probability of seeing the complementary event E is at least  $1-4t^2\left(\frac{2}{|\mathcal{Y}|}-\frac{1}{|\mathcal{Y}|^2}\right)$ , as claimed.  $\square$ 

**Lemma C.4.** Let  $G: \mathcal{X} \to \mathcal{Y}$  and  $H: \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$  be random oracles implemented by the compressed oracle technique. Define the function  $F: \mathcal{X} \to \mathcal{Z}$  by F(x) = H(x, G(x)). Consider running an interaction of an oracle algorithm with F until query t, then measuring the internal state of G and H to obtain  $D_G$  and  $D_H$ .

Let  $E_t$  be the event that after the measurement at time t, for all  $(x||y,z) \in D_H$ , there exists a entry  $(x,y) \in D_G$ . Then

$$\Pr[E_t] \ge 1 - 4t^2 \left( \frac{2}{|\mathcal{Y}|} - \frac{1}{|\mathcal{Y}|^2} \right)$$

*Proof.* The proof requires changing just a few lines of the proof of Lemma C.1. The first modification is that  $U_H$  acts on all three registers of  $Q = (Q_X, Q_Y, Q_Z)$ , although it only modifies  $Q_Z$ . This change does not affect the proof of Claim C.2.

Second, in the proof of Claim C.3, the projectors are slightly modified.  $E_{Y,\perp}$  now requires  $D_H(x\|y) = \bot$ , instead of  $D_H(y) = \bot$ . Additionally,  $E_{Y1,Z}$  and  $E_{Y+,Z}$  are replaced by a single new projector  $E_{Y,Z}$ , which projects onto states  $|a\rangle\otimes|x,0,u\rangle\otimes|D_G,D_H\rangle$  where (1)  $D_G$  and  $D_H$  satisfy E, (2)  $D_G(x) = y$  for some y, and (3)  $D_H(x\|y) = z$  for some z. The fourth condition of uniqueness is no longer necessary because  $x\|y$  fully specifies x. The analysis of  $\|\overline{E}\cdot \mathsf{Decomp}_G\cdot E_{Y,Z}|\psi'\rangle\|$  is almost exactly the same as that of  $\|\overline{E}\cdot \mathsf{Decomp}_G\cdot E_{Y1,Z}|\psi'\rangle\|$  in the original proof, with the minor syntactic change that  $D_H$  contains a tuple  $(x\|y,z)$  instead of (y,z).

## C.3 Knowledge of Preimage for Expanding Random Oracles

Here we show that if a random oracle G is sufficiently expanding and an adversary making polynomially many queries to G knows elements y that appear in the image of G, then G's compressed database should also contain a corresponding entry. In fact, we will show a more general statement which takes into account an adversary which knows partial preimages as well. In the case where  $\mathcal{X}_1 = \emptyset$  in the following lemma, this corresponds to an adversary simply finding elements from the range.

**Lemma C.5.** Let  $G: \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{Y}$  be a random function where  $|\mathcal{X}_2| < |\mathcal{Y}|$ . Consider an oracle algorithm A makes q of queries to G, then outputs two vector of k values  $\mathbf{x}^{(1)} = \left(x_1^{(1)}, \dots, x_k^{(1)}\right)$  and  $\mathbf{y} = (y_1, \dots, y_k)$ . Let p be the probability that for every i, there exists an  $x_i^{(2)} \in \mathcal{X}$  such that  $G\left(x_i^{(1)}, x_i^{(2)}\right) = y_i$ .

Now consider running the same experiment where G is instead implemented as a compressed oracle, and measuring its database register after A outputs to obtain D. Let p' be the probability that for every i, there exists an  $x_i^{(2)} \in \mathcal{X}_2$  such that  $D\left(x_i^{(1)} \| x_i^{(2)}\right) = y_i$ . If k and q are  $\operatorname{poly}(\lambda)$  and  $|\mathcal{X}_2|^k/|\mathcal{Y}| = \operatorname{negl}(\lambda)$ , then  $t \in \mathbb{N}$ 

$$p \le p' + \mathsf{negl}(\lambda)$$

<sup>&</sup>lt;sup>17</sup>We remark that the reliance on the number of queries is unlikely to be tight. A tighter bound might be achieved by performing a direct computation of the effects of querying G on every  $x \in \mathcal{X}$  at the end of the experiment.

*Proof.* Consider the adversary B which attempts to find k input/output pairs by running A to obtain  $\mathbf{x^{(1)}}$  and  $\mathbf{y}$ , then guessing a uniform  $\mathbf{x^{(2)}} \in \mathcal{X}_2^k$  to construct the vector  $\mathbf{x} = \left(x_1^{(1)} \| x_1^{(2)}, \dots, x_k^{(1)} \| x_k^{(2)} \right)$  and outputting  $(\mathbf{x}, \mathbf{y})$ . Denote  $p_B$  as the probability that it outputs  $(\mathbf{x}, \mathbf{y})$  such that  $G(x_i^{(1)}, x_i^{(2)}) = y_i$  for all i. Since  $\mathbf{x^{(2)}}$  is independent of A, we have  $p_B \geq p/|\mathcal{X}_2|^k$ .

Now consider running B with a compressed oracle, then measuring the compressed oracle to obtain a database D. Note that this is the same distribution over databases as running A. Denote  $p_B'$  as the probability that it successfully outputs  $(\mathbf{x}, \mathbf{y})$  such that  $(x_i^{(1)} || x_i^{(2)}, y_i) \in D$ . We may decompose the corresponding event into two mutually exclusive components:

- Let  $E'_{B,+}$  be the event that  $(x_i^{(1)} || x_i^{(2)}, y_i) \in D$  for all i and D contains a collision  $(x_0^*, y^*)$  and  $(x_1^*, y^*)$ .
- Let  $E'_{B,1}$  be the event that  $(x_i^{(1)}||x_i^{(2)},y_i) \in D$  for all i and D does not contain a collision.

By definition,  $p_B' = \Pr[E_{B,+}'] + \Pr[E_{B,1}']$ . We may define analogous events  $E_{A,+}'$  and  $E_{A,1}'$  when A is run and the database is measured, where the first condition is changed to the existence of  $x_i$ , rather than requiring A to find it.  $\Pr[E_{B,+}']$  and  $\Pr[E_{A,+}']$  are both bounded by the probability of D containing a collision, which Lemma 3.5 bounds by  $O(q^3/|\mathcal{Y}|)$ . Furthermore, since there is a unique "solution" to  $\mathbf{y}$  in the event  $E_{A,1}'$ , we have  $\Pr[E_{B,1}'] = \Pr[E_{A,1}']/|\mathcal{X}_2|^k$ . Thus we can relate  $p_A'$  and  $p_B'$  by  $p_B' \leq O(q^3/|\mathcal{Y}|) + \Pr[E_{A,1}']/|\mathcal{X}_2|^k \leq O(q^3/|\mathcal{Y}|) + p_A'/|\mathcal{X}_2|^k$ . By Lemma 3.4,

$$\sqrt{p_B} \le \sqrt{p_B'} + \sqrt{k/|\mathcal{Y}|}$$

Combining these with our bounds on  $p_B$  and  $p'_B$ , we have

$$\begin{split} \sqrt{p_A/|\mathcal{X}|^k} & \leq \sqrt{O(q^3/|\mathcal{Y}|) + p_A'/|\mathcal{X}|^k} + \sqrt{k/|\mathcal{Y}|} \\ \sqrt{p_A} & \leq \sqrt{O(q^3 \cdot |\mathcal{X}|^k/|\mathcal{Y}|) + p_A'} + \sqrt{k \cdot |\mathcal{X}|^k/|\mathcal{Y}|} \end{split}$$

Since k and q are polynomial in  $\lambda$  and  $|\mathcal{X}|^k/|\mathcal{Y}| = \text{negl}$ , we obtain the desired result by squaring both sides of the inequality.

### **D** Additional Prelims

We give some additional preliminaries in this section.

#### D.1 NIZK

A NIZK for NP scheme should satisfy the following properties:

**Correctness** A NIZK proof (Setup, Delegate, Prove, Verify) is correct if there exists a negligible function  $negl(\cdot)$  such that for all  $\lambda \in \mathbb{N}$ , all  $x \in L$ , and all  $w \in \mathcal{R}_L(x)$  it holds that

$$\Pr[\mathsf{Verify}(\mathsf{CRS},\mathsf{Prove}(\mathsf{CRS},\mathsf{token},\,w,x),x)=1]=1-\mathsf{negl}(\lambda)$$

where (CRS, msk)  $\leftarrow$  Setup( $1^{\lambda}$ ), token  $\leftarrow$  Delegate(msk).

**Computational Soundness** A one-time NIZK proof (Setup, Prove, Verify) is computationally sound if there exist a negligible function  $negl(\cdot)$  such that for all unbounded adversaries  $\mathcal A$  and all  $x \notin L$ , it holds that:

$$\Pr[\mathsf{Verify}(\mathsf{CRS}, \pi \leftarrow \mathcal{A}(\mathsf{CRS}, x, \mathsf{token}), x) = 1] = \mathsf{negl}(\lambda)$$
 where (CRS, msk)  $\leftarrow \mathsf{Setup}(1^{\lambda})$ , token  $\leftarrow \mathsf{Delegate}(\mathsf{msk})$ .

Computational Zero Knowledge A one-time NIZK proof (Setup, Delegate, Prove, Verify) is computationally zero-knowledge if there exists a simulator S such that for all non-uniform QPT adversaries with quantum advice  $\{\rho_{\lambda}\}_{{\lambda}\in\mathbb{N}}$ , all statements  $x\in L$  and all witnesses  $w\in\mathcal{R}_L(x)$ , it holds that  $S(1^{\lambda},x)\approx_c \operatorname{Prove}(\operatorname{CRS},\operatorname{token},w,x)$  where  $\operatorname{CRS}\leftarrow\operatorname{Setup}(1^{\lambda})$ , token  $\leftarrow\operatorname{Delegate}(\operatorname{msk})$ .