On Best-Possible One-Time Programs

Aparna Gupte
MIT

Jiahui Liu
Fujitsu Research

Luowen Qian
Northeastern University

Justin Raizes
NTT Research

Bhaskar Roberts
University of California, Berkeley

Mark Zhandry
Stanford University

Abstract

One-time programs (OTPs) aim to let a user evaluate a program on a single input while revealing nothing else. Classical OTPs require hardware assumptions. Quantum measurement offers a possible physical basis for one-time use: obtaining an output may irreversibly disturb the program needed for further computation. Yet deterministic functionalities remain impossible due to gentle-measurement attacks (Broadbent, Gutoski and Stebila, 2013). While recent works achieve positive results for randomized functionalities with high-entropy outputs (Gunn, Movassagh 2025; Gupte, Liu, Raizes, Roberts, Vaikuntanathan 2025), the fundamental limits and the strongest achievable security notions remain poorly understood.

Inspired by classical obfuscation, we ask for a "best-possible" one-time compiler: a generic transformation that, for any functionality, leaks the least amount of information compared with any other one-time implementation. We show that a generic best-possible one-time compiler cannot exist even for classical randomized functionalities assuming $\mathsf{SZK} \not\subseteq \mathsf{BQP}$.

Given this impossibility, we identify a natural subclass of one-time compilers called "testable one-time program" compilers, which output quantum states augmented with reflection programs for themselves. (1) We formulate a simplified, generalized *Single-Effective-Query (SEQ)* simulation security notion for quantum channels. The definition uses self-adjoint implementations whose behavior under arbitrary quantum interactions depends only on the channel. We show that SEQ security implies best-possible testable one-time security. (2) We construct SEQ-secure OTPs for *all* quantum functionalities in the classical oracle model, yielding the first positive results for arbitrary quantum channels beyond classical randomized functionalities. Thus, SEQ security could serve as a testable one-time analogue of virtual black-box (VBB) security in the many-time obfuscation setting.

Finally, we propose stateful quantum indistinguishability obfuscation (stateful quantum iO) — quantum state obfuscation for stateful quantum programs. It implies best-possible testable OTPs and is achievable in the classical oracle model, thus giving us a promising approach towards best-possible testable OTPs in the plain model.

Contents

1 Introduction **3**
- 1.1 Defining Correctness . . . 6
- 1.2 Impossibility of Best-Possible One-Time Programs . . . 8
- 1.3 Why Unrestricted Best-Possible One-Time Compilers Are Impossible . . . 9
- 1.4 From Impossibility to Testable Programs . . . 10
- 1.5 One-Time Security for Quantum Functionalities . . . 12
- 1.6 SEQ As Best-Possible Testable Security . . . 14
- 1.7 Constructing SEQ-Secure Programs . . . 14
- 1.8 Technical Overview of Positive Results . . . 15
- 1.9 Discussion and Future Directions . . . 18
- 1.10 One-Time Security in Previous Works . . . 19

2 Preliminaries **20**
- 2.1 Quantum Computation . . . 20
- 2.2 Quantum Authentication Schemes . . . 20
- 2.3 Ideal Obfuscation of Unitaries . . . 21

3 Impossibility of Best-Possible One-Time Compilers **22**
- 3.1 Definition of a Best-Possible One-Time Program Compiler . . . 22
- 3.2 Impossibility from Statistical Zero-Knowledge Hardness . . . 24
- 3.3 Impossibility from Public-Coin Perfect Zero-Knowledge Hardness . . . 25
- 3.4 Consequences under Concrete Assumptions . . . 27

4 SEQ Implies Best-Possible Testable One-Time Security **27**

5 Stateful Quantum State Obfuscation: Towards a Plain Model Construction **33**
- 5.1 Stateful Quantum Indistinguishability Obfuscation . . . 33
- 5.2 Stateful Quantum iO Implies Best-Possible Testable One-Time Programs . . . 34
- 5.3 Stateful Oracles in the Classical Oracle Model . . . 35

A Lossy Public-Key Encryption **45**
- A.1 Hard Zero-Knowledge Problems from Lossy Encryption . . . 46
- A.2 Statistically Lossy Encryption from LWE . . . 46
- A.3 Construction from Group Actions . . . 46

B Classical Single Effective Query (CSEQ) **48**
- B.1 Classical Single Effective Query (CSEQ) Model . . . 48
- B.2 When is CSEQ a special case of SEQ . . . 49

C Impossibility of one-time correct sampling QSIO **66**

D One-Time VBB Security When Possible **68**

E SEQ Is Occasionally Strictly More Secure than Testable Security **69**
- E.1 A Separation from Testable Implementations . . . 69
- E.2 Recognizable Outputs in the SEQ Interface . . . 71

1 Introduction

One-time programs (OTPs), introduced by Goldwasser, Kalai, and Rothblum [GKR08], aim to give a user a program that can be evaluated on one input of the user's choice and then becomes useless afterwards. In the classical plain model this is impossible, since the user can simply copy the program before running it. Quantum information offers a possible way around this copying attack: unknown quantum states cannot in general be cloned, and extracting an output may disturb the state that encodes a program. However, Broadbent, Gutoski, and Stebila [BGS13] showed that quantum states do not generically solve the one-time program problem: for deterministic classical functionalities, gentle-measurement techniques still allow many evaluations.

The Cryptographic Perspective. The central cryptographic challenge for OTPs is to identify the most appropriate security goal in the presence of unavoidable attacks.

Ben-David and Sattath [BS23] constructed "quantum one-time signature tokens", which can be viewed as quantum OTPs for the signing functionality. This circumvents the impossibility of [BGS13] because the signing functionality is *randomized*, with high-entropy classical outputs. More recently, two works [GLR+25, GM24] extended this approach to give quantum OTP constructions for general randomized classical functionalities. They proposed security notions and proved guarantees for functionalities with entropic outputs, obtaining the first general positive results for OTPs.

Randomness alone, however, does not make the intuitive ideal of "revealing the output of a single input" achievable. As pointed out in [GLR+25], if some piece of information can be deterministically extracted from the probabilistic output, then a generalization of the attack of [BGS13] extracts that information on multiple inputs. For example, this applies when all possible outputs on a given input have the same parity. Such attacks seem to violate the very intuition behind a one-time program.

The prior works [GLR+25, GM24] do provide one-time security guarantees for their constructions, but these come in several forms: some are tailored to particular applications like signing, while the simulation-based definitions are very involved. In particular, [GLR+25] propose a strong simulation-based notion called *Single-Effective Query* (SEQ) security, which respects a quantum adversary's ability to uncompute its own evaluations. It remains unclear both what these guarantees mean for the one-time security of general programs and whether they are the strongest security one could hope for. Ideally, we would have a guarantee that a program is one-time except for certain classes of unavoidable attacks. We discuss the prior security definitions further in Section 1.10.

This leads to a fundamental question:

For a given family of functionalities,
what is the strongest achievable one-time security notion for quantum OTPs?

For inspiration, consider classical obfuscation, which, like OTPs, seeks to protect the implementation details, but unlike OTPs, permits arbitrarily many evaluations. A similar difficulty arises even with this weaker goal. Virtual black-box (VBB) obfuscation requires that the behavior of any adversary receiving an obfuscated program be indistinguishable from that of a simulator with black-box oracle access to the functionality. This is provably impossible for certain functionalities [BGI+01].

Two approaches have emerged from this impossibility. The first is the weaker notion of *indistinguishability obfuscation* (iO), proposed in [BGI+01], which roughly requires that obfuscations of two equivalent programs be indistinguishable. This notion avoids the impossibility and is potentially achievable (e.g. [GGH+16, JLS26] in the pre-quantum setting, or [BGMZ18, HJL25]). A priori, it is unclear what iO guarantees about a particular program: we usually care about the security of that program, and comparing it with equivalent programs does not directly explain what information is hidden. Indeed, historically we did not know how to construct useful applications of iO until more than a decade after the definition was proposed [SW14]. The other approach retains the VBB goal and shows that it *is* achievable for some very specific functionalities [CD08, LPS04, WZ17, GKW17].

Unfortunately, there is a wide gulf between what is known to be VBB obfuscatable and what is known to be un-obfuscatable even after decades of research. Thus both approaches have major limitations.

Goldwasser and Rothblum [GR07] provide a satisfying way to unify them through the lens of "best-possible" obfuscation. Their starting question is: *what is the least information any obfuscated program can reveal?* A best-possible obfuscator reveals no more than any other program computing the same functionality, except for the unavoidable leakage of an upper bound on the program length. Informally, best-possible security requires that whatever can be efficiently learned from the obfuscated program must also be learnable from every such alternative implementation (of comparable length). In particular, if a functionality can be VBB-obfuscated securely by *any* obfuscator, a best-possible obfuscator must also provide VBB security for it. Goldwasser and Rothblum show that best-possible obfuscation is actually *equivalent* to iO. With the subsequent emergence of iO constructions, this gives best-possible obfuscation for all programs. To establish VBB security for a new functionality, it therefore suffices to show that *some* VBB obfuscator exists: any iO scheme then provides that guarantee.

The Quantum Information Perspective. A fundamental problem at the intersection of quantum information and computation is to understand how destructive measurements limit the amount of information extractable from a quantum state. In the information-theoretic setting, Holevo's theorem [Hol73] gives a foundational limitation: measuring an n-qubit encoding of a classical message can reveal at most n bits of information about that message. This bound on the total information extracted does not rule out giving the recipient a choice of *which* information to extract. Different choices can require incompatible measurements, so extracting one part of the information may destroy access to another.

This possibility motivates the study of quantum random access codes in quantum one-way communication complexity [ANTV02, Nay99]. This study asks how compactly a classical truth table can be encoded into a quantum state while allowing any chosen entry to be recovered with bounded error. Such an encoding serves as a quantum program for evaluating the corresponding function on a single input of the user's choice. For example, two bits can be encoded into one qubit so that either chosen bit can be recovered with probability about 0.85 [ANTV02]. The measurements for the two choices are incompatible: decoding one bit erases the information needed to decode the other. The asymptotic savings are limited, however: encoding an arbitrary N-bit truth table so that any chosen entry can be recovered with constant bounded error still requires $\Omega(N)$ qubits [Nay99].

Quantum complexity theory generalizes quantum random access codes to evaluating arbitrary programs through the perspective of quantum advice [Aar05]. The comparison between e.g. $\mathsf{BQP/poly}$ and $\mathsf{BQP/qpoly}$ asks whether access to a quantum program can enable a quantum computer to perform computations beyond those possible with a classical program. Here, as with random access codes, correctness is required only for a single evaluation, and the pursuit of more efficient encodings naturally leads to examples of one-time behavior. For certain search problems, which allow multiple valid outputs, prior work has shown that using a quantum program can be exponentially more efficient than a classical one [Liu23, ABK24]. Broadbent, Karvonen, and Lord [BKL24] further show that some promise problems admit *uncloneable quantum programs*: their computational usefulness cannot be duplicated by uniform efficient strategies for two separate recipients. Gavinsky [Gav12] makes this one-time behavior explicit: a two-qubit instance of the hidden-matching encoding allows either of two chosen relational queries to be answered with certainty, while any strategy answering both succeeds with probability at most $3/4$ on a random instance. While the efficiency and computational power of these programs have been extensively studied, their one-time behavior remains less systematically understood.

A separate line of work investigates the role of destructive measurement through a counterfactual question: what would change if measurement outcomes could be sampled without disturbing the quantum state? The resulting models of computation with *non-collapsing measurements* have

been studied for their computational power [ABFL16, MP25] and the power of quantum verification [AGI+25, BM25]. For the random-access task above, Aaronson [Aar18] shows that with these measurements, an arbitrary truth table can instead be encoded in a quantum state of polylogarithmic size while allowing efficient recovery of any chosen entry, in contrast to the linear lower bound for ordinary measurements. These models have also been used to formulate cryptographic hardness assumptions [MSY25] and to separate cryptographic primitives [ÇKM+26].

Quantum one-time programs offer a complementary and operational approach to study this question, by bringing the destructive nature of quantum measurement into the spotlight:

How and when can obtaining the result of a chosen computation come at an unavoidable cost to the ability to perform others?

Measurement can make information extraction destructive, but coherent quantum evolution is reversible. The recipient of a quantum program can control how it is evaluated: they may postpone measurements, evaluate on superpositions, or reverse an evaluation instead of following the intended procedure. As mentioned above, techniques like gentle measurement allow repeated evaluation of deterministic classical functionalities, ruling out one-time programs for them [Aar05, BGS13]. For a randomized output with sufficient entropy, retaining the observed sample may instead obstruct this recovery. The positive results for randomized functionalities discussed above show that restrictions on further information extraction can indeed be enforced. However, the positive results are still limited, and there is a wide gap in our understanding of how this phenomenon manifests for other randomized functionalities.

These results leave open how strongly information extraction can be restricted for a general computation. While the recipient (or adversary) is free to choose how to evaluate a given program, the designer (or cryptographer) is free to choose which correct implementation to provide. Information exposed by one quantum program may be hidden by another program for the same functionality. To understand the limits of one-time computation, we therefore need to distinguish information that every correct implementation must reveal from information exposed by a particular implementation. We would like to understand the limits of the strongest one-time programs and find a generic way to achieve this security.

Our Question. Both perspectives therefore motivate the study of the best achievable security for quantum OTPs: programs that reveal no more information than any correct one-time implementation of comparable size for the same functionality. More succinctly,

Is there a best-possible one-time compiler?

Remark 1.1 (Deterministic Functionalities)**.** *The question remains meaningful even for deterministic functionalities, an important special case and the original setting considered by Goldwasser, Kalai, and Rothblum. Here, best-possible one-time security is collapses to best-possible obfuscation. Existing attacks make repeated evaluation unavoidable, so revealing the least amount of information among all one-time correct implementations is equivalent to revealing the least among all* many-time *correct implementations.*

Since our comparison class of possible obfuscations of a program includes quantum programs, the corresponding notion is quantum-state *obfuscation; restricting the comparison to classical programs recovers the classical notion of Goldwasser and Rothblum. Assuming quantum state indistinguishability obfuscation [CG24], best-possible compilers are achievable for deterministic functionalities; existing ideal obfuscation results also give constructions in the classical oracle model [BBV24]. However, this is an unsatisfying answer to the one-time question: achieving best-possible security here means giving up one-time security altogether.*

For the rest of this work, we therefore focus on the more general setting of randomized and even quantum functionalities, where learning an output may irreversibly disturb the program and allow stronger security than best-possible many-time obfuscation.

Organization. Comparing implementations first requires specifying which behavior makes them implementations of the same functionality. We therefore begin in Section 1.1 by clarifying the definition of correctness and addressing a common confusion. Sections 1.2 and 1.3 then state and explain our impossibility result for unrestricted best-possible one-time programs. This motivates the notion of *testable* programs, introduced in Section 1.4 to capture a natural subclass of one-time compilers. We introduce SEQ for quantum channels, show that it is best possible among testable programs, and present our constructions in Sections 1.5 to 1.7, respectively. Section 1.8 sketches the proofs of the positive results, and Section 1.9 discusses their implications and open questions.

1.1 Defining Correctness

In this section, we clarify our choice of correctness definition for obfuscation and OTPs to address a common confusion. Recall the original, informal goal of obfuscation: preserve a functionality's input–output behavior while hiding everything else as much as possible [BGI+01]. Correctness specifies the behavior that must be preserved; security concerns what a recipient should not learn from the implementation. For one-time programs, correctness concerns only the first evaluation. The *ideal* security then should require that anything that is not preserved by the correctness requirement should be hidden.

For Deterministic Functionalities. We first recall the existing definitions for deterministic functionalities. Consider a deterministic classical function $f : \mathcal{X} \to \mathcal{Y}$. A quantum program $P = (\rho, \mathsf{Eval})$ consists of a possibly mixed state ρ and a public evaluation circuit Eval. The recipient can run Eval coherently and in reverse[1]. Directly formalizing the prescribed input–output behavior gives the following definition.

Definition 1.2 (Classical correctness)**.** *A program P is (one-time)* classically correct *for f if evaluation returns $f(x)$ on each classical input x with certainty.*

Another prevalent definition in the literature requires coherent evaluation.

Definition 1.3 (Coherent correctness)**.** *A program P is (one-time)* coherently correct *for f if its evaluator implements the unitary map*

$$U_f : |x\rangle\,|z\rangle \longmapsto |x\rangle\,|z \oplus f(x)\rangle\,.$$

We set aside other subtleties of defining correctness here to focus on the distinction between these two notions.

In the black-box setting, the recipient can only query the supplied interface. Here the two notions are inequivalent: coherent queries can reveal properties of f that cannot be learned efficiently through classical queries. For example, recovering the hidden period in Simon's problem takes polynomially many coherent queries but exponentially many classical queries [Sim94].

Obfuscation instead gives the recipient white-box access to an implementation $P = (\rho, \mathsf{Eval})$. The public description of Eval permits coherent execution of both Eval and Eval^{-1}, even when correctness is promised only on classical inputs. The standard uncomputation argument underlying the impossibility of deterministic OTPs (see, e.g. [BGS13]) gives the following equivalence.

Fact 1.4 (White-box equivalence of correctness notions)**.** *For deterministic functionalities, classical and coherent correctness are equivalent up to a uniform efficient transformation of the public evaluator using black-box calls to* Eval *and* Eval^{-1}.

[1] We take Eval to be unitary without loss of generality.

Thus coherent access is unavoidable for a quantum recipient of a deterministic program, motivating the use of coherent oracle access in models such as the QROM [BDF+11].

For deterministic functionalities, we draw three conclusions:

1. Definitions 1.2 and 1.3 are equivalent in the standard white-box model.
2. Definition 1.2 is *the* definition of correctness that follows from first principles and should be the default.
3. In the black-box model, however, Definition 1.3 more faithfully captures what a quantum recipient can learn from a correct implementation. This unavoidable access is *the only* reason to prefer Definition 1.3 in that setting.[2]

For Sampling Functionalities. Next, we turn to what it means to correctly implement a sampling functionality. For a classical sampling task represented by a randomized circuit $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$, the specified behavior is

$$x \longmapsto f(x; r), \qquad r \xleftarrow{\$} \mathcal{R}.$$

Write D_x for the resulting output distribution. The user chooses one classical input and receives one classical sample; the random coins remain internal to the functionality.

As with Definition 1.2, the following definition arises from first principles.

Definition 1.5 (Sampling correctness)**.** *A program P is (one-time)* sampling correct *for f if evaluation produces exactly the distribution D_x on each classical input x.*

This definition does not require preserving the output's dependence on a particular random tape. Similarly, deterministic correctness does not require preserving the original circuit's internal wire values. For obfuscation, correctness requires preserving the map from inputs to output distributions. The security goal is to hide everything else as much as possible, including the output's dependence on the random tape. We emphasize that Definition 1.5 has long been the default in the classical literature. It dates back to the original definition of sampling obfuscation by Barak et al. [BGI+01]. Later examples include the probabilistic obfuscation notions of Canetti et al. [CLTV15] and Dodis et al. [DHRW16, Definition 4.1]. Classical sampling is also the correctness interface used in prior work on randomized classical OTPs [GLR+25, GM24]. It appears in applications such as signature tokens and one-time NIZKs [BS23, GLR+25].

The uncomputation argument in Fact 1.4, however, breaks down for sampling functionalities. Retaining a random output can correlate it with the program and workspace, preventing correct uncomputation. The argument therefore does not yield a well-defined analogue of Definition 1.3 for sampling functionalities.[3]

One might nevertheless wonder whether there is an unknown generic attack for sampling functionalities, analogous to Fact 1.4. Such an attack might suggest a more "appropriate" analogue of Definition 1.3. It would not, however, change the best-possible comparison in the standard white-box model since the attacker is still permitted to carry out such attacks under Definition 1.5. More precisely, suppose a uniform efficient procedure converts any one-time sampling-correct program into one satisfying a stronger coherent correctness requirement. The conversion must work from

[2] For deterministic functionalities, the white-box equivalence makes coherent correctness a convenient default. The literature can thus leave implicit the distinction between what correctness requires and what a recipient can derive. This convention can obscure the significance of that distinction when moving to sampling functionalities.

[3] Prior work by Gupte et al. [GLR+25] prescribed a particular coherent interface for certain sampling functionalities in its simulation security notion. Extending the QROM is a natural starting point for a construction that makes extensive use of a random oracle. As we show later, their notion is a special case of our generalized SEQ framework. It corresponds to one particular quantum-channel extension of a sampling functionality; other extensions can also be used. This interface is therefore not intrinsic to the classical sampling task.

a single program instance and preserve the sampling functionality. Now consider an implementation that reveals no more than any program satisfying the stronger requirement. A recipient of any one-time sampling-correct program can first convert it into such a program. The implementation therefore reveals no more than the original sampling-correct program either. If the proposed notion is white-box equivalent to Definition 1.5, these conversions work in both directions. The two notions of best-possible one-time security are then equivalent as well. Therefore, for our purposes, using Definition 1.5 is both justified by the intended use of obfuscation and technically sufficient.

Finally, a future OTP application may require a particular coherent extension of a sampling functionality. We believe the cleanest approach is then to specify that extension as a quantum channel. Looking ahead, our positive results construct OTPs for arbitrary quantum channels.

1.2 Impossibility of Best-Possible One-Time Programs

For a classical sampling functionality, the most literal definition of best-possible one-time compilers says the following. A one-time compiler OTP^* is best possible if, for every adversary attacking $\mathsf{OTP}^*(f)$ and for every efficient one-time program P that implements the same sampling functionality f, there is a simulator given P that reproduces the adversary's view. In other words, $\mathsf{OTP}^*(f)$ should leak no more than any possible one-time implementation of f, since if P achieves a certain security guarantee, then $\mathsf{OTP}^*(f)$ should also achieve that guarantee by the existence of the simulator.

Definition 1.6 (Informal; Definition 3.3)**.** *A one-time compiler* OTP^* *is* best-possible *for a family* $\mathcal{F}$ *of sampling functionalities if there exists a simulator* Sim *such that for any sampling functionality* $f \in \mathcal{F}$*, and for any quantum program* P *that* one-time *implements* f*,* $\mathsf{OTP}^*(f, 1^{|P|})$ *is computationally indistinguishable from* $\mathsf{Sim}(P, 1^{|f|})$*.*

The length of the program is also unavoidably leaked similar to indistinguishability obfuscation and best-possible obfuscation [GR07].

This definition is deliberately unrestricted. A competing implementation P may be a mixed quantum state, and it only needs to implement the sampling functionality in the one-query classical sense as in Definition 1.5. We emphasize that this is not an arbitrary technical choice. Our central question of investigating "the least amount of information leaked for any one-time implementation" should include reasonable implementation that correctly samples $\{D_x\}$. Thus the first-principle best-possible definition therefore has to compare against all of them.

Our first result shows that an unrestricted best-possible one-time compiler is impossible. Following the notation of Goldreich, Sahai, and Vadhan [GSV98], let $\mathsf{HVPZK}|_{\mathrm{am}}$ denote the class of promise problems that admit public-coin honest-verifier perfect zero knowledge proof systems, with strict polynomial-time simulation as specified in Section 3.3. Similarly, $\mathsf{HVSZK}|_{\mathrm{am}}$ only requires statistical zero knowledge, and $\mathsf{HVSZK}|_{\mathrm{am}} = \mathsf{SZK}$ [GSV98, Theorem 2], where SZK is the standard class without either the honest-verifier or public-coin restriction.

Theorem 1.7 (Informal; Theorems 3.4 and 3.6)**.** *If* $\mathsf{SZK} \not\subseteq \mathsf{BQP}$ *(respectively,* $\mathsf{HVPZK}|_{\mathrm{am}} \not\subseteq \mathsf{BQP}$*), no best-possible one-time compiler with statistical (respectively, perfect) equivalence exists for all classical sampling functionalities with a single input bit.*

These assumptions are also implied by various post-quantum cryptographic assumptions, giving the following concrete consequences.

Corollary 1.8 (Informal; Theorem 3.7)**.** *Assuming the post-quantum hardness of LWE, best-possible one-time compilers do not exist for classical sampling functionalities under statistical equivalence. Assuming post-quantum weakly pseudorandom effective group actions, best-possible one-time compilers do not exist even under perfect equivalence.*

To establish the second consequence, we construct a perfectly lossy encryption scheme from the same group action assumption, which may be of independent interest (Theorem A.6).

1.3 Why Unrestricted Best-Possible One-Time Compilers Are Impossible

We now explain the intuition behind the impossibility result. Looking ahead, this section will also help motivate the positive results. We refer the reader to Section 3.2 for a formal proof.

At a high level, the key insight relies on identifying two kinds of sampling functionalities that a best-possible compiler would have to treat differently.

1. The first kind is essentially input-independent: the functionality samples from a distribution that does not depend on the input. Such a task has an extremely secure (similar to virtual black-box) one-time implementation: sample once and hardwire the sample into a constant program. Any best-possible compiler for this task must leak no more than this constant implementation. We call this the *lossy* case, since the output carries essentially no information about the input.

2. The second kind is input-dependent: the output uniquely identifies the input except with negligible probability, so intuitively a correct implementation cannot simply return a sample fixed before seeing the input. We call this the *injective* case, where injectivity refers to recovering the input from the output, possibly inefficiently.

We construct these two kinds of functionalities assuming the worst-case hardness of the SZK-complete problem *Statistical Difference* [SV03]. After polarization, a hard instance gives two classical sampling circuits C_0, C_1 whose output distributions D_0, D_1 are either $\exp(-n)$-close or $(1-\exp(-n))$-far. Consider the sampling functionality

$$f(b; r) := C_b(r), \qquad b \in \{0,1\}.$$

In the close case, sampling from D_0 regardless of b implements f up to negligible error, giving the lossy case above. In the far case, the two distributions put almost all their mass on disjoint sets, so the output identifies b, giving the injective case. Thus, if an efficient best-possible compiler lets us distinguish these two kinds of functionalities, we obtain an efficient quantum algorithm for Statistical Difference, contradicting $\mathsf{SZK} \not\subseteq \mathsf{BQP}$. Figure 1 summarizes this contradiction.

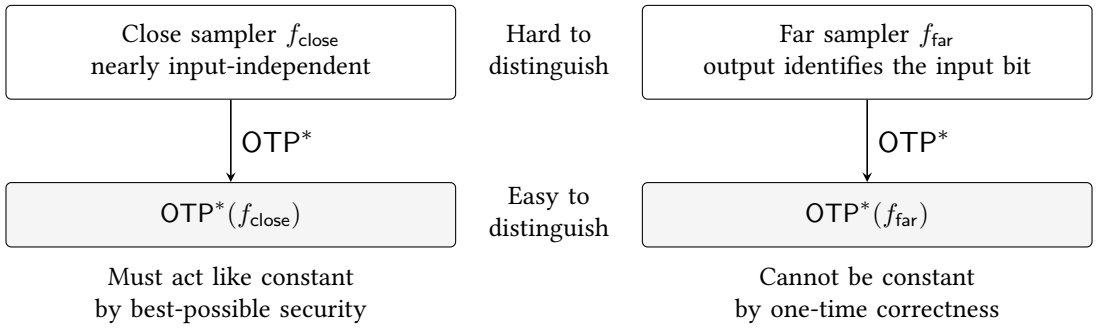


Figure 1: Distinguishing the close and far cases from sampler descriptions is hard under $\mathsf{SZK} \not\subseteq \mathsf{BQP}$. The same best-possible compiler OTP^* followed by the EPR test would distinguish these cases, implying that OTP^* cannot be efficient.

Remark 1.9 (One-time VBB when possible)**.** *Under the same assumptions, the same proof idea rules out an even weaker security notion. We call this notion "one-time VBB when possible": if the family of functionalities admits a one-time VBB compiler then we also require one-time VBB security; otherwise, there is no security required. We formalize this notion in Appendix D.*

Proof idea in slightly more detail. Suppose we have a best-possible one-time compiler OTP^* and are given a polarized Statistical Difference instance (C_0, C_1). We compile $f(b; r) = C_b(r)$ and test whether evaluating the resulting program reveals the input bit to the output. To do so, prepare an EPR pair $(|00\rangle + |11\rangle)/\sqrt{2}$ on the input and an untouched reference register, evaluate the program, copy its output to a fresh register, and uncompute the evaluation. Finally, test whether the input and reference have returned to their initial EPR state. The test uses the public evaluation circuit and its inverse, so it is efficient even though the program state may be unknown.

In the injective case, correctness forces the copied output to identify the original input bit in the reference register, except with negligible probability. This effectively measures the reference bit, and uncomputation cannot undo its correlation with the copied output because it touches neither register. Consequently, the EPR test fails with probability at least $1/2 - \mathsf{negl}(\lambda)$.

In the lossy case, correctness alone does not force $\mathsf{OTP}^*(f)$ to return a fixed sample. The essential observation is that f has a one-time implementation that is a mixture of constant programs: sample $y \leftarrow D_0$ once, store y, and return it on either input. For each fixed y, let $c_y(b) = y$. Correctness forces $\mathsf{OTP}^*(c_y)$ to output y, up to negligible error, so copying its output causes negligible disturbance and uncomputation restores the EPR pair. Thus the test passes with overwhelming probability on every compiled constant program.

To relate this behavior back to the behavior of $\mathsf{OTP}^*(f)$, we invoke best-possible security. Let P_y be the program storing y, and let P be its mixture for $y \leftarrow D_0$, with a common evaluator that copies the stored value. Security supplies one simulator for P and every P_y, and linearity gives

$$\mathsf{OTP}^*(f) \approx_c \mathsf{Sim}(P) = \{\mathsf{Sim}(P_y)\}_{y \leftarrow D_0} \approx_c \{\mathsf{OTP}^*(c_y)\}_{y \leftarrow D_0}.$$

Since the EPR test is efficient, it must pass for $\mathsf{OTP}^*(f)$ with overwhelming probability as well. Repeating the test with freshly compiled programs therefore distinguishes the close and far cases of Statistical Difference.

For perfect equivalence, we only need the two distributions in the lossy case to be exactly equal, so that the mixture of constant programs implements f perfectly. Malka's reduction [Mal15, Lemma 4.4 of the full version] shows that the identical-versus-far variant of Statistical Difference is $\mathsf{HVPZK}|_{\mathrm{am}}$-hard under randomized polynomial-time reductions. The same EPR argument therefore gives the perfect-equivalence impossibility. We formalize the perfect variant in Section 3.3.

Remark 1.10. *We note that our impossibility can be dequantized to state that best-possible one-time programs are also impossible with classical obfuscators assuming the existence of the same (but classically secure) lossy encryptions used for the concrete consequences in Appendix A. The only step that needs to be changed is that rather than using a quantum distinguisher that tests if the input superposition collapses, we can simply rewind the classical program to test if it is a constant function.*

This exact step is also the place where the impossibility breaks down when hardware assumptions or memory coherence assumptions are introduced, and these are the two settings where prior works have successfully constructed strong one-time programs.

1.4 From Impossibility to Testable Programs

We established that unrestricted best-possible one-time compilers are impossible to hope for. The natural next step is to ask whether there is a weaker but still meaningful notion of best-possible one-time compilers that can be achieved. One possible direction is to restrict the class of functionalities to avoid the examples used by the impossibility result. In this rest of this work, we instead restrict the comparison class of one-time compilers.

A valid takeaway from the impossibility result is that mixed one-time implementations are very secure since they can sometimes even achieve the ideal one-time functionality, but on the other hand, it is also impossible to efficiently identify when mixed-output techniques can be applied given only

the circuit description as input. Thus, a natural idea to get around the impossibility is to restrict the comparison class to compilers that output pure-state programs only.

Remark 1.11. *Restricting to pure-state programs may be very tempting because this is closer in spirit to how best-possible obfuscation is traditionally defined [GR07]: one compares against fixed obfuscated programs rather than arbitrary mixtures. However, notice that for perfectly correct obfuscation of deterministic functionalities, the distinction between fixed programs vs mixture of programs collapses: if a mixed state (many-time) implements a deterministic functionality with zero error, then every pure state in its support implements the same functionality, so quantifying over mixed states only makes the definition conceptually more involved. Turning back to one-time programs of randomized functionalities, the distinction need not collapse: a mixed implementation can be correct only in aggregate while its pure components are not individually correct.*

With that said, the mixed state definition is still the most reasonable definition for best-possible one-time compilers since that is the most literal interpretation of the ideal goal of comparing against all one-time correct implementations.

However, simply restricting the comparison class to pure-state implementations does not evade the impossibility. To see the issue, let $C(r)$ be a classical circuit sampling the input-independent distribution in the impossibility, using ℓ uniformly random bits r. For any $s \in \{0,1\}^\ell$, the pure state

$$|\psi_s\rangle = 2^{-\ell/2} \sum_{r\in\{0,1\}^\ell} (-1)^{s\cdot r} |r, C(r)\rangle \tag{1}$$

together with the evaluation procedure that copies the $C(r)$ register to the output register is already a pure-state implementation of this sampling task. Each state has a polynomial-size exact preparation circuit: prepare the signed uniform superposition over r and compute $C(r)$ reversibly, retaining the random tape. Thus the constant-sampler side of the impossibility can be represented using efficiently preparable pure program states.

Indeed, averaging uniformly over s cancels every off-diagonal term between distinct random tapes, giving exactly the mixed state that samples r and stores $|r, C(r)\rangle$. By linearity, any simulator's average behavior on the pure implementations $|\psi_s\rangle$ is the same as its behavior on this mixed implementation. But this mixed implementation is also the average of the basis-state programs $|r, C(r)\rangle$, each of which is a pure implementation of the constant function with value $C(r)$. Consequently, the same averaging argument used in the impossibility still forces the EPR test to pass on the compiled functionality in the lossy case, while correctness forces it to fail with constant probability in the injective case. The test would therefore still distinguish the two cases of Statistical Difference. Bare purity therefore does not give a meaningful escape from the unrestricted impossibility.

There is also a representation issue: if a compiler outputs a mixed state, there is no canonical decomposition of that mixed state into pure states. A definition that asks whether the "pure components" are correct therefore depends on a representation that is not part of the physical object.

These two issues motivate the notion of *testability*. Informally, a testable quantum program consists of a pure program state $|\psi\rangle$ (which can be interpreted as the input to a universal quantum circuit), and a reflection program for $|\psi\rangle$, i.e. it implements the unitary $I - 2|\psi\rangle\langle\psi|$.

Definition 1.12 (Informal; Definition 4.1)**.** *A quantum program is* testable *if its program state is a pure state $|\psi\rangle$ accompanied by a reflection program for $|\psi\rangle$. A one-time compiler is testable if its outputs are testable programs and every $|\psi\rangle$ in the support one-time implements the input functionality.*

The reflection program in the testability definition gives a physical and well-defined way to pin down the pure program state output by the compiler. Crucially, testability is not just purity: changing the hidden phases in (1) also changes the reflection interface. Thus the random-phase averaging trick is no longer merely a different decomposition of the same program object. This is

the extra structure that breaks the impossibility argument and lets the simulator coherently recognize whether the program has returned to its designated initial state.

We highlight that testability turns out to be not artificial from the perspective of known constructions of one-time compilers. There are concrete and recurring ways to modify existing one-time compiler constructions to become testable. For starters, a trivial case is the classical-state case: if each issued program is individually correct and $|\tilde{P}\rangle$ is always a computational-basis string, then one can first read out that string (equivalently, measure in the computational basis) and then efficiently implement the exact reflection $I - 2|\tilde{P}\rangle\langle\tilde{P}|$ coherently via an equality check and phase kickback. The most common pattern for designing a one-time compiler is to issue an uncloneable token that collapses upon use, along with an obfuscated classical program that only works when the two-basis measurement outcome on the token is correct. For subspace and coset states, coherent membership tests in complementary bases suffice to implement reflection about the designated token [BS23, GLR+25]. In particular, both the plain-model coset-state construction of [GLR+25] and the construction of [GM24] use a distinguished abort output, making these acceptance tests accessible through the evaluator. Thus, these existing constructions admit testable variants preserving their established correctness and security guarantees.

We further conjecture that the program implicitly having a reflection oracle might be an unavoidable property for many natural function classes, since outputs such as signatures and one-time NIZKs admit efficient verification. The SEQ interface becomes testable as soon as we know an efficiently preparable state that is orthogonal to every channel output. This structure can be generically added by extending the query register with a success bit whose incoming value is ignored and whose output value is always set to 1. We refer the readers to Appendix E.2 for more details.

1.5 One-Time Security for Quantum Functionalities

Before we investigate best-possible one-time compilers in the restricted testable setting, we first turn to extending the study of one-time security from classical sampling functionalities to all quantum (channel) functionalities. As far as we are aware, our work is the first to study positive results for constructing one-time programs for arbitrary quantum channels without hardware assumptions.

The work of Gupte et al. [GLR+25] proposed a simulation-based definition called Single Effective Query (SEQ) security for one-time programs for classical sampling functionalities, and showed that it is achievable in the classical oracle model. The core idea is to require the adversary's view to be simulatable with only an ideal query interface for the protected functionality Ψ. This interface permits computations and uncomputations that have not extracted information, but it tries its best to detect when the output is meaningfully disturbed and subsequently refuse to cooperate with a subsequent query. This is in contrast to older simulation-based one-time security notions such as in [GKR08, BGS13] where the interface allows literally one physical query, regardless what has been learned in the query.

In this section, we recast this idea in a simpler form that also applies to arbitrary quantum channels. For the rest of this work, we call our generalized notion simply SEQ and refer to the original version from [GLR+25] as classical SEQ (CSEQ) to distinguish.

Without hardware assumptions, an adversary could always compute and uncompute the unitary implementation of the interface. This motivates the following definition of a "self-adjoint implementation" of a quantum channel.

Definition 1.13 (Informal; Definition 4.2)**.** *Consider a quantum channel Ψ using input/output register $\mathcal{X}$ which is implemented by some unitary U_Ψ which might use a private auxiliary register $\mathcal{A}$ that is initialized to 0. Let $\mathcal{C}$ be a qubit register acting as a counter. Let the* self-adjoint implementation $S(U_\Psi)$ *of Ψ be the unitary that swaps*

$$|x\rangle_{\mathcal{X}}|0\rangle_{\mathcal{A}}\otimes|0\rangle_{\mathcal{C}} \leftrightarrow (U_\Psi|x\rangle_{\mathcal{X}}|0\rangle_{\mathcal{A}})\otimes|1\rangle_{\mathcal{C}}$$

and acts as identity everywhere else.

Fact 1.14. *$S(U_\Psi)$ is a well-defined self-adjoint unitary and can be efficiently implemented given controlled query access to U_Ψ.*

Figure 2 gives this implementation and separates the public query register from the persistent private registers. Its middle gate checks whether uncomputation has restored the auxiliary register before allowing another computation.

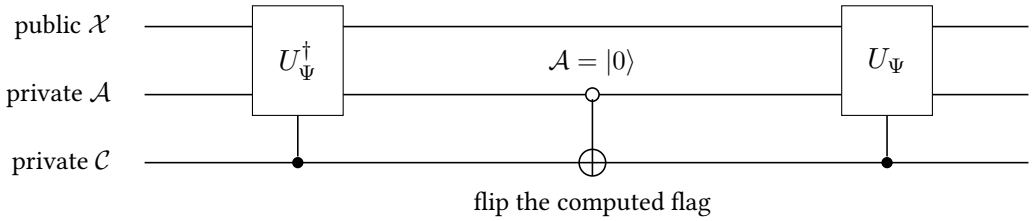


Figure 2: The self-adjoint SEQ interface $S(U_\Psi)$. The caller controls only $\mathcal{X}$; $\mathcal{A}$ and $\mathcal{C}$ persist privately between calls. The middle gate flips $\mathcal{C}$ exactly when the inverse computation has restored $\mathcal{A}$ to zero.

Definition 1.15 (Informal; Definition 4.3)**.** *A one-time compiler is SEQ-secure if $\mathsf{OTP}(P)$ can be efficiently (computationally indistinguishably) simulated by a simulator that only gets access to P via querying the self-adjoint implementation for P without having access to its $\mathcal{A}, \mathcal{C}$ registers.*

It is so called "Single Effective Query" because (1) to compute Ψ a second time, you must first return to the state $|x\rangle_\mathcal{X} |0\rangle_\mathcal{A} \otimes |0\rangle_\mathcal{C}$ by uncomputing it; but more importantly, (2) if the output is "meaningfully" disturbed, then further evaluations become effectively impossible because the evaluator cannot return to the initial state $|x\rangle_\mathcal{X} |0\rangle_\mathcal{A} \otimes |0\rangle_\mathcal{C}$.

A helpful example is a channel that ignores the input and outputs n uniformly random bits. An untouched output can be uncomputed, but measuring all n bits breaks the coherence with the hidden purification. For large n, the next query then acts very close to identity.[4]

Theorem 1.16 (Informal; Theorem B.19)**.** *For every non-trivial randomized classical functionality $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ satisfying the nontriviality condition that $|\mathcal{X}| > 1$, there is a quantum channel Φ_f such that the classical query interface O_f^{CSEQ} and the generalized query interface $O_{\Phi_f}^{\mathsf{SEQ}}$ are efficiently inter-simulatable. Consequently, classical CSEQ security is a special case of generalized SEQ security.*

Thus our generalized SEQ definition should be read as extending the prior CSEQ framework rather than replacing it with an incomparable notion.

As the prior work [GLR+25] has already shown, SEQ as a simulation-based security is unachievable for certain randomized circuits in the plain model by generalizing the impossibility of VBB [BGI+01]. Despite this VBB-style impossibility, we believe that SEQ on its own is still a very useful notion for the topic of one-time programs. Of course, the VBB-style impossibility does not rule out SEQ for *all* interesting functionalities; but more importantly, similar to the role VBB plays today, SEQ provides a relatively simple yet *realistic ceiling* on the one-time security that one could hope for in the plain model for a given functionality using *testable* OTP obfuscators.

[4] In the canonical purification, the computed state contains $2^{-n/2} \sum_r |r\rangle_{\text{out}} |r\rangle_{\text{private}}$. After measuring the output, its squared projection onto the computed subspace is 2^{-n}, so the next SEQ query changes the state by trace distance $O(2^{-n/2})$. This also matches the intuition that the higher the entropy of the measured output, the more destructive the effect on the quantum program.

1.6 SEQ As Best-Possible Testable Security

Next, we study best-possible security relative to the class of testable one-time compilers. Similarly, we define a one-time compiler to be *best possible among testable programs* for a class of quantum channels if whatever can be learned from its output can be simulated given any testable one-time implementation of the same channel. Here the compiler being constructed need not itself be testable.

Our main positive theorem is that this simulation notion is best possible for the testable comparison class.

Theorem 1.17 (Informal; Theorem 4.5)**.** *Any one-time compiler satisfying* SEQ *security for a class of quantum channels is best possible among testable one-time programs for that class.*

Recovering Prior Security Guarantees. Together, our two simulation theorems give SEQ a unifying role among the known approaches to one-time security. First, the CSEQ guarantee of [GLR+25] is recovered directly as a special case of SEQ by Theorem 1.16. The other one-time constructions [BS23, GLR+25, GM24] admit testable implementations through their existing acceptance checks. Thus a one-time compiler satisfying SEQ security would also straightforwardly achieve the security guarantees from these works. Taking a different perspective, our theorem also gives a way to establish new applications for SEQ security: it suffices to exhibit a testable implementation that provides the desired information-hiding guarantee, and any SEQ-secure compiler for the same channel inherits that guarantee.

For testable quantum one-time programs, SEQ is therefore a ceiling on achievable privacy, including that of future constructions. This is analogous to how VBB provides a ceiling to security achievable by iO in the many-time obfuscation setting. Complementarily, if one aims to construct a one-time compiler that achieves security unachievable by even SEQ, the OTP construction *must actively* prevent its output program from being testable.

1.7 Constructing SEQ-Secure Programs

We next show that the SEQ benchmark is achievable in idealized models, generalizing [GLR+25] from sampling functionalities to arbitrary non-oracular quantum functionalities.

Since we have already established that SEQ is sufficient for best-possible testable security, a natural high level strategy to construct one-time compilers with this security is to simply (many-time) obfuscate the SEQ interface and let the SEQ interface handle one-time security. Since the SEQ interface maintains hidden registers across queries, the main tool our construction uses is a new notion called stateful quantum obfuscation. We formalize a notion of stateful quantum indistinguishability obfuscation (iO). Two stateful quantum programs are equivalent if they induce the same behavior over every sequence of evaluations. Stateful quantum iO asks that equivalent stateful programs have indistinguishable obfuscations.

Theorem 1.18 (Informal; Theorem 5.4)**.** *Assuming stateful quantum indistinguishability obfuscation, there exists a one-time compiler for all quantum channels that is best possible among testable programs*[5]*.*

We also give evidence that this is a reasonable idealized route by constructing stateful quantum obfuscation in an oracle model.

[5] As usual for positive results on oracle-model obfuscation, the oracle is an implementation aid for the compiler and evaluator. Thus, when we say "all quantum channels" in the oracle model, we mean all non-oracular quantum channels; allowing the protected channel itself to query the same oracle would reintroduce the usual VBB-style impossibilities for obfuscating oracle-aided functionalities [BGI+01, GLR+25]. For comparison, our impossibility results for non-oracular sampling functionalities hold relative to any unitary oracle O, with BQP replaced by BQP^O in the hardness assumptions. The oracle used here is efficiently simulatable, so the new assumptions are equivalent to the non-oracular assumptions.

Security notion	Impossibilities	Constructions
Single physical query, simulation-based, OTP for quantum functionalities [BGS13]	Strong impossibility for most functionalities relative to all oracles	For single physical-query learnable (trivial) functions only [BGS13]; For constant-distribution functionalities (implicit in this work)
Classical SEQ, simulation-based, OTP for classical functionalities [GLR+25]	VBB-style impossibility for contrived functionalities in the plain model [GLR+25]	For all classical functions, relative to a classical oracle [GLR+25]
Generalized SEQ, simulation-based, OTP for quantum functionalities (this work)		For all quantum functionalities relative to a classical oracle (this work)
Best-possible OTP compiler (this work)	Impossibility for generic compilers under our zero-knowledge hardness assumptions (this work)	
Best-possible OTP compiler among only testable programs (this work)		From either SEQ security or quantum stateful iO (this work)

Figure 3: One-time program security notions with impossibilities and constructions.

Theorem 1.19 (Informal; Theorem 5.8 and corollary 5.10)**.** *Ideal stateful quantum obfuscation exists in the classical oracle model. Consequently, there exist* SEQ*-secure one-time programs for all non-oracular quantum channels in the classical oracle model.*

This gives, to our knowledge, the first positive results for one-time programs for arbitrary non-oracular quantum channels without hardware assumptions, albeit in the classical oracle model. It also suggests an approach for a plain-model instantiation of best-possible testable one-time programs by constructing stateful quantum obfuscation in the plain model.

1.8 Technical Overview of Positive Results

We now give a high-level overview of the main arguments for the positive results in the last two sections.

Best-Possible Testable One-Time Programs. We first sketch that a one-time program achieving SEQ security is as secure as *any* testable one-time program.

To show that SEQ security is best-possible for testable one-time programs, we must show how to simulate $\mathsf{SEQ}(f)$ using an alternative quantum program g which is equivalent to f for one query and has a reflection oracle R. The idea of the simulator is straightforward: we simply implement $\mathsf{SEQ}(f)$ by implementing $\mathsf{SEQ}(g)$ instead. The intuition is that the SEQ interface would hide the implementation details of whether f or g is being queried since they are one-time equivalent.

To formalize our intuition above, we establish the following lemma.

Lemma 1.20 (Informal; Lemma 4.10)**.** *For any two unitary implementations U_Ψ, U'_Ψ of the same channel Ψ, $S(U_\Psi)$ and $S(U'_\Psi)$ are perfectly indistinguishable given unbounded query access to only the input/output register $\mathcal{X}$.*

Thus this lemma establishes that the self-adjoint implementation is a generic compiler to make quantum channel evaluation reversible without any dependence on the implementation of the channel itself.

This already suffices to prove the statement if f and g are classically described programs. However, there is one subtlety we need to address, which is that g may be a quantum state describing a program $|g\rangle$. After evaluating it once, the program state may be disturbed. Unlike classically, we cannot say that the program state is readonly due to no cloning.

This is where the reflection oracle R plays a crucial role: the reflection oracle R well defines the quantum program $|g\rangle$. In particular, we modify the middle gate in Figure 2 to also control on the quantum program register being $|g\rangle$, which can be efficiently implemented using the reflection oracle. The intuitive reason that this works is that this way, we are effectively implementing the following quantum program that does *not* involve a quantum auxiliary input $|g\rangle$ and yet is perfectly equivalent to g:

1. Perform a swap $|0\rangle \leftrightarrow |g\rangle$ in the program register.
2. Carry out the original computation.

Observe that the SEQ for this program would reflect around all zeroes on both the auxiliary register and the program register, which is equivalent to reflecting around all zeroes on the auxiliary register and $|g\rangle$ on the program register if we do not do the first step.

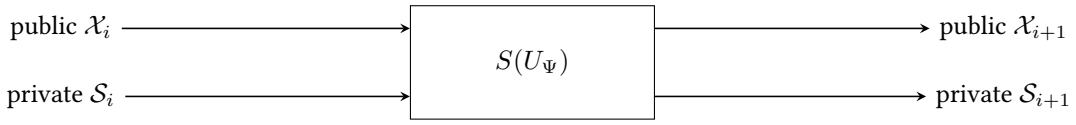
Achieving SEQ Security with Oracles. Note that our Lemma 1.20 gives a straightforward way for instantiating SEQ-secure one-time programs with *stateful unitary* ideal obfuscation. In such an obfuscation scheme, the obfuscator takes as input a unitary U that acts on a public register $\mathcal{X}$ and an internal auxiliary register $\mathcal{A}$ that is initialized to some state that is given; and it outputs $\tilde{U}$ which can only be queried in either the forward or backward direction where the internal register is inaccessible. Then to one-time protect f, we can simply use the ideal obfuscation to obfuscate and give out $\mathsf{SEQ}(f)$.

Intuitively, such a stateful unitary ideal obfuscation scheme can be implemented in a (stateless) unitary oracle model, or by ideal unitary obfuscation. The idea is that we can use a quantum authentication scheme to protect the internal state so that we can delegate the internal state management to the adversary without compromising security. As shown in Figure 4, the recipient holds the authenticated memory, while an obfuscated update circuit holds the authentication key. On valid memory, the circuit coherently decodes, applies the SEQ update, and re-authenticates. On invalid memory, it undoes the authentication check and acts as identity; both branches remain coherent so that the entire update is unitary. Furthermore, such a scheme can be ported to the classical oracle model using the compiler of [HT25]. While this compiler introduces quantum states into the program, this does not matter for our application since our obfuscated programs are allowed to have quantum states.

Towards the Plain Model via Stateful Quantum Obfuscation. Even though SEQ security is achievable using ideal obfuscation or classical oracles, we already know that there are randomized classical functionalities that cannot be SEQ-securely one-time protected in the plain model [GLR+25]. A natural question then is whether we can nevertheless achieve best-possible testable one-time programs in the plain model. Perhaps this is achievable through an alternative notion of obfuscation which plausibly exists in the plain model, similar to how for many-time security, iO and best-possible obfuscation are equivalent.

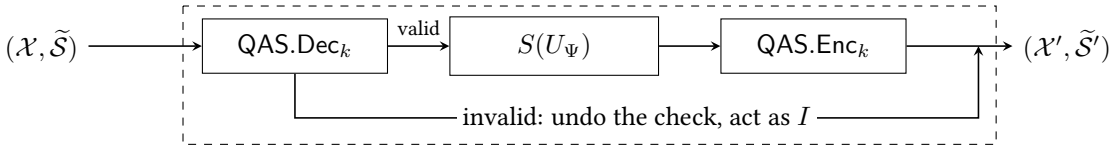
Previous works propose natural definitions for indistinguishability obfuscation of quantum circuits that compute either pseudodeterministic functions [BBV24, CG24] or unitaries [HT25] — for every two quantum programs that compute the same pseudodeterministic function (or unitary), their obfuscations should be indistinguishable. A natural attempt at generalizing this to arbitrary quantum *channels* would be to ask that if two quantum programs output similar mixed states, their obfuscations should look indistinguishable. However, if two programs produce similar outputs for the first evaluation, but then degrade so that subsequent evaluations are distinguishable, it should be hard to

(a) The ideal interface keeps its memory between calls



$\mathcal{S} = (\mathcal{A}, \mathcal{C})$ is the persistent private memory

(b) In the real world, the recipient holds authenticated memory



obfuscate the coherent update unitary with k hardwired

Figure 4: Implementing the stateful SEQ interface. The compiler authenticates the initial memory and gives out an obfuscation of the update unitary. Authentication protects the memory held by the recipient, while ideal unitary obfuscation protects the key. Both branches of the authentication check are coherent. Ideal stateful obfuscation yields SEQ security; stateful iO suffices for best-possible security among testable programs.

obfuscate them into programs that are indistinguishable. In fact, our previously discussed impossibility result formalizes this intuition.

This is a uniquely quantum issue for two reasons. First, quantum programs with auxiliary quantum states inherently maintain state over evaluations that involve non-gentle measurements. Second, quantum circuits have the ability to self-produce randomness, and there is no general way to "de-randomize" them, like in the classical case. Classical descriptions of circuits do not degrade with evaluations, and it is always possible to de-randomize a probabilistic circuit $C(x; r)$ and obfuscate $C(x; F_k(x))$, so that the obfuscations of two equivalent classical probabilistic circuits C_1, C_2 will always produce the same output across evaluations. To avoid the impossibility, we explicitly consider obfuscating programs whose state may change over time.

Definition 1.21 (Stateful iO, Informal; Definition 5.3)**.** *Stateful indistinguishability obfuscation allows two stateful quantum programs to be indistinguishably obfuscated if and only if they produce the same output distributions over* any *sequence of evaluations.*

To show that stateful iO evades our impossibility result, we show that it is implied by ideal unitary obfuscation. Intuitively, a unitary program can authenticate a private register to itself in order to maintain state across multiple queries. Using ideal obfuscation, the authentication key is protected, so the register remains private from the evaluator. Similarly as before, we can further port this to the classical oracle model using the compiler of [HT25].

Furthermore, unlike ideal obfuscation, there does not seem to be an inherent barrier to constructing stateful iO in the plain model. Because the output distributions are required to be close even for query sequences that can depend on the programs being obfuscated, stateful iO avoids the self-referential techniques used to rule out ideal obfuscation [BGI+01].

Finally, we show that stateful iO implies best-possible testable one-time programs. Thus, if one could construct stateful iO in the plain model, they would also construct best-possible testable one-time programs in the plain model. This again uses Lemma 1.20, which when combined with the security guarantee of stateful iO, shows that whatever could be learned from the stateful obfuscation of $\mathsf{SEQ}(f)$ could also be learned from g by statefully obfuscating $\mathsf{SEQ}(g)$.

1.9 Discussion and Future Directions

Prevalence of Our Best-Possible OTP Impossibility. Beyond the deterministic case, our best-possible impossibility identifies two classes of randomized functionalities that no single best-possible compiler can handle simultaneously. A different idea for getting around the impossibility might be to restrict the functionality class to exclude one of these classes (this is orthogonal to restricting the comparison class of one-time compilers as we've done for testable one-time compilers). Inspecting our impossibility further, the lossy class is probably unlikely to be of interest for one-time programs, where the randomized program samples the same distribution for every input.

While this could be a valid way of getting around the impossibility, we argue that this direction seems to require restricting the functionality class to be very small. For starters, consider the randomized functionality $f(x;r) \to (y_1, y_2)$ where y_1 is sampled from a fixed distribution independent of x. Then, this is a functionality that is not lossy yet (a suitably adapted version of) our impossibility still applies. One could even further consider more involved variants of this where the lossy structure is even less apparent, such as applying a pseudorandom permutation to the output. Given these examples, we suspect that it is unlikely to identify a meaningful and large class of genuinely randomized functionalities where best-possible OTPs are possible, since any class that is closed under augmentation of such structures is also susceptible to our impossibility.

Implications for Defining Obfuscation of Quantum Programs. Previous work propose natural definitions for indistinguishability obfuscation for quantum circuits that compute pseudodeterministic classical functions [BBV24, CG24] and unitaries [HT25] — for every two quantum programs that compute the same pseudodeterministic function (or unitary), their obfuscations should be indistinguishable. A natural attempt at generalizing this to quantum *channels* would be to ask that if two quantum programs output (approximately) the same mixed state, their obfuscations should look indistinguishable. Note that this natural attempt only asks the two programs to produce samples from same mixed state for the *first* evaluation. This is because for (approximately) classical functions or more generally unitaries, one-time equivalence is equivalent to many-time equivalence by gentle measurement. However, when the functionality is not unitary and the program is described by a program state, the program state itself may evolve after the first query. Crucially, we can rule out this one-time security notion by invoking our best-possible OTP impossibility, since intuitively if such an object exists, then it would give a best-possible OTP similar to how iO is equivalent to best-possible obfuscation. We formalize this in Appendix C.

Our definition of stateful iO gets around this problem by considering *many-time* equivalence of two programs rather than *one-time* equivalence. Thus, when defining (many-time) obfuscation for quantum sampling programs or general quantum channels, a notion which takes into account behavior on sequential queries, like stateful iO, seems necessary.

Future Directions. Our results leave several directions open. The most direct is to find plain-model one-time compilers with best-possible testable security, perhaps through a candidate for stateful quantum obfuscation. On the (many-time) obfuscation front, does unitary indistinguishability obfuscation imply stateful iO? The main difficulty in lifting our proof from ideal obfuscation is to adapt the argument that the obfuscator protects the authentication key. Furthermore, does classical indistinguishability obfuscation imply unitary iO?

Another direction is to explore security beyond those achievable by testable compilers, in other words, to explore *untestable* one-time program constructions. Our impossibility already constructs a one-time VBB obfuscator for lossy-mode encryptors or constant-distribution samplers. It is not hard to see that for these functionalities, testable implementations would yield strictly worse security. The SEQ interface can also offer more privacy than testable implementations for certain functionalities (Theorem E.1), but extending the query register with a bit that is overwritten with a fixed success

value generically makes the interface testable (Appendix E.2).

To further motivate one-time program security beyond testable compilers, we present a generic attack against any testable one-time program f. The attack takes a polynomial-size list of input–observable pairs $(x_i, O_i)_{i=1}^N$, where each input state is efficiently preparable and each observable is efficiently measurable with $\|O_i\|_\infty \leq 1$. It estimates $\mathsf{Tr}(O_i f(x_i))$ for every i up to inverse polynomial error in expected polynomial time. (For example, we can estimate how often specified output bits are 0 on selected inputs.) While this attack is not always useful (say for forging multiple signatures against a signature token), it allows a user to estimate output statistics on many chosen inputs from a single testable program.

This attack follows from the single-copy tomography result of Farhi et al. [FGH+10, Theorem 2]. For each input–observable pair, evaluation followed by the binary output measurement with effect $(I + O_i)/2$ defines an efficiently implementable POVM on the program state. Their algorithm estimates its acceptance probability while restoring the program state using the reflection oracle, allowing the same copy to be reused for all pairs.

1.10 One-Time Security in Previous Works

For completeness, we survey one-time security notions studied in prior works in this section.

- The work by Gupte et al. [GLR+25] proposes a notion called "single effective query (SEQ) simulation-based one-time security" that circumvents the impossibility results (without hardware assumptions) of the simulation-based one-time security defined in preceding works [GKR08, BJ15]. In this work, we revise and generalize their SEQ definition to all functionalities and, to distinguish, we refer to their original (classical) version as CSEQ.

  More concretely, the CSEQ one-time security notion requires that the adversary's view after maliciously utilizing the one-time program of f can be simulated by only querying a restricted stateful query interface to f. Such an interface attempts to record prior queries made to f and only answers the query if no prior queries are recorded.

  They show that this notion is achievable for *all* functionalities in the classical oracle model, which is both good and also unsatisfactory. It is good because it is a single notion that captures all functionalities. It is however unsatisfactory because it delegates the problem of identifying what level of security the definition guarantees to the problem of what can be learned through the CSEQ interface. For example, the CSEQ interface would allow an unlimited number of evaluations for a deterministic functionality thus keeping consistency with the impossibility result [BGS13]. However, for a general randomized functionality, it is not clear what can be learned through the CSEQ interface since the interface is somewhat complicated. Our SEQ notion somewhat mitigates this issue since our SEQ security is much simpler.

- Apart from CSEQ security, [GLR+25] presents several security notions in the plain model, but these notions are restricted to specific settings or target specific applications. These notions can be viewed as extensions of the security requirement for one-time signature tokens in [BS23] to more functionalities. [BS23] states that it is impossible to produce two valid signatures for two distinct messages with respect to the signature verification algorithm. This security can be generalized to any unforgeable functionalities such as one-time NIZKs [GLR+25]. Similarly, the security of one-time PRFs (where part of the PRF input is sampled randomly) [GLR+25] only states that it is impossible to simultaneously distinguish the outputs on two different inputs from random.

- The work by Gunn and Movassagh [GM24] showed that for their construction, a somewhat natural class of attackers (including the attack in [BGS13]) cannot even produce a second output as long as the functionality samples a high min entropy output on every input. However,

it is possible that a more malicious attacker can still produce multiple outputs: an example of this can be seen from the plain-model impossibility in [GLR+25, Theorem 7.7].

These prior results are covered by the two simulation arguments in Sections 1.5 and 1.6. CSEQ is recovered as a special case of SEQ, while the other constructions discussed here admit testable implementations using their acceptance checks. Best-possible testable security then guarantees that an SEQ-secure program reveals no more information than those implementations of the same channel, in their respective models.

2 Preliminaries

2.1 Quantum Computation

We fix canonical description formats for all objects. For a classical randomized function f, we write $|f|$ for the bit-length of its canonical description (excluding any oracle access it might carry). For a quantum sampling program $P = (\rho, C)$, we write $|P|$ for the length of a canonical description consisting of a description of the state ρ together with the description of the (possibly oracle-aided) circuit C; oracle access is not counted toward length. For a stateful quantum program $(U, |\psi\rangle)$, we similarly write $|(U, |\psi\rangle)|$ for the length of a canonical description of U together with a description of $|\psi\rangle$.

When comparing programs by length (e.g., in indistinguishability obfuscation), we allow *padding* to equalize lengths without changing behavior: padding may add ancilla qubits initialized to $|0\rangle$ and insert no-op gates that leave the program's input–output behavior unchanged.

When we say a result relativizes to any unitary oracle, we mean that it holds in the model where all parties receive query access to a family of unitaries including inverse and control as is standard [Zha25].

We will need a few facts about the trace distance of two pure states whose amplitudes are defined by classical probability distributions.

Definition 2.1 (Hellinger Distance)**.** *For two probability density functions f, g over a finite domain $\mathcal{X}$, the squared Hellinger distance between f and g is defined as*

$$H^2(f,g) = \frac{1}{2}\sum_{x\in\mathcal{X}}\left(\sqrt{f(x)} - \sqrt{g(x)}\right)^2 = 1 - \sum_{x\in\mathcal{X}}\sqrt{f(x)g(x)}.$$

Lemma 2.2. *Let D_1, D_2 be two probability density functions over a finite domain $\mathcal{X}$, then*

$$H^2(D_1, D_2) \le \mathrm{TV}(D_1, D_2) \le \sqrt{2}\, H(D_1, D_2).$$

Lemma 2.3. *Let $\mathcal{X}$ be a finite set and let D_1, D_2 be probability densities on $\mathcal{X}$. Let*

$$|\psi_1\rangle = \sum_{x\in\mathcal{X}}\sqrt{D_1(x)}\,|x\rangle \quad \text{and} \quad |\psi_2\rangle = \sum_{x\in\mathcal{X}}\sqrt{D_2(x)}\,|x\rangle \ .$$

Then, the trace distance between the pure states $|\psi_1\rangle, |\psi_2\rangle$ is

$$T(|\psi_1\rangle\langle\psi_1|, |\psi_2\rangle\langle\psi_2|) = \sqrt{1 - (1 - H^2(D_1, D_2))^2}.$$

2.2 Quantum Authentication Schemes

Definition 2.4 (Quantum Authentication Scheme)**.** *A **quantum authentication scheme** is a tuple of QPT algorithms* (KeyGen, Enc, Dec, Ver) *with the following behavior.*

- $k \leftarrow \mathsf{KeyGen}(1^\lambda, n)$ *takes as input the security parameter* λ *and a length* $n \in \mathbb{N}$*, then outputs a key* k*. Here,* n *is the length of the state to be authenticated.*
- $|\psi'\rangle \leftarrow \mathsf{Enc}_k(|\psi\rangle)$ *takes as input a key* k *and an* n*-qubit state* $|\psi\rangle$*, then outputs another state* $|\psi'\rangle$*.*
- $|\psi''\rangle / \bot \leftarrow \mathsf{Dec}_k(|\psi'\rangle)$ *takes as input a key* k *and a state* $|\psi'\rangle$*, then outputs another state* $|\psi''\rangle$ *or* $\bot$*.*
- $\mathsf{Acc}/\mathsf{Rej} \leftarrow \mathsf{Ver}_k(|\psi'\rangle)$ *takes as input a key* k *and a state* $|\psi'\rangle$*, then outputs* Acc *(accept) or* Rej *(reject).* Ver_k*'s behavior is identical to* Dec_k*'s, except* Ver_k *outputs* Acc *whenever* Dec_k *would output* $|\psi\rangle$ *and outputs* Rej *whenever* Dec_k *would output* $\bot$*.*

A quantum authentication scheme must satisfy correctness and security properties.

- ***Correctness.*** *For every* k *in the support of* KeyGen *and every state* $|\psi\rangle$*,*

$$\mathsf{Dec}_k(\mathsf{Enc}_k(|\psi\rangle)) = |\psi\rangle$$

- ***Security.*** *For every QPT adversary* $\mathcal{A}$*, there exists an* $\varepsilon(\lambda) \in [0,1]$ *such that for every* $|\psi\rangle$*,*

$$\left\{ \mathsf{Dec}_k(\rho) \;\middle|\; \begin{array}{c} k \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \rho \leftarrow \mathcal{A}(\mathsf{Enc}_k(|\psi\rangle)) \end{array} \right\} \approx (1-\varepsilon)\{|\psi\rangle\} + \varepsilon\{\bot\}$$

If this holds given oracle access to Ver_k *(or if there is a public key* vk *that allows implementing* Ver_k *and it holds given access to* vk*, we say that the scheme is publicly verifiable.*

[BBV24] shows how to construct publicly verifiable quantum authentication using coset states.

2.3 Ideal Obfuscation of Unitaries

In this section, we will deal with quantum programs $(|\psi\rangle, Q)$ that approximately implement a unitary transform $\rho \mapsto U\rho U^\dagger$.

Definition 2.5. *A quantum program* $(|\psi\rangle, Q)$ *is an* ε*-approximation of a unitary transformation* U *if*

$$\|Q(\cdot \otimes |\psi\rangle) - U(\cdot)\|_\diamond \leq \varepsilon.$$

Definition 2.6 (Quantum State Obfuscation for Unitaries)**.** *A* quantum state ideal obfuscation *for the class of approximately-unitary quantum programs in the classical oracle model is a pair of QPT algorithms* $(\mathsf{QObf}, \mathsf{QEval})$ *with the following syntax:*

- $\mathsf{QObf}(1^\lambda, (|\psi\rangle, Q)) \to (|\widetilde{\psi}\rangle, \mathsf{F})$*: The obfuscator takes as input the security parameter* 1^λ *and a quantum program* $(|\psi\rangle, Q)$ *in the plain model*[6]*, and outputs an obfuscated program specified by a state* $|\widetilde{\psi}\rangle$ *and a classical function* F*.*
- $\mathsf{QEval}^{\mathsf{F}}(\rho_{\mathsf{in}}, |\widetilde{\psi}\rangle) \to \rho_{\mathsf{out}}$*: The evaluation algorithm executes the obfuscated program on quantum input* ρ_{in} *by making use of the state* $|\widetilde{\psi}\rangle$ *and making superposition queries to the classical oracle* F*, and produces the quantum output* ρ_{out}*.*

These algorithms have to satisfy the following properties.

[6] To circumvent impossibility results of [BGI+01], we say that Q does not make use oracles.

- ***Functionality-Preserving:*** *For every quantum program $(|\psi\rangle, Q)$ which is a negligible approximation of a unitary transformation $\mathcal{U}$, there exist negligible functions μ, ν such that*

$$\Pr_{(|\widetilde{\psi}\rangle,\mathsf{F})\leftarrow\mathsf{QObf}(1^\lambda,(|\psi\rangle,Q))}\left[\left\|\mathsf{QEval}^{\mathsf{F}}(\cdot\otimes|\widetilde{\psi}\rangle)-\mathcal{U}(\cdot)\right\|_\diamond\leq\mu(\lambda)\right]\geq 1-\nu(\lambda).$$

Thus, with overwhelming probability over compilation, the sampled program is correct uniformly over all inputs, including inputs entangled with a reference register.

- ***Ideal Obfuscation:*** *There exists a QPT simulator* Sim *such that for every QPT adversary $\mathcal{A}$ and quantum program $(|\psi\rangle, Q)$ that* $\mathsf{negl}(\lambda)$*-approximates some unitary U, and every QPT distinguisher D,*

$$\Big|\Pr\left[1\leftarrow D(\mathcal{A}(\mathsf{QObf}(1^\lambda,(|\psi\rangle,Q))))\right] - \Pr\left[1\leftarrow D(\mathsf{Sim}^{U,U^\dagger}(1^\lambda,\mathcal{A},n(\lambda),m(\lambda)))\right]\Big| = \mathsf{negl}(\lambda).$$

Here, n is the input length of the quantum program, m is the size of the quantum program.

Theorem 2.7 (Adapted from Theorem 7.2 of [HT25])**.** *There exists a quantum state ideal obfuscation for the class of approximately unitary programs with quantum inputs and outputs in the classical oracle model.*[7]

Strengthening correctness. Security follows from the cited theorem. We verify the per-output correctness required here for its construction. Write $\mathcal{Q}_\psi = Q(\cdot\otimes|\psi\rangle)$ and suppose $\|\mathcal{Q}_\psi - \mathcal{U}\|_\diamond \leq \delta$, where δ is negligible. Fix a sampled obfuscation ω, and let $\mathcal{E}_\omega$ be its evaluation channel. In the construction of [HT25], the initial teleportation outcome i is uniform independently of the input, and its signature s is generated using an independent signing token. The oracle uses labels $H_\omega(j,b,i,s)$ for measurement index j and outcome bit b. Let t be the number of measurements and set

$$\beta_\omega = \Pr_{i,s}\left[\exists j<t:\ H_\omega(j,0,i,s)=H_\omega(j,1,i,s)\right].$$

For every fixed (i,s) outside this event, all labels decode unambiguously. Authentication correctness is an exact identity of measurement instruments for every authentication key, and token verification accepts honest signatures perfectly. Applying this identity successively through the evaluator therefore gives exactly $\mathcal{Q}_\psi$, including on entangled inputs. Since the distribution of (i,s) is input-independent, for some channel $\mathcal{B}_\omega$ we have

$$\mathcal{E}_\omega = (1-\beta_\omega)\mathcal{Q}_\psi + \beta_\omega\mathcal{B}_\omega, \qquad \|\mathcal{E}_\omega - \mathcal{U}\|_\diamond \leq \delta + 2\beta_\omega.$$

If H is a truly random function with κ-bit outputs, then $\mathbb{E}_\omega\beta_\omega \leq (t-1)2^{-\kappa}$, which is negligible for $\kappa \geq \lambda + \lceil\log_2 t\rceil$. For a quantum-secure pseudorandom function, the same bound holds up to a negligible term: sampling (i,s) and checking these $t-1$ equalities is an efficient test. Thus $\nu := \mathbb{E}_\omega\beta_\omega + 2^{-\lambda}$ is negligible. Markov's inequality gives $\beta_\omega \leq \sqrt{\nu}$ with probability at least $1-\sqrt{\nu}$, proving the required per-output diamond-norm bound $\delta + 2\sqrt{\nu}$. □

3 Impossibility of Best-Possible One-Time Compilers

3.1 Definition of a Best-Possible One-Time Program Compiler

Randomized Functions. A randomized function $f : \mathcal{X}\times\mathcal{R}\to\mathcal{Y}$ takes an input $x\in\mathcal{X}$, samples randomness $r \xleftarrow{\$} \mathcal{R}$ and outputs $y = f(x;r)\in\mathcal{Y}$. Let $f(x)$ be the distribution over y-values that

[7] The oracle itself is efficient outside of the part of using a truly random function, which can be heuristically instantiated via quantum-secure PRFs.

results from this procedure. By considering f as a circuit description of itself, f is computable in time $\mathsf{poly}(|f|)$, where $|f|$ is its description length.

Let $\mathcal{F}$ be a set of randomized functions, and for each $\lambda \in \mathbb{N}$, let $\mathcal{F}_\lambda$ be the set of all functions in $\mathcal{F}$ with description length λ.

Sampling Programs. Given a randomized function $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$, a sampling program P that implements f is a quantum program $P = (\rho, \mathsf{Eval})$ comprising a (possibly mixed) quantum state ρ and a (possibly oracle-aided) quantum unitary circuit Eval. We write $|P|$ for its description length.

P can be evaluated on any $x \in \mathcal{X}$ by computing

$$\mathsf{Eval}(|x\rangle\langle x|_{\mathcal{X}} \otimes |0\rangle\langle 0|_{\mathcal{Y}} \otimes \rho_{\mathcal{P}})$$

and measuring the $\mathcal{Y}$ register to obtain output y. Let $P(x)$ refer to the distribution over y-values that results from this evaluation procedure.

Definition 3.1 (Correctness of Sampling Programs)**.** *Let $\mu(\lambda)$ be a negligible function. Given a randomized function $f \in \mathcal{F}_\lambda$ and a sampling program P, we say that P **implements** f **with error** μ if for every $x \in \mathcal{X}$, the statistical distance between $P(x)$ and $f(x)$ is $\leq \mu(\lambda)$.*

Our notion of correctness is really a notion of one-time correctness. It says that the first time the program is evaluated, it will sample from approximately the desired distribution. However, measuring the output of the program may destroy the program state ρ, so there is no guarantee that the second evaluation of the program will be correct.

One-Time Program Compilers. A one-time program compiler OTP takes a randomized function f and outputs a sampling program P that implements f. The compiler is best-possible if $\mathsf{OTP}(f)$ can be simulated by any program implementing f for which $|f| = |P|$.

Definition 3.2 (One-Time Program Compiler)**.** *Let $\mathcal{F}$ be a family of randomized functions. A **one-time program compiler for** $\mathcal{F}$ is a QPT machine OTP that takes as input the description of a function $f \in \mathcal{F}$ and outputs a sampling program $P = \mathsf{OTP}(f)$. For correctness, we require that there is a negligible function $\mu(\lambda)$ such that for any $\lambda \in \mathbb{N}$ and any $f \in \mathcal{F}_\lambda$, $\mathsf{OTP}(f)$ implements f with error $\mu(\lambda)$.*

The following definition says that a one-time program compiler is best-possible if $\mathsf{OTP}(f)$ can be simulated given any program P that implements f with low error. We consider two notions of equivalence: perfect equivalence requires P to implement f with 0 error, and statistical equivalence allows P to implement f with a non-zero, but still negligible, error.

Definition 3.3 (Best-Possible One-Time Program Compiler)**.** *Let $\mathcal{F}$ be a family of randomized functions, and let OTP^* be a one-time program compiler for $\mathcal{F}$.*

*OTP^* is a **best-possible one-time program compiler for** $\mathcal{F}$ with statistical/perfect equivalence if for every QPT adversary $\mathcal{A}$ and any function $\mu(\lambda)$ that is (respectively) negligible/identically zero, there is a QPT simulator Sim such that for every QPT distinguisher D, there is a negligible function $\nu(\lambda)$ such that for any $\lambda \in \mathbb{N}$, any $f \in \mathcal{F}_\lambda$, and any sampling program P that implements f with error $\mu(\lambda)$ and satisfies $|P| = |f|$,*

$$\left|\Pr\left[1 \leftarrow D(1^\lambda, f, \mathcal{A}(1^\lambda, \mathsf{OTP}^*(f)))\right] - \Pr\left[1 \leftarrow D(1^\lambda, f, \mathsf{Sim}(1^\lambda, P))\right]\right| \leq \nu(\lambda).$$

The distinguisher receives f as auxiliary input, while the simulator receives only P. The negligible bound may depend on D, but is uniform over the admissible choices of f and P.

In the study of best-possible obfuscation, [GR07] requires perfect equivalence. In other words, the simulator only needs to work correctly when P implements f with 0 error. Statistical equivalence

allows μ to be a non-zero, but negligible, function. Requiring statistical equivalence yields a stronger notion of security.

We rule out these two notions under the hardness of statistical zero knowledge and public-coin honest-verifier perfect zero knowledge, respectively (Sections 3.2 and 3.3). The resulting impossibilities also follow from the post-quantum hardness of LWE and weak pseudorandomness of group actions, respectively (Theorem 3.7).

3.2 Impossibility from Statistical Zero-Knowledge Hardness

We construct the two kinds of sampling functionalities from Section 1.3 from Statistical Difference: in the lossy case, the output distribution is nearly independent of the input; in the injective case, the output almost determines the input. A best-possible compiler would let an efficient quantum test distinguish the two cases, giving a quantum polynomial-time algorithm for this SZK-complete problem. We use the promise-problem versions of these complexity classes, so a BQP algorithm need only be correct on instances satisfying the promise.

Theorem 3.4. *If there exists a best-possible one-time program compiler with statistical equivalence for all classical randomized functions with a single input bit, then* $\mathsf{SZK} \subseteq \mathsf{BQP}$.

Proof. Suppose such a compiler OTP^* exists. By the completeness of Statistical Difference and the polarization lemma of Sahai and Vadhan [SV03, Theorem 3.1 and Lemma 3.3], an instance of length n can be transformed in polynomial time into two classical sampling circuits C_0, C_1 whose output distributions D_0, D_1 satisfy one of the promises

$$\Delta(D_0, D_1) \le \varepsilon \qquad \text{or} \qquad \Delta(D_0, D_1) \ge 1 - \varepsilon, \qquad \varepsilon = 2^{-n},$$

where Δ denotes statistical distance. It suffices to distinguish these two cases in quantum polynomial time. Define the sampling functionality $f(b; r) = C_b(r)$ for $b \in \{0, 1\}$, padding the randomness inputs if necessary. For each output string y, also consider the constant functionality $c_y(b) = y$. Pad these descriptions and the comparison programs below to a common length $\lambda = q(n)$ for a fixed polynomial q, chosen large enough for all of them. Thus ε is negligible as a function of λ, and the equal-length requirement of Definition 3.3 is satisfied.

The adversary. Given a sampling program $P = (\rho, \mathsf{Eval})$, the adversary $\mathcal{A}$ prepares the two-qubit EPR state

$$|\Phi\rangle_{\mathcal{X}\mathcal{X}'} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

and initializes output registers $\mathcal{Y}, \mathcal{Y}'$ to zero. It applies Eval to the input $\mathcal{X}$, output $\mathcal{Y}$, and program register $\mathcal{P}$, copies $\mathcal{Y}$ to $\mathcal{Y}'$ using CNOT gates, and applies $\mathsf{Eval}^\dagger$ to $\mathcal{X}\mathcal{Y}\mathcal{P}$. Finally, it projects $\mathcal{X}\mathcal{X}'$ onto $|\Phi\rangle$, outputting 0 if the test passes and 1 otherwise. We call $\mathcal{X}'$ the reference bit and it is untouched throughout, and uncomputation also leaves the copied output $\mathcal{Y}'$ untouched. All these operations are efficient because the evaluation circuit is public.

Constant functions. First consider $\mathcal{A}$ on $\mathsf{OTP}^*(c_y)$. By correctness, evaluation on either classical input produces y except with negligible probability, uniformly over y. The same holds on the EPR input, since the two input branches have orthogonal reference states. Projecting the output register onto $|y\rangle$ therefore changes the state by negligible trace distance. On this projected state, copying the output only adds the fixed value y to $\mathcal{Y}'$. Uncomputation consequently restores the initial EPR state up to negligible error, giving

$$\Pr[\mathcal{A}(1^\lambda, \mathsf{OTP}^*(c_y)) \to 1] \le \mathsf{negl}(\lambda) \tag{2}$$

uniformly over y.

Close distributions: the lossy case. Let $P_y = (|y\rangle\langle y|, \mathsf{Eval}_{\mathsf{copy}})$, where the common evaluator ignores the input bit and copies the program register into the output register. Each P_y perfectly implements c_y, while

$$P = \left(\sum_y D_0(y) \, |y\rangle\langle y| \, , \mathsf{Eval}_{\mathsf{copy}} \right)$$

implements f with error at most ε. Fix a negligible error bound covering ε and use the same simulator Sim from Definition 3.3 for P and every P_y. By (2) and best-possible security, applied to the distinguisher that reads the output bit, we obtain $\Pr[\mathsf{Sim}(1^\lambda, P_y) \to 1] = \mathsf{negl}(\lambda)$ uniformly over y. By linearity in the program state,

$$\Pr[\mathsf{Sim}(1^\lambda, P) \to 1] = \sum_y D_0(y) \Pr[\mathsf{Sim}(1^\lambda, P_y) \to 1] = \mathsf{negl}(\lambda).$$

Applying best-possible security once more, now to f and P, shows that $\Pr[\mathcal{A}(1^\lambda, \mathsf{OTP}^*(f)) \to 1] = \mathsf{negl}(\lambda)$.

Far distributions: the injective case. There are disjoint sets $\mathcal{Y}_0, \mathcal{Y}_1$ such that $D_b(\mathcal{Y}_b) \geq 1 - \varepsilon$ for each $b \in \{0,1\}$. These sets are only used in the analysis; the adversary does not need to compute them. By correctness, after evaluation and copying the output, the reference bit b and copied output y satisfy $y \in \mathcal{Y}_b$ except with negligible probability. Projecting onto this event changes the state by negligible trace distance. Uncomputation leaves the reference and copied output untouched, so the projected state remains supported on

$$\Pi_{\mathsf{sep}} = \sum_{b \in \{0,1\}} \sum_{y \in \mathcal{Y}_b} \mathbb{I}_{\mathcal{X}} \otimes |b\rangle\langle b|_{\mathcal{X}'} \otimes |y\rangle\langle y|_{\mathcal{Y}'} \, .$$

In the EPR projector, the cross terms between $b = 0$ and $b = 1$ vanish when multiplied by Π_{sep} on both sides, because $\mathcal{Y}_0$ and $\mathcal{Y}_1$ are disjoint. The resulting operator is $1/2$ times a projector, so

$$\left\| \Pi_{\mathsf{sep}} \left(|\Phi\rangle\langle\Phi|_{\mathcal{X}\mathcal{X}'} \otimes \mathbb{I}_{\mathcal{Y}'} \right) \Pi_{\mathsf{sep}} \right\| \leq \frac{1}{2}.$$

Thus the EPR test accepts with probability at most $1/2 + \mathsf{negl}(\lambda)$, and $\Pr[\mathcal{A}(1^\lambda, \mathsf{OTP}^*(f)) \to 1] \geq 1/2 - \mathsf{negl}(\lambda)$.

Repeating the experiment a constant number of times with freshly compiled programs distinguishes the close and far cases with bounded error for all sufficiently large n. The finitely many smaller instances can be hardwired, giving a BQP algorithm for Statistical Difference and hence $\mathsf{SZK} \subseteq \mathsf{BQP}$. □

This proof uses statistical equivalence because P need only approximate f in the close case. Section 3.3 obtains an impossibility with perfect equivalence from public-coin perfect zero-knowledge hardness.

3.3 Impossibility from Public-Coin Perfect Zero-Knowledge Hardness

Following the notation of Goldreich, Sahai, and Vadhan [GSV98], let $\mathsf{HVPZK}|_{\mathrm{am}}$ denote the class of promise problems having public-coin honest-verifier perfect zero-knowledge proofs. Here the verifier and simulator run in strict polynomial time, and on YES instances the simulator's output is identically distributed to the honest verifier's view. Equivalently, one may allow the simulator to abort with probability at most $1/2$ and require exact simulation conditioned on not aborting: Malka's transformation [Mal15, Lemma 5.2 of the full version] removes such aborts and preserves public coins for the honest verifier. Perfect completeness is not required.

Lemma 3.5. *Let $\mathsf{SD}_{0,1/2}$ be the promise problem whose YES instances are pairs of classical sampling circuits with identical output distributions, and whose NO instances have output distributions at statistical distance at least $1/2$. Then $\mathsf{SD}_{0,1/2}$ is $\mathsf{HVPZK}|_{\mathrm{am}}$-hard under bounded-error randomized polynomial-time many-one reductions.*

Proof. For any promise problem $\Pi = (\Pi_Y, \Pi_N)$ in $\mathsf{HVPZK}|_{\mathrm{am}}$, Malka's reduction [Mal15, Lemma 4.4 of the full version] maps an instance x to classical sampling circuits X, Y, Z satisfying

$$x \in \Pi_Y \implies \Delta(X,Y) = 0 \quad \text{and} \quad \Pr[Z=1] \geq \frac{2}{3},$$
$$x \in \Pi_N \implies \Delta(X,Y) \geq \frac{1}{4} \quad \text{or} \quad \Pr[Z=1] \leq \frac{1}{3}.$$

Taking a constant number of independent copies of X and Y preserves equality and increases distance at least $1/4$ to at least $1/2$ [SV03, Lemma 3.4]. Estimate $\Pr[Z=1]$ to additive error less than $1/6$ with probability at least $9/10$. If the estimate is below $1/2$, output a fixed pair of circuits with disjoint output distributions; otherwise, output the amplified pair. On a YES instance, the reduction outputs an identical pair with probability at least $9/10$. On a NO instance, either both possible outputs are far pairs, or $\Pr[Z=1] \leq 1/3$ and the fixed far pair is output with probability at least $9/10$. Thus the reduction outputs an instance on the correct side of the promise with probability at least $9/10$. □

Theorem 3.6. *If there exists a best-possible one-time program compiler with perfect equivalence for all classical randomized functions with a single input bit, then $\mathsf{HVPZK}|_{\mathrm{am}} \subseteq \mathsf{BQP}$.*

Proof. Suppose such a compiler OTP^* exists. By Lemma 3.5 and closure of BQP under bounded-error randomized polynomial-time reductions, it suffices to give a BQP algorithm for $\mathsf{SD}_{0,1/2}$. Given an instance (X,Y) of length n, take $t = O(n)$ independent copies to obtain sampling circuits C_0, C_1 with output distributions D_0, D_1. Equality is preserved exactly, while the NO case gives $\Delta(D_0, D_1) \geq 1 - 2^{-n}$ by the direct-product bound [SV03, Lemma 3.4].

Define $f(b;r) = C_b(r)$, padding the randomness inputs if necessary, and $c_y(b) = y$ for each output string y. As in Theorem 3.4, pad these descriptions and the comparison programs to a common length $\lambda = q(n) \geq n$ for a fixed polynomial q. Use the same padding of the state registers and the same evaluator for every comparison program. Let $\mathcal{A}$ be the EPR adversary used in that proof.

When $D_0 = D_1$, the program

$$P = \left(\sum_y D_0(y) \left|y\right\rangle \left\langle y\right| , \mathsf{Eval}_{\mathsf{copy}} \right)$$

implements f exactly. Its state has a polynomial-size exact preparation circuit: sample a uniform random tape, evaluate C_0, and discard the tape. Each $P_y = (\left|y\right\rangle \left\langle y\right| , \mathsf{Eval}_{\mathsf{copy}})$ likewise has an exact preparation circuit and perfectly implements c_y. The preparation descriptions can be padded to the same length; the simulator receives the prepared state and the common evaluator. Apply Definition 3.3 with error bound 0, using one simulator Sim for P and every P_y and the distinguisher that reads its output bit. Linearity in the program state and the uniform negligible bound ν give

$$\begin{aligned}\Pr[\mathcal{A}(1^\lambda, \mathsf{OTP}^*(f)) \to 1] &\leq \sum_y D_0(y) \Pr[\mathcal{A}(1^\lambda, \mathsf{OTP}^*(c_y)) \to 1] + 2\nu(\lambda) \\ &= \mathsf{negl}(\lambda),\end{aligned}$$

where the last equality uses the constant-function bound (2). When $\Delta(D_0, D_1) \geq 1 - 2^{-n}$, the far-distribution analysis in Theorem 3.4 instead gives

$$\Pr[\mathcal{A}(1^\lambda, \mathsf{OTP}^*(f)) \to 1] \geq \frac{1}{2} - \mathsf{negl}(\lambda).$$

Thus a constant number of repetitions with freshly compiled programs decides $\mathsf{SD}_{0,1/2}$ with bounded error for all sufficiently large n. Hardwiring the finitely many smaller instances gives $\mathsf{SD}_{0,1/2} \in \mathsf{BQP}$ and hence the claimed containment. □

3.4 Consequences under Concrete Assumptions

The zero-knowledge hardness assumptions above follow from familiar cryptographic assumptions. We record the resulting consequences here and give the constructions and reductions in Appendix A.

Theorem 3.7. *Assuming the post-quantum hardness of LWE for the parameter choices given in Theorem A.3, there exists a family of randomized functions $\mathcal{F}$ for which there is no best-possible one-time program compiler with statistical equivalence (Definition 3.3).*

Assuming the post-quantum weak pseudorandomness of group actions (Assumption A.5), there exists a family of randomized functions $\mathcal{F}$ for which there is no best-possible one-time program compiler with perfect equivalence (Definition 3.3).

Proof. By Lemma A.2, statistically/perfectly lossy encryption implies the corresponding zero-knowledge hardness assumption in Theorems 3.4 and 3.6. The result follows from the statistically lossy construction under LWE (Theorem A.3) and the perfectly lossy construction from group actions (Theorem A.6). □

The same arguments also rule out sampling quantum state iO and one-time VBB security when possible; see Appendices C and D.

4 SEQ Implies Best-Possible Testable One-Time Security

Definition 4.1 (Testable quantum program)**.** *A quantum program $P = (|\psi\rangle, \mathsf{Eval})$, specified by a pure state $|\psi\rangle$ and a public unitary evaluation circuit* Eval*, is* testable *if it is accompanied by white-box access to a quantum program $\mathcal{R}_\psi = (|\eta\rangle, V)$ implementing the reflection unitary*[8]

$$R = \mathsf{X} \otimes |\psi\rangle\langle\psi| + I \otimes (I - |\psi\rangle\langle\psi|)$$

Here the recipient receives the auxiliary state $|\eta\rangle$ and the classical description of the unitary circuit V. With this auxiliary state, tracing out V's auxiliary output must realize the channel $R(\cdot)R^\dagger$ on every flag–program input, including inputs entangled with a reference. The auxiliary state may change during this evaluation. We include the reflection program in the size of the testable program and abbreviate the resulting object as $P' = (|\psi\rangle, \mathsf{Eval}, R)$.

A testable one-time program compiler is a one-time program compiler that outputs testable quantum programs.

In a relativized model the public circuits may use the shared oracles. We also use the term *testable interface* for an ideal oracle interface that provides the same coherent test of its designated initial hidden state; the black-box simulation arguments below apply to such interfaces as well.

Next we revise and generalize the SEQ definition of [GLR+25] to all quantum functionalities and adopt the name SEQ for this generalized notion. To distinguish from the prior work, we refer to their original, classical version as CSEQ. Intuitively, the SEQ oracle embeds a single effective application of the purified channel into a globally unitary, repeatable interface.

[8] The usual phase reflection $R' = I - 2|\psi\rangle\langle\psi|$ is equivalent under coherent control. Compute–copy–uncompute makes R reusable and controllable while restoring its auxiliary registers, which we suppress below.

Stinespring form and notation. Let $\Phi : \mathsf{L}(\mathcal{H}_Q) \to \mathsf{L}(\mathcal{H}_Q)$ be a quantum channel. Without loss of generality (by padding with dummy qubits if needed), assume the Stinespring dilation has matching input/output dimensions,[9] so we fix a Stinespring unitary $U_\Phi : \mathcal{H}_Q \otimes \mathcal{H}_P \to \mathcal{H}_Q \otimes \mathcal{H}_P$ with the private register P initialized to $|0^m\rangle_P$ as expected by the purification. Tracing out the private register yields Φ on the query register. We represent Φ by a circuit for such a dilation and write $|\Phi|$ for the length of this description. This length is public, and descriptions may be padded as in the preliminaries.

Definition 4.2 (The Single Effective Query Oracle (SEQ))**.** *For a channel Φ with fixed purification U_Φ as above, the single effective query oracle O_Φ^{SEQ} acts on a query register $\mathcal{Q}$ and maintains a private register $\mathcal{P}$ and a one-qubit computed flag $\mathcal{C}$. The registers are initialized to $|0^m\rangle_\mathcal{P} \otimes |0\rangle_\mathcal{C}$. On each query, the oracle applies the following unitary on $(\mathcal{Q}, \mathcal{P}, \mathcal{C})$:*

1. *Controlled on $\mathcal{C}$=1, apply $U_\Phi^\dagger$ to $(\mathcal{Q}, \mathcal{P})$.*
2. *Apply the swap on the subspace spanned by $|0\rangle_\mathcal{C} \otimes |0^m\rangle_\mathcal{P}$ and $|1\rangle_\mathcal{C} \otimes |0^m\rangle_\mathcal{P}$ (and act as the identity on the orthogonal subspace). Equivalently, flip $\mathcal{C}$ controlled on $\mathcal{P}$= $|0^m\rangle$.*
3. *Controlled on $\mathcal{C}$=1, apply U_Φ to $(\mathcal{Q}, \mathcal{P})$.*

O_Φ^{SEQ} may also be called $O_{U_\Phi}^{\mathsf{SEQ}}$ to make the particular Stinespring dilation U_Φ explicit.

It is often unnecessary to write $O_{U_\Phi}^{\mathsf{SEQ}}$, which makes the particular dilation U_Φ explicit, and instead O_Φ^{SEQ} suffices. Lemma 4.10 says that any two Stinespring dilations U, U' of Φ will produce indistinguishable oracles O_U^{SEQ} and $O_{U'}^{\mathsf{SEQ}}$.

The above makes the SEQ interface unitary-by-construction. Intuitively, the first time we query, Step 1 is inactive, Step 2 flips the computed flag, and Step 3 applies U_Φ. On subsequent queries, Steps 1 and 3 cancel on $(\mathcal{Q}, \mathcal{P})$, and Step 2 flips the flag only in the $\mathcal{P}$= $|0^m\rangle$ subspace; the overall interface remains unitary and well-defined.

In fact, we can verify that this unitary exactly maps the well-initialized input to its output and vice versa, while acting as identity on everything else.

Definition 4.3 (SEQ-based simulation security for one-time programs)**.** *A one-time program compiler* OTP *satisfies* SEQ*-based simulation security for a class $\mathcal{C}$ of quantum channels if there exists a q.p.t. simulator* Sim *such that for every $\Phi \in \mathcal{C}$ and for every q.p.t. distinguisher D, there exists a negligible function* $\mathsf{negl}(\lambda)$ *with*

$$\left|\Pr\left[1 \leftarrow D(1^\lambda, \Phi, \mathsf{OTP}(1^\lambda, \Phi))\right] - \Pr\left[1 \leftarrow D(1^\lambda, \Phi, \mathsf{Sim}^{O_\Phi^{\mathsf{SEQ}}}(1^\lambda))\right]\right| \leq \mathsf{negl}(\lambda).$$

As before, giving Φ to D means that D may depend on Φ (e.g., via a classical description or black-box evaluation access); this redundancy matches the order of quantifiers.

We now define the notion of a best-possible one-time program compiler, which produces one-time programs that are "best-possible" among all programs that implement the same sampling task and are testable.

Definition 4.4 (Best-possible OTP among testable programs)**.** *Let $\mathcal{C}$ be a family of quantum channels. A one-time program compiler* OTP^* *is* ***best-possible among testable programs*** *for $\mathcal{C}$ if there exists a q.p.t. simulator* Sim *such that for every $\Phi \in \mathcal{C}$, for every testable quantum program P that implements Φ and satisfies $|P| = |\Phi|$, and for every q.p.t. distinguisher D,*

$$\left|\Pr\left[1 \leftarrow D(1^\lambda, \Phi, \mathsf{OTP}^*(1^\lambda, \Phi))\right] - \Pr\left[1 \leftarrow D(1^\lambda, \Phi, \mathsf{Sim}(1^\lambda, P))\right]\right| \leq \mathsf{negl}(\lambda).$$

Here, "P implements Φ" means that tracing out P's private register after applying Eval *to the program state $|\psi\rangle$ realizes the channel Φ on the external interface.*

[9] Padding the output can in some cases weaken SEQ security by making the interface testable; see Appendix E.2.

Testability restricts the competing programs P in this definition; OTP^* need not itself output testable programs. We use *best-possible testable security* as shorthand for security against this comparison class.

Theorem 4.5. *Any one-time program compiler OTP^* satisfying SEQ security for a class of quantum channels (Definition 4.3) is a best-possible one-time program compiler among testable programs for that class (Definition 4.4).*

Proof. Since OTP^* satisfies SEQ security, there exists a q.p.t. simulator Sim such that for every channel Φ and every q.p.t. distinguisher D,

$$\left|\Pr[1 \leftarrow D(1^\lambda, \Phi, \mathsf{OTP}^*(1^\lambda, \Phi))] - \Pr[1 \leftarrow D(1^\lambda, \Phi, \mathsf{Sim}^{O_\Phi^{\mathsf{SEQ}}}(1^\lambda))]\right| \leq \mathsf{negl}(\lambda).$$

It therefore suffices to show that given any testable implementation P of Φ, we can simulate oracle access to O_Φ^{SEQ} using P. Composing Sim with this wrapper yields the simulator required by Definition 4.4.

Let $P = (|\psi\rangle, \mathsf{Eval}, R)$ be a testable quantum program that implements Φ, where $R = \mathsf{X} \otimes |\psi\rangle\langle\psi| + I \otimes (I - |\psi\rangle\langle\psi|)$ acts on a one-qubit flag and the program register. Without loss of generality, we can think about $|\psi\rangle$ padded with zero qubits which are used for the auxiliary wires, and the reflection unitary is extended to also reflect around zeroes for those qubits. The following simulator Sim' takes any such testable program P that implements Φ and simulates O_Φ^{SEQ}.

Definition 4.6 (Simulator $\mathsf{Sim}'(P)$ for O_Φ^{SEQ} from a testable program.)**.** *The oracle $\mathsf{Sim}'(P)$ maintains internally the program register $\mathcal{P}$ initialized to $|\psi\rangle$ and a one-qubit computed flag $\mathcal{C}$ initialized to $|0\rangle$. On each query on register $\mathcal{Q}$, $\mathsf{Sim}'(P)$ applies the following unitary on $(\mathcal{Q}, \mathcal{P}, \mathcal{C})$:*

1. *Controlled on $\mathcal{C}$=1, apply $\mathsf{Eval}^\dagger$ to $(\mathcal{Q}, \mathcal{P})$, mapping back to the input space of Eval.*
2. *Apply the reflection-controlled flip R to $(\mathcal{C}, \mathcal{P})$, i.e., flip $\mathcal{C}$ iff the program register equals $|\psi\rangle$.*
3. *Controlled on $\mathcal{C}$=1, apply Eval to $(\mathcal{Q}, \mathcal{P})$.*

We now argue that $\mathsf{Sim}'(P)$ is perfectly indistinguishable from O_Φ^{SEQ}. We write $\mathcal{A}$ for the adversary's private workspace register, which the adversary may initialize and act upon arbitrarily; the oracle's hidden registers $(\mathcal{C}, \mathcal{P})$ remain inaccessible.

Lemma 4.7. *Let P be a testable quantum program or a testable oracle interface[10] that implements Φ. Then oracles for $\mathsf{Sim}'(P)$ (definition 4.6) and O_Φ^{SEQ} are perfectly indistinguishable after any number of quantum queries.*

Proof. Let U_Φ be the canonical Stinespring dilation unitary for Φ. O_Φ^{SEQ} involves applying U_Φ. We will refer to O_Φ^{SEQ} as $O_{U_\Phi}^{\mathsf{SEQ}}$ to distinguish it from a similar oracle that applies a different unitary.

We will show that $\mathsf{Sim}'(P)$ is equivalent to $O_{U'_\Phi}^{\mathsf{SEQ}}$, for a particular Stinespring dilation U'_Φ of the channel Φ. Note that the Stinespring dilation needs $\mathcal{P}$ to be initialized to $|0^m\rangle$, whereas $\mathsf{Sim}'(P)$ initializes it to $|\psi\rangle$. We handle this discrepancy by conjugating each step of $\mathsf{Sim}'(P)$ with a swap operation that swaps the states $|0^m\rangle$ and $|\psi\rangle$.

Let U'_Φ be the following unitary:

1. Swap $|0^m\rangle_\mathcal{P}$ and $|\psi\rangle_\mathcal{P}$.
2. Apply Eval to $(\mathcal{Q}, \mathcal{P})$.

[10] The interface may maintain a hidden quantum state, provided its coherent test reflects around the designated initial state, including that hidden state.

3. Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.

Lemma 4.8. *U'_Φ is a valid Stinespring dilation of Φ.*

Proof. Given a state ρ on the query register $\mathcal{Q}$, consider the following three procedures for handling the query:

1. Apply channel Φ to $\rho_{\mathcal{Q}}$.
2. Apply Eval to $\rho_{\mathcal{Q}} \otimes |\psi\rangle_{\mathcal{P}}$ and then trace out $\mathcal{P}$.
3. Apply U'_Φ to $\rho_{\mathcal{Q}} \otimes |0^m\rangle_{\mathcal{P}}$ and then trace out $\mathcal{P}$.

Procedure 1 is equivalent to procedure 2 because P implements Φ. Next, procedure 2 is equivalent to procedure 3 because U'_Φ maps $|0^m\rangle_{\mathcal{P}} \to |\psi\rangle_{\mathcal{P}}$ before applying Eval. This shows that U'_Φ is a Stinespring dilation of Φ. □

Let $O^{\mathsf{SEQ}}_{U'_\Phi}$ be the SEQ oracle (definition 4.2) that uses unitary U'_Φ instead of U_Φ.

Lemma 4.9. *$O^{\mathsf{SEQ}}_{U'_\Phi}$ and $\mathsf{Sim}'(P)$ are perfectly indistinguishable after any number of quantum queries.*

Proof. Let's consider the following hybrids, which transform $O^{\mathsf{SEQ}}_{U'_\Phi}$ to $\mathsf{Sim}'(P)$:

Hybrid 1 – $O^{\mathsf{SEQ}}_{U'_\Phi}$ – Start with state $|0\rangle_{\mathcal{C}} |0^m\rangle_{\mathcal{P}}$ and handle each query as follows:

1. Controlled on $\mathcal{C}$=1, apply ${U'_\Phi}^\dagger$ as follows:
   (a) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.
   (b) Apply $\mathsf{Eval}^\dagger$ to $(\mathcal{Q}, \mathcal{P})$.
   (c) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.
2. Controlled on $\mathcal{P}$=$|0^m\rangle$, flip $\mathcal{C}$.
3. Controlled on $\mathcal{C}$=1, apply U'_Φ as follows:
   (a) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.
   (b) Apply Eval to $(\mathcal{Q}, \mathcal{P})$.
   (c) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.

Hybrid 2: Start with state $|0\rangle_{\mathcal{C}} |0^m\rangle_{\mathcal{P}}$ and handle each query as follows:

1. Controlled on $\mathcal{C}$=1:
   (a) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.
   (b) Apply $\mathsf{Eval}^\dagger$ to $(\mathcal{Q}, \mathcal{P})$.
   (c) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.
2. (a) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.
   (b) Controlled on $\mathcal{P}$=$|\psi\rangle$, flip $\mathcal{C}$.
   (c) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.
3. Controlled on $\mathcal{C}$=1:

(a) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.

(b) Apply Eval to $(\mathcal{Q}, \mathcal{P})$.

(c) Swap $|0^m\rangle_{\mathcal{P}}$ and $|\psi\rangle_{\mathcal{P}}$.

Hybrids 1 and 2 are equivalent. The only difference is step 2. In hybrid 1, we control on $\mathcal{P} = |0^m\rangle$. In hybrid 2, we swap $|0^m\rangle$ with $|\psi\rangle$ and control on $\mathcal{P} = |\psi\rangle$. These procedures implement the same operation.

Hybrid 3 – $\mathsf{Sim}'(P)$ – Start with state $|0\rangle_{\mathcal{C}} |\psi\rangle_{\mathcal{P}}$ and handle each query as follows:

1. Controlled on $\mathcal{C}$=1, apply $\mathsf{Eval}^\dagger$ to $(\mathcal{Q}, \mathcal{P})$.
2. Controlled on $\mathcal{P}$=$|\psi\rangle$, flip $\mathcal{C}$.
3. Controlled on $\mathcal{C}$=1, apply Eval to $(\mathcal{Q}, \mathcal{P})$.

The difference between hybrids 2 and 3 is that in hybrid 3, we have omitted all the swap operations (the steps that swap $|0^m\rangle$ and $|\psi\rangle$), and the initial state of $\mathcal{P}$ is $|\psi\rangle$, not $|0^m\rangle$.

We will argue that hybrids 2 and 3 are perfectly indistinguishable. Given any user that submits a sequence of queries to the oracle in hybrids 2 or 3, we can view their sequence of queries as a sequence of invocations of steps 1, 2, and 3. We will prove the following invariant: at the start or end of any step in the sequence, the current state of the system in hybrid 2 can be mapped to the current state of the system in hybrid 3 by applying the swap operation to $\mathcal{P}$, which swaps $|0^m\rangle$ and $|\psi\rangle$.

First, before the first step of the first query, the state of the oracle's registers are $|0\rangle_{\mathcal{C}} |0^m\rangle_{\mathcal{P}}$ in hybrid 2 and $|0\rangle_{\mathcal{C}} |\psi\rangle_{\mathcal{P}}$ in hybrid 3, so the invariant is satisfied at this point.

Second, let us assume the invariant is satisfied at the start of some invocation of step 1, and let's step through the execution of step 1 to show that the invariant is satisfied at the end. If $\mathcal{C} = 0$, then step 1 acts as the identity in both hybrids, so the invarinat will be satisfied at the end. Next, let's consider the case where $\mathcal{C} = 1$. In hybrid 2, step 1 applies the swap operation, which transforms the state to match the initial state of hybrid 3. Then it applies $\mathsf{Eval}^\dagger$, as is done in hybrid 3. At this point, the state is the same in the two hybrids. Finally, in hybrid 2, we apply the swap operation again so the final state in hybrid 2 could be transformed into the final state in hybrid 3 by another swap operation. Therefore the invariant is satisfied at the end of step 1.

Third, let us assume the invariant is satisfied at the start of some invocation of step 2 or 3. We can show that the invariant is still satisfied at the end of this step using similar reasoning to our argument for step 1.

In conclusion, before or after any step of any query, the current state of the system in hybrid 2 can be mapped to the current state of the system in hybrid 3 by applying a swap operation to $\mathcal{P}$. The swap operation is applied to an internal register of the oracle, which is not part of the user's view. If a computationally unbounded user makes an unbounded number of queries to either the hybrid 2 or hybrid 3 oracle and outputs a final state on their registers, this state will be the same once we trace out $\mathcal{P}$. Therefore, hybrids 2 and 3 are perfectly indistinguishable.

In total, we have shown that $O^{\mathsf{SEQ}}_{U'_\Phi}$ and $\mathsf{Sim}'(P)$ are perfectly indistinguishable after any number of queries. □

It just remains to show that $O^{\mathsf{SEQ}}_{U_\Phi}$ and $O^{\mathsf{SEQ}}_{U'_\Phi}$ are perfectly indistinguishable. This is implied by the following lemma.

Lemma 4.10 (Self-adjoint implementations hide dilation differences)**.** *Let U, U' be two Stinespring dilations of Φ with the same ancilla register $\mathcal{P}$ initialized to $|0^m\rangle_{\mathcal{P}}$. Then the oracles O^{SEQ}_U and $O^{\mathsf{SEQ}}_{U'}$ (definition 4.2) are perfectly indistinguishable after any number of queries.*

Proof. Let $P_0 := I_{\mathcal{Q}} \otimes |0^m\rangle\langle 0^m|_{\mathcal{P}}$ and $P_\perp := I - P_0$. Writing O_U^{SEQ} in 2×2 block form on $\mathcal{C}$ using the identities

$$\mathrm{C}_{\mathcal{C}}U = \begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix}, \quad \mathrm{C}_{\mathcal{C}}U^\dagger = \begin{pmatrix} I & 0 \\ 0 & U^\dagger \end{pmatrix}, \quad \mathrm{C}_{\mathcal{P}=|0^m\rangle}\mathrm{NOT}_{\mathcal{C}} = \begin{pmatrix} P_\perp & P_0 \\ P_0 & P_\perp \end{pmatrix},$$

where we use the shorthand $\mathrm{C}_{\mathcal{P}=|0^m\rangle}\mathrm{NOT}_{\mathcal{C}}$ to denote the controlled-NOT on target $\mathcal{C}$ with control projector P_0 on $\mathcal{P}$, namely $\mathrm{C}_{\mathcal{P}=|0^m\rangle}\mathrm{NOT}_{\mathcal{C}} := \mathsf{X}_{\mathcal{C}} \otimes P_0 + I_{\mathcal{C}} \otimes P_\perp$. Writing $\mathsf{X}_{\mathcal{C}} = |0\rangle\langle 0|\,1_{\mathcal{C}} + |1\rangle\langle 1|\,0_{\mathcal{C}}$ and $I_{\mathcal{C}} = |0\rangle\langle 0|\,0_{\mathcal{C}} + |1\rangle\langle 1|\,1_{\mathcal{C}}$, the resulting 2×2 block form on $\mathcal{C}$ is precisely $\left(\begin{smallmatrix} P_\perp & P_0 \\ P_0 & P_\perp \end{smallmatrix}\right)$. We obtain

$$O_U^{\mathsf{SEQ}} = \begin{pmatrix} P_\perp & P_0 U^\dagger \\ U P_0 & U P_\perp U^\dagger \end{pmatrix}_{\mathcal{C};(\mathcal{P},\mathcal{Q})}. \tag{3}$$

By Stinespring uniqueness (for equal-dimension ancillae), there exists a unitary $V_{\mathcal{P}}$ on $\mathcal{P}$ such that $U' P_0 = (I_{\mathcal{Q}} \otimes V_{\mathcal{P}}) U P_0$. This equality concerns the initialized subspace; the full unitary extensions need not agree elsewhere. Let $W := I_{\mathcal{Q}} \otimes V_{\mathcal{P}}$. Since $P_\perp = I - P_0$, we also have

$$W U P_\perp U^\dagger W^\dagger = I - W U P_0 U^\dagger W^\dagger = I - U' P_0 (U')^\dagger = U' P_\perp (U')^\dagger.$$

Define a unitary on $(\mathcal{C}, \mathcal{P}, \mathcal{Q})$ that is controlled by $\mathcal{C}$:

$$S := |0\rangle\langle 0|_{\mathcal{C}} \otimes I_{\mathcal{P}\mathcal{Q}} + |1\rangle\langle 1|_{\mathcal{C}} \otimes W\,.$$

Then, using the block form from (3),

$$\begin{aligned} S\, O_U^{\mathsf{SEQ}}\, S^\dagger &= \begin{pmatrix} I & 0 \\ 0 & W \end{pmatrix} \begin{pmatrix} P_\perp & P_0 U^\dagger \\ U P_0 & U P_\perp U^\dagger \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & W^\dagger \end{pmatrix} \\ &= \begin{pmatrix} P_\perp & P_0 U^\dagger W^\dagger \\ W U P_0 & W U P_\perp U^\dagger W^\dagger \end{pmatrix} \\ &= \begin{pmatrix} P_\perp & P_0 (U')^\dagger \\ U' P_0 & U' P_\perp (U')^\dagger \end{pmatrix} = O_{U'}^{\mathsf{SEQ}}. \end{aligned}$$

Thus $O_{U'}^{\mathsf{SEQ}} = S O_U^{\mathsf{SEQ}} S^\dagger$ with S acting only on the hidden registers $(\mathcal{C}, \mathcal{P})$ and trivially on $(\mathcal{A}, \mathcal{Q})$.

Consider any t-query adversary that interleaves local operations V_i on $(\mathcal{A}, \mathcal{Q})$ with oracle calls. Writing $O_U^{\mathrm{op}} := O_U^{\mathsf{SEQ}} \otimes I_{\mathcal{A}}$ and similarly for $O_{U'}^{\mathsf{SEQ}}$, we have

$$\begin{aligned} \mathcal{U}_{U'} &= V_t O_{U'}^{\mathrm{op}} \cdots V_1 O_{U'}^{\mathrm{op}} = V_t (S O_U^{\mathrm{op}} S^\dagger) \cdots V_1 (S O_U^{\mathrm{op}} S^\dagger) \\ &= S \left(V_t O_U^{\mathrm{op}} \cdots V_1 O_U^{\mathrm{op}} \right) S^\dagger = S\, \mathcal{U}_U\, S^\dagger, \end{aligned}$$

since S acts only on $(\mathcal{C}, \mathcal{P})$ and commutes with each V_i (which acts on $(\mathcal{A}, \mathcal{Q})$). The joint initial state is $|\psi\rangle_{\mathcal{A}} |0\rangle_{\mathcal{C}} |0^m\rangle_{\mathcal{P}} |\phi\rangle_{\mathcal{Q}}$, and S fixes it because S acts as identity on the $\mathcal{C}{=}0$ subspace. Therefore $\mathcal{U}_{U'} |\mathrm{init}\rangle = S\, \mathcal{U}_U |\mathrm{init}\rangle$. Tracing out the inaccessible registers $(\mathcal{C}, \mathcal{P})$ yields identical reduced states on $(\mathcal{A}, \mathcal{Q})$ in the two worlds (partial trace is invariant under local conjugation on the traced-out subsystem). Hence no test on $(\mathcal{A}, \mathcal{Q})$ can distinguish O_U^{SEQ} from $O_{U'}^{\mathsf{SEQ}}$, even across multiple queries. □

Lemmas 4.8 to 4.10 imply that oracle access to $\mathsf{Sim}'(P)$ and O_Φ^{SEQ} are perfectly indistinguishable. □

Giving the SEQ-security simulator Sim query access to $\mathsf{Sim}'(P)$, instead of O_Φ^{SEQ}, completes the proof of the theorem.

□

5 Stateful Quantum State Obfuscation: Towards a Plain Model Construction

Now that we have a strong definition for OTPs which makes sense in the plain model, we would like to actually construct it there. For a moment, let us suppose that we had a candidate construction. How would we show that any functionally equivalent program reveals no more information than the candidate? In the classical setting, we can prove that iO is best-possible by using iO to obfuscate the other, allegedly "better" program. However, in the quantum setting, the other program may use a quantum state - so at a minimum we need quantum state iO [BBV24, CG24]. Further than that, the other program *might change its behavior as it is queried* – for example by refusing to answer a second query. So it seems that any security proof would need the ability to obfuscate such programs.

In this section, we define and investigate the feasibility of quantum state obfuscation for programs which may modify their behavior as they are queried.

Definition 5.1 (Stateful Quantum Programs)**.** *A stateful quantum program is specified by a tuple $(U, |\psi\rangle)$ consisting of a unitary U and an initial state $|\psi\rangle$ contained in register $\mathcal{R}$. To evaluate a stateful quantum program on a state contained in register $\mathcal{Q}$, apply U to register $(\mathcal{Q}, \mathcal{R})$ and output register $\mathcal{Q}$.*

We define oracle access to P as follows. We initialize an internal register $\mathcal{R}$ with the state $|\psi\rangle$. A query can be either a forward or inverse query. On a forward query in register $\mathcal{Q}$, the operation U is performed on registers $(\mathcal{Q}, \mathcal{R})$, and the query register $\mathcal{Q}$ is returned. On an inverse query in register $\mathcal{Q}$, the operation U^{-1} is performed on registers $(\mathcal{Q}, \mathcal{R})$ and $\mathcal{Q}$ is returned. Following our unitary-oracle convention, either query may also be coherently controlled by a qubit supplied by the caller; the inactive branch leaves both $\mathcal{Q}$ and $\mathcal{R}$ unchanged. For a quantum oracle algorithm A, we denote oracle access to program P as A^P.

Definition 5.2 (Functional Equivalence Between Stateful Programs)**.** *Two stateful quantum programs $P_1 = (U_1, |\psi_1\rangle)$ and $P_2 = (U_2, |\psi_2\rangle)$ are (s, ε)-functionally equivalent if for every quantum oracle algorithm A_s making at most s oracle queries, the trace distance between the state of A_s given P_1 and its state when its given P_2 is at most ε, that is,* [11]

$$\frac{1}{2}\left\|A_s^{P_1} - A_s^{P_2}\right\|_1 \leq \varepsilon$$

If this holds for every $s = \mathsf{poly}(\lambda)$, we say they are $(\mathsf{poly}(\lambda), \varepsilon)$-functionally equivalent.

A stateful quantum program essentially implements a quantum channel *with memory*. As the program is queried, the state on register $\mathcal{R}$ may evolve, changing the behavior on future queries. As an example, imagine the "counting program" given by $|\psi\rangle = |0^n\rangle$ and U which increments the contents of $\mathcal{R}$ by 1 modulo 2^n, then classical-copies the result to register $\mathcal{Q}$. If one were to query this program twice on $|0\rangle$, the program would return $|1\rangle$ the first time and $|2\rangle$ the second time.

5.1 Stateful Quantum Indistinguishability Obfuscation

Using the notion of stateful functional equivalence, the definition of stateful iO is quite natural: if two programs have stateful functional equivalence, they can be indistinguishably obfuscated.

Definition 5.3 (Stateful Quantum Indistinguishability Obfuscation)**.** *A stateful quantum state indistinguishability obfuscator is a QPT compiler Obfs which takes in a stateful quantum program $(U, |\psi\rangle)$, then outputs another stateful quantum program $(\widetilde{U}, |\widetilde{\psi}\rangle)$:*

$$(\widetilde{U}, |\widetilde{\psi}\rangle) \leftarrow \mathsf{Obfs}(1^\lambda, (U, |\psi\rangle))$$

It must satisfy the following two properties.

[11] Note that the order of quantifiers permits A_s to depend on the programs. For example, A_s could evaluate P_1 on a description of U_1.

- ***Correctness:*** *There exists some negligible function μ such that for every stateful program $(U, |\psi\rangle)$, the obfuscation $(\widetilde{U}, |\widetilde{\psi}\rangle) \leftarrow \mathsf{Obfs}(1^\lambda, (U, |\psi\rangle))$ is $(\mathsf{poly}(\lambda), \mu(\lambda))$-functionally equivalent to $(U, |\psi\rangle)$ with probability $1 - \mathsf{negl}(\lambda)$.*
- ***Security:*** *For every pair of stateful programs $(U_1, |\psi_1\rangle)$ and $(U_2, |\psi_2\rangle)$ which are $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$-functionally equivalent and satisfy $|(U_1, |\psi_1\rangle)| = |(U_2, |\psi_2\rangle)|$,*

$$\{\mathsf{Obfs}(1^\lambda, (U_1, |\psi_1\rangle))\} \approx \{\mathsf{Obfs}(1^\lambda, (U_2, |\psi_2\rangle))\}$$

It is useful to compare this to the classical notion of indistinguishability obfuscation. In the classical setting, two programs P_1 and P_2 are considered to be functionally equivalent if $P_1(x) = P_2(x)$ for *every* x. In the quantum setting, the functional equivalence requirement is usually relaxed to allow negligible error in implementing the program [BK21, HT25], because it is generally quite difficult to exactly implement a quantum program.

This behavior can be seen as requiring that once P_1 and P_2 are fixed, *no* possible query x exists that would allow distinguishing black-box access to P_1 from access to P_2. We emphasize that in this view, x is still allowed to depend on the code of the two program even though the adversary otherwise can only make black-box queries to the selected program. As a sanity check, allowing the adversary to depend on the code ensures that this view still avoids impossibility results like [BGI+01, ABDS21] which are based on "feeding the program to itself".

In the stateful setting, the definition needs to account for the ability of the program to change over the course of several queries. We allow this by considering an adversary which may submit a polynomial number of queries in sequence. If the program did not update its state, then this would be equivalent to quantifying over all single queries, since sequential queries would have the same effect as single queries (up to negligible error due to allowing imperfect implementations). On the other hand, it might be the case that two stateful programs have identical outputs for, say, 10 queries, before suddenly changing behavior on the 11'th. Since even an adversary with black-box access could perform 11 queries and notice the difference on the last query, it does not make sense to claim that the two programs can be indistinguishably obfuscated.

The restriction to a polynomial number of queries is also quite natural. Although any polynomial-time adversary could certainly reach a polynomially-late program state by simply performing that many queries, to reach a superpolynomially-late program state they would need to interact with the program in a non-black-box manner. Thus, we argue that the ability to skip to a superpolynomially-late program state should be considered as a major break in the security of the scheme. At that point, we should simply consider obfuscating a state*less* program which allows anyone to skip to any point in the future.

Nonetheless, we do note that strengthening the functional equivalence requirement to a superpolynomial number of queries also makes sense as a (weaker) definition, if obfuscation for polynomial functional equivalence turns out to be impossible. We provide evidence in Section 5.3 that obfuscation is possible for programs which are only polynomially-equivalent by constructing such a scheme in an idealized model.

5.2 Stateful Quantum iO Implies Best-Possible Testable One-Time Programs

We show that such stateful quantum iO is actually sufficient to achieve quantum one-time programs that are best possible among testable programs. Thus, if one were to construct stateful quantum indistinguishability obfuscation in the plain model, they would also obtain good one-time programs.

Theorem 5.4. *Assuming stateful quantum indistinguishability obfuscation, there exists a quantum one-time program compiler for quantum channels that is best possible among testable programs (Definition 4.4).*

Proof. Let $\Phi : L(\mathcal{H}_{\mathcal{Q}}) \to L(\mathcal{H}_{\mathcal{Q}})$ be a quantum channel and let $P = (|0^m\rangle, \mathsf{Eval})$ be a canonical Stinespring dilation of Φ. Equip P with the circuit that reflects around $|0^m\rangle$. Fix a public polynomial L bounding the size of this canonical SEQ wrapper for channel descriptions of length s, as well as the size of $\mathsf{SEQ}(P')$ for testable programs P' of length s, including their reflection circuits and auxiliary states. Throughout this proof, pad each wrapper to length $L(s)$. The construction is $\mathsf{sqiO}(\mathsf{SEQ}(P))$, with $s = |\Phi|$, where $\mathsf{SEQ}(P)$ is a program implementing the SEQ oracle on P (see Section 4).

Given a testable program P' that implements Φ and satisfies $|P'| = |\Phi|$, the simulator constructs $\mathsf{SEQ}(P')$ from the supplied evaluation and reflection circuit descriptions, including the reflection auxiliary state in its private memory. This is a plain-model stateful program. The simulator outputs $\mathsf{sqiO}(\mathsf{SEQ}(P'))$, padded using the public size $s = |\Phi|$. The two wrappers therefore have the same length. By Lemma 4.7, they are $(t, 0)$-functionally equivalent for every $t \in \mathbb{N}$. Stateful iO security now gives the required indistinguishability.

□

5.3 Stateful Oracles in the Classical Oracle Model

Next, we provide evidence that stateful obfuscation is possible by constructing it in the classical oracle model. Unfortunately, we do not yet know how to construct stateful quantum state indistinguishability obfuscation in the plain model. Even giving a plausible candidate would resolve a major open question since it implies weaker notions of obfuscation which have not yet been constructed, such as iO for pseudodeterministic circuits or for unitaries.

Another interpretation of our construction is that *oracle models with memory are just as good as stateless oracle models.* For example, our Single Effective Query (SEQ) security is a query interface that maintains internal state across queries. Although it might seem somewhat odd to allow an oracle to maintain state, our result shows that this behavior can be achieved in the more standard classical oracle model.

Definition 5.5 (Ideal Stateful Quantum State Obfuscation)**.** *An ideal stateful quantum state obfuscator is a QPT compiler* Obfs *which takes as input a stateful quantum program* $(U, |\psi\rangle)$ *and outputs another stateful quantum program* $(\widetilde{U}, |\widetilde{\psi}\rangle)$*.*

$$(\widetilde{U}, |\widetilde{\psi}\rangle) \leftarrow \mathsf{Obfs}(1^\lambda, (U, |\psi\rangle))$$

In an oracle model, Obfs *may also output a description of an oracle algorithm matching the model (e.g. a unitary for the unitary oracle model), and all parties are given oracle access to the resulting operation. In the classical oracle model, this algorithm may use a truly random function internally; it is efficient relative to that function.*

It must satisfy the following two properties.

- ***Correctness:*** *There exists some* $\varepsilon = \mathsf{negl}(\lambda)$ *such that for every stateful program* $(U, |\psi\rangle)$*, the obfuscation* $(\widetilde{U}, |\widetilde{\psi}\rangle) \leftarrow \mathsf{Obfs}(1^\lambda, (U, |\psi\rangle))$ *is* $(\mathsf{poly}(\lambda), \varepsilon)$*-functionally equivalent to* $(U, |\psi\rangle)$ *with probability* $1 - \mathsf{negl}(\lambda)$*.*

- ***Security:*** *There exists a simulator* Sim *such that for every stateful program* $P = (U, |\psi\rangle)$*, every QPT adversary* $\mathcal{A}$*, and every QPT distinguisher* D*,*

$$\Big|\Pr[1 \leftarrow D(\mathcal{A}(\mathsf{Obfs}(1^\lambda, P)))] - \Pr[1 \leftarrow D(\mathsf{Sim}^P(1^\lambda, \mathcal{A}, n(\lambda), m(\lambda)))]\Big| = \mathsf{negl}(\lambda).$$

  where U *has at most* $n(\lambda)$ *gates,* $|\psi\rangle$ *has at most* $n(\lambda)$ *qubits, and* P *takes as input an* $m(\lambda)$*-sized register.*

A straightforward hybrid argument shows that ideal stateful obfuscator is also a stateful indistinguishability obfuscator. First, ideal obfuscation allows replacing an obfuscation of P_1 by a simulator who only has oracle access to P_1. Then, since the simulator makes only a polynomial number of queries, it cannot distinguish between oracle access to P_1 or P_2. Finally, ideal obfuscation allows switching back from the simulator with oracle access to P_2 to an obfuscation of P_2.

Claim 5.6. *Any ideal stateful quantum state obfuscator is also a stateful quantum state indistinguishability obfuscator.*

We now show how to construct ideal stateful obfuscation in the unitary oracle model. As a corollary of [HT25], it can also be constructed in the classical oracle model.

Construction 5.7. *The construction uses a publicly verifiable quantum authentication scheme* QAS *(e.g. the coset authentication scheme from [BBV24]) and an ideal unitary obfuscation* Obfs' *(e.g. Theorem 2.7 [HT25]).*

Without loss of generality, the QAS scheme supports Pauli key updates since we can always attach a uniformly random Pauli correction to the key. Such an update preserves the key distribution and changes only the decoded logical Pauli frame, hence preserving the accepting and rejecting subspaces of the authentication scheme. [BBV24]'s authentication scheme also supports Pauli key updates natively. We implement $\mathsf{QAS.Dec}_k$ *and* $\mathsf{QAS.Enc}_k$ *as reversible circuits that act unitarily on an enlarged work space and return all ancillas to* $|0\rangle$ *(i.e., they are coherently implemented isometries), so the following overall procedure is unitary. Assume that* U *does not make any use of oracles.*

On input $P = (U, |\psi\rangle)$ *and security parameter* 1^λ*:*

1. *Sample an authentication key* $k \leftarrow \mathsf{QAS.KeyGen}(1^\lambda)$.
2. *Compute* $|\widetilde{\psi}\rangle := \mathsf{QAS.Enc}_k(|\psi\rangle)$ *on register* $\mathcal{R}$.
3. *Let* $\widetilde{U}$ *be the following unitary:*
   (a) *Take as input two registers* $(\mathcal{Q},\ \mathcal{R})$
   (b) *Apply* $\mathsf{QAS.Dec}_k$ *to register* $\mathcal{R}$*. If the result is* $\perp$*, uncompute* $\mathsf{QAS.Dec}_k$ *and return the input registers immediately.*
   (c) *Otherwise, apply* U *to* $(\mathcal{Q}, \mathcal{R})$.
   (d) *Apply* $\mathsf{QAS.Enc}_k$ *to register* $\mathcal{R}$.
   (e) *Return* $(\mathcal{Q}, \mathcal{R})$.
4. *Compute the unitary obfuscation* $(|\widetilde{\phi}\rangle\,, C) \leftarrow \mathsf{Obfs}'(\widetilde{U})$.[12]
5. *Output* $|\widetilde{\psi}\rangle\,, \mathsf{Obfs}'(\widetilde{U})$.

Theorem 5.8. *Ideal stateful quantum obfuscation exists in the classical oracle model.*

Proof. Correctness follows from the perfect correctness of authentication and the per-output, worst-case correctness of Theorem 2.7. With overwhelming probability over the unitary obfuscation, its reusable evaluator implements the authenticated update and its controlled forward and inverse applications with negligible error over any polynomial number of queries. On honestly authenticated memory these updates implement the original stateful program, giving the required functional equivalence.

[12] Here and below, $\mathsf{Obfs}'(V)$ includes the following adaptation: obfuscate $W = I \oplus V$, and use $Q^\dagger X_b Q X_b$, where Q is its reversible evaluator and the added qubit b starts in $|0\rangle$. By [HT25, Lemma 6.4], this restores the auxiliary state and supports reusable controlled forward and inverse evaluation with negligible error. The resulting controlled oracle queries are handled throughout the simulation and hybrids below by conjoining their controls with authentication acceptance.

Let n be the size of the internal state register $\mathcal{R}$, in qubits. Let Adv be the adversary. Let Sim' be the simulator of the ideal unitary obfuscation scheme Obfs'. Let Adv' be the adversary obtained as $\mathsf{Adv}' = \mathsf{Sim}'(\mathsf{Adv}, \cdot)$.

The simulator $\mathsf{Sim}^P(\mathsf{Adv}')$ first samples $k \leftarrow \mathsf{Auth.KeyGen}(1^\lambda)$. Prepare n EPR pairs in registers $(\mathcal{A}_1, \mathcal{B}_1), \ \ldots, \ (\mathcal{A}_n, \mathcal{B}_n)$ and apply $\mathsf{Auth.Enc}_k$ to register $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_n)$, expanding the register as necessary, to obtain

$$\propto \sum_{x \in \{0,1\}^n} |x\rangle_{\mathcal{A}} \otimes \mathsf{Auth.Enc}_k(|x\rangle)_{\mathcal{B}}$$

It runs $\mathsf{Adv}'^{(\cdot)}(\mathcal{B})$, answering its oracle queries as follows. Verification and its control on the oracle call are performed coherently, and the verification workspace is uncomputed before returning.

1. Parse the query register as $(\mathcal{Q}, \mathcal{B}')$.
2. Run $\mathsf{Auth.Ver}_k(\mathcal{B}')$. If the result is $\perp$, uncompute $\mathsf{Auth.Ver}_k$ and return immediately. Otherwise continue.
3. Query P on $\mathcal{Q}$. Return the result.

We show that $\mathsf{Sim}^P(\mathsf{Adv})$ is indistinguishable from $\mathsf{Adv}^{\widetilde{U}}(|\widetilde{\psi}\rangle)$ for any $q = \mathsf{poly}(\lambda)$ queries via a series of hybrid experiments.

- $\mathsf{Hyb}_{-1} = \mathsf{Adv}(|\widetilde{\psi}\rangle, \mathsf{Obfs}'(\widetilde{U}))$.
- $\mathsf{Hyb}_0 = \mathsf{Sim}'^{\widetilde{U},\widetilde{U}^\dagger}(\mathsf{Adv}, |\widetilde{\psi}\rangle) = \mathsf{Adv}'^{\widetilde{U},\widetilde{U}^\dagger}(|\widetilde{\psi}\rangle)$.
- Hyb_1: Prepare the half-authenticated EPR pairs

  $$\propto \sum_{x \in \{0,1\}^n} |x\rangle_{\mathcal{A}} \otimes \mathsf{Auth.Enc}_k(|x\rangle)_{\mathcal{B}}$$

  as in the simulator. Teleport $|\psi\rangle$ into the authenticated register $\mathcal{B}$ using the EPR halves in register $\mathcal{A}$. Run $\mathsf{Sim}'^{\widetilde{U}_{k'},\widetilde{U}^\dagger_{k'}}(\mathsf{Adv}, \mathcal{B}) = \mathsf{Adv}'^{\widetilde{U}_{k'},\widetilde{U}^\dagger_{k'}}(\mathcal{B})$. Here, $\widetilde{U}_{k'}$ denotes that the oracle uses key k'.
- Hyb_2: Instead of performing the teleportation and key update $k \mapsto k'$ at initialization, perform them when Sim' submits its first query (before answering the query). Until that point, store the internal state, which is initialized to $|\psi\rangle$, in register $\mathcal{R}$.
- Hyb_{2+i} for $i = 1$ to q: Initialize Sim' as in Hyb_2. For the first i queries, answer the query as in Sim, performing P internally using register $\mathcal{R}$ and controlled on authentication acceptance. Then, upon receiving query $i+1$, teleport the state of register $\mathcal{R}$ using $\mathcal{A}$ and update the key $k \mapsto k'$. Then continue answering queries as in the real scheme. If no further query is made, omit the teleportation and key update.
- $\mathsf{Sim} = \mathsf{Hyb}_{2+q}$.

$\mathsf{Hyb}_{-1} \approx \mathsf{Hyb}_0$ by the security of the ideal obfuscation of unitary $\widetilde{U}$ (see Definition 2.6).

$\mathsf{Hyb}_0 = \mathsf{Hyb}_1$ by the correctness of teleportation and the ability to perform Pauli key updates on the authentication scheme. $\mathsf{Hyb}_1 = \mathsf{Hyb}_2$ since the teleportation and key updates are performed on disjoint registers from those hold by Adv. The main step is to show that $\mathsf{Hyb}_{2+i} \approx \mathsf{Hyb}_{2+i+1}$.

Claim 5.9. *For every* $0 \leq i < q$,

$$\mathsf{Hyb}_{2+i} \approx \mathsf{Hyb}_{2+i+1}$$

Proof. Set $j = i + 1$. We further divide the transition into a sequence of hybrid experiments. For an inverse-oracle query, replace U by $U^\dagger$ throughout.

- $\mathsf{Hyb}_{2+i,1}$: For the first i queries, answer the query as in Sim. The difference from Hyb_{2+i} happens upon receiving query j.

  Upon receiving query j, coherently compute $\mathsf{QAS.Dec}_k$ on the queried register $\mathcal{B}'$. Let Π_{acc} project onto successful decoding, and let Π_{EPR} project the decoded message register and $\mathcal{A}$ onto

  $$|\Omega\rangle_{\mathcal{A}\mathcal{B}'} = 2^{-n/2} \sum_{x\in\{0,1\}^n} |x\rangle_{\mathcal{A}} \otimes |x\rangle_{\mathcal{B}'} .$$

  Measure the two-outcome projection with bad projector

  $$\Pi_{\mathrm{bad}} = \Pi_{\mathrm{acc}}(I - \Pi_{\mathrm{EPR}}).$$

  Abort and output $\perp$ only on the bad outcome. The good outcome includes both rejection and acceptance with intact EPR pairs, preserving coherence between these two subspaces. Uncompute the decoder and continue as in Hyb_{2+i}, starting with teleportation using $\mathcal{A}$ and the key update.

- $\mathsf{Hyb}_{2+i,2}$: This is the same as $\mathsf{Hyb}_{2+i,1}$ except that after the good outcome on query j, the order of computation and teleportation is swapped. While $\mathcal{B}'$ is decoded, apply U to $(\mathcal{Q}, \mathcal{R})$ controlled on authentication acceptance. Uncompute the decoder, then teleport the contents of $\mathcal{R}$ using $\mathcal{A}$ and update $k \mapsto k'$. Return the query registers and answer subsequent queries as in the real scheme.

- $\mathsf{Hyb}_{2+i,3}$: This is the same as $\mathsf{Hyb}_{2+i,2}$, except that the early abort measurement on query j is *not* performed. The computation of U remains controlled on authentication acceptance.

  Note that the decoded message in $\mathcal{B}'$ is not used for query j; only the coherent acceptance flag controls U on $(\mathcal{Q}, \mathcal{R})$.

- Hyb_{2+i+1}: Replace coherent decoding, control on acceptance, and uncomputation by coherent verification, the same control, and uncomputation. Defer teleportation and the key update until immediately before query $j + 1$, omitting them if there is no further query.

Publicly verifiable authentication bounds the probability η of Π_{bad} by a negligible function. To see the entangled-state guarantee explicitly, let σ_{acc} be the subnormalized decoded state on $(\mathcal{A}, \mathcal{B}')$ on acceptance, averaging over the authentication key. The n EPR pairs are the unique joint $+1$ eigenstate of the $2n$ commuting Bell stabilizers $X_{\mathcal{A}_j} X_{\mathcal{B}'_j}$ and $Z_{\mathcal{A}_j} Z_{\mathcal{B}'_j}$. Their projectors therefore give the union bound

$$\eta = \mathrm{Tr}[(I - |\Omega\rangle\langle\Omega|)\sigma_{\mathrm{acc}}] \le \sum_{\ell=1}^{n} \sum_{S\in\{X,Z\}} \mathrm{Tr}\left[\frac{I - S_{\mathcal{A}_\ell} S_{\mathcal{B}'_\ell}}{2}\, \sigma_{\mathrm{acc}}\right].$$

For either S-basis, measuring $\mathcal{A}$ at initialization prepares a uniformly random product S-basis state in the authenticated message. Since $\mathcal{A}$ is untouched before teleportation, each summand equals the average probability that authentication accepts but the decoded message's ℓth S-basis bit differs from that input bit. Choosing (ℓ, S) uniformly gives one efficient test with failure probability ε, so $\eta \le 2n\varepsilon$. This test uses only product X/Z-basis preparations and single-bit tests, and publicly verifiable mapping security of [BBV24] makes ε negligible. Thus η is negligible, and inserting the bad-outcome measurement changes the state by at most $O(\sqrt{\eta})$ by gentle measurement, proving $\mathsf{Hyb}_{2+i} \approx \mathsf{Hyb}_{2+i,1}$. The inserted measurement detects accepted corruption; it does not measure

the authentication acceptance flag itself. Its abort probability is therefore η, even when ordinary authentication rejection has large probability, and its good outcome preserves coherence between rejection and intact acceptance. Public verifiability is necessary because oracle access to Ver_k is used to answer the first i queries.

$\mathsf{Hyb}_{2+i,1} = \mathsf{Hyb}_{2+i,2}$ holds on the entire good subspace. On its accepting component, $(\mathcal{A}, \mathcal{B}')$ contains n EPR pairs, so teleporting $\mathcal{R}$ and then applying U is equivalent to applying U to $(\mathcal{Q}, \mathcal{R})$ and then teleporting. On its rejecting component, both procedures skip U and perform the same teleportation. The Pauli key update preserves the authentication subspaces, so rejection before teleportation remains rejection afterward. These identities hold for each teleportation outcome and hence also preserve coherence between the accepting and rejecting components.

$\mathsf{Hyb}_{2+i,2} \approx \mathsf{Hyb}_{2+i,3}$ follows by removing the same gentle measurement, again at cost $O(\sqrt{\eta})$.

Finally, $\mathsf{Hyb}_{2+i,3} = \mathsf{Hyb}_{2+i+1}$ because coherent verification implements the same acceptance projector as decoding, and the decoded message is otherwise unused. Teleportation acts only on the simulator's registers $(\mathcal{A}, \mathcal{R})$, so it can be deferred past the adversary's computation until the next query, or omitted at the end. Summing over the q transitions gives negligible total error $O(q\sqrt{\eta})$, taking η to bound the bad-outcome probability in every transition. □

□

By using Theorem 5.8 and obfuscating the SEQ oracle, which is a stateful quantum program, we achieve one-time programs with SEQ security for all quantum channels in the classical oracle model.

Corollary 5.10. *There exist one-time programs with SEQ security for all quantum channels in the classical oracle model.*

Use of Generative AI

Generative AI tools, mainly Codex with various versions of GPT, were used in an iterative workflow involving evolving manuscript drafts and author feedback. Assistance included organizing and expressing author-supplied ideas and arguments, drafting and revising passages, and checking and clarifying technical arguments. Intellectual contributions included identifying a reference for the Stinespring uniqueness fact used in the proof of Lemma 4.10, pointing out that the assumption for the impossibility can be weakened to hardness of Statistical Difference (originally lossy encryptions), and developing the separation proof of Theorem E.1 in Appendix E.1. The authors reviewed and revised the resulting material and take responsibility for the final work.

Acknowledgements

AG was supported in part by DARPA under Agreement No. HR00112020023, NSF CNS-2154149 and a Simons Investigator Award. This work was done in part while AG was at the Simons Institute and participating in the Challenge Institute for Quantum Computation at UC Berkeley.

References

[Aar05] Scott Aaronson. "Limitations of Quantum Advice and One-Way Communication". In: *Theory of Computing* 1.1 (2005), pp. 1–28. DOI: 10.4086/toc.2005.v001a001 (cit. on pp. 4, 5).

[Aar18] Scott Aaronson. "PDQP/qpoly = ALL". In: *Quantum Information and Computation* 18.11–12 (2018), pp. 901–909. DOI: 10.26421/QIC18.11-12-1. arXiv: 1805.08577 (cit. on p. 5).

[ABDS21] Gorjan Alagic, Zvika Brakerski, Yfke Dulek, and Christian Schaffner. "Impossibility of Quantum Virtual Black-Box Obfuscation of Classical Circuits". In: *Advances in Cryptology—CRYPTO 2021, Part I*. 2021, pp. 497–525. DOI: 10.1007/978-3-030-84242-0_18. arXiv: 2005.06432 (cit. on p. 34).

[ABFL16] Scott Aaronson, Adam Bouland, Joseph Fitzsimons, and Mitchell Lee. "The Space "Just Above" BQP". In: *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*. 2016, pp. 271–280. DOI: 10.1145/2840728.2840739. arXiv: 1412.6507 (cit. on p. 5).

[ABK24] Scott Aaronson, Harry Buhrman, and William Kretschmer. "A Qubit, a Coin, and an Advice String Walk into a Relational Problem". In: *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*. Vol. 287. 2024, 1:1–1:24. DOI: 10.4230/LIPIcs.ITCS.2024.1. arXiv: 2302.10332 (cit. on p. 4).

[AFMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. "Cryptographic Group Actions and Applications". In: *Advances in Cryptology—ASIACRYPT 2020, Part II*. 2020, pp. 411–439. DOI: 10.1007/978-3-030-64834-3_14 (cit. on p. 47).

[AGI+25] Scott Aaronson, Sabee Grewal, Vishnu Iyer, Simon C. Marshall, and Ronak Ramachandran. "PDQMA = DQMA = NEXP: QMA with Hidden Variables and Non-Collapsing Measurements". In: *45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025)*. Vol. 360. 2025, 3:1–3:17. DOI: 10.4230/LIPIcs.FSTTCS.2025.3. arXiv: 2403.02543 (cit. on p. 5).

[ANTV02] Andris Ambainis, Ashwin Nayak, Amnon Ta-Shma, and Umesh Vazirani. "Dense Quantum Coding and Quantum Finite Automata". In: *Journal of the ACM* 49.4 (2002), pp. 496–511. DOI: 10.1145/581771.581773 (cit. on p. 4).

[BBV24] James Bartusek, Zvika Brakerski, and Vinod Vaikuntanathan. "Quantum State Obfuscation from Classical Oracles". In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*. 2024, pp. 1009–1017. DOI: 10.1145/3618260.3649673. Cryptology ePrint Archive: 2024/082 (cit. on pp. 5, 16, 18, 21, 33, 36, 38).

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *Advances in Cryptology—ASIACRYPT*. 2011, pp. 41–69. DOI: 10.1007/978-3-642-25385-0_3 (cit. on p. 7).

[BDRV18] Itay Berman, Akshay Degwekar, Ron D. Rothblum, and Prashant Nalini Vasudevan. "From Laconic Zero-Knowledge to Public-Key Cryptography". In: *Advances in Cryptology – CRYPTO 2018*. Vol. 10993. 2018, pp. 674–697. DOI: 10.1007/978-3-319-96878-0_23. Cryptology ePrint Archive: 2018/548. URL: https://eccc.weizmann.ac.il/report/2017/172/ (cit. on p. 46).

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. "On the (Im)possibility of Obfuscating Programs". In: *Advances in Cryptology — CRYPTO 2001*. 2001, pp. 1–18. DOI: 10.1007/3-540-44647-8_1 (cit. on pp. 3, 6, 7, 13, 14, 17, 21, 34).

[BGMZ18] James Bartusek, Jiaxin Guan, Fermi Ma, and Mark Zhandry. *Preventing Zeroizing Attacks on GGH15*. 2018. Cryptology ePrint Archive: 2018/511 (cit. on p. 3).

[BGS13] Anne Broadbent, Gus Gutoski, and Douglas Stebila. "Quantum One-Time Programs". In: *Advances in Cryptology—CRYPTO 2013, Part II*. 2013, pp. 344–360. DOI: 10.1007/978-3-642-40084-1_20. arXiv: 1211.1080 (cit. on pp. 3, 5, 6, 12, 15, 19).

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. "Possibility and Impossibility Results for Encryption and Commitment Secure under Selective Opening". In: *Advances in Cryptology – EUROCRYPT 2009*. Vol. 5479. 2009, pp. 1–35. DOI: 10.1007/978-3-642-01001-9_1 (cit. on p. 46).

[BJ15] Anne Broadbent and Stacey Jeffery. "Quantum Homomorphic Encryption for Circuits of Low T-Gate Complexity". In: *Advances in Cryptology—CRYPTO 2015, Part II*. 2015, pp. 609–629. DOI: 10.1007/978-3-662-48000-7_30. arXiv: 1412.8766 (cit. on p. 19).

[BK21] Anne Broadbent and Raza Ali Kazmi. "Constructions for Quantum Indistinguishability Obfuscation". In: *Progress in Cryptology—LATINCRYPT 2021*. 2021, pp. 24–43. DOI: 10.1007/978-3-030-88238-9_2. arXiv: 2005.14699 (cit. on p. 34).

[BKL24] Anne Broadbent, Martti Karvonen, and Sébastien Lord. "Uncloneable Quantum Advice". In: *IACR Communications in Cryptology* 1.3 (2024). DOI: 10.62056/abe0fhbmo. arXiv: 2309.05155 (cit. on p. 4).

[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 (STOC)*. 2023, pp. 1567–1578. DOI: 10.1145/3564246.3585179. Cryptology ePrint Archive: 2023/252 (cit. on p. 66).

[BM25] Roozbeh Bassirian and Kunal Marwaha. "Superposition detection and QMA with non-collapsing measurements". In: *Quantum* 9 (2025), p. 1839. DOI: 10.22331/q-2025-08-28-1839. arXiv: 2403.02532 (cit. on p. 5).

[BS23] Shalev Ben-David and Or Sattath. "Quantum Tokens for Digital Signatures". In: *Quantum* 7, 901 (2023). DOI: 10.22331/q-2023-01-19-901. arXiv: 1609.09047 (cit. on pp. 3, 7, 12, 14, 19).

[CD08] Ran Canetti and Ronny Ramzi Dakdouk. "Obfuscating Point Functions with Multibit Output". In: *Advances in Cryptology—EUROCRYPT 2008*. 2008, pp. 489–508. DOI: 10.1007/978-3-540-78967-3_28 (cit. on p. 3).

[CG24] Andrea Coladangelo and Sam Gunn. "How to Use Quantum Indistinguishability Obfuscation". In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*. 2024, pp. 1003–1008. DOI: 10.1145/3618260.3649779. arXiv: 2311.07794 (cit. on pp. 5, 16, 18, 33).

[ÇKM+26] Alper Çakan, Dakshita Khurana, Tomoyuki Morimae, Yuki Shirakawa, Kabir Tomer, and Takashi Yamakawa. "On the Cryptographic Futility of Non-collapsing Measurements". In: *Advances in Cryptology – EUROCRYPT 2026*. Vol. 16541. 2026, pp. 603–632. DOI: 10.1007/978-3-032-25291-3_21. arXiv: 2510.05055 (cit. on p. 5).

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. "Obfuscation of Probabilistic Circuits and Applications". In: *Theory of Cryptography – TCC 2015, Part II*. Vol. 9015. 2015, pp. 468–497. DOI: 10.1007/978-3-662-46497-7_19. Cryptology ePrint Archive: 2014/882 (cit. on p. 7).

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. "Spooky Encryption and Its Applications". In: *Advances in Cryptology – CRYPTO 2016, Part III*. Vol. 9816. 2016, pp. 93–122. DOI: 10.1007/978-3-662-53015-3_4. Cryptology ePrint Archive: 2016/272 (cit. on p. 7).

[FGH+10] Edward Farhi, David Gosset, Avinatan Hassidim, Andrew Lutomirski, Daniel Nagaj, and Peter Shor. "Quantum State Restoration and Single-Copy Tomography for Ground States of Hamiltonians". In: *Physical Review Letters* 105.19, 190503 (2010). Full version. DOI: 10.1103/PhysRevLett.105.190503. arXiv: 0912.3823v2 (cit. on p. 19).

[Gav12] Dmytro Gavinsky. "Quantum Money with Classical Verification". In: *2012 IEEE 27th Annual Conference on Computational Complexity*. 2012, pp. 42–52. DOI: 10.1109/CCC.2012.10. arXiv: 1109.0372 (cit. on p. 4).

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. "Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits". In: *SIAM Journal on Computing* 45.3 (2016), pp. 882–929. DOI: 10.1137/14095772X (cit. on p. 3).

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "One-Time Programs". In: *Advances in Cryptology—CRYPTO 2008*. 2008, pp. 39–56. DOI: 10.1007/978-3-540-85174-5_3 (cit. on pp. 3, 12, 19).

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. "Lockable Obfuscation". In: *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. 2017, pp. 612–621. DOI: 10.1109/FOCS.2017.62. Cryptology ePrint Archive: 2017/274 (cit. on p. 3).

[GLR+25] Aparna Gupte, Jiahui Liu, Justin Raizes, Bhaskar Roberts, and Vinod Vaikuntanathan. "Quantum One-Time Programs, Revisited". In: *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC)*. 2025, pp. 213–221. DOI: 10.1145/3717823.3718204. arXiv: 2411.01876 (cit. on pp. 3, 7, 12–16, 19, 20, 27, 48, 52).

[GM24] Sam Gunn and Ramis Movassagh. *Quantum One-Time Protection of Any Randomized Algorithm*. 2024. Cryptology ePrint Archive: 2024/1798 (cit. on pp. 3, 7, 12, 14, 19, 66).

[GR07] Shafi Goldwasser and Guy N. Rothblum. "On Best-Possible Obfuscation". In: *Theory of Cryptography Conference (TCC)*. 2007, pp. 194–213. DOI: 10.1007/978-3-540-70936-7_11 (cit. on pp. 4, 8, 11, 23).

[GSV98] Oded Goldreich, Amit Sahai, and Salil Vadhan. "Honest-Verifier Statistical Zero-Knowledge Equals General Statistical Zero-Knowledge". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. 1998, pp. 399–408. DOI: 10.1145/276698.276852 (cit. on pp. 8, 25).

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin. "Lattice-Based Post-Quantum iO from Circular Security with Random Opening Assumption". In: *Advances in Cryptology—CRYPTO 2025, Part VII*. 2025, pp. 3–38. DOI: 10.1007/978-3-032-01907-3_1. Cryptology ePrint Archive: 2025/390 (cit. on p. 3).

[HLOV11] Brett Hemenway, Benoît Libert, Rafail Ostrovsky, and Damien Vergnaud. "Lossy Encryption: Constructions from General Assumptions and Efficient Selective Opening Chosen Ciphertext Security". In: *Advances in Cryptology—ASIACRYPT 2011*. 2011, pp. 70–88. DOI: 10.1007/978-3-642-25385-0_4. Cryptology ePrint Archive: 2009/088 (cit. on p. 45).

[Hol73] A. S. Holevo. "Bounds for the Quantity of Information Transmitted by a Quantum Communication Channel". In: *Problems of Information Transmission* 9.3 (1973), pp. 177–183. URL: https://www.mathnet.ru/eng/ppi903 (cit. on p. 4).

[HT25] Mi-Ying Huang and Er-Cheng Tang. *Obfuscation of Unitary Quantum Programs*. 2025. arXiv: 2507.11970 (cit. on pp. 16–18, 22, 34, 36).

[JLS26] Aayush Jain, Huijia Lin, and Amit Sahai. "Indistinguishability Obfuscation from Well-Founded Assumptions". In: *Journal of the ACM* 73.1 (2026). DOI: 10.1145/3785007. arXiv: 2008.09317 (cit. on p. 3).

[Liu23] Qipeng Liu. "Non-uniformity and Quantum Advice in the Quantum Random Oracle Model". In: *Advances in Cryptology—EUROCRYPT*. 2023, pp. 117–143. DOI: 10.1007/978-3-031-30545-0_5 (cit. on p. 4).

[LPS04] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. "Positive Results and Techniques for Obfuscation". In: *Advances in Cryptology—EUROCRYPT 2004*. 2004, pp. 20–39. DOI: 10.1007/978-3-540-24676-3_2. Cryptology ePrint Archive: 2004/060 (cit. on p. 3).

[Mal15] Lior Malka. "How to Achieve Perfect Simulation and a Complete Problem for Non-interactive Perfect Zero-Knowledge". In: *Journal of Cryptology* 28.3 (2015), pp. 533–550. DOI: 10.1007/s00145-013-9165-6. URL: https://eccc.weizmann.ac.il/report/2009/132/ (cit. on pp. 10, 25, 26).

[MP25] David Miloschewsky and Supartha Podder. "New Lower-Bounds for Quantum Computation with Non-Collapsing Measurements". In: *40th Computational Complexity Conference (CCC 2025)*. Vol. 339. 2025, 12:1–12:23. DOI: 10.4230/LIPIcs.CCC.2025.12. arXiv: 2411.04085 (cit. on p. 5).

[MSY25] Tomoyuki Morimae, Yuki Shirakawa, and Takashi Yamakawa. *Quantum Cryptography and Hardness of Non-Collapsing Measurements*. To appear in ASIACRYPT 2026. 2025. arXiv: 2510.04448 (cit. on p. 5).

[Nay99] Ashwin Nayak. "Optimal Lower Bounds for Quantum Automata and Random Access Codes". In: *40th Annual Symposium on Foundations of Computer Science*. 1999, pp. 369–376. DOI: 10.1109/SFFCS.1999.814608. arXiv: quant-ph/9904093 (cit. on p. 4).

[PW08] Chris Peikert and Brent Waters. "Lossy Trapdoor Functions and Their Applications". In: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*. 2008, pp. 187–196. DOI: 10.1145/1374376.1374406. Cryptology ePrint Archive: 2007/279 (cit. on p. 46).

[Sim94] Daniel R. Simon. "On the power of quantum computation". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 116–123. DOI: 10.1109/SFCS.1994.365701 (cit. on p. 6).

[SV03] Amit Sahai and Salil Vadhan. "A Complete Problem for Statistical Zero Knowledge". In: *Journal of the ACM* 50.2 (2003), pp. 196–249. DOI: 10.1145/636865.636868. Cryptology ePrint Archive: 2000/056 (cit. on pp. 9, 24, 26).

[SW14] Amit Sahai and Brent Waters. “How to Use Indistinguishability Obfuscation: Deniable Encryption, and More”. In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*. 2014, pp. 475–484. DOI: 10.1145/2591796.2591825. Cryptology ePrint Archive: 2013/454 (cit. on p. 3).

[Vad99] Salil Pravin Vadhan. “A Study of Statistical Zero-Knowledge Proofs”. PhD thesis. Massachusetts Institute of Technology, 1999. URL: https://www.wisdom.weizmann.ac.il/~oded/PSX/salil-phd.pdf (cit. on p. 46).

[WZ17] Daniel Wichs and Giorgos Zirdelis. “Obfuscating Compute-and-Compare Programs under LWE”. In: *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. 2017, pp. 600–611. DOI: 10.1109/FOCS.2017.61. Cryptology ePrint Archive: 2017/276 (cit. on p. 3).

[Zha25] Mark Zhandry. “How to Model Unitary Oracles”. In: *Advances in Cryptology—CRYPTO 2025, Part II*. 2025, pp. 237–268. DOI: 10.1007/978-3-032-01878-6_8. Cryptology ePrint Archive: 2025/1072 (cit. on p. 20).

Supplementary Material

A Lossy Public-Key Encryption

This appendix records how lossy public-key encryption (lossy PKE) gives hard problems in the zero-knowledge classes used in our impossibility results. We recall the standard construction from LWE by citation and give the perfectly lossy construction from group actions.

In injective mode, lossy PKE functions like normal PKE, in which Enc hides the message computationally, but not statistically. In lossy mode, Enc hides the message statistically or perfectly, so the output distributions of $\mathsf{Enc}(0)$ and $\mathsf{Enc}(1)$ are (respectively) statistically close or identical. Furthermore, the injective and lossy modes are indistinguishable to an adversary who is given the public encryption key.

The following definition is based on [HLOV11, Definition 2]. We allow a negligible fraction of bad keys, while requiring perfect correctness for every message on the remaining injective keys. We also distinguish statistical from perfect lossiness and require indistinguishability against quantum adversaries.

Definition A.1 (Statistically/Perfectly Lossy Public-Key Encryption)**.** *A* ***statistically/perfectly lossy public-key encryption*** *(lossy PKE) scheme is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax.*

Let $\mathcal{M}$ *be the message space, and let* $\mathcal{R}_{\mathsf{Gen}}, \mathcal{R}_{\mathsf{Enc}}, \mathcal{R}_{\mathsf{Dec}}$ *be the sample space of the randomness for* $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$*, respectively. The sizes of these sets may depend on* λ*. Also let* $r_{\mathsf{Gen}} \xleftarrow{\$} \mathcal{R}_{\mathsf{Gen}}, r_{\mathsf{Enc}} \xleftarrow{\$} \mathcal{R}_{\mathsf{Enc}}, r_{\mathsf{Dec}} \xleftarrow{\$} \mathcal{R}_{\mathsf{Dec}}$*. Next,*

- $\mathsf{Gen}(1^\lambda, \mathsf{mode}; r_{\mathsf{Gen}})$*: Takes a security parameter* $\lambda \in \mathbb{N}$ *and a* $\mathsf{mode} \in \{\mathsf{inj}, \mathsf{lossy}\}$ *and outputs keys* $(\mathsf{pk}, \mathsf{sk})$*. Additionally, let* $\mathsf{GenPK}(1^\lambda, \mathsf{mode}; r_{\mathsf{Gen}})$ *compute* $\mathsf{Gen}(1^\lambda, \mathsf{mode}; r_{\mathsf{Gen}})$ *but only output* pk*.*
- $\mathsf{Enc}(\mathsf{pk}, m; r_{\mathsf{Enc}})$*: Takes a public key* pk *and a message* $m \in \mathcal{M}$ *and outputs a ciphertext* c*.*
- $\mathsf{Dec}(\mathsf{sk}, c; r_{\mathsf{Dec}})$*: Takes a secret key* sk *and a ciphertext* c*, decrypts* c*, and outputs the message* m*.*

We will often omit the random inputs in our notation, for instance, writing $\mathsf{Enc}(\mathsf{pk}, m)$ *instead of* $\mathsf{Enc}(\mathsf{pk}, m; r_{\mathsf{Enc}})$*.*

Next, the encryption scheme satisfies the following properties:

- ***Correctness:*** *With overwhelming probability over the sampling of* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{inj})$*, the following is true for every* $m \in \mathcal{M}$*:*

$$\Pr_{r_{\mathsf{Enc}}, r_{\mathsf{Dec}}} [m = \mathsf{Dec}[\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)]] = 1$$

- ***Statistical/Perfect Lossiness:*** *There is a function* $\mu(\lambda)$ *that is (respectively) negligible/identically zero such that with overwhelming probability over the sampling of* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{lossy})$*, the following is true for every* $m_0, m_1 \in \mathcal{M}$*: the statistical distance between* $\mathsf{Enc}(\mathsf{pk}, m_0)$ *and* $\mathsf{Enc}(\mathsf{pk}, m_1)$ *is* $\leq \mu(\lambda)$*.*
- ***Indistinguishability of Modes:*** *The output distributions of* $\mathsf{GenPK}(1^\lambda, \mathsf{inj})$ *and* $\mathsf{GenPK}(1^\lambda, \mathsf{lossy})$ *are indistinguishable to any QPT adversary. Formally, we require that for any QPT adversary* $\mathcal{A}$*, there is a negligible function* $\nu(\lambda)$ *such that:*

$$\left| \Pr\left[\mathcal{A}(1^\lambda, \mathsf{GenPK}(1^\lambda, \mathsf{inj})) \to 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathsf{GenPK}(1^\lambda, \mathsf{lossy})) \to 1\right] \right| \leq \nu(\lambda)$$

A.1 Hard Zero-Knowledge Problems from Lossy Encryption

The standard implication from statistically lossy encryption to hard problems in SZK is recalled in [BDRV18, Appendix A.1 of the full version]. Perfect lossiness gives the corresponding implication for public-coin honest-verifier perfect zero knowledge.

Lemma A.2. *If statistically lossy PKE exists, then* $\mathsf{SZK} \not\subseteq \mathsf{BQP}$. *If perfectly lossy PKE exists, then* $\mathsf{HVPZK}|_{\mathrm{am}} \not\subseteq \mathsf{BQP}$. *In both cases, the message space is assumed to contain two distinct efficiently specifiable messages.*

Proof. Given a public key pk, construct polynomial-size sampling circuits $Q_b(r) = \mathsf{Enc}(\mathsf{pk}, m_b; r)$ for two distinct messages m_0, m_1. For all but a negligible fraction of injective keys, correctness implies that Q_0 and Q_1 have disjoint supports. For all but a negligible fraction of lossy keys, their distributions are statistically close, or identical in the perfectly lossy case. Distinguishing statistically close distributions from distributions with disjoint supports is a promise problem in SZK. The promise problem with identical distributions as YES instances and disjoint supports as NO instances has a public-coin honest-verifier perfect zero-knowledge proof [Vad99, Proposition 6.2.6]; its simulator samples the public challenge and the selected circuit's random input, so runs in strict polynomial time. A BQP algorithm for the corresponding promise problem would distinguish the two key-generation modes, contradicting Definition A.1. □

A.2 Statistically Lossy Encryption from LWE

Statistically lossy PKE with ℓ-bit messages follows from (n, k)-lossy trapdoor functions whenever $k - \ell = \omega(\log \lambda)$ [BHY09, Section 4.2]. The construction preserves perfect correctness on good injective keys and applies to almost-always lossy trapdoor functions. Its key-indistinguishability reduction also applies to QPT adversaries. Combining it with [PW08, Theorem 6.4 of the full version] gives the following standard consequence.

Theorem A.3 ([BHY09, PW08])**.** *Let* $q = 4\lambda^{10}$, $\alpha = 1/(32\lambda^{10})$, *and* $\chi = \overline{\Psi}_\alpha$. *Assuming the post-quantum hardness of* $\mathsf{LWE}_{q,\chi}$, *there exists a statistically lossy PKE scheme (Definition A.1) with message space* $\mathcal{M} = \{0,1\}^\lambda$.

Indeed, the parameters $d = \lambda$, $c_1 = 4$, and $c_2 = c_3 = 2$ in [PW08, Theorem 6.4 of the full version] give input length $n = \lambda^2$ and lossiness $k \geq n/2 - o(n)$, sufficient for the standard encryption construction.

A.3 Construction from Group Actions

In this section, we describe a perfectly lossy PKE assuming the post-quantum weak pseudorandomness of effective group actions.

Notation. For a regular and abelian group action $\star : \mathbb{G} \times \mathbb{X} \to \mathbb{X}$, we use additive notation to denote the group operation in $\mathbb{G}$.

Definition A.4 (Effective Group Action)**.** *A regular and abelian group action* $(\mathbb{G}, \mathbb{X}, \star)$ *is effective if it satisfies the following properties.*

1. *The group* $\mathbb{G}$ *is finite and there exist efficient p.p.t. algorithms for membership testing (deciding whether a binary string represents a group element), equality testing, group operation and computing the inverse of any element. We require exact uniform sampling in* $\mathbb{G}$ *in strict polynomial time.*

2. *The set $\mathbb{X}$ is finite and there exist efficient algorithms for membership testing (to check if a binary string represents a valid set element), and unique representation.*

3. *There exists a distinguished element $x_0 \in \mathbb{X}$ with known representation.*

4. *There exists an efficient algorithm that given any $g \in \mathbb{G}$ and any $x \in \mathbb{X}$, outputs $g \star x$.*

Assumption A.5 (Post-Quantum Weak Pseudorandomness of an Effective Group Action, strengthening [AFMP20])**.** *Suppose $(\mathbb{G}, \mathbb{X}, \star)$ is a family of effective group actions indexed by λ, with $|\mathbb{G}|^{-1} = \mathsf{negl}(\lambda)$. For every polynomial $t = t(\lambda)$, the distributions $\{(x_i, g \star x_i)\}_{i=1}^t$ and $\{(x_i, y_i)\}_{i=1}^t$ are computationally indistinguishable to QPT adversaries, where $g \leftarrow \mathbb{G}$ and all $x_i, y_i \leftarrow \mathbb{X}$ are sampled independently and uniformly at random.*

Theorem A.6. *Suppose there exists an effective group action for which Assumption A.5 holds. Then, there exists a perfectly lossy public key encryption scheme (Definition A.1) with message space $\mathcal{M} = \{0,1\}^\lambda$.*

Construction A.7. *Let $(\mathbb{G}, \mathbb{X}, \star)$ be a group action for which Assumption A.5 holds. Let $x_* \in \mathbb{X}$ be a distinguished point that is specified as a public parameter. We first describe encryption of one bit.*

- $\mathsf{Gen}(1^\lambda, \mathsf{mode})$*: Sample $v \leftarrow \mathbb{G}$ and set $x_0 = v \star x_*$, which is uniform in $\mathbb{X}$ by regularity.*
  - *If* $\mathsf{mode} = \mathsf{lossy}$*, then sample random $g, h \leftarrow \mathbb{G}$ and set* $\mathsf{pk} := (x_0, g \star x_0, h \star x_0, (g+h) \star x_0)$ *and* $\mathsf{sk} = \bot$*.*
  - *If* $\mathsf{mode} = \mathsf{inj}$*, then sample independent random $g, h, k \leftarrow \mathbb{G}$. Set* $\mathsf{pk} = (x_0, g \star x_0, h \star x_0, k \star x_0)$ *and* $\mathsf{sk} = (g, k - h)$*.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*:*
  - *Sample a random $u \leftarrow \mathbb{G}$. Parse* $\mathsf{pk} = (x_0, x_1, x_2, x_3)$*.*
  - *If $m = 0$, output $c = (u \star x_0, u \star x_1)$.*
  - *If $m = 1$, output $c = (u \star x_2, u \star x_3)$.*
- $\mathsf{Dec}(\mathsf{sk}, c)$*:*
  - *Parse injective secret key as* $\mathsf{sk} = (g, g')$ *and ciphertext $c = (y_1, y_2)$.*
  - *If $g \star y_1 = y_2$ then output $m = 0$.*
  - *Else if $g' \star y_1 = y_2$ then output $m = 1$.*
  - *Otherwise, output $\bot$.*

To encrypt a message in $\{0,1\}^\lambda$, encrypt each bit independently under the same public key and decrypt componentwise.

Proof. We will prove that the scheme described in Construction A.7 is a perfectly lossy PKE scheme, assuming the weak pseudorandomness assumption of the group action.

Correctness. Except when $k = g + h$, an event of probability $1/|\mathbb{G}| = \mathsf{negl}(\lambda)$, this encryption scheme satisfies perfect decryption correctness on injective keys. In the injective mode, an encryption of 0 looks like

$$(u \star x_0, u \star x_1) = (u \star x_0, (u + g) \star x_0),$$

for a random $u \leftarrow \mathbb{G}$, while an encryption of 1 looks like

$$(u \star x_2, u \star x_3) = ((u + h) \star x_0, (u + k) \star x_0).$$

In order for there to be a collision between encryptions of 0 and 1, we would need some w, v satisfying

$$w \star x_0 = (v + h) \star x_0 \quad \text{and} \quad (w + g) \star x_0 = (v + k) \star x_0.$$

Because we are working with a regular group action, this implies $w = v + h$ and $w + g = v + k$, hence $g + h = k$. Therefore, on every key with $k \neq g + h$, decryption is perfectly correct, as required by Definition A.1.

Perfect Lossiness. In the lossy mode, note that an encryption of 1 takes the form

$$\begin{aligned}(u \star x_2, u \star x_3) &= ((u + h) \star x_0,\ (u + g + h) \star x_0) \\ &= (u' \star x_0, (u' + g) \star x_0) \\ &= (u' \star x_0, u' \star x_1).\end{aligned}$$

Here $u' = u + h$. Since u is sampled uniformly at random, so is u', and thus the distribution of an encryption of 1 is identical to that of an encryption of 0. Hence, encryptions of 0 and 1 are identically distributed, and the scheme is perfectly lossy.

Indistinguishability of modes. Apply Assumption A.5 to two samples and order the resulting tuple as $(x_0, g \star x_0, x_2, g \star x_2)$. Since x_0, x_2 are independent and uniform, regularity lets us write $x_2 = h \star x_0$ for a uniform $h \in \mathbb{G}$ independent of x_0, g. Thus this is exactly a lossy public key. In the random case, all four entries are independent and uniform, which is exactly the injective public-key distribution, proving indistinguishability against QPT adversaries.

Finally, encrypting each bit independently preserves perfect correctness on good injective keys and perfect lossiness for every message in $\{0, 1\}^\lambda$, while the public-key distributions remain unchanged. □

B Classical Single Effective Query (CSEQ)

B.1 Classical Single Effective Query (CSEQ) Model

In this section, we recall the *single effective query* security definition from [GLR+25] and, to distinguish it from our generalized notion, refer to this classical variant as CSEQ.

Definition B.1 (The Single Effective Query Oracle)**.** *For a randomized function $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$, we define the classical single effective query oracle O_f^{CSEQ} as implementing the following algorithm:*

- *We assume it has oracle access to O_f, which acts as*

$$|x, r, u\rangle \mapsto |x, r, u \oplus f(x; r)\rangle\,.$$

- *The oracle maintains two internal registers: a workspace register $\mathcal{W}$ to perform intermediate computations, and a database register $\mathcal{D}$ to implement the database for the compressed random oracle. These registers are initialized to $|0\rangle_{\mathcal{W}} \otimes |\emptyset\rangle_{\mathcal{D}}$.*

- *To describe its behavior on queries, all we need to do is describe its behavior on basis states. We will maintain the invariant that the workspace register is always $|0\rangle_{\mathcal{W}}$. On query $|x, u, b\rangle_{\mathcal{Q}}$ on query register $\mathcal{Q}$, the oracle implements an isometry specified by the following steps on each basis state $|x, u, b\rangle_{\mathcal{Q}} \otimes |D\rangle_{\mathcal{D}} \otimes |0\rangle_{\mathcal{W}}$:*

  1. *If there exists some $x' \neq x$ such that $(x', r) \in D$ for some $r \in \mathcal{R}$, skip the following steps.*

2. *Copy x into the workspace register*

$$|x,u,b\rangle_{\mathcal{Q}} \otimes |D\rangle_{\mathcal{D}} \otimes |0\rangle_{\mathcal{W}} \mapsto |x,u,b\rangle_{\mathcal{Q}} \otimes |D\rangle_{\mathcal{D}} \otimes |x,0\rangle_{\mathcal{W}}\,.$$

3. *Apply a compressed random oracle query isometry with query register $\mathcal{W}$ and database register $\mathcal{D}$.*
4. *On the basis states apply the following unitary map that acts on registers $\mathcal{Q},\mathcal{W}$:*

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

5. *Uncompute the workspace register: first, query the compressed oracle again with the query register being $\mathcal{W}$ and the database register being $\mathcal{D}$, and then copy the input x.*

Definition B.2 (CSEQ-based simulation security for one-time programs)**.** *A one-time program compiler* OTP *satisfies* CSEQ*-based simulation security for a class $\mathcal{F}$ of randomized functions if there exists a q.p.t. simulator* Sim *such that for every $f\in\mathcal{F}$ and for every q.p.t. distinguisher D, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr\left[1\leftarrow D(1^\lambda,f,\mathsf{OTP}(1^\lambda,f))\right]-\Pr\left[1\leftarrow D(1^\lambda,f,\mathsf{Sim}^{O_f^{\mathsf{CSEQ}}}(1^\lambda))\right]\right|\leq \mathsf{negl}(\lambda).$$

By giving f as input to the distinguishing algorithm, we mean that D gets access to f in a way that it can evaluate on any (x,r) of its choice. Note that this is actually redundant, because the order of quantifiers is such that D is allowed to depend on the choice of f.

B.2 When is CSEQ a special case of SEQ

We will show that for any classical functionality f with a domain size $|\mathcal{X}|>1$, there is a channel Φ_f such that the classical SEQ oracle O_f^{CSEQ} reveals the same information as the (channel) SEQ oracle $O_{\Phi_f}^{\mathsf{SEQ}}$.

The channel Φ_f computes f on any input x and uses a (compressed) random oracle H to sample r. Given a function $f:\mathcal{X}\times\mathcal{R}\to\mathcal{Y}$ and a random oracle $H:\mathcal{X}\to\mathcal{R}$, Φ_f maps $x\to f(x;H(x))$ and also flips a bit b to indicate that it has executed. Φ_f is essentially the same as the classical SEQ oracle O_f^{CSEQ}, except without step 1 of O_f^{CSEQ}.

Definition B.3 (Channel Φ_f)**.** *Given a randomized classical function $f:\mathcal{X}\times\mathcal{R}\to\mathcal{Y}$, let Φ_f be the following quantum channel.*

- *The channel's internal register has two parts $\mathcal{P}=\mathcal{D}\times\mathcal{W}$. $\mathcal{D}$ is a database register for the compressed random oracle, and $\mathcal{W}$ is a workspace register to perform intermediate computations. $\mathcal{D}$ is initialized to $|\emptyset\rangle_{\mathcal{D}}$, and we maintain the invariant that $\mathcal{W}$ is in the state $|0,0\rangle_{\mathcal{W}}$ at the start of each query.*
- *The channel's Stinespring unitary U_{Φ_f} acts as follows on each basis state $|x,u,b\rangle_{\mathcal{Q}}\otimes|D\rangle_{\mathcal{D}}\otimes|0,0\rangle_{\mathcal{W}}$:*
  1. *Copy x into the workspace register:*

$$|x,u,b\rangle_{\mathcal{Q}} \otimes |D\rangle_{\mathcal{D}} \otimes |0,0\rangle_{\mathcal{W}} \mapsto |x,u,b\rangle_{\mathcal{Q}} \otimes |D\rangle_{\mathcal{D}} \otimes |x,0\rangle_{\mathcal{W}}$$

  2. *Apply the query operation of the compressed oracle with $\mathcal{W}$ as the query register and $\mathcal{D}$ as the database register.*

3. *Apply the unitary that acts as follows on the basis states:*

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

4. *Uncompute step 2 and then step 1.*

Note that f and Φ_f provide no one-time guarantee. They never reject queries, so the user may make many effective queries. There are two ways to limit the user's queries: we can implement f with the classical SEQ oracle O_f^{CSEQ} and implement Φ_f with the SEQ oracle $O_{\Phi_f}^{\mathsf{SEQ}}$. The purpose of both oracles is to restrict the user to just one effective query.

Which oracle reveals more information about f? In fact, they reveal the same information. We formalize this by showing that O_f^{CSEQ} can be used to simulate $O_{\Phi_f}^{\mathsf{SEQ}}$, and vice versa as long as $|\mathcal{X}| > 1$ (theorem B.19).

To prove the first simulation indistinguishability in theorem B.19, we first define the CSEQ-wrapper simulator Sim'.

The simulator Sim' below has the following internal registers. Let $\mathcal{P}$ store the program state of P, which is initialized to $|\psi\rangle$. Let $\mathcal{D}$ be an internal database register that stores a query $x \in \mathcal{X} \cup \{\emptyset\}$ and is initialized to $|\emptyset\rangle$. Let $\mathcal{W}_x \times \mathcal{W}_y \times \mathcal{W}_b$ be work registers that store values $(x,y,b) \in (\mathcal{X} \cup \{\emptyset\}) \times (\mathcal{Y} \cup \{\emptyset\}) \times \{0,1\}$. $\mathcal{W}_x \times \mathcal{W}_y \times \mathcal{W}_b$ are initialized to $|\emptyset,\emptyset,0\rangle$.

We represent elements of $\mathcal{X} \cup \{\emptyset\}$ as the following binary strings. Each $x \in \mathcal{X}$ becomes $x\|1$ and $\emptyset$ becomes $0^n\|0$. In particular, $x = 0^n$ is represented as $0^n\|1$, which is different from the representation $0^n\|0$ of $\emptyset$. Therefore, we have that $\emptyset + x = x$, and $x + x = \emptyset$. The same kind of representation is used for $\mathcal{Y} \cup \{\emptyset\}$.

$\mathsf{Sim}'(P)$ works as follows. First, it uses the EvalAndCopy operation to evaluate $P(x)$. Rather than evaluating the program on registers $\mathcal{Q}_x \times \mathcal{Q}_u$ directly, EvalAndCopy evaluates it on work registers $\mathcal{W}_x \times \mathcal{W}_y$ instead, and copies the inputs and outputs between $\mathcal{Q}_x \times \mathcal{Q}_u$ and $\mathcal{W}_x \times \mathcal{W}_y$. This provides insulation from any pathological behavior of Eval. Even if Eval modifies or reads from its query register arbitrarily, we are guaranteed that EvalAndCopy only reads from $\mathcal{Q}_x$ and only writes to $\mathcal{Q}_y$.

Next, $\mathsf{Sim}'(P)$ stores a query x' in the database $\mathcal{D}$ and rejects any new query x if $x' \notin \{x,\emptyset\}$. This is analogous O_f^{CSEQ}'s database. The ReflectAndCopy operation updates $\mathsf{Sim}'(P)$'s database by copying x from $\mathcal{Q}_x$ to $\mathcal{D}$ if and only if the remaining internal registers $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y \times \mathcal{W}_b$ are in their initial state (lemma B.5).

Definition B.4 (Simulator Sim' for O_f^{CSEQ})**.** *The simulator also applies to testable oracle interfaces, using their evaluation and coherent-test operations directly. All subsequent simulation lemmas in this appendix cover both forms of access.*

- Sim' *takes as input a testable quantum program* $P = (|\psi\rangle, \mathsf{Eval}, R)$.
- *Initialize the internal state to* $|\psi\rangle_{\mathcal{P}} \otimes |\emptyset\rangle_{\mathcal{D}} \otimes |\emptyset\rangle_{\mathcal{W}_x} \otimes |\emptyset\rangle_{\mathcal{W}_y} \otimes |0\rangle_{\mathcal{W}_b}$.
- *Given a query* $|x,u,b\rangle_{\mathcal{Q}}$ *and database* $|x'\rangle_{\mathcal{D}}$, Sim' *acts as follows.*
  1. *If* $x' \notin \{x,\emptyset\}$*, do nothing and ignore the remaining steps.*
  2. ReflectAndCopy*: If* $\mathcal{W}_x \times \mathcal{W}_y$ *contains* $(\emptyset,\emptyset)$*, then do the following:*
     (a) *Apply the reflection oracle* R *to registers* $(\mathcal{W}_b, \mathcal{P})$.
     (b) *If* $\mathcal{W}_b$ *contains* 1*, then* CNOT *the value* x *from* $\mathcal{Q}_x$ *onto* $\mathcal{D}$.
     (c) *Apply the reflection oracle* R *to registers* $(\mathcal{W}_b, \mathcal{P})$.
  3. EvalAndCopy*:*
     (a) CNOT *the value* x *from* $\mathcal{Q}_x$ *to* $\mathcal{W}_x$.

(b) *Apply* Eval *to registers* $(\mathcal{W}_x, \mathcal{W}_y, \mathcal{P})$.
(c) CopyOutput: CNOT *the contents of* $\mathcal{W}_y$ *onto* $\mathcal{Q}_u$.
(d) *Apply* $\mathsf{Eval}^\dagger$ *to registers* $(\mathcal{W}_x, \mathcal{W}_y, \mathcal{P})$.
(e) CNOT *the value* x *from* $\mathcal{Q}_x$ *to* $\mathcal{W}_x$.

4. *Apply* ReflectAndCopy *(step 2) again.*
5. Bitflip: *Apply* X *to the* $\mathcal{Q}_b$ *register.*

In summary, the simulator acts as follows: it acts as the identity on states $|x, u, b\rangle |D\rangle |\cdots\rangle$ if $x' \in D$ for some $x' \neq x$, and on the rest of the state space it acts as the unitary

$$\mathsf{Bitflip} \circ \mathsf{ReflectAndCopy} \circ \mathsf{EvalAndCopy} \circ \mathsf{ReflectAndCopy}.$$

The next lemma says that ReflectAndCopy acts similarly to a reflection oracle on registers $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y \times \mathcal{W}_b$. It CNOTS x from $\mathcal{Q}_x$ to $\mathcal{D}$ if and only if $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y \times \mathcal{W}_b$ is in its initial state.

Lemma B.5. *If* P *is a testable quantum program, then* ReflectAndCopy *(step 2 of* $\mathsf{Sim}'(P)$*) is equivalent to the operation that applies* CNOT *from* $\mathcal{Q}_x$ *to* $\mathcal{D}$ *controlled on* $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y \times \mathcal{W}_b$ *being in their initial state* $|\psi, \emptyset, \emptyset, 0\rangle$.

Additionally, after any number of queries to $\mathsf{Sim}'(P)$, $\mathcal{W}_b$ *is in state* $|0\rangle$.

Proof. First, $\mathcal{W}_b$ is initialized to $|0\rangle$. Next, note that $\mathcal{W}_b$ is only modified during the ReflectAndCopy operation. Let us assume that at the start of a given ReflectAndCopy operation, $\mathcal{W}_b$ contains $|0\rangle$. We will show that at the end of this operation, $\mathcal{W}_b$ still contains $|0\rangle$. This establishes that $\mathcal{W}_b = |0\rangle$ after every query to $\mathsf{Sim}'(P)$.

Let us consider how ReflectAndCopy acts on the following basis states.

Case 1: $\mathcal{W}_x \times \mathcal{W}_y$ contains a state orthogonal to $|\emptyset, \emptyset\rangle$. Then ReflectAndCopy acts as the identity, so at the end of ReflectAndCopy, $\mathcal{W}_b = |0\rangle$, and $\mathcal{Q}_x \times \mathcal{D} \times \mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ are unchanged.

Case 2: $\mathcal{W}_x \times \mathcal{W}_y = |\emptyset, \emptyset\rangle$, but $\mathcal{P}$ contains a state that is orthogonal to $|\psi\rangle$. ReflectAndCopy first applies R to $(\mathcal{W}_b, \mathcal{P})$, resulting in $\mathcal{W}_b = |0\rangle$. Then the CNOT operation from $\mathcal{Q}_x$ to $\mathcal{D}$ is controlled on 0, so we do not copy x to $\mathcal{D}$, and we can replace this operation with the identity. Finally, the second application of R also leaves the $\mathcal{W}_b = |0\rangle$. At the end of ReflectAndCopy, $\mathcal{W}_b = |0\rangle$, and $\mathcal{Q}_x \times \mathcal{D} \times \mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ are unchanged.

Case 3: $\mathcal{W}_x \times \mathcal{W}_y = |\emptyset, \emptyset\rangle$, and $\mathcal{P} = |\psi\rangle$. ReflectAndCopy first applies R to $(\mathcal{W}_b, \mathcal{P})$, resulting in $\mathcal{W}_b = |1\rangle$. Then, the CNOT operation copies x from $\mathcal{Q}_x$ to $\mathcal{D}$. Since the CNOT step acts only on $\mathcal{W}_b \times \mathcal{Q}_x \times \mathcal{D}$, this does not affect the $\mathcal{P}$ register, which will still be in the state $|\psi\rangle$. Finally, the second application of the reflection oracle R will XOR the $\mathcal{W}_b$ register with 1, returning it to $|0\rangle$. At the end of ReflectAndCopy, $\mathcal{W}_b = |0\rangle$, and $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ are unchanged.

In summary, we've shown that for each type of basis state listed above, ReflectAndCopy keeps $\mathcal{W}_b = |0\rangle$, and it performs a CNOT from $\mathcal{Q}_x \to \mathcal{D}$ controlled on $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y = |\psi, \emptyset, \emptyset\rangle$. Any state on registers $\mathcal{Q}_x \times \mathcal{D} \times \mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ is in the span of the basis states above, so we have completely characterized the behvaior of ReflectAndCopy. □

Next, let us define a testable quantum program $P_f = (|+\rangle, \mathsf{Eval}_f, R_+)$ as follows:

- $|+\rangle = \frac{1}{\sqrt{|\mathcal{R}|}} \cdot \sum_{r \in \mathcal{R}} |r\rangle$
- Eval_f is a unitary that maps $|x, u\rangle \otimes |r\rangle \mapsto |x, u \oplus f(x; r)\rangle \otimes |r\rangle$
- R_+ is the reflection about $|+\rangle$

Note that P_f is a testable quantum program that implements f with 0 error.

Lemma B.6. *Oracles for* $\mathsf{Sim}'(P_f)$ *and* O_f^{CSEQ} *are perfectly indistinguishable to any adversary making an unbounded number of quantum queries.*

Proof. The following hybrids transform O_f^{CSEQ} into $\mathsf{Sim}'(P_f)$.

Hybrid 0. This is the CSEQ oracle, except that the size of the compressed oracle database is fixed to record at most 1 query. This is equivalent to the CSEQ oracle because the CSEQ oracle holds at most one entry in the database at any point in time, by Claim 4.7 of [GLR+25].

Recall that StdDecomp applied to $\mathcal{W} \times \mathcal{D}$ swaps the states $|x, r\rangle_{\mathcal{W}} \otimes |\emptyset, \emptyset\rangle_{\mathcal{D}}$ and $|x, r\rangle_{\mathcal{W}} \otimes |x, +\rangle_{\mathcal{D}}$, and acts as the identity on the rest of the space. Additionally, CStO′ applied to $\mathcal{W} \times \mathcal{D}$ maps $|x, r\rangle_{\mathcal{W}} \otimes |D\rangle_{\mathcal{D}} \mapsto |x, r \oplus D(x)\rangle \otimes |D\rangle$.

The oracle initializes its internal registers $\mathcal{W} \times \mathcal{D} = |0\rangle \times |\emptyset\rangle$. On query $|x, u, b\rangle_{\mathcal{Q}}$, the oracle acts as follows:

1. If there exists some $x' \neq x$ such that $(x', r) \in D$ for some $r \in \mathcal{R}$, skip the following steps.
2. Copy x into the workspace register
$$|x\rangle_{\mathcal{Q}_x} \otimes |0\rangle_{\mathcal{W}} \mapsto |x\rangle_{\mathcal{Q}_x} \otimes |x, 0\rangle_{\mathcal{W}}.$$
3. Apply the compressed random oracle to registers $\mathcal{W} \times \mathcal{D}$ by executing the following unitaries:
$$\mathsf{StdDecomp} \circ \mathsf{CStO}' \circ \mathsf{StdDecomp}$$
4. Apply the following unitary map:
$$|u, b\rangle_{\mathcal{Q}_u \times \mathcal{Q}_b} \otimes |x, r\rangle_{\mathcal{W}} \mapsto |u \oplus f(x; r), b \oplus 1\rangle_{\mathcal{Q}_u \times \mathcal{Q}_b} \otimes |x, r\rangle_{\mathcal{W}}$$
5. Query the compressed oracle again:
$$\mathsf{StdDecomp} \circ \mathsf{CStO}' \circ \mathsf{StdDecomp}$$
6. Copy x into the workspace register
$$|x\rangle_{\mathcal{Q}_x} \otimes |x, 0\rangle_{\mathcal{W}} \mapsto |x\rangle_{\mathcal{Q}_x} \otimes |0\rangle_{\mathcal{W}}$$

Hybrid 1. We've canceled out operations with their inverse whereever possible. We have also moved the bitflip on $\mathcal{Q}_b$ to the final step.

The oracle initializes its internal registers $\mathcal{W} \times \mathcal{D} = |0\rangle \times |\emptyset\rangle$. On query $|x, u, b\rangle_{\mathcal{Q}}$, the oracle acts as follows:

1. If there exists some $x' \neq x$ such that $(x', r) \in D$ for some $r \in \mathcal{R}$, skip the following steps.
2. Apply StdDecomp to registers $\mathcal{Q}_x \times \mathcal{D}$.
3. Apply the following unitary map:
$$|x, u, b\rangle_{\mathcal{Q}} \otimes |x, r\rangle_{\mathcal{D}} \mapsto |x, u \oplus f(x; r), b\rangle_{\mathcal{Q}} \otimes |x, r\rangle_{\mathcal{D}}$$
4. Apply StdDecomp to registers $\mathcal{Q}_x \times \mathcal{D}$.
5. Apply X (bitflip) to $\mathcal{Q}_b$.

Claim B.7. *The oracles in hybrids 0 and 1 are perfectly indistinguishable after any number of queries.*

Proof. First, in hybrid 0, StdDecomp applies $\mathsf{StdDecomp}_x$ to $\mathcal{D}$ controlled on the value of x written on $\mathcal{W}$. In hybrid 1, StdDecomp applies $\mathsf{StdDecomp}_x$ to $\mathcal{D}$ controlled on the value of x written on $\mathcal{Q}_x$. These operations are equivalent because during any query in hybrid 0, the value of x written on $\mathcal{Q}_x$ matches the value written on $\mathcal{W}$ whenever StdDecomp is called.

Let us modify hybrid 0 so that StdDecomp is controlled on $\mathcal{Q}_x$ instead of $\mathcal{W}$.

Second, StdDecomp now commutes with items 2, 4 and 6 of hybrid 0. This is because StdDecomp applies $\mathsf{StdDecomp}_x$ to $\mathcal{D}$ controlled on the computational-basis value of $\mathcal{Q}_x$. Step 2 applies a unitary to $\mathcal{W}$ controlled on the computational-basis value of $\mathcal{Q}_x$. StdDecomp and step 2 commute because they are controlled by the computational-basis value of the same register $\mathcal{Q}_x$, and they apply a controlled unitary to disjoint registers, $\mathcal{D}$ and $\mathcal{W}$ respectively. The same reasoning shows that StdDecomp commutes with step 6. Step 4 does not act on $\mathcal{Q}_x$ or D, so it commutes with StdDecomp because they act on disjoint registers.

Let us switch the order of step 2 and the first StdDecomp in step 3. Let us also switch the order of step 6 and the last StdDecomp in step 5.

Third, the final application of StdDecomp in step 3 cancels out with the first application of StdDecomp in step 5. This is because $\mathsf{StdDecomp}^{-1} = \mathsf{StdDecomp}$, and the only operation that occurs in between these two applications of StdDecomp is step 4, which commutes with StdDecomp.

Fourth, after making the changes above, we have that hybrid 0 is equivalent to the following operations:

1. If there exists some $x' \neq x$ such that $(x', r) \in D$ for some $r \in \mathcal{R}$, skip the following steps.
2. Apply StdDecomp to $\mathcal{Q}_x \times \mathcal{D}$.
3. Step 2: $|x\rangle_{\mathcal{Q}_x} \otimes |0\rangle_{\mathcal{W}} \to |x\rangle_{\mathcal{Q}_x} \otimes |x, 0\rangle_{\mathcal{W}}$
4. CStO': $|x, 0\rangle_{\mathcal{W}} \otimes |x, r\rangle_{\mathcal{D}} \to |x, r\rangle_{\mathcal{W}} \otimes |x, r\rangle_{\mathcal{D}}$
5. Step 4: $|u, b\rangle_{\mathcal{Q}_u \times \mathcal{Q}_b} \otimes |x, r\rangle_{\mathcal{W}} \mapsto |u \oplus f(x; r), b \oplus 1\rangle_{\mathcal{Q}_u \times \mathcal{Q}_b} \otimes |x, r\rangle_{\mathcal{W}}$
6. CStO': $|x, r\rangle_{\mathcal{W}} \otimes |x, r\rangle_{\mathcal{D}} \to |x, 0\rangle_{\mathcal{W}} \otimes |x, r\rangle_{\mathcal{D}}$
7. Step 6: $|x\rangle_{\mathcal{Q}_x} \otimes |x, 0\rangle_{\mathcal{W}} \to |x\rangle_{\mathcal{Q}_x} \otimes |0\rangle_{\mathcal{W}}$
8. Apply StdDecomp to $\mathcal{Q}_x \times \mathcal{D}$.

Note that the middle steps, Step 6 $\circ$ CStO' $\circ$ Step 4 $\circ$ CStO' $\circ$ Step 2, are all classical operations, and they are equivalent to the following operation:

$$|x, u, b\rangle_{\mathcal{Q}} \otimes |0\rangle_{\mathcal{W}} \otimes |x, r\rangle_{\mathcal{D}} \to |x, u \oplus f(x; r), b \oplus 1\rangle_{\mathcal{Q}} \otimes |0\rangle_{\mathcal{W}} \otimes |x, r\rangle_{\mathcal{D}}$$

Note that $\mathcal{W}$ is returned to the $|0\rangle$ state by the end of this sequence of operations, so we can ignore it.

Fifth, let us delay applying the bitflip to $\mathcal{Q}_b$ until after all the other steps have finished. This is equivalent to the steps above because the bitflip commutes with all other steps.

Now we have that hybrid 0 is equivalent to the following steps:

1. If there exists some $x' \neq x$ such that $(x', r) \in D$ for some $r \in \mathcal{R}$, skip the following steps.
2. Apply StdDecomp to $\mathcal{Q}_x \times \mathcal{D}$.
3. Apply the following unitary map:
$$|x, u, b\rangle_{\mathcal{Q}} \otimes |x, r\rangle_{\mathcal{D}} \to |x, u \oplus f(x; r), b\rangle_{\mathcal{Q}} \otimes |x, r\rangle_{\mathcal{D}}$$
4. Apply StdDecomp to $\mathcal{Q}_x \times \mathcal{D}$.
5. Apply X (bitflip) to $\mathcal{Q}_b$.

This is exactly hybrid 1, so we've shown that hybrids 0 and 1 are perfectly indistinguishable. □

Hybrid 2. Let us split the database register $\mathcal{D}$ into two registers, called $\mathcal{D} \times \mathcal{P}$ to match the eventual notation of $\mathsf{Sim}'(P_f)$. $\mathcal{D}$ will store the database's x-value, and $\mathcal{P}$ will store its r-value.

Let Record be the unitary that swaps the states $|x\rangle_{\mathcal{Q}_x} \otimes |\emptyset\rangle_{\mathcal{D}} |+\rangle_{\mathcal{P}}$ and $|x\rangle_{\mathcal{Q}_x} \otimes |x\rangle_{\mathcal{D}} |+\rangle_{\mathcal{P}}$, and acts as the identity on the rest of the space. Additionally, let Decomp be the unitary that swaps $|\emptyset\rangle_{\mathcal{P}}$ and $|+\rangle_{\mathcal{P}}$.

In this hybrid, the oracle initializes database registers $\mathcal{D} \times \mathcal{P} = |\emptyset, \emptyset\rangle$ and work registers $\mathcal{W}_x \times \mathcal{W}_y \times \mathcal{W}_b = |\emptyset, \emptyset, 0\rangle$, although the oracle never operates on $\mathcal{W}_x \times \mathcal{W}_y \times \mathcal{W}_b$.

Next, the oracle answers queries on $|x, u, b\rangle_{\mathcal{Q}} \otimes |x'\rangle_{\mathcal{D}}$ as follows:

1. If $x' \notin \{x, \emptyset\}$, then skip the following steps.
2. Apply Record $\circ$ Decomp to the $(\mathcal{Q}_x, \mathcal{D}, \mathcal{P})$ registers.
3. Apply Eval_f on the $(\mathcal{Q}_x, \mathcal{Q}_u, \mathcal{P})$ registers:
$$|x, u\rangle_{\mathcal{Q}_x \times \mathcal{Q}_u} \otimes |r\rangle_{\mathcal{P}} \to |x, u \oplus f(x; r)\rangle_{\mathcal{Q}_x \times \mathcal{Q}_u} \otimes |r\rangle_{\mathcal{P}}$$
4. Apply Decomp $\circ$ Record on the $(\mathcal{Q}_x, \mathcal{D}, \mathcal{P})$ registers.
5. Apply X (bitflip) to $\mathcal{Q}_b$.

Claim B.8. *The oracles in hybrids 1 and 2 are perfectly indistinguishable after any number of quantum queries.*

Proof. We claim that the unitaries defined in hybrids 1 and 2 are identical. We argue this by choosing a suitable basis for states on registers $\mathcal{Q}_x \times \mathcal{D}$ (hybrid 1) or $\mathcal{Q} \times \mathcal{P} \times \mathcal{D}$ (hybrid 2) and then tracing how the unitaries StdDecomp, Eval_f, Record $\circ$ Decomp and Decomp $\circ$ Record transform these basis states.

The unitaries in hybrids 1 and 2 both act as the identity on basis states of the form $|x\rangle_{\mathcal{Q}_x} |x'\rangle_{\mathcal{D}} |\cdots\rangle$ for $x' \notin \{x, \emptyset\}$, so we will restrict our attention to states supported on $|x\rangle_{\mathcal{Q}_x} |x\rangle_{\mathcal{D}}$ and $|x\rangle_{\mathcal{Q}_x} |\emptyset\rangle_{\mathcal{D}}$. For the $\mathcal{P}$ register, our orthogonal "basis" will be $|\emptyset\rangle$, $|+\rangle$ and $|-\rangle$, where, in an abuse of notation, we write $|-\rangle$ to denote any state in the span of $\{\frac{1}{\sqrt{|\mathcal{R}|}} \cdot \sum_{r \in \mathcal{R}} (-1)^{r \cdot u} |r\rangle\}_{u \neq 0}$.

- The starting state of the first query is $|x\rangle_{\mathcal{Q}_x} |\emptyset\rangle_{\mathcal{D}} |\emptyset\rangle_{\mathcal{P}}$, and both StdDecomp and Record $\circ$ Decomp map it to $|x\rangle |x\rangle |+\rangle$.
- Eval_f maps $|x\rangle |x\rangle |+\rangle$ and $|x\rangle |x\rangle |-\rangle$ to superpositions of $|x\rangle |x\rangle |+\rangle$ and $|x\rangle |x\rangle |-\rangle$ (where $|-\rangle$ does not necessarily refer to the same state as before the application of Eval_f).
- Both StdDecomp and Decomp $\circ$ Record map these states as follows:
$$|x\rangle |x\rangle |+\rangle \mapsto |x\rangle |\emptyset\rangle |\emptyset\rangle \,, \text{ and}$$
$$|x\rangle |x\rangle |-\rangle \mapsto |x\rangle |x\rangle |-\rangle \,.$$
- Note that StdDecomp, Record$\circ$Decomp, and Decomp$\circ$Record each leave $|x\rangle |x\rangle |-\rangle$ invariant.

Let us define the set of *reachable* states to be states whose database is supported on eigenstates of the form: $|x, -\rangle, |\emptyset, \emptyset\rangle$. We claim that the database is in a reachable state at the start of any query. First, the state of the database at the start of the first query is reachable. Second, the observations above imply that if the database is in a reachable state at the start of a query, then it will end the query in a reachable state.

The only difference in the operations applied by hybrids 1 and 2 is that hybrid 1 uses StdDecomp twice, and hybrid 2 uses Record $\circ$ Decomp and Decomp $\circ$ Record instead.

Let us step through a query in hybrids 1 or 2 when the starting state is reachable. The first time that StdDecomp or Record$\circ$Decomp is called during the query, the database is supported on states of the form $|\emptyset,\emptyset\rangle, |x,-\rangle$. We've already argued that StdDecomp, Record$\circ$Decomp act equivalently on such states. After this operation, the database is supported on states of the form $|x,+\rangle, |x,-\rangle$. Then Eval_f maps these states to superpositions of $|x,+\rangle, |x,-\rangle$. Next, StdDecomp or Decomp$\circ$Record is called. We've argued that StdDecomp and Decomp$\circ$Record act equivalently on database states of the form $|x,+\rangle, |x,-\rangle$.

This shows that the unitaries in hybrids 1 and 2 act equivalently on any reachable state, so hybrids 1 and 2 are perfectly indistinguishable. □

Hybrid 3. This is $\mathsf{Sim}'(P_f)$.

In this hybrid, the oracle initializes internal registers $\mathcal{D}\times\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y\times\mathcal{W}_b = |\emptyset\rangle\otimes|+\rangle\otimes|\emptyset,\emptyset,0\rangle$, and answers queries on $|x,u,b\rangle\otimes|x'\rangle_{\mathcal{D}}$ as follows:

1. If $x'\notin\{x,\emptyset\}$, do nothing and ignore the remaining steps.
2. Apply ReflectAndCopy to the $(\mathcal{Q}_x,\mathcal{D},\mathcal{P},\mathcal{W}_x,\mathcal{W}_y,\mathcal{W}_b)$ registers.
3. Apply EvalAndCopy on the $(\mathcal{Q}_x,\mathcal{Q}_u,\mathcal{P},\mathcal{W}_x,\mathcal{W}_y)$ registers.
4. Apply ReflectAndCopy to the $(\mathcal{Q}_x,\mathcal{D},\mathcal{P},\mathcal{W}_x,\mathcal{W}_y,\mathcal{W}_b)$ registers.
5. Apply Bitflip to $\mathcal{Q}_b$.

Claim B.9. *The oracles in hybrids 2 and 3 are perfectly indistinguishable after any number of quantum queries.*

Proof. First, consider program P_f. For this program, Record is equivalent to ReflectAndCopy. The latter copies x from $\mathcal{Q}_x$ to $\mathcal{D}$ controlled on $\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y\times\mathcal{W}_b$ being in the initial state (lemma B.5). Record copies x from $\mathcal{Q}_x$ to $\mathcal{D}$ controlled on $\mathcal{P}$ being in its initial state $|+\rangle$. Furthermore, every time that Record is called in hybrid 2, the $\mathcal{W}_x\times\mathcal{W}_y\times\mathcal{W}_b$ registers are in their initial state because hybrid 2 does not operate on these registers after initializing them. Therefore, we can replace Record with ReflectAndCopy.

Second, Eval_f is equivalent to EvalAndCopy for program P_f. If $\mathcal{W}_x\times\mathcal{W}_y$ are in the state $|\emptyset,\emptyset\rangle$ at the start of EvalAndCopy, then EvalAndCopy acts as follows:

$$\begin{aligned}
&|x,u\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u}\otimes|\emptyset,\emptyset\rangle_{\mathcal{W}_x\times\mathcal{W}_y}\otimes|r\rangle_{\mathcal{P}}\\
&\to|x,u\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u}\otimes|x,\emptyset\rangle_{\mathcal{W}_x\times\mathcal{W}_y}\otimes|r\rangle_{\mathcal{P}}\\
&\to|x,u\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u}\otimes|x,f(x;r)\rangle_{\mathcal{W}_x\times\mathcal{W}_y}\otimes|r\rangle_{\mathcal{P}}\\
&\to|x,u\oplus f(x;r)\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u}\otimes|x,f(x;r)\rangle_{\mathcal{W}_x\times\mathcal{W}_y}\otimes|r\rangle_{\mathcal{P}}\\
&\to|x,u\oplus f(x;r)\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u}\otimes|x,\emptyset\rangle_{\mathcal{W}_x\times\mathcal{W}_y}\otimes|r\rangle_{\mathcal{P}}\\
&\to|x,u\oplus f(x;r)\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u}\otimes|\emptyset,\emptyset\rangle_{\mathcal{W}_x\times\mathcal{W}_y}\otimes|r\rangle_{\mathcal{P}}
\end{aligned}$$

We claim that $\mathcal{W}_x\times\mathcal{W}_y=|\emptyset,\emptyset\rangle$ at the start of every EvalAndCopy query. Before any queries have been made, $\mathcal{W}_x\times\mathcal{W}_y=|\emptyset,\emptyset\rangle$. Next, the computational-basis values of $\mathcal{W}_x\times\mathcal{W}_y$ are only modified during EvalAndCopy. We showed above that EvalAndCopy returns $\mathcal{W}_x\times\mathcal{W}_y$ to $|\emptyset,\emptyset\rangle$, so $\mathcal{W}_x\times\mathcal{W}_y=|\emptyset,\emptyset\rangle$ at the start of every EvalAndCopy query.

We also showed that EvalAndCopy is equivalent to

$$|x,u\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u}\otimes|r\rangle_{\mathcal{P}}\to|x,u\oplus f(x;r)\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u}\otimes|r\rangle_{\mathcal{P}}$$

which is the application of Eval_f to $\mathcal{Q}_x\times\mathcal{Q}_u\times\mathcal{P}$.

Third, let us remove all the Decomp operations from hybrid 2 as follows. The Decomp operation at the end of one query cancels with the Decomp operation at the start of the next one. This is because $\mathsf{Decomp} \circ \mathsf{Decomp} = I$, and Decomp operates only on the internal registers of the oracle.

Next, we can remove the first Decomp from the first query. This Decomp operation acts on $|\emptyset\rangle_{\mathcal{P}}$ to produce the state $|+\rangle_{\mathcal{P}}$. In hybrid 3, we remove this Decomp operation and instead have the oracle start with state $|+\rangle_{\mathcal{P}}$.

Finally, we can remove the final Decomp from the final query. Decomp acts only on the internal registers of the oracle, so the view of a distinguisher that has query access to the oracle is unchanged.

In summary, we have transformed hybrid 2 into hybrid 3 and shown that the two hybrids are equivalent. □

□

Lemma B.10. *If P_1 and P_2 are testable quantum programs that implement the same functionality with 0 error, then oracles for $\mathsf{Sim}'(P_1)$ and $\mathsf{Sim}'(P_2)$ are perfectly indistinguishable to any adversary making an unbounded number of quantum queries.*

The intuition for this proof is that given two program states $|\psi^1\rangle, |\psi^2\rangle$ that implement the same sampling functionality, for each x, we can decompose each program state into a superposition over an orthonormal basis of programs that give deterministic output values. Since the two programs implement the same functionality, these bases are equivalent up to a rotation/labeling.

This nice picture breaks down when you consider querying the program on multiple x values, but the database that Sim' uses to record queries allows us to get around this issue and still make the argument go through.

Proof. First, for any program P and any $(x, y) \in \mathcal{X} \times \mathcal{Y}$, let $\alpha_{x,y} = \sqrt{\Pr[P(x) \to y]}$.

In $\mathsf{Sim}'(P)$, the initial state of $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ is $|\psi, \emptyset, \emptyset\rangle$. The following claim says that we can decompose $|\psi, \emptyset, \emptyset\rangle$ into a basis defined by the y-values that the program outputs. For each y-value, $|\psi_{x,y}\rangle$ is the component of $|\psi, \emptyset, \emptyset\rangle$ that produces output y when EvalAndCopy is called.

Claim B.11. *For any $x \in \mathcal{X}$, there exists a set of orthonormal states $\{|\psi_{x,y}\rangle\}_{y \in \mathcal{Y}}$ on registers $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ such that for any $u \in \mathcal{Y} \cup \{\emptyset\}$ and any y in the support of $P(x)$,*

$$\mathsf{EvalAndCopy}\,|x, u\rangle_{\mathcal{Q}_x \times \mathcal{Q}_u} \otimes |\psi_{x,y}\rangle = |x, u + y\rangle_{\mathcal{Q}_x \times \mathcal{Q}_u} \otimes |\psi_{x,y}\rangle$$

Additionally, $|\psi, \emptyset, \emptyset\rangle$ is in the span of $\{|\psi_{x,y}\rangle\}_y$:

$$|\psi, \emptyset, \emptyset\rangle = \sum_{y \in \mathcal{Y}} \alpha_{x,y} |\psi_{x,y}\rangle$$

Proof. First, note that EvalAndCopy does not modify the computational-basis value of $\mathcal{Q}_x$. It just applies an operation that is controlled by $\mathcal{Q}_x$. When $\mathcal{Q}_x$ contains value x, the EvalAndCopy step acts as follows on registers $\mathcal{Q}_u \times \mathcal{W}_x \times \mathcal{W}_y \times \mathcal{P}$:

$$\mathsf{Eval}_x^\dagger \circ \mathsf{CopyOutput} \circ \mathsf{Eval}_x$$

where we define Eval_x to be the operation that CNOTs x onto register $\mathcal{W}_x$ and then applies Eval to $\mathcal{W}_x \times \mathcal{W}_y \times \mathcal{P}$.

Second, for each $x \in \mathcal{X}$ and each y in the support of $P(x)$, let $|\psi_{x,y}\rangle$ be the component of $|\psi, \emptyset, \emptyset\rangle$ that produces output y when the program is queried on x. Formally, for each $x \in \mathcal{X}$ and each y in the support of $P(x)$,

$$\text{let } |\psi_{x,y}\rangle = \frac{1}{\alpha_{x,y}} \cdot \mathsf{Eval}_x^\dagger \cdot (\mathbb{I}_{\mathcal{P} \times \mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot (|\psi, \emptyset, \emptyset\rangle_{\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y})$$

This state is well-defined because for all y in the support of $P(x)$, $\alpha_{x,y} \neq 0$.

The state $|\psi_{x,y}\rangle$ defined above has unit norm. We know that when P is evaluated on x, the value written to $\mathcal{W}_y$ will be y with probability $\alpha_{x,y}^2 = \Pr[P(x) \to y]$. That means

$$\left\| (\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle \right\| = \alpha_{x,y}$$

Next, since Eval_x is a unitary,

$$\begin{aligned}
\||\psi_{x,y}\rangle\| &= \left\| \frac{1}{\alpha_{x,y}} \cdot \mathsf{Eval}_x^\dagger \cdot (\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle \right\| \\
&= \frac{1}{\alpha_{x,y}} \cdot \left\| (\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle \right\| \\
&= \frac{1}{\alpha_{x,y}} \cdot \alpha_{x,y} = 1
\end{aligned}$$

Third, let us define $|\psi_{x,y}\rangle$ for any y not in the support of $P(x)$. The definition above will not suffice here because $\alpha_{x,y} = 0$.

$$\text{Let } |\psi_{x,y}\rangle = \mathsf{Eval}_x^\dagger \cdot (|\psi\rangle_{\mathcal{P}} |\emptyset\rangle_{\mathcal{W}_x} |y\rangle_{\mathcal{W}_y})$$

This state is well-defined and has unit norm because $\mathsf{Eval}_x^\dagger$ is a unitary.

Now we will prove some properties of $\{|\psi_{x,y}\rangle\}_{y\in\mathcal{Y}}$.

Fourth, for any $x \in \mathcal{X}$, the states $\{|\psi_{x,y}\rangle\}_{y\in\mathcal{Y}}$ are orthogonal. This is because for any $y \in \mathcal{Y}$, $\mathsf{Eval}_x |\psi_{x,y}\rangle$ is in the span of $\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}$. For any two distinct values $y, y' \in \mathcal{Y}$, $\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}$ and $\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y'\rangle \langle y'|_{\mathcal{W}_y}$ project onto orthogonal subspaces, so $\mathsf{Eval}_x |\psi_{x,y}\rangle$ and $\mathsf{Eval}_x |\psi_{x,y'}\rangle$ are orthogonal. Finally, Eval_x is a unitary, so it preserves inner products, and $|\psi_{x,y}\rangle$ and $|\psi_{x,y'}\rangle$ are orthogonal as well.

Fifth, $|\psi, \emptyset, \emptyset\rangle$ is in the span of $\{|\psi_{x,y}\rangle\}_{y\in\mathcal{Y}}$.

$$\begin{aligned}
|\psi, \emptyset, \emptyset\rangle &= \mathsf{Eval}_x^\dagger \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle \\
&= \mathsf{Eval}_x^\dagger \cdot (\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes \mathbb{I}_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle \\
&= \mathsf{Eval}_x^\dagger \cdot \left(\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes \sum_{y\in\mathcal{Y}\cup\{\emptyset\}} |y\rangle \langle y| \right) \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle \\
&= \sum_{y\in\mathcal{Y}\cup\{\emptyset\}} \mathsf{Eval}_x^\dagger \cdot (\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|) \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle \\
&= \sum_{y\in\mathcal{Y}} \alpha_{x,y} |\psi_{x,y}\rangle
\end{aligned}$$

We used the fact that

$$\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes \left(\sum_{y\in\mathcal{Y}\cup\{\emptyset\}} |y\rangle \langle y| \right) = \sum_{y\in\mathcal{Y}\cup\{\emptyset\}} \mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|$$

We also used the fact that $(\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |\emptyset\rangle \langle\emptyset|) \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle = \mathbf{0}$ because $P(x)$ never outputs $\emptyset$. Likewise, for any y not in the support of $P(x)$, $(\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|) \cdot \mathsf{Eval}_x \cdot |\psi, \emptyset, \emptyset\rangle = \mathbf{0}$.

Sixth, let us show that for any y in the support of $P(x)$,

$$\mathsf{EvalAndCopy} |x, u\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u} \otimes |\psi_{x,y}\rangle = |x, u + y\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u} \otimes |\psi_{x,y}\rangle$$

$\mathsf{EvalAndCopy}$ does not modify the computational basis value of $\mathcal{Q}_x$ and acts as $\mathsf{Eval}_x^\dagger \circ \mathsf{CopyOutput} \circ \mathsf{Eval}_x$ on the remaining registers. Let us apply Eval_x:

$$\begin{aligned}\mathsf{Eval}_x(|u\rangle_{\mathcal{Q}_u} \otimes |\psi_{x,y}\rangle) &= |u\rangle_{\mathcal{Q}_u} \otimes \mathsf{Eval}_x \cdot \left(\frac{1}{\alpha_{x,y}} \cdot \mathsf{Eval}_x^\dagger \cdot (\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot |\psi,\emptyset,\emptyset\rangle\right) \\ &= \frac{1}{\alpha_{x,y}} \cdot |u\rangle_{\mathcal{Q}_u} \otimes \left((\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot |\psi,\emptyset,\emptyset\rangle\right)\end{aligned}$$

Next, if we apply $\mathsf{CopyOutput}$, this copies y from $\mathcal{W}_y$ to $\mathcal{Q}_u$:

$$\mathsf{CopyOutput} \cdot \mathsf{Eval}_x \cdot (|u\rangle \otimes |\psi_{x,y}\rangle) = \frac{1}{\alpha_{x,y}} \cdot |u+y\rangle \otimes \left((\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot |\psi,\emptyset,\emptyset\rangle\right)$$

Finally, let us apply $\mathsf{Eval}_x^\dagger$:

$$\begin{aligned}\mathsf{Eval}_x^\dagger \cdot \mathsf{CopyOutput} \cdot \mathsf{Eval}_x \cdot (|u\rangle \otimes |\psi_{x,y}\rangle) &= \frac{1}{\alpha_{x,y}} \cdot |u+y\rangle \otimes \left(\mathsf{Eval}_x^\dagger \cdot (\mathbb{I}_{\mathcal{P}\times\mathcal{W}_x} \otimes |y\rangle \langle y|_{\mathcal{W}_y}) \cdot \mathsf{Eval}_x \cdot |\psi,\emptyset,\emptyset\rangle\right) \\ &= |u+y\rangle \otimes |\psi_{x,y}\rangle\end{aligned}$$

In summary,

$$\mathsf{EvalAndCopy}\,|x,u\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u} \otimes |\psi_{x,y}\rangle = |x,u+y\rangle_{\mathcal{Q}_x\times\mathcal{Q}_u} \otimes |\psi_{x,y}\rangle$$

□

Let S_x be the span of $\{|\psi_{x,y}\rangle\}_{y\in\mathcal{Y}}$, and let $S_{\perp,x}$ be the subspace of S_x that is orthogonal to $|\psi,\emptyset,\emptyset\rangle$. Since $|\psi_{x,y}\rangle \in S_x$, $|\psi_{x,y}\rangle$ can be written as a superposition of $|\psi,\emptyset,\emptyset\rangle$ and a state $|\psi_{x,y}^\perp\rangle \in S_{\perp,x}$.

For programs $P_1 = (|\psi^1\rangle, \mathsf{Eval}_1, R_1)$ and $P_2 = (|\psi^2\rangle, \mathsf{Eval}_2, R_2)$, let us use superscripts or subscripts for the variables defined above. For example, let $|\psi_{x,y}^1\rangle$ and $|\psi_{x,y}^2\rangle$ be the state $|\psi_{x,y}\rangle$ defined for programs P_1 and P_2, respectively. Let $\mathsf{ReflectAndCopy}_1, \mathsf{EvalAndCopy}_1, \mathsf{ReflectAndCopy}_2, \mathsf{EvalAndCopy}_2$ be defined analogously.

For each x, let us define a unitary U_x that maps

$$U_x : |\psi_{x,y}^1\rangle \mapsto |\psi_{x,y}^2\rangle$$

for all $y \in \mathcal{Y}$. Such a unitary exists because $\{|\psi_{x,y}^1\rangle\}_y$ and $\{|\psi_{x,y}^2\rangle\}_y$ are each orthonormal. Next, let us define CU to be a controlled version of U_x that reads the value of x from $\mathcal{Q}_x$ and applies U_x to $\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y$.

Claim B.12. *For any $x \in \mathcal{X}$, $U_x \cdot |\psi^1,\emptyset,\emptyset\rangle = |\psi^2,\emptyset,\emptyset\rangle$.*

Proof. First, for any (x,y), $\alpha_{x,y}^1 = \sqrt{\Pr[P_1(x)\to y]} = \sqrt{\Pr[P_2(x)\to y]} = \alpha_{x,y}^2$.

Second,

$$\begin{aligned}U_x \cdot |\psi^1,\emptyset,\emptyset\rangle &= U_x \cdot \sum_y \alpha_{x,y}^1 |\psi_{x,y}^1\rangle \\ &= \sum_y \alpha_{x,y}^1 \cdot U_x \cdot |\psi_{x,y}^1\rangle \\ &= \sum_y \alpha_{x,y}^1 \cdot |\psi_{x,y}^2\rangle \\ &= \sum_y \alpha_{x,y}^2 |\psi_{x,y}^2\rangle \\ &= |\psi^2,\emptyset,\emptyset\rangle\end{aligned}$$

□

Now let us define some hybrids to transform $\mathsf{Sim}'(P_1)$ into $\mathsf{Sim}'(P_2)$.

Hybrid 1. $\mathsf{Sim}'(P_1)$

Hybrid 2. At a high level, in this hybrid we operate on initial state $|\psi^1\rangle$ but on each query it transforms the state into $|\psi^2\rangle$ and acts with the P_2 evaluation unitary. An important detail is that transforming from $|\psi^1\rangle$ to $|\psi^2\rangle$ can only be done with respect to some x – the x-value given as input to the query.

- Initialize the program register with $|\psi^1\rangle$.
- On queries of the form $|x, u, b\rangle\,|D\rangle$ with $x' \in D$ such that $x' \neq x$, act as the identity and skip the following steps.
- Apply CU, which applies U_x to $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ controlled on the value x written on $\mathcal{Q}_x$.
- Apply the unitary

$$\mathsf{Bitflip} \circ \mathsf{ReflectAndCopy}_2 \circ \mathsf{EvalAndCopy}_2 \circ \mathsf{ReflectAndCopy}_2$$

- Apply $\mathsf{CU}^\dagger$.

Hybrid 3. $\mathsf{Sim}'(P_2)$

Before showing the indistinguishability of these hybrids, we make the following claim.

Claim B.13. *At the end of any query in hybrids 1 and 2, the database and program registers maintain the following invariant: $D = \emptyset$ if and only if $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ contain the initial program state $|\psi^1, \emptyset, \emptyset\rangle$, and $D = x$ if and only if $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ are in some state $|\phi\rangle \in S^1_{\perp,x}$.*

Proof. We first prove this invariant for hybrid 1 using induction. For the base case: the oracle is initialized in the state $|\emptyset\rangle_{\mathcal{D}}\,|\psi^1, \emptyset, \emptyset\rangle_{\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y}$, so the invariant is satisfied to begin with. For the inductive case, let us assume that at the start of a query, the invariant is satisfied. We will show that the invariant is still satisfied at the end of the query.

First, if $\mathcal{Q}_x = |x\rangle$, $\mathcal{D} = |x'\rangle$, and $x' \notin \{\emptyset, x\}$, then this query aborts, and the state on $\mathcal{D} \times \mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ still satisfy the invariant.

Next, let us consider the case where $\mathcal{Q}_x = |x\rangle$, and $\mathcal{D}$ contains either x or $\emptyset$. Let us also assume that the state satisfies the invariant at the start of the query. First, by lemma B.5, $\mathsf{ReflectAndCopy}$ acts as follows.

$$\begin{aligned}\mathsf{ReflectAndCopy} : |\emptyset\rangle\,|\psi^1, \emptyset, \emptyset\rangle &\mapsto |x\rangle\,|\psi^1, \emptyset, \emptyset\rangle \\ |x\rangle\,|\phi\rangle &\mapsto |x\rangle\,|\phi\rangle\end{aligned}$$

where $|\phi\rangle \in S^1_{\perp,x}$. Therefore, at the end of $\mathsf{ReflectAndCopy}$, $\mathcal{D} = |x\rangle$, and $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ are in S^1_x.

Second, apply $\mathsf{EvalAndCopy}$ to the state

$$|x\rangle_{\mathcal{D}}\,|\psi^1_{x,y}\rangle_{\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y}.$$

By claim B.11, this leaves the state on the $\mathcal{D} \times \mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ registers unchanged. This implies that $\mathsf{EvalAndCopy}$ maps states in S^1_x to states in S^1_x. Additionally, $\mathsf{EvalAndCopy}$ does not touch $\mathcal{D}$. Therefore, at the end of $\mathsf{EvalAndCopy}$, $\mathcal{D}$ still contains x, and $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ still lies in S^1_x.

Third, we apply $\mathsf{ReflectAndCopy}$ again. At the start of this operation, $\mathcal{D} \times \mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ is in the span of $|x\rangle\,|\psi^1, \emptyset, \emptyset\rangle$ and $|x\rangle\,|\phi\rangle$ for states $|\phi\rangle \in S^1_{\perp,x}$. $\mathsf{ReflectAndCopy}$ acts on these basis states as follows:

$$\mathsf{ReflectAndCopy} : |x\rangle\,|\psi^1,\emptyset,\emptyset\rangle \mapsto |\emptyset\rangle\,|\psi^1,\emptyset,\emptyset\rangle$$
$$|x\rangle\,|\phi\rangle \mapsto |x\rangle\,|\phi\rangle$$

Therefore, the state at the end of this operation is in the span of states that satisfy the invariant.

Fourth, we apply Bitflip. This operation acts only on $\mathcal{Q}_b$, so the state at the end of this operation will still satisfy the invariant.

A very similar argument also shows that this invariant holds for hybrid 2. □

Claim B.14. *At the start of any invocation of* $\mathsf{EvalAndCopy}_1$ *in hybrid 1, the state of* $\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y$ *is in* S^1_x*, where* x *is the value written on* $\mathcal{Q}_x$*.*

Proof. Claim B.13 says that at the start of any query, the state of $\mathcal{D}\times\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y$ is in $|\emptyset\rangle\,|\psi^1,\emptyset,\emptyset\rangle$ or $|x'\rangle\,|\phi\rangle$ for some $|\phi\rangle\in S^1_{\perp,x'}$. In the second case, we can assume that $x'=x$ because otherwise, the query will immediately abort, and $\mathsf{EvalAndCopy}_1$ will not be executed.

Next, the query to $\mathsf{Sim}'(P)$ applies $\mathsf{ReflectAndCopy}_1$, which acts as follows (lemma B.5):

$$\mathsf{ReflectAndCopy}_1 : |\emptyset\rangle\,|\psi^1,\emptyset,\emptyset\rangle \mapsto |x\rangle\,|\psi^1,\emptyset,\emptyset\rangle$$
$$|x\rangle\,|\phi\rangle \mapsto |x\rangle\,|\phi\rangle$$

for any $|\phi\rangle\in S^1_{\perp,x}$. In either case, $\mathsf{ReflectAndCopy}_1$ maps states in S^1_x to states in S^1_x.

Next, the only time that $\mathsf{EvalAndCopy}_1$ is invoked is right after the first invocation of $\mathsf{ReflectAndCopy}$. We've shown that at the start of $\mathsf{EvalAndCopy}_1$, the state of $\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y$ is in S^1_x. □

Claim B.15. *Hybrids 1 and 2 are perfectly indistinguishable.*

Proof. First, we make the following observations:

1. $\mathsf{CU}^\dagger\circ\mathsf{Bitflip}\circ\mathsf{CU}=\mathsf{Bitflip}$. This is because Bitflip acts only on $\mathcal{Q}_b$, and CU acts only on $\mathcal{Q}_x\times\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y$. Since they act on disjoint registers, they commute, so $\mathsf{CU}^\dagger\circ\mathsf{Bitflip}\circ\mathsf{CU}=\mathsf{CU}^\dagger\circ\mathsf{CU}\circ\mathsf{Bitflip}=\mathsf{Bitflip}$.

2. $\mathsf{CU}^\dagger\circ\mathsf{ReflectAndCopy}_2\circ\mathsf{CU}=\mathsf{ReflectAndCopy}_1$.

   First, $\mathsf{ReflectAndCopy}_2$ is equivalent to the operation that applies CNOT from $\mathcal{Q}_x$ to $\mathcal{D}$ controlled on $\mathcal{P}\times\mathcal{W}_x\times\mathcal{W}_y\times\mathcal{W}_b$ being in the state $|\psi^2,\emptyset,\emptyset,0\rangle$ (lemma B.5). Second, CU acts as follows (claim B.12):
   $$|x\rangle_{\mathcal{Q}_x}\,|\psi^1,\emptyset,\emptyset,0\rangle \to |x\rangle_{\mathcal{Q}_x}\,|\psi^2,\emptyset,\emptyset,0\rangle$$
   That implies that $\mathsf{CU}^\dagger\circ\mathsf{ReflectAndCopy}_2\circ\mathsf{CU}$ applies CNOT from $\mathcal{Q}_x$ to $\mathcal{D}$ controlled on the state $|\psi^1,\emptyset,\emptyset,0\rangle$. This is the same as $\mathsf{ReflectAndCopy}_1$.

3. For every x,y,
   $$\begin{aligned}\mathsf{EvalAndCopy}_1(|x,u\rangle\,|\psi^1_{x,y}\rangle) &= |x,u+y\rangle\,|\psi^1_{x,y}\rangle\\ &= \mathsf{CU}^\dagger\circ\mathsf{EvalAndCopy}_2(|x,u\rangle\,|\psi^2_{x,y}\rangle)\\ &= \mathsf{CU}^\dagger\circ\mathsf{EvalAndCopy}_2\circ\mathsf{CU}(|x,u\rangle\,|\psi^1_{x,y}\rangle)\end{aligned}$$

4. For every x,y, and every state $|\phi\rangle\in S^1_x$, we have that
   $$\mathsf{EvalAndCopy}_1(|x,u\rangle\,|\phi\rangle) = \mathsf{CU}^\dagger\circ\mathsf{EvalAndCopy}_2\circ\mathsf{CU}(|x,u\rangle\,|\phi\rangle)$$
   This follows from the previous observation (Item 3).

By claim B.14, in hybrid 1, $\mathsf{EvalAndCopy}_1$ is only invoked on states in S^1_x, and on these states, $\mathsf{EvalAndCopy}_1$ acts the same as $\mathsf{CU}^\dagger \circ \mathsf{EvalAndCopy}_2 \circ \mathsf{CU}$. Therefore, in hybrid 1, we can replace $\mathsf{EvalAndCopy}_1$ with $\mathsf{CU}^\dagger \circ \mathsf{EvalAndCopy}_2 \circ \mathsf{CU}$, and the change will be perfectly indistinguishable.

Let us put everything together. We can replace

$$\mathsf{Bitflip} \circ \mathsf{ReflectAndCopy}_1 \circ \mathsf{EvalAndCopy}_1 \circ \mathsf{ReflectAndCopy}_1$$

from hybrid 1 with

$$\mathsf{CU}^\dagger \circ \mathsf{Bitflip} \circ \mathsf{CU} \circ \mathsf{CU}^\dagger \circ \mathsf{ReflectAndCopy}_2 \circ \mathsf{CU} \circ \mathsf{CU}^\dagger \circ \mathsf{EvalAndCopy}_2 \circ \mathsf{CU} \circ \mathsf{CU}^\dagger \circ \mathsf{ReflectAndCopy}_2 \circ \mathsf{CU}$$

Then when we cancel out adjacent applications of CU and $\mathsf{CU}^\dagger$, the operation becomes:

$$\mathsf{CU}^\dagger \circ \mathsf{Bitflip} \circ \mathsf{ReflectAndCopy}_2 \circ \mathsf{EvalAndCopy}_2 \circ \mathsf{ReflectAndCopy}_2 \circ \mathsf{CU}$$

This is the sequence of operations found in hybrid 2. Therefore, hybrids 1 and 2 are perfectly indistinguishable.

□

Claim B.16. *Hybrids 2 and 3 are perfectly indistinguishable.*

Proof. In hybrid 2, the first application of CU during the first query to the simulator converts $|\psi^1, \emptyset, \emptyset\rangle$ into $|\psi^2, \emptyset, \emptyset\rangle$, which is consistent with the initial state of $|\psi^2, \emptyset, \emptyset\rangle$ in hybrid 3. We would now like to argue that the applications of CU and $\mathsf{CU}^\dagger$ cancel each other, but this is not true in general: in between queries, the adversary can act on the input query register, so that we are effectively applying a $U_{x'} \circ U_x^\dagger$ operation on the internal state of the oracle, which is not the identity operation in general.

We make use of the invariant from Claim B.13: In every branch of the superposition of hybrid 2, after the end of a query, either $D = \emptyset$ and $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y = |\psi^1, \emptyset, \emptyset\rangle$, or D contains x, and $\mathcal{P} \times \mathcal{W}_x \times \mathcal{W}_y$ contains $|\phi\rangle \in S^1_{\perp,x}$.

In the branch with $|\psi^1, \emptyset, \emptyset\rangle$, CU acts the same no matter what the control register is, and always transforms $|\psi^1, \emptyset, \emptyset\rangle$ to $|\psi^2, \emptyset, \emptyset\rangle$, In this branch, the applications of CU and $\mathsf{CU}^\dagger$ cancel each other. In the branch with $|\phi\rangle\,|x\rangle$, where $|\phi\rangle \in S^1_{\perp,x}$, if $x' \neq x$, the next query acts as the identity on this branch, so we can ignore it. □

This shows that $\mathsf{Sim}'(P_1)$ and $\mathsf{Sim}'(P_2)$ are perfectly indistinguishable and completes the proof of Lemma B.10. □

Lemma B.17 (Simulation of the CSEQ oracle)**.** *Given any (possibly oracle-aided) testable quantum program P that implements f with 0 error, an oracle for $\mathsf{Sim}'(P)$ (definition B.4) is perfectly indistinguishable from O_f^{CSEQ} to any adversary making an unbounded number of quantum queries.*

Proof. First, Sim' only needs black-box access to the Eval and R operations of P. It does not access $|\psi\rangle$ directly. P may even be oracle-aided, and the oracle may maintain an internal pure state, which is considered part of $|\psi\rangle$.

Second, P_f (defined in lemma B.6) and P are testable quantum programs. P_f implements f with 0 error. Lemma B.6 says that $\mathsf{Sim}'(P_f)$ and O_f^{CSEQ} are perfectly indistinguishable after an unbounded number of quantum queries.

Third, since P also implements f with 0 error, lemma B.10 implies that oracles for $\mathsf{Sim}'(P_f)$ and $\mathsf{Sim}'(P)$ are perfectly indistinguishable after an unbounded number of quantum queries. Therefore, $\mathsf{Sim}'(P)$ and O_f^{CSEQ} are also perfectly indistinguishable after an unbounded number of quantum queries. □

Remark B.18. *For classical functionalities, any OTP compiler that satisfies CSEQ security is best-possible among testable programs, in the sense of definition 4.4. This follows from Lemma B.17; the compiler's own outputs need not be testable.*

Theorem B.19.

1. *For any sets of bitstrings $\mathcal{X}, \mathcal{R}, \mathcal{Y}$, there exists a Q.P.T. simulator Sim_1 such that for every randomized function $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$, O_f^{CSEQ} and $\mathsf{Sim}_1^{O_{\Phi_f}^{\mathsf{SEQ}}}$ are perfectly indistinguishable.*
2. *For any sets of bitstrings $\mathcal{X}, \mathcal{R}, \mathcal{Y}$, **for which** $|\mathcal{X}| > 1$, there exists a Q.P.T. simulator Sim_2 such that for every randomized function $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$, $\mathsf{Sim}_2^{O_f^{\mathsf{CSEQ}}}$ and $O_{\Phi_f}^{\mathsf{SEQ}}$ are perfectly indistinguishable.*

Proof.

Simulating O_f^{CSEQ} with Sim_1: Let Sim' be the simulator from definition B.4. By lemma B.17, if Sim' is given black-box access to any testable quantum program P that implements f with 0 error, then $\mathsf{Sim}'(P)$ is perfectly indistinguishable from O_f^{CSEQ}. The program P may be oracle-aided, and the oracle may maintain an internal pure state.

Next, $O_{\Phi_f}^{\mathsf{SEQ}}$ can be used to construct such a testable oracle interface that implements f with 0 error (lemma B.20). Composing these two procedures yields Sim_1.

Simulating $O_{\Phi_f}^{\mathsf{SEQ}}$ with Sim_2: Sim_2 mainly uses the simulator Sim' from definition 4.6 (this is a different Sim' than the one used to construct Sim_1). This Sim' takes any testable quantum program that implements Φ_f and perfectly simulates $O_{\Phi_f}^{\mathsf{SEQ}}$ (lemma 4.7). The program can be oracle-aided, and the oracle can even maintain an internal state. Sim' only needs black-box access to the program's Eval and R operations.

Next, we can use O_f^{CSEQ} to construct a testable oracle interface that implements Φ_f as long as $|\mathcal{X}| > 1$ (lemma B.24). Then if we run Sim' on this interface, it will perfectly simulate $O_{\Phi_f}^{\mathsf{SEQ}}$. □

Lemma B.20. *$O_{\Phi_f}^{\mathsf{SEQ}}$ can be used to construct a testable oracle interface that implements f with 0 error.*

The program is oracle-aided since it makes queries to $O_{\Phi_f}^{\mathsf{SEQ}}$, and the program state includes the internal state of the oracle.

Proof. Let us construct a testable oracle interface $P = (|\psi\rangle, \mathsf{Eval}, \mathsf{R})$ that implements f.

- Program State $|\psi\rangle$: Let the program state be the internal state of $O_{\Phi_f}^{\mathsf{SEQ}}$ concatenated with a cache register $\mathcal{B}$ that is initialized to $|0\rangle$. Note that the internal state of $O_{\Phi_f}^{\mathsf{SEQ}}$ is defined as a pure state, so the initial program state $|\psi\rangle$ is indeed pure.
- Eval: The program's Eval operation simply queries $O_{\Phi_f}^{\mathsf{SEQ}}$. Eval takes an external query register $\mathcal{Q}$ with basis states of the form $|x, u\rangle$ where $(x, u) \in \mathcal{X} \times \mathcal{Y}$. Then Eval queries $O_{\Phi_f}^{\mathsf{SEQ}}$ on $\mathcal{Q} \times \mathcal{B}$:

  $$|x, u\rangle_{\mathcal{Q}} \otimes |b\rangle_{\mathcal{B}} \to |x, u \oplus y\rangle_{\mathcal{Q}} \otimes |b \oplus b'\rangle_{\mathcal{B}}$$

  Finally, Eval outputs $\mathcal{Q}$.
- R: The reflection operation R takes an external register $\mathcal{E}$. If $\mathcal{B}$ stores 0, then R applies X to $\mathcal{E}$.

Claim B.21. *The program P defined above implements f with 0 error (definition 3.1).*

Proof. Given a query of the form $|x, 0\rangle$, the program's Eval operation queries $O^{\mathsf{SEQ}}_{\Phi_f}$ on $|x, 0, b\rangle$. The first time $O^{\mathsf{SEQ}}_{\Phi_f}$ is queried, it applies the unitary U_{Φ_f}, which uses a compressed oracle to compute $y = f(x; r)$ for a uniformly random $r \xleftarrow{\$} \mathcal{R}$. This is exactly the output distribution of $f(x)$. □

The following claim shows that P is indeed testable.

Claim B.22. *After any number of* Eval *and R operations, applying R is equivalent to applying the function $\mathsf{X} \otimes |\psi\rangle\langle\psi| + I \otimes (I - |\psi\rangle\langle\psi|)$ to $\mathcal{E}$ and the current program state.*

Proof. Here is some intuition for why R works as the reflection oracle. First, $O^{\mathsf{SEQ}}_{\Phi_f}$'s internal state includes a register $\mathcal{C}$ that is $|0\rangle$ if and only if $O^{\mathsf{SEQ}}_{\Phi_f}$ is in its initial state. Second, any time that $\mathcal{C}$ is changed, $O^{\mathsf{SEQ}}_{\Phi_f}$ flips the output bit b. This implies that $\mathcal{C}$ and $\mathcal{B}$ always store the same value after any query, and that $\mathcal{B} = |0\rangle$ if and only if the program state is in its initial state. Next, we will make this argument formal.

The program state comprises three registers: $\mathcal{C}$ and $\mathcal{P}$, which contain $O^{\mathsf{SEQ}}_{\Phi_f}$'s state, and $\mathcal{B}$. The initial state of $\mathcal{B} \times \mathcal{C} \times \mathcal{P}$ is $|\psi\rangle := |0\rangle \otimes |0\rangle \otimes |0^m\rangle$.

We will show that after any number of Eval and R operations, $\mathcal{B} = |0\rangle$ if and only if $\mathcal{B} \times \mathcal{C} \times \mathcal{P} = |\psi\rangle$.

After any number of Eval and R operations, $\mathcal{B}$ and $\mathcal{C}$ store the same computational-basis value. First, $\mathcal{B} \times \mathcal{C}$ are initialized to $|0\rangle \otimes |0\rangle$. Next, the only time that $\mathcal{C}$'s computational-basis value changes is potentially during an Eval operation, specifically during step 2 of the query to $O^{\mathsf{SEQ}}_{\Phi_f}$. If $\mathcal{C}$ is not flipped in step 2, then this call to $O^{\mathsf{SEQ}}_{\Phi_f}$ acts as the identity, and $\mathcal{B}$ is not flipped either. On the other hand, if $\mathcal{C}$ is flipped in step 2, then this call to $O^{\mathsf{SEQ}}_{\Phi_f}$ applies U_{Φ_f} or $U^\dagger_{\Phi_f}$, which are equal according to claim B.23. U_{Φ_f} flips the value of b. Since the program records the value of b on register $\mathcal{B}$, that means that $\mathcal{B}$ is flipped on any query that flips $\mathcal{C}$. Therefore, after any number of Eval or R operations, $\mathcal{B} \times \mathcal{C}$ store $(0, 0)$ or $(1, 1)$.

After any number of Eval and R operations, if $\mathcal{C} = |0\rangle$ then $\mathcal{P} = |0^m\rangle$. First, $\mathcal{C} \times \mathcal{P}$ are initialized to $|0\rangle \otimes |0^m\rangle$, so the invariant is initially true. Second, R does not change the state of $\mathcal{C} \times \mathcal{P}$. Third, the first Eval query flips $\mathcal{C}$ to $|1\rangle$ with certainty. If any future Eval query flips $\mathcal{C}$ back to $|0\rangle$, then the check in step 2 of $O^{\mathsf{SEQ}}_{\Phi_f}$ must have found that $\mathcal{P} = |0^m\rangle$ and flipped $\mathcal{C}$ to $|0\rangle$. Then step 3 would have acted as the identity because $\mathcal{C} = |0\rangle$, so at the end of the query, it is still true that $\mathcal{P} = |0^m\rangle$. This argument extends to show that after any number of queries, either $\mathcal{C} = |1\rangle$ or $\mathcal{C} \times \mathcal{P} = |0\rangle \otimes |0^m\rangle$.

The previous discussion implies that after any number of Eval and R operations, if $\mathcal{B} = |0\rangle$, then $\mathcal{C} = |0\rangle$, and $\mathcal{P} = |0^m\rangle$, so $\mathcal{B} \times \mathcal{C} \times \mathcal{P} = |\psi\rangle$. Next, if $\mathcal{B} = |1\rangle$, then $\mathcal{B} \times \mathcal{C} \times \mathcal{P} \neq |\psi\rangle$. Therefore, $\mathcal{B} = |0\rangle$ if and only if $\mathcal{B} \times \mathcal{C} \times \mathcal{P} = |\psi\rangle$.

Finally, R applies X to $\mathcal{E}$ if $\mathcal{B} = |0\rangle$. This is equivalent to applying X to $\mathcal{E}$ if $\mathcal{B} \times \mathcal{C} \times \mathcal{P}$ is in state $|\psi\rangle$.

□

Claim B.23. *For the unitary U_{Φ_f} defined in definition B.3, $U^\dagger_{\Phi_f} = U_{\Phi_f}$.*

Proof. First, steps 1 - 3 of U_{Φ_f} are each their own inverses. Step 1 CNOTs x from $\mathcal{Q}$ to $\mathcal{W}$, which can be uncomputed by applying step 1 a second time.

Step 2 queries the compressed oracle. The compressed oracle is perfectly indistinguishable from a random oracle when queried as a black box, and any query to a random oracle can be uncomputed by querying the random oracle a second time. Therefore, step 2 can be uncomputed by computing step 2 a second time.

Step 3 CNOTs some values that are computed from $\mathcal{W}$ onto $\mathcal{Q}$. This can be uncomputed by applying step 3 a second time.

Second, U_{Φ_f} and $U_{\Phi_f}^\dagger$ apply the following sequences of operations:

$$\begin{aligned}
U_{\Phi_f} &= \text{step } 1^\dagger \cdot \text{step } 2^\dagger \cdot \text{step } 3 \cdot \text{step } 2 \cdot \text{step } 1 \\
U_{\Phi_f}^\dagger &= \left(\text{step } 1^\dagger \cdot \text{step } 2^\dagger \cdot \text{step } 3 \cdot \text{step } 2 \cdot \text{step } 1\right)^\dagger \\
&= \text{step } 1^\dagger \cdot \text{step } 2^\dagger \cdot \text{step } 3^\dagger \cdot \left(\text{step } 2^\dagger\right)^\dagger \cdot \left(\text{step } 1^\dagger\right)^\dagger \\
&= \text{step } 1^\dagger \cdot \text{step } 2^\dagger \cdot \text{step } 3 \cdot \text{step } 2 \cdot \text{step } 1 \\
&= U_{\Phi_f}
\end{aligned}$$

□

□

Lemma B.24. *If $|\mathcal{X}| > 1$, then O_f^{CSEQ} can be used to construct a testable oracle interface that implements Φ_f.*

Proof. Let us construct a testable oracle interface $P = (|\psi\rangle, \mathsf{Eval}, \mathsf{R})$ that implements Φ_f.

- Program State $|\psi\rangle$: Let the program state $|\psi\rangle$ be the internal state of O_f^{CSEQ} concatenated with several work registers. Let $x_0, x_1 \in \mathcal{X}$ be two distinct values, and then initialize the work registers as follows:

  $$\mathcal{Q}_0 = |x_0, 0, 0\rangle, \mathcal{Q}_1 = |x_1, 0, 0\rangle, \mathcal{B}_0 = |0\rangle, \mathcal{B}_1 = |0\rangle$$

  Note that the internal state of O_f^{CSEQ} is defined as a pure state, so $|\psi\rangle$ is indeed pure.

- Eval: The program's Eval operation simply queries O_f^{CSEQ}.

- R: The reflection operation R should test whether the program state is in its initial state. Part of the program state is contained in the oracle O_f^{CSEQ}, so our construction cannot access it directly. Instead, we can make queries to O_f^{CSEQ} to test if its internal state is in its initial state. Our strategy is to query O_f^{CSEQ} on two different inputs, while uncomputing between queries. If the oracle answers both queries, then its state began in the initial state. If it rejects at least one query, then its database must already record a query.

  Formally, R takes as input a single-qubit register $\mathcal{E}$ and acts as follows.

  1. Query O_f^{CSEQ} on $\mathcal{Q}_0$. The resulting state is $|x_0, y_0, b_0\rangle_{\mathcal{Q}_0}$ for some $y_0 \in \mathcal{Y}$ and $b_0 \in \{0,1\}$.
  2. Copy b_0 onto $\mathcal{B}_0$.
  3. Uncompute item 1.
  4. Query O_f^{CSEQ} on $\mathcal{Q}_1$. The resulting state is $|x_1, y_1, b_1\rangle_{\mathcal{Q}_1}$ for some $y_1 \in \mathcal{Y}$ and $b_1 \in \{0,1\}$.
  5. Copy b_1 onto $\mathcal{B}_1$.
  6. Uncompute item 4.
  7. If $|b_0, b_1\rangle_{\mathcal{B}_0 \times \mathcal{B}_1} = |1,1\rangle$, then apply X (bitflip) to register $\mathcal{E}$.
  8. Uncompute steps 1 - 6.

Claim B.25. *P implements Φ_f (with 0 error).*

Proof. When we apply Eval to the user's query register $\mathcal{Q}$ and the initial program state $|\psi\rangle$, it queries O_f^{CSEQ}. Since the database of O_f^{CSEQ} is initialized to $|\emptyset\rangle$, step 1 of O_f^{CSEQ} does nothing during this query.

If we omit step 1 of O_f^{CSEQ}, then the remaining steps are exactly the same as U_{Φ_f}. Therefore, applying Eval with program state $|\psi\rangle$ computes the same operation as Φ_f. □

The following claim shows that P is indeed testable.

Claim B.26. *After any number of* Eval *and R operations, applying R is equivalent to applying the function $\mathsf{X} \otimes |\psi\rangle\langle\psi| + I \otimes (I - |\psi\rangle\langle\psi|)$ to $\mathcal{E}$ and the current program state.*

Proof. Let us consider the case where P's program state is in its initial state $|\psi\rangle$. We will show that in this case, R applies X to $\mathcal{E}$ and restores the program state to its initial state.

Step 1 of R queries O_f^{CSEQ} on input $|x_0, 0, 0\rangle$. Since we are in the initial state, O_f^{CSEQ} responds to the query and flips b_0 to 1 with certainty. Step 2 copies b_0 to another register $\mathcal{B}_0$, but this step does not change the joint state on $\mathcal{Q}_0$ and the internal registers of O_f^{CSEQ} because the value $b_0 = 1$ is deterministic. Step 3 uncomputes step 1. Steps 1 and 3 act only on $\mathcal{Q}_0$ and the internal state of O_f^{CSEQ}, and the state on these registers is unchanged by step 2. Therefore the state of $\mathcal{Q}_0$ and the internal state of O_f^{CSEQ} is the same at the end of step 3 as it was at the start of step 1. The only difference in the program's state is that at the end of step 3, $\mathcal{B}_0$ contains $|1\rangle$.

Next, steps 4 - 6 are the same as steps 1 - 3 except the query register is $\mathcal{Q}_1 = |x_1, 0, 0\rangle$. By the same argument as before, the state at the end of step 6 is the same as the state at the start of step 1 except that $\mathcal{B}_0 = |1\rangle$ and $\mathcal{B}_1 = |1\rangle$.

Next, step 7 applies X to $\mathcal{E}$ with certainty. This step does not change the state on the program's registers because the application of X occurs with probability 1. Finally, step 8 uncomputes all steps except for the application of X. At the end of R, the program state is the initial state $|\psi\rangle$, and the only change from the beginning of R is the application of X to $\mathcal{E}$.

Next, let us consider the case where the program state is orthogonal to $|\psi\rangle$. We only need to consider states that are reachable from a sequence of Eval and R operations. We will show that in this case, R acts as the identity.

If the program state is orthogonal to $|\psi\rangle$, then the database register $\mathcal{D}$ must contain a non-empty database D. The program state comprises the registers: $\mathcal{W} \times \mathcal{D}$, which are the internal state of O_f^{CSEQ}, as well as $\mathcal{Q}_0 \times \mathcal{Q}_1 \times \mathcal{B}_0 \times \mathcal{B}_1$. After any query to O_f^{CSEQ}, the $\mathcal{W}$ register is in the $|0\rangle$ state because any values written to it have been uncomputed during the query. Likewise, the state of $\mathcal{Q}_0 \times \mathcal{Q}_1 \times \mathcal{B}_0 \times \mathcal{B}_1$ at the start of any R operation is the same as its initial state (claim B.27). After any number of queries to Eval and R, the only register that might not be in its initial state is $\mathcal{D}$. If the program state is orthogonal to $|\psi\rangle$, then $\mathcal{D}$ must contain a non-empty database. We will consider two types of non-empty databases: (1) every entry of D (the only entry, really) is of the form (x_0, r) for some $r \in \mathcal{R}$, and (2) D contains (x', r) for some $x' \neq x_0$ and some $r \in \mathcal{R}$.

Now let us step through R in the first case, where every entry of D is of the form (x_0, r) for some $r \in \mathcal{R}$. Step 1 queries O_f^{CSEQ} on input $|x_0, 0, 0\rangle$, and the query is answered ($b_0 = 1$). Step 2 copies b_0 to $\mathcal{B}_0$, and step 3 uncomputes step 1. The state of the program at the end of step 3 is the same as the state at the beginning of step 1 except that $\mathcal{B}_0 = |1\rangle$. Next, step 4 queries O_f^{CSEQ} on x_1. This query is rejected ($b_1 = 0$) because D contains (x_0, r). By step 7, $\mathcal{B}_0 \times \mathcal{B}_1 = |1, 0\rangle$, so step 7 acts as the identity. Finally, step 8 uncomputes steps 1 - 6, so R acts as the identity on the program state.

Finally, let us step through R in the second case, where D contains (x', r) for some $x' \neq x_0$ and some $r \in \mathcal{R}$. Then the query to O_f^{CSEQ} in step 1 will be rejected ($b_0 = 0$). Skipping ahead, at the

start of step 7, $\mathcal{B}_0 = |0\rangle$, so this step acts as the identity. Finally, step 8 uncomputes steps 1 - 6, so R acts as the identity on the program state.

In summary, we've shown that for every program state that is reachable by a sequence of Eval and R operations, if the program state is orthogonal to $|\psi\rangle$, then R acts as the identity on the program state. □

Claim B.27. *After any number of* Eval *and R operations, the work registers are in their initial state:*

$$\mathcal{Q}_0 = |x_0, 0, 0\rangle, \mathcal{Q}_1 = |x_1, 0, 0\rangle, \mathcal{B}_0 = |0\rangle, \mathcal{B}_1 = |0\rangle$$

Proof. Before any Eval and R operations have been executed, the work registers $\mathcal{Q}_0 \times \mathcal{Q}_1 \times \mathcal{B}_0 \times \mathcal{B}_1$ are in their initial state. Next, Eval does not modify the work registers. Finally, it suffices to analyze an R operation and show that if the work registers are in their initial state at the start of the operation, then they will return to their initial state at the end of the operation.

Let us split up the program's registers into two groups. Let register $\mathcal{R}$ comprise all components of the work registers $\mathcal{Q}_0 \times \mathcal{Q}_1 \times \mathcal{B}_0 \times \mathcal{B}_1$ except the values x_0 and x_1 written on $\mathcal{Q}_0$ and $\mathcal{Q}_1$ respectively. Let $\mathcal{S}$ comprise the internal registers of O_f^{CSEQ} and the values x_0 and x_1 written on $\mathcal{Q}_0$ and $\mathcal{Q}_1$.

Next, no step of R or O_f^{CSEQ} modifies the computational basis value of x_0 and x_1 written on $\mathcal{Q}_0$ and $\mathcal{Q}_1$. Therefore these values remain in unchanged by R.

Steps 1 - 6 of R may modify register $\mathcal{R}$, but they only do so by CNOT-ing a value onto $\mathcal{R}$ that was computed from the state on register $\mathcal{S}$. Next, step 7 applies an X operation to $\mathcal{E}$ controlled on the value of $\mathcal{R}$. However, step 7 does not change the state of $\mathcal{S}$. Finally, step 8 uncomputes steps 1 - 6. Again, this entails CNOT-ing a value onto $\mathcal{R}$ that is computed from the state on $\mathcal{S}$. The value that is CNOT-ed in step 8 is the same as the value that was CNOT-ed during steps 1 - 6 because the state on $\mathcal{S}$ is unchanged. Therefore step 8 returns the state of $\mathcal{R}$ to its initial state.

In summary, all components of $\mathcal{Q}_0 \times \mathcal{Q}_1 \times \mathcal{B}_0 \times \mathcal{B}_1$ are returned to their initial state at the end of R. □

□

C Impossibility of one-time correct sampling QSIO

In this section, we show that our impossibility for a best-possible one-time compiler in Section 3 rules out the existence of obfuscation schemes that satisfy a natural definition of quantum state indistinguishability obfuscation for *sampling* programs. Previously, quantum state indistinguishability obfuscation has been studied only in the setting where the quantum program implements a deterministic classical functionality with negligible error [BKNY23, GM24].

A natural generalization of the definition of quantum state IO to quantum programs designed for sampling tasks is the following. It requires obfuscations of any two programs that implement the same sampling task to be indistinguishable. Crucially, for two programs to be equal in this sense, they need to evaluate the same distributions on the first query, but there are no guarantees on their equivalence on subsequent queries. A priori, one can image that it might be possible to indistinguishably obfuscate such a pair of programs by enforcing some kind of one-time guarantee, so both obfuscations stop working after one evaluation. The impossibility in this section (Corollary C.3) says that this is not possible. Indeed, if it was possible, then it would be a best-possible one-time program. This is formalized in Lemma C.2.

We leave open the question of coming up with a feasible notion of quantum state IO for sampling programs. Our notion of stateful obfuscation in Section 5.3 (a notion of quantum state IO where the two programs must have the same behavior on polynomially many (forward and inverse) queries, instead of just the first one) can be seen as an attempt to make progress in this direction.

Definition C.1 (Quantum State Obfuscation for Quantum Sampling Programs)**.** *A quantum state obfuscator for quantum sampling programs is a q.p.t. algorithm* QObf *with the following syntax:*

$$\mathsf{QObf}(1^\lambda, \rho, C) \to (\tilde{\rho}, \tilde{C}).$$

The obfuscator takes as input a security parameter 1^λ *and a quantum sampling program* (ρ, C) *and outputs an obfuscated sampling program* $(\tilde{\rho}, \tilde{C})$*. Both program states may be mixed, and* C *and* $\tilde{C}$ *are unitary evaluation circuits, as in Definition 3.1.*

- ***Correctness:*** *Suppose the program family* $\{(\rho_\lambda, C_\lambda)\}_{\lambda\in\mathbb{N}}$ *implements a family of sampling functionalities* $\{D_\lambda : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}_{\lambda\in\mathbb{N}}$ *for every* $x \in \{0,1\}^{n(\lambda)}$ *up to negligible error* $\mathsf{negl}(\lambda)$*. Then, the family of quantum sampling programs* $\{(\tilde{\rho}_\lambda, \tilde{C}_\lambda)\}$ *where*

  $$(\tilde{\rho}_\lambda, \tilde{C}_\lambda) \leftarrow \mathsf{QObf}(1^\lambda, \rho_\lambda, C_\lambda)$$

  also implements the same sampling functionality up to negligible error.

- ***Indistinguishability Obfuscation:*** *For every pair of families of quantum sampling programs* $\{(\rho_{\lambda,0}, C_{\lambda,0})\}_{\lambda\in\mathbb{N}}$ *and* $\{(\rho_{\lambda,1}, C_{\lambda,1})\}_{\lambda\in\mathbb{N}}$ *that both implement the same sampling functionality* $\{D_\lambda : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}_{\lambda\in\mathbb{N}}$ *up to negligible error in* λ*, and furthermore satisfy that the program descriptions have the same length* $|\rho_{\lambda,0}, C_{\lambda,0}| = |\rho_{\lambda,1}, C_{\lambda,1}|$ *for every* λ*, for every q.p.t. adversary* $\{A_\lambda\}_{\lambda\in\mathbb{N}}$ *and every family of polynomial-length classical auxiliary inputs* $\{z_\lambda\}_{\lambda\in\mathbb{N}}$*, there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

  $$\Big|\Pr\left[1 \leftarrow A_\lambda\left(1^\lambda, z_\lambda, \mathsf{QObf}(1^\lambda, \rho_{\lambda,0}, C_{\lambda,0})\right)\right] - \Pr\left[1 \leftarrow A_\lambda\left(1^\lambda, z_\lambda, \mathsf{QObf}(1^\lambda, \rho_{\lambda,1}, C_{\lambda,1})\right)\right]\Big| \leq \mathsf{negl}(\lambda).$$

Lemma C.2. *Suppose there exists a quantum state indistinguishability obfuscator for quantum sampling programs for a class of classical randomized functionalities* $\mathcal{F}$*. Then, there must exist a one-time compiler that satisfies the best-possible one-time security guarantee of Definition 3.3 for* $\mathcal{F}$*.*

Proof. By assumption, let QObf be an obfuscator that satisfies quantum state indistinguishability obfuscation for quantum sampling programs for $\mathcal{F}$. Consider the following quantum sampling program $P_f = (|\psi_f\rangle\langle\psi_f|, C_f)$ that implements classical randomized functions $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ for $f \in \mathcal{F}$: the program state is a description of f, along with a uniform superposition over $\mathcal{R}$,

$$|\psi_f\rangle = |f\rangle \otimes \frac{1}{\sqrt{|\mathcal{R}|}} \sum_{r\in\mathcal{R}} |r\rangle .$$

The circuit C_f maps

$$|x, u\rangle_{\mathcal{Q}} \otimes |f, r\rangle_{\mathcal{P}} \mapsto |x, u \oplus f(x; r)\rangle_{\mathcal{Q}} \otimes |f, r\rangle_{\mathcal{P}} .$$

Fix a public polynomial L such that $L(\lambda) \geq \lambda$ and $|P_f| \leq L(\lambda)$ for every $f \in \mathcal{F}_\lambda$. Such a polynomial exists because this canonical implementation has size polynomial in $|f|$. Let $\mathsf{Pad}_{L(\lambda)}$ pad any shorter program to length $L(\lambda)$ without changing its functionality. Define $\mathsf{OTP}^*(f) := \mathsf{QObf}(1^\lambda, \mathsf{Pad}_{L(\lambda)}(P_f))$ for $f \in \mathcal{F}_\lambda$. Correctness follows from the correctness of QObf. We claim that OTP^* satisfies the best-possible one-time security guarantee of Definition 3.3. For every q.p.t. adversary $\mathcal{A}$ and negligible correctness error μ, define

$$\mathsf{Sim}(1^\lambda, P) := \mathcal{A}(1^\lambda, \mathsf{QObf}(1^\lambda, \mathsf{Pad}_{L(\lambda)}(P))).$$

Every admitted competitor P satisfies $|P| = |f| = \lambda$, so the two inputs to QObf have the same length $L(\lambda)$. They also implement the same sampling functionality up to error μ. Thus, for every q.p.t. distinguisher D, there is a negligible function ν such that for every λ, every $f \in \mathcal{F}_\lambda$, and every such P,

$$\left|\Pr[1 \leftarrow D(1^\lambda, f, \mathcal{A}(1^\lambda, \mathsf{OTP}^*(f)))] - \Pr[1 \leftarrow D(1^\lambda, f, \mathsf{Sim}(1^\lambda, P))]\right| \leq \nu(\lambda).$$

Otherwise, choosing violating pairs (f_λ, P_λ) along an infinite subsequence of parameters, and taking identical programs at the remaining parameters, would give two equal-length program families contradicting quantum state indistinguishability obfuscation. The corresponding adversary composes $\mathcal{A}$ with D and uses $z_\lambda = f_\lambda$ as its classical auxiliary input. □

Corollary C.3. *Assuming $\mathsf{SZK} \not\subseteq \mathsf{BQP}$, there exists a class of sampling functionalities $\mathcal{D}$ such that there is no quantum state indistinguishability obfuscator (as defined in Definition C.1) for $\mathcal{D}$.*

This follows from Lemma C.2 and theorem 3.4.

D One-Time VBB Security When Possible

We formalize the weakening discussed in Remark 1.9 and explain how the proofs of Theorems 3.4 and 3.6 also rule it out. The phrase "when possible" refers to security on a family of functionalities: a simulator must work throughout that family, rather than being chosen separately for each circuit.

One-query simulation. Fix the public input and output spaces at each security parameter. For a classical sampling functionality f, let $\mathcal{O}_f^{(1)}$ allow at most one classical query x and return a classical sample $f(x; r)$ for fresh internal randomness r. It supplies neither a coherent extension nor the circuit description of f. We use the sampling-program syntax and public size convention of Definition 3.3; in particular, a competing program must satisfy $|P| = |f|$. Compiler randomness is included in the output sampling program, so correctness can hold for that mixed program as a whole.

Definition D.1 (One-time VBB security)**.** *A one-time compiler K has* one-time VBB security *on a family $\mathcal{G}$ if for every QPT adversary $\mathcal{A}$ there is a QPT simulator S making at most one classical oracle query such that*

$$\mathcal{A}(1^\lambda, \mathsf{K}(f)) \approx_c \mathsf{S}^{\mathcal{O}_f^{(1)}}(1^\lambda, 1^{|f|})$$

uniformly over $f \in \mathcal{G}_\lambda$. The simulator may depend on $\mathcal{A}$, K, and $\mathcal{G}$, but not on the particular f. Indistinguishability uses the same distinguisher convention as Definition 3.3.

Definition D.2 (One-time VBB when possible)**.** *A one-time compiler K for $\mathcal{F}$ achieves* one-time VBB when possible *with statistical/perfect equivalence if the following holds for every subfamily $\mathcal{G} \subseteq \mathcal{F}$. If there is a one-time VBB-secure compiler J on $\mathcal{G}$ whose output $\mathsf{J}(f)$ is an admissible competitor for f in Definition 3.3, then K has one-time VBB security on $\mathcal{G}$. Here admissibility means $|\mathsf{J}(f)| = |f|$ and correctness with a common negligible/zero error bound, respectively.*

The definition imposes correctness on all of $\mathcal{F}$, while its security requirement applies only on subfamilies admitting one-query simulation. It does not require the compiler to recognize those subfamilies efficiently. The public size bound prevents the comparison from providing additional program-length information; all circuits and sample-containing programs in the argument below can be padded to satisfy it.

Lemma D.3. *A best-possible one-time compiler with statistical/perfect equivalence achieves one-time VBB when possible with the corresponding equivalence notion.*

Proof. Fix a subfamily $\mathcal{G}$, a witnessing compiler J, and an adversary $\mathcal{A}$ against OTP^*. Best-possible security gives a QPT simulator B such that

$$\mathcal{A}(1^\lambda, \mathsf{OTP}^*(f)) \approx_c \mathsf{B}(1^\lambda, \mathsf{J}(f)) \approx_c \mathsf{S}^{\mathcal{O}_f^{(1)}}(1^\lambda, 1^{|f|}) \qquad (f \in \mathcal{G}_\lambda).$$

The second relation applies the one-time VBB security of J to the adversary B. Both simulators work uniformly over $\mathcal{G}$, and the resulting S makes at most one classical query. □

Theorem D.4. *If* $\mathsf{SZK} \not\subseteq \mathsf{BQP}$ *(respectively,* $\mathsf{HVPZK}|_{\mathrm{am}} \not\subseteq \mathsf{BQP}$*), no one-time compiler for all classical sampling functionalities with a single input bit achieves one-time VBB when possible with statistical (respectively, perfect) equivalence.*

Proof sketch. Suppose K is a correct compiler with the claimed security. For statistical equivalence, use the polarized sampling circuits C_0, C_1, functionality $f(b; r) = C_b(r)$, constants $c_y(b) = y$, and EPR adversary $\mathcal{A}$ from Theorem 3.4. Use the same common polynomial padding, so all public lengths and input/output spaces agree. The correctness arguments in that proof give

$$\Pr[\mathcal{A}(\mathsf{K}(c_y)) = 1] \le \mathsf{negl}(\lambda), \qquad \Pr[\mathcal{A}(\mathsf{K}(f)) = 1] \ge \tfrac{1}{2} - \mathsf{negl}(\lambda) \quad \text{in the far case.}$$

Let $\mathcal{G}$ contain all the close-case functionalities and all the constants used in the reduction. This subfamily admits a one-time VBB-secure compiler J: sample $y \leftarrow f(0)$ and output a padded program that always returns y. The common negligible bound on the distance between D_0 and D_1 gives correctness, and one classical query to f exactly simulates $\mathsf{J}(f)$. Thus Definition D.2 supplies a single one-query simulator S for $\mathcal{A}$ throughout $\mathcal{G}$.

Fix a close instance. For either possible query b, the response $f(b)$ is negligibly close to $c_y(b)$ averaged over $y \leftarrow D_0$. Because S makes at most one classical query and does not receive the circuit description, its final states in these two experiments are negligibly close. Suppressing the common public length and security parameter, we obtain

$$\begin{aligned}\Pr[\mathcal{A}(\mathsf{K}(f)) = 1] &\approx \Pr[\mathsf{S}^{\mathcal{O}_f^{(1)}} = 1] \\ &\approx \mathbb{E}_{y \leftarrow D_0} \Pr[\mathsf{S}^{\mathcal{O}_{c_y}^{(1)}} = 1] \\ &\approx \mathbb{E}_{y \leftarrow D_0} \Pr[\mathcal{A}(\mathsf{K}(c_y)) = 1] \le \mathsf{negl}(\lambda).\end{aligned}$$

Each approximation here has negligible additive error, uniformly over the indicated programs. The first and third use the same simulator on $\mathcal{G}$; the middle one uses only the one-query response distributions. Consequently, a constant number of repetitions with freshly compiled programs distinguishes close from far distributions with bounded error for all sufficiently large parameters, giving $\mathsf{SZK} \subseteq \mathsf{BQP}$.

For perfect equivalence, use the product construction for $\mathsf{SD}_{0,1/2}$ in Theorem 3.6 and let $\mathcal{G}$ contain the identical-distribution functionalities and the constants. The compiler J is now perfectly correct, and the middle comparison above is exact. The same test decides $\mathsf{SD}_{0,1/2}$, so Lemma 3.5 gives $\mathsf{HVPZK}|_{\mathrm{am}} \subseteq \mathsf{BQP}$. In either case, the finitely many smaller instances can be hardwired. □

E SEQ Is Occasionally Strictly More Secure than Testable Security

E.1 A Separation from Testable Implementations

SEQ security implies best-possible testable security, but the SEQ interface need not itself be testable. We give a simple pair of classical sampling functionalities whose SEQ implementations reveal strictly less information than any testable implementations. The channels use only an input qubit and an

output qubit. We then show how a fixed output marker makes the SEQ interface testable, whereas one-time VBB security survives such output padding.

Consider the sampling functionalities

$$f_0(x;r) = 0, \qquad f_1(x;r) = (1-x)r, \qquad x, r \in \{0,1\}, \tag{4}$$

where r is uniform. Thus both functionalities output zero at $x = 1$, while at $x = 0$ they output zero and a fair bit, respectively. Let $\mathcal{Q} = \mathcal{X} \otimes \mathcal{Y}$ consist of an input qubit and an output accumulator, and define

$$W\left|x, y\right\rangle = \left|x, y \oplus (1-x)\right\rangle, \qquad \Phi_0(\rho) = \rho, \qquad \Phi_1(\rho) = \frac{\rho + W\rho W^\dagger}{2}. \tag{5}$$

The first channel does nothing; the second applies the controlled bit flip W with probability $1/2$. On input $|x, 0\rangle$, measuring $\mathcal{Y}$ after one channel evaluation gives exactly $f_0(x;r)$ or $f_1(x;r)$.

Theorem E.1 (SEQ can reveal less than testable implementations)**.** *Given the SEQ interface for an unknown channel from* (5)*, every adversary has distinguishing advantage at most* $2/3$*, even with arbitrarily many adaptive, controlled, or inverse queries. In contrast, any individually correct testable implementation of the corresponding sampling functionality in* (4) *permits distinguishing advantage* 1*.*

Proof. The SEQ bound follows from a private subspace on which every query acts as the identity. In both experiments, the initial private state has a constant component in such a subspace, and these components give the adversary exactly the same interaction.

Write $|\eta\rangle = |0\rangle_{\mathcal{P}} |0\rangle_{\mathcal{C}}$ for the initial private state, and let

$$|a\rangle = |0\rangle_{\mathcal{P}} |1\rangle_{\mathcal{C}}, \qquad |b\rangle = |1\rangle_{\mathcal{P}} |1\rangle_{\mathcal{C}}.$$

Consider the following isometries:

$$\begin{aligned} J\left|\varphi\right\rangle &= \left|\varphi\right\rangle\left|\eta\right\rangle, \\ V_0\left|\varphi\right\rangle &= \left|\varphi\right\rangle\left|a\right\rangle, \\ V_1\left|\varphi\right\rangle &= \frac{\left|\varphi\right\rangle\left|a\right\rangle + W\left|\varphi\right\rangle\left|b\right\rangle}{\sqrt{2}}. \end{aligned}$$

Tracing out the private register in V_i gives Φ_i. Since the ranges of J and V_i are orthogonal, the SEQ oracle of Definition 4.2 is precisely

$$O_i = I - 2M_iM_i^\dagger, \qquad M_i = \frac{J - V_i}{\sqrt{2}}. \tag{6}$$

Indeed, this unitary swaps $J\left|\varphi\right\rangle$ with $V_i\left|\varphi\right\rangle$ and fixes the orthogonal complement of their ranges.

Now define the private states

$$|\zeta_0\rangle = \frac{|\eta\rangle + |a\rangle}{\sqrt{2}}, \qquad |\zeta_1\rangle = \frac{|\eta\rangle + \sqrt{2}\,|a\rangle}{\sqrt{3}}.$$

For every public state $|\varphi\rangle$, direct substitution gives

$$V_i^\dagger(\left|\varphi\right\rangle\left|\zeta_i\right\rangle) = \langle\eta|\zeta_i\rangle\left|\varphi\right\rangle = J^\dagger(\left|\varphi\right\rangle\left|\zeta_i\right\rangle).$$

Thus $M_i^\dagger(\left|\varphi\right\rangle\left|\zeta_i\right\rangle) = 0$, so O_i acts as the identity on $\mathcal{H}_{\mathcal{Q}} \otimes \mathrm{span}\{|\zeta_i\rangle\}$. Its orthogonal complement is also invariant. Every operation on public registers, including controlled or inverse oracle queries, preserves this private-subspace decomposition. Consequently, the coherence between the two private subspaces vanishes when the private registers are traced out at the end of any interaction.

Fix an arbitrary adversary. Let σ be its final state when every oracle query acts as the identity, and let τ_0, τ_1 be its final states conditioned on the complementary private subspaces in the two experiments. Since

$$|\langle\zeta_0|\eta\rangle|^2 = \frac{1}{2}, \qquad |\langle\zeta_1|\eta\rangle|^2 = \frac{1}{3},$$

the adversary's actual final states satisfy

$$\rho_0 = \frac{1}{2}\sigma + \frac{1}{2}\tau_0 = \frac{1}{3}\sigma + \frac{2}{3}\left(\frac{1}{4}\sigma + \frac{3}{4}\tau_0\right),$$
$$\rho_1 = \frac{1}{3}\sigma + \frac{2}{3}\tau_1.$$

They share the same component $\sigma/3$. Their trace distance, and hence the distinguishing advantage, is at most $2/3$. The argument places no bound on the adversary's running time or number of queries.

For a testable implementation, initialize the evaluator on $x = 0$, with fresh zero output and work registers and its designated pure program state. Call the joint initial state $|\Omega\rangle$. Apply Eval, apply Z to the output bit, apply $\mathsf{Eval}^\dagger$, and test the projector $|\Omega\rangle\langle\Omega|$ using the reflection and the known input/workspace initialization. If the output-one probability is p, this test accepts with probability

$$\left|\langle\Omega|\,\mathsf{Eval}^\dagger \cdot Z \cdot \mathsf{Eval}\,|\Omega\rangle\right|^2 = (1-2p)^2. \tag{7}$$

For perfect correctness this is 1 for f_0 and 0 for f_1. □

The example separates the SEQ interface from every testable implementation even though its honest output distributions are elementary. Its privacy bound concerns the entire interaction with one initialized oracle, rather than only one oracle query.

E.2 Recognizable Outputs in the SEQ Interface

In this section, we establish a sufficient condition on the channel for the SEQ interface to be testable: if every channel output is orthogonal to some efficiently preparable pure state. This condition also prevents a probe in that state from entering the computed subspace and causing a reverse evaluation.

Lemma E.2 (Reflection from a missing output direction)**.** *Let Φ be a channel on $\mathcal{Q}$. Suppose we are given an efficient preparation circuit for a pure state $|\chi\rangle \in \mathcal{Q}$ satisfying*

$$\Phi^\dagger(|\chi\rangle\langle\chi|) = 0, \tag{8}$$

where $\Phi^\dagger$ is the adjoint map. Then two queries to O_Φ^{SEQ} suffice to implement the coherent test for its designated initial private state, with all probe registers restored. In particular, the SEQ interface is testable.

Proof. Let $|\eta\rangle = |0^m\rangle_{\mathcal{P}}|0\rangle_{\mathcal{C}}$ be the initial private state. Define isometries

$$J|\varphi\rangle = |\varphi\rangle|\eta\rangle, \qquad V|\varphi\rangle = U_\Phi(|\varphi\rangle|0^m\rangle)|1\rangle_{\mathcal{C}}.$$

Their ranges are orthogonal, and the SEQ unitary swaps $J|\varphi\rangle$ with $V|\varphi\rangle$, acting as the identity on the orthogonal complement of these two ranges.

Condition (8) is equivalent to

$$V^\dagger(|\chi\rangle\langle\chi| \otimes I)V = 0.$$

Consequently, $|\chi\rangle \otimes \mathcal{H}_{\mathcal{PC}}$ is orthogonal to the entire computed subspace. For any private vector $a|\eta\rangle + |h\rangle$, where $\langle\eta|h\rangle = 0$, a query on the probe $|\chi\rangle$ gives

$$O_\Phi^{\mathsf{SEQ}}\big(|\chi\rangle(a|\eta\rangle + |h\rangle)\big) = aV|\chi\rangle + |\chi\rangle|h\rangle.$$

The first term has its query register orthogonal to $|\chi\rangle$, and the second has query register exactly $|\chi\rangle$. Coherently flip a fresh test bit when the query register is orthogonal to $|\chi\rangle$, then uncompute the oracle query. The probe returns to $|\chi\rangle$, and the test bit flips precisely on the private component $|\eta\rangle$. The probe can be prepared and tested efficiently by hypothesis. □

For any channel Φ on $\mathcal{Q}$, one can enforce this condition by extending the query register with a success bit whose incoming value is ignored:

$$\widehat{\Phi}(\rho_{\mathcal{Q}b}) = \Phi\big(\mathrm{Tr}_b\, \rho_{\mathcal{Q}b}\big) \otimes |1\rangle\langle 1|_b\,. \tag{9}$$

Since the success bit is independent of Φ itself, intuitively this means that the extra security one may expect from SEQ (compared to testable programs) is simply more often preventing the adversary from knowing when the evaluation succeeded. Here $\widehat{\Phi}$ is a channel on the entire enlarged query register, including arbitrary incoming contents of b.

Finally, we point out that one-time VBB security (Definition D.1) is preserved when a success bit is appended to the output, unlike SEQ. Given a one-time VBB compiler for f, keep its compiled program and wrap its evaluator to append 1, thereby implementing $\widehat{f}(x;r) = (f(x;r), 1)$. An attack on this program is also an attack on the original program with the public wrapper applied. Its one-time VBB simulator can answer its single query to f by querying $\widehat{f}$ and discarding the bit. Thus the same output padding that makes the SEQ interface testable does not affect one-time VBB security.