# Towards compressed permutation oracles

Dominique Unruh

University of Tartu

May 26, 2023

**Abstract**

Compressed oracles (Zhandry, Crypto 2019) are a powerful technique to reason about quantum random oracles, enabling a sort of lazy sampling in the presence of superposition queries. A long-standing open question is whether a similar technique can also be used to reason about random (efficiently invertible) permutations.

In this work, we make a step towards answering this question. We first define the compressed permutation oracle and illustrate its use. While the soundness of this technique (i.e., the indistinguishability from a random permutation) remains a conjecture, we show a curious 2-for-1 theorem: If we use the compressed permutation oracle methodology to show that some construction (e.g., Luby-Rackoff) implements a random permutation (or strong qPRP), then we get the fact that this methodology is actually sound for free.

## Contents

**1 Introduction** **1**

**2 Preliminaries** **4**

**3 Compressed function oracles** **5**

**4 Compressed permutation oracles** **11**

**5 Towards compressed permutations** **13**

## 1 Introduction

The random oracle [6] is a powerful heuristic[1] for cryptographic security proofs. It allows us to abstract from the gritty details of the definition of a hash function and to imagine it to be just a random function. We can then use powerful reasoning techniques such as lazy sampling to make security proofs simpler or, in many cases, possible in the first place. (Lazy sampling refers to the technique of choosing the outputs of the random oracle "on demand", when they are first accessed.) These techniques are useful even if we are not in the random oracle model. For example, when working with a pseudorandom function, the first step in a proof is often to replace it by a fictitious random function. Quite similar to the random oracle are random permutations (to model cryptographically-strong permutations), or ideal ciphers (a heuristic model for block ciphers, basically a key-indexed family of random permutations). In the standard model, random permutations occur in security proofs involving pseudorandom permutations (e.g., in protocols involving block ciphers). In such proofs, we often consider invertible random permutations, i.e., we also give the adversary access also to the inverse of the permutation. All of this can be handled very nicely using lazy sampling.

At least, this is the situation in classical cryptography. Once quantum (or post-quantum) cryptography enters the picture, using the random oracle becomes much harder. This is because the quantum random oracle gives the adversary superposition-access to the random oracle. That is, the adversary can query

[1] In general, this heuristic is not sound: There are contrived protocols which are secure in the random oracle model but insecure when the oracle is instantiated with *any* hash function [11]. However, in practice the random oracle model has proven to be a very good heuristic. Readers who reject heuristics in security proofs may still enjoy the results in this work as a result about generic query complexity, or as a technique for security proofs involving pseudorandom permutations.

the random oracle on a superposition of many different values. Then lazy sampling as in the classical case does not work any more: The adversary could query the oracle on a superposition of all inputs already in the very first query. If we were to sample the oracle at all the sampled positions, this would mean sampling the whole function in one go. But that goes against the very idea of lazy sampling. Furthermore, we cannot just measure where the oracle is queried as this would disturb the adversary state, and we need to make sure that our technique does not influence the way in which the adversary is entangled with the random oracle (in a way that the adversary can notice).

The above does not mean that the random oracle is unusable in the quantum setting. A number of techniques have been developed for handling the random oracle (history-free reductions, $2q$-wise independent functions, semi-constant distributions, small-range distributions, one-way to hiding (O2H) theorems, polynomial method, adversary method, see the related work below). However, none of these have the general applicability of the lazy sampling method, and they are often much harder to use. Then, surprisingly, Zhandry [30] discovered that a variant of lazy sampling is actually possible with quantum random oracles, although it is not as simple (and as general) as in the classical case. We refer to this technique as Zhandry's "compressed oracle technique". (We give more details about it below.)

However, when talking about (invertible) random permutations, the situation is much more limited. The abovementioned tools are specific to the random function case.[2] To the best of our knowledge, no hardness results are known about invertible random permutations, not even simple query complexity results such as the hardness of searching an input with certain properties. As a consequence, we do not know anything about the post-quantum security of cryptosystems built from invertible permutations, such as the industry-standard SHA3 [22].

The present work attempts *a first step* towards closing this gap. We present a sufficient condition for a variant of the compressed oracle technique to work also for random permutations.

**The compressed oracle model.** Zhandry [30] presented a different way to see the random oracle. The traditional quantum random oracle is modeled by giving an adversary access to the unitary operation $|x,y\rangle \mapsto |x, y \oplus h(x)\rangle$ where $h$ is a uniformly randomly chosen function. (I.e., all outputs of $h$ are chosen independently.) Given such a unitary, the adversary can evaluate $h$ in superposition. Now Zhandry showed that the random oracle can be replaced (in an indistinguishable way) by a random oracle that keeps a lazily evaluated function in a separate register $H$ (inaccessible to the adversary). That is, initially that register $H$ contains $|\varnothing\rangle$ where $\varnothing$ represents the empty partial function. Then, upon a query with input $x = x_0$, the function will be updated to contain a superposition of all $|x_0 \mapsto y\rangle$ (for different $y$) and $y$ will be the result of the query. (Here $x_0 \mapsto y$ is the partial function defined only at input $x_0$.) Further queries can add more entries to this function, and if, say, $H$ contains $|h\rangle$ and $h(x) \neq \bot$, and we query the oracle at $x$, we get $h(x)$. When the adversary uncomputes some information that it computed before, the corresponding output in $h$ can become undefined again. And all of this is possible in superposition between different inputs. Now all of this is extremely simplified, and hold only up to some error terms that are annoying but necessary. The advantage of this model is that we can, within limits (due to the annoying error terms), treat the random oracles as if it did lazy sampling even in the quantum setting. We give more details in Section 3.

**Compressed permutations.** In this work, we ask the question whether we can extend the idea above to random permutations. After all, in the classical case, lazy sampling works for random permutations is almost as easy as for random functions. However, the compressed oracle has so far withstood all attempts to be ported to the permutation setting. (See the related work below.) But before we come to our contribution, let us first make explicit why we are interested in a compressed oracle for random permutations (compressed permutation oracle, CPO). There are two main ways in which we could use this technique:

- We are analyzing a cryptographic scheme that *uses* an invertible permutation. And we wish to model that permutation in an idealized way (random oracle like). E.g., we might want show that the Sponge construction [7], where the block function is an invertible permutation (as is the case with SHA3) implements a pseudorandom random function. We would then replace the invertible permutation in the proof by a CPO and then use the features of the CPO (such as "lazy sampling") to make the proof simpler.

[2] Except for the O2H theorem. Some variants of the O2H theorem apply to arbitrarily distributed functions [2], in particular to invertible permutations. (An invertible permutation can be modeled as a function $f : \{0,1\} \times \{0,1\}^n \to \{0,1\}^n$, uniformly sampled from the set of all functions where $f(0,\cdot)$ is a permutation and $f(1,\cdot)$ its inverse.) However, we are not aware of any work that makes use of this.

Analogously, we can use the technique for an analysis of a scheme using an ideal cipher. (Since the ideal cipher is simply a family of invertible permutations.)

- We are analyzing a cryptographic scheme that *implements* an invertible permutation or a strong pseudorandom permutation (PRP, i.e., a secure blockcipher). For example, in the classical case, the four-round Luby-Rackoff construction [21] is known to be a strong PRP if the round function is a (noninvertible) pseudorandom function [21]; we do not know yet whether an analogous result holds in the quantum case.[3] To show this, it is sufficient to show that Luby-Rackoff, using a random function as the round function, is indistinguishable from a random permutation (given queries in both directions).

  Now there can be different approaches for the latter, but one promising avenue for this would be: (a) Show that Luby-Rackoff is indistinguishable from a CPO (using the fact that the CPO gives us an explicit list of all queries to the random permutation in the proof). (b) Use that a CPO is indistinguishable from a random invertible permutation.

In this paper, we are specifically interested in the second use case. The problem with that use case is that, even if we show (a), we still do not know whether a CPO is indistinguishable from a random permutation (i.e., (b)). We show that this is not a problem. Specifically, we show the following almost circular seeming result:

> **Main contribution:** If some construction (say based on a random oracle) is indistinguishable from a CPO (i.e., we have (a)), then the CPO is indistinguishable from an invertible permutation (i.e., we get (b)).

So, if we show (a), we get (b) for free!

This has two benefits:

- If we are in the second use case, we do not need to worry whether the CPO is indeed indistinguishable from an invertible permutation. We simply can focus on the (admittedly still rather hard) problem of analyzing the construction.
- This gives a new approach towards showing that the CPO technique works – if we can show (a) for any construction (even some practically irrelevant one), then we know that the CPO is indistinguishable from an invertible permutation in general. So we can then also use it, e.g., in the first use case (say, quantum security of SHA3).

This brings us a step closer towards being able to handle invertible permutations in the quantum setting.

**Related work.** *Quantum random oracles.* [26, 19] showed that finding preimages in the random oracle is hard ([10] showed this in worst-case setting.) [9] introduced "history-free reductions" which basically amounts to replacing the random oracle by a different function right from the start. [31] showed that random oracles can be simulated using $2q$-wise independent functions. Based on this, [26] introduces a technique for extracting preimages of the random oracle. [31] introduces the "semi-constant distributions" technique that allows us to program the random oracle in many random locations with a given challenge value without the adversary noticing. [29] improves upon this with the "small-range distribution" technique that allows us to simulate random oracles using random looking functions with a small range. [28] shows that random oracles are collision resistant (this is generalized by [23, 16, 4] to the case of non-uniformly distributed functions with independently sampled outputs). Collision-resistance of the random oracle is generalized to the "collapsing property" which allows us to show that measuring the output of the random oracle effectively measures the input [25]. More general methods for problems in quantum query complexity (not limited to random oracles) include the polynomial method [5] and the adversary method [1]. [3] shows that the difficulties of using the quantum random oracle are not just a matter of missing proof techniques, but that in certain cases classically secure schemes are not secure in the quantum random oracle model.

*Compressed oracles.* Compressed oracles were introduced in [30] and used there to show indifferentiability of the Merkle-Damgård construction, as well as security of the Fujisaki-Okamoto transform. [12] generalizes [30] to Fourier transforms over abelian groups, thus allowing random functions with a range different from $\{0,1\}^n$. Different from [30], they do not have a compression/decompression algorithm but instead reason using invariants that are expressed in a basis different from the computational basis. They also introduce support for parallel queries. [15] generalizes [30] to non-uniformly distributed functions,

[3] We know that four rounds are not sufficient [20], but nothing excludes that, e.g., five-round Luby-Rackoff could be a strong qPRP. [18] proves that four-round Luby-Rackoff is a qPRP (but not a strong one) but their result contains a flaw and the fix is work in progress [17].

but only for the case where all outputs are independently sampled. (This is similar to what we achieve in our reformulation of [30] in Section 3, although we additionally get rid of the Fourier transform.)

*Random permutations.* [28] shows that random functions are indistinguishable from (noninvertible) random permutations. This allows us to derive results for random permutations from results for random functions. [27] shows the existence of quantum-secure pseudorandom permutations (qPRP, secure under superposition-queries of the function and its inverse) from quantum one-way functions. In particular, this implies that a random invertible permutation can be efficiently simulated.[4] However, [27] does not give us any technique for analyzing schemes that *use* a qPRP. When analyzing such a scheme we would replace the qPRP by an invertible random function in the proof, and the techniques from the present paper could be helpful.

*Security of the sponge construction.* The sponge construction was proposed by [7]. In the classical random oracle model, security of the sponge construction was shown by [8], both when the sponge is based on random functions and on invertible random permutations. They showed indifferentiability, which implies many other properties such as collision-resistance, pseudorandomness, and more. In the quantum setting, collision-resistance and the collapsing property from [25] were shown in [13] in the random function case. Quantum pseudorandomness of the sponge was shown by [14] but only in the case where the underlying round function is secret (the adversary cannot query it). Indifferentiability in the quantum setting was shown by [15] in the random function case. All those results immediately imply the corresponding results in the non-invertible random permutation case since random functions and permutations are indistinguishable [28]. However, for invertible random permutations, no quantum results are known.[5]

**Organization.** In Section 2 we introduce relevant notational conventions. In Section 3, we present compressed oracles for random functions. Specifically, we recap and give a new, more streamlined view on Zhandry's technique. (We recommend readers familiar with Zhandry's technique to at least skim it because it will introduce some of the formalism for later.) We also give a short example how it is used. In Section 4, we formulate the compressed permutation oracle and give an example how to use it. In Section 5, we prove our main result: If some construction implements the compressed permutation oracle, then the compressed permutation oracle is indistinguishable from a random permutation. In the back matter, we provide a list of theorems, a symbol and a keyword index, as well as the bibliography.

# 2 Preliminaries

**Total and partial functions.** Throughout this work, we will extensively deal with total and partial functions to describe states, queries, and invariants. For sets $D, R$, let $D \to R$ be the *total functions* from $D$ to $R$, and $D \hookrightarrow R$ the *total injections* (i.e., injective total functions) from $D$ to $R$. Furthermore, $D \rightharpoonup R$ are the *partial functions*. For a partial function $f : D \rightharpoonup R$, $\operatorname{dom} f \subseteq D$ is the domain (inputs on which $f$ is defined) and $\operatorname{im} f$ is the image of $f$.

$\varnothing$ is the *empty partial function* (defined nowhere).

**Quantum-related notation.** Quantum states are elements of a (not necessarily finite-dimensional) Hilbert space $\mathcal{H}$. We usually represent quantum states with greek letters (e.g., $\psi$) and use ket-notation ($|x\rangle$) to refer to basis states of the computational basis unless specified otherwise, and $\langle x|$ is the adjoint of $|x\rangle$ ($\langle x| = |x\rangle^\dagger$). (I.e., $|x\rangle$ for $x \in X$ form an orthonormal basis of $\mathbb{C}^X$.) $\|\psi\|$ is the norm of $\psi \in \mathcal{H}$, and $|||A|||$ is the operator norm of the bounded operator $A : \mathcal{H} \to \mathcal{H}'$. For $S \subseteq \mathcal{H}$, $\operatorname{span} S$ is the *(closed) span* of $S$, i.e., the smallest topologically closed subspace of $\mathcal{H}$ containing $S$. *Projector* always means orthogonal projector.

We will often need to consider the distance between a vector and a subspace: For two vectors $\psi, \psi' \in \mathcal{H}$, we write $\psi \overset{\varepsilon}{\approx} \psi'$ to denote $\|\psi - \psi'\| \leq \varepsilon$. And if $S$ is a closed subspace of $\mathcal{H}$, then we write $\psi \overset{\varepsilon}{\approx} S$ to denote $\exists \psi' \in S.\ \psi \overset{\varepsilon}{\approx} \psi'$.[6]

[4] If we implement the underlying quantum one-way function using a random oracle, and we simulate that random oracle with the method from [31] or [30], then we even get a simulation without computational assumptions.

[5] [24] has a proof of collision resistance but it was found to be flawed and withdrawn.

[6] Or equivalently: $\|(1 - P)\psi\| \leq \varepsilon$ where $P$ is the projector onto $S$.

**Quantum oracle queries.** Throughout the paper, we will frequently refer to oracle queries. Thus, we fix some variables once and for all: $D$ always refers to the domain and $R$ to the range of the function. (I.e., queries are always made to a function $h : D \to R$.) We will also fix once and for all the sizes of $D$ and $R$ as $M := |D|$ and $N := |R|$. (In particular, $D, R$ are assumed to be finite.)

We furthermore assume a commutative group operation $\oplus$ on $D$ and on $R$ with the property $x \oplus x = 0$. (For example, $D = R = \{0,1\}^n$ and $\oplus$ is bit-wise XOR.)

A query to a fixed function $f : D \to R$ can then be implemented by the unitary $U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$. (The fact that this is unitary follows from the fact that $\oplus$ is a group operation.) However, we will more often be interested in queries to a function that is also stored in a quantum register: For a set $\mathsf{Func} \subseteq D \rightharpoonup R$ that will always be clear from the context, define the unitary $\mathsf{StO}$ (for "standard oracle") on $\mathbb{C}^D \otimes \mathbb{C}^R \otimes \mathbb{C}^{\mathsf{Func}}$ by:

$$\mathsf{StO}|x\rangle|y\rangle|h\rangle = |x\rangle \left|y \oplus h(x)\right\rangle |h\rangle$$

Here $y \oplus h(x)$ is defined to be $y$ when $h(x) = \bot$. (This latter case only arises if $\mathsf{Func}$ contains partial functions.)

# 3 Compressed function oracles

We will now recapitulate and rephrase *Zhandry's compressed oracle technique* [30]. trying to emphasize more the separation between implementation issues (encoding via "databases" etc.) and the core concepts. Then we proceed to give a different view on the technique that does not involve Fourier transforms and which is, in our opinion, conceptually simpler.

We will often refer to the compressed oracle as *compressed function oracle* or *CFO* to distinguish it from the compressed permutation oracle introduced later.

A reader who wishes to skip this part and to directly learn our new technique can skip ahead to the mini-summary at the end of this section (page 11).

In a nutshell, the compressed oracle technique is a way to simulate/implement a quantum random oracle (i.e., a uniformly random function $h : D \to R$ to which an adversary or quantum algorithm has superposition query access) in a way that has the following crucial features:

- *The adversary cannot distinguish between the original random oracle and the simulation.* This allows us to use the simulation in proofs instead of the original oracle.
- *The simulation uses an internal state that has a small representation.* This is not the case for trivial implementations of the random oracle: Those would have to pick and store the value table of the random function at the beginning. This value table would require $|D| \cdot \log|R|$ classical bits which is infeasible for typical size of the domain $D$. In contrast, the compressed oracle only requires roughly $q(\log|D| + \log|R|)$ qubits after $q$ queries to the random oracle.
- *The simulation keeps track where the random oracle was already queried, and what the result of that query is.* E.g., if the adversary queries $h(x)$ (possibly in superposition between different values $x$), and gets $y := h(x)$, then the simulation will keep a record that a query $x \mapsto y$ was performed (or a superposition of such records). While this is trivial in the classical case, it is highly surprising that this is possible in the quantum case: Naively keeping a record of the queries would entangle the adversary's state with the state of the compressed oracle, something the adversary might detect.[7] Having this record is the arguably the main advantage of the compressed oracle technique as a proof technique. For example, it allows us to formulate invariants such as "the adversary has not yet queried an $x$ with $h(x) = 0$".
- *The simulation is efficient.* That is, its runtime is polynomial in the number of queries performed by the adversary, and the bitlengths of the inputs and outputs of $h$. This is closely related to

[7] For example, the adversary might initialize a register $X$ with $\sum_x \frac{1}{\sqrt{M}}|x\rangle$, then perform a superposition query with input $x$. The compressed oracle needs to record the query $x \mapsto y$ (in superposition between different $x$). Now the register $X$ is entangled with the compressed oracle's record. (Or, if the compressed oracle would measure the query input, the register $X$ would collapse to a single value.) Now the adversary might wish to distinguish whether the compressed oracle records its queries or not. For that purpose, the adversary uncomputes the previous query. Now $X$ would be in the original state when using the original random oracle; the adversary can check whether this is the case. But if the compressed oracle keeps a record of the query $x \mapsto y$, the state $X$ will not be in its original state but entangled with the compressed oracle. So in order to be indistinguishable, the compressed oracle needs to forget the query (i.e., erase the record $x \mapsto y$ from its state). In other words, the compressed oracle needs to not only record queries, but also "unrecord" queries in case of uncomputations. Since the compressed oracle does not know a priori whether a given query is a computation or an uncomputation (or something in between), it would seem impossible to solve this problem. The surprising fact of the compressed oracle technique is that it does solve this problem, almost as a side effect.

the fact that the internal state has a small representation. Previous approaches for efficiently implementing/simulating the quantum random oracle either required computational assumptions (simulation via quantum pseudorandom functions [29]) or required the simulator to know the number of queries that the adversary will perform at the outset of the simulation (simulation via $2q$-wise independent functions [32]).

Through the rest of this section, we present our reformulation of Zhandry's technique before considering permutations.

**Standard oracle.** Consider the original quantum random oracle. This oracle initially classically samples a random function $h \xleftarrow{\$} (D \to R)$. And then a query to the function $h$ is implemented by a unitary $U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus h(x)\rangle$ on the adversary's query registers $X, Y$.

It is easy to see that this is perfectly indistinguishable from the following construction (called the *standard oracle* in [30]):[8] An additional quantum register $H$ is initialized with $\sum_{h \in D \to R} \frac{1}{\sqrt{|D \to R|}} |h\rangle$, i.e., with the uniform superposition of all possible functions. The adversary does not get access to this register $H$, but instead the oracle query is changed to be the unitary $\mathsf{StO} : |x\rangle|y\rangle|h\rangle \mapsto |x\rangle|y \oplus h(x)\rangle|h\rangle$ on registers $X, Y, H$.

To better understand the following steps, imagine that the register $H$ consists of many separate registers $H_x$ ($x \in D$), each $H_x$ storing the output $h(x)$. (That is, $h$ is represented as a value table in $H$ with $H_x$ being the table entries.) Each $H_x$ has Hilbert space $\mathbb{C}^R$.

**Compressed oracle.** Next, we transform the oracle into yet another representation. First, we extend the registers $H_x$ to allow for a value $|\bot\rangle$, i.e., the Hilbert space of a single $H_x$ is $\mathbb{C}^{R \cup \{\bot\}}$. This means that the register $H$ now contains not only total functions $h$, but can also contain superpositions of partial functions. ($\bot$ denoting an undefined output.)

Intuitively, $|\bot\rangle$ in some $H_x$ will mean that the corresponding $h$-output is not yet determined, i.e., that any value is still possible. In particular, having $|\bot\rangle$ in all registers $H_x$ (i.e., having the empty partial function $|\varnothing\rangle$ in $H$) should correspond to the initial state of the random oracle.

We make this more formal by defining an encoding/decoding operation to map between states that do not use $|\bot\rangle$ (as in the standard oracle) and states that do use $|\bot\rangle$ (compressed states).

Let $Q$ denote the quantum Fourier transform on $\mathbb{C}^R$. We extend it to work on the register $H_x$ by defining $Q|\bot\rangle := |\bot\rangle$. (In [30], the specific case $R = \{0,1\}^n$ is considered. In this case the quantum Fourier transform is simply a Hadamard gate on each qubit in $H_x$. But in [12], the case for general abelian groups $R$ is considered and other quantum Fourier transforms are used.) Let $U_\bot$ be the unitary with $U_\bot|\bot\rangle = |0\rangle$, $U_\bot|0\rangle = |\bot\rangle$, $U_\bot|y\rangle = |y\rangle$ for $y \neq 0$. Let $\mathsf{Decomp}_1 := Q \cdot U_\bot \cdot Q^\dagger$ (the *decompression operation*).

In the standard oracle, $H_x$ has initial state $\sum_y \frac{1}{\sqrt{|R|}} |y\rangle$. If we apply $\mathsf{Decomp}_1^\dagger$ to it, we get

$$\sum_y \frac{1}{\sqrt{|R|}} |y\rangle \overset{Q^\dagger}{\longmapsto} |0\rangle \overset{U_\bot^\dagger}{\longmapsto} |\bot\rangle \overset{Q}{\longmapsto} |\bot\rangle.$$

Thus, by applying $\mathsf{Decomp}_1^\dagger$ to all registers $H_x$ in the initial state of the standard oracle, we get $|\bot\rangle$ in every $H_x$. This leads to the following idea: Initialize all $H_x$ with $|\bot\rangle$. And whenever we want to perform an oracle query, we decompress all $H_x$ by applying $\mathsf{Decomp}_1$ (for the initial state, this gives the initial state of the standard oracle). Then we apply $\mathsf{StO}$ (the standard oracle).[9] And then we compress all $H_x$ again by applying $\mathsf{Decomp}_1^\dagger$. This will lead to exactly the same behavior as the standard oracle (since successive $\mathsf{Decomp}_1$, $\mathsf{Decomp}_1^\dagger$ pairs cancel out).

In other words, we define the compressed oracle to be the oracle with the initial state $|\bot\rangle \otimes \cdots \otimes |\bot\rangle$

[8] The indistinguishability formally follows from the fact that the query commutes with a computational basis measurement of the register $H$, and the fact that if that computational basis measurement is performed at the beginning of the execution, then it is equivalent to uniformly (classically) sampling $h$.

[9] Since we extended the space of the register $H_x$ to be $R \cup \{\bot\}$, $\mathsf{StO}$ must be well-defined also on states where one or many of the $H_x$ are $|\bot\rangle$, i.e., on superpositions of partial functions. See the preliminaries for the precise definition for that case. Note, however, that since we start from an initial state that is a superposition of total functions, this case never arises no matter what queries to $\mathsf{StO}$ we perform, *unless we apply some other operations to the oracle register $H$ directly.*

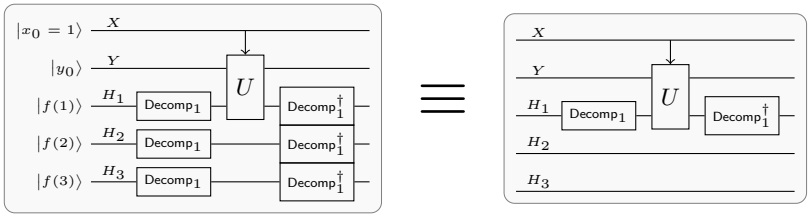



Figure 1: Operation of CFO for fixed $x_0 := 1$. $U$ denotes the operation $|y_0\rangle|y\rangle \mapsto |y_0 \oplus y\rangle|y\rangle$.

in register $H$, and that applies the following unitary to $X, Y, H$ on each query:

$$\mathsf{CFO} := (I_X \otimes I_Y \otimes \mathsf{Decomp}^\dagger) \cdot \mathsf{StO} \cdot (I_X \otimes I_Y \otimes \mathsf{Decomp}) \quad \text{with} \quad \mathsf{Decomp} := \bigotimes_{x \in D} \mathsf{Decomp}_1 \,. \quad (1)$$

Now CFO is perfectly indistinguishable from the standard oracle.

**The size of the compressed oracle.** So far, we have seen that the compressed oracle CFO simulates the standard oracle. But why is it useful? To see this, we will think of the register $H$ as containing partial functions: The basis states of $H$ are $|y_{x_1}, \ldots, y_{x_N}\rangle$ for $y_{x_1} \in R \cup \{\perp\}$ where $x_1, \ldots, x_N$ are the elements of $D$. This is the value table of a partial function $f : D \rightharpoonup R$. We will identify $|y_{x_1}, \ldots, y_{x_N}\rangle$ with $|f\rangle$. In particular, the initial state of CFO is then $|\perp, \ldots, \perp\rangle = |\varnothing\rangle$. ($\varnothing$ is the completely undefined partial function.)

Consider a state $|x_0\rangle|y_0\rangle|f\rangle$ before a query to the compressed oracle, with $|\mathrm{dom}\, f| \leq \ell$. (The initial state has $\ell = 0$.)

Applying CFO to this state will decompress all $H_x$ (which does not affect $|x\rangle$) apply StO (which does not affect $H_x$ for $x \neq x_0$ for this particular state), and then compress all $H_x$ again. This is illustrated in the left side of Figure 1 for $x_0 := 1$. On all $H_x$ with $x \neq x_0$, $\mathsf{Decomp}_1$ and $\mathsf{Decomp}_1^\dagger$ cancel out (see the right side of Figure 1). Thus, no matter what $x_0$, $y_0$, $f$ are, the resulting state will be a superposition of $f'$ with $f' = f$ except on $x_0$. In particular, $|\mathrm{dom}\, f'| \leq |\mathrm{dom}\, f| + 1 \leq \ell + 1$. Since this holds for any $|x_0, y_0, f\rangle$ with $|\mathrm{dom}\, f| \leq \ell$, this also holds for any superposition of such states. Thus we have shown[10] that any state of the compressed oracle that is a superposition of $|f\rangle$ of size $\leq \ell$ will, after a query, be a superposition of $|f\rangle$ of size $\leq \ell + 1$.

In particular, after $q$ queries, the compressed oracle state is a superposition of partial functions of size $\leq q$. Such a partial function can be represented in approximately $q(\log|D| + \log|R|)$ bits, hence the state of the compressed oracle indeed has a much smaller representation. This justifies the name "compressed oracle".

And we also can see that it indeed "records" queries in some sense: if the state of the oracle contains $|f\rangle$, then every $x \in \mathrm{dom}\, f$ must have been queried. Otherwise we would have $f(x) = \perp$ as in the initial state. The converse does not hold, though, because queries can be uncomputed and thus removed from $f$.

**Efficient implementation.** So far, we do not have an efficiently simulatable oracle because we represent the state of the compressed oracle by giving the complete value table for the partial functions $f$. (Each potential output is stored in a different register $H_x$.) However, an algorithm implementing CFO is free to store the partial functions in a more efficient way, namely as sorted lists of input/output pairs (called a *database* in [30]), leading to a compact state. And an efficient circuit for the unitary CFO can be constructed by only applying $\mathsf{Decomp}_1$ on those entries of the database that are involved in the present query. This then gives the oracle defined in [30]. We omit the details here as they are not relevant for the rest of this paper.

The advantage of separating the definition of the efficient encoding of the compressed oracle state from the conceptual encoding as a partial function is that proofs will not have to consider the concrete encoding with ordered association lists only when analyzing the runtime of the simulation and can use the mathematically simpler concept of partial functions everywhere else. In particular, in information-theoretical proofs, we do not need to consider the efficient encoding at all.

[10] Actually, we have handwavingly sketched it but a formal proof is easy and follows the same ideas.

**Getting rid of the Fourier transform.** So far, we have described the compressed oracle as in Zhandry's original work (although with a different presentation). As presented originally, it would seem that the Fourier transform is an integral part of the idea of the compressed oracle.[11] We will now show that there is a different view which does not involve the Fourier transform at all. Recall the definition of $\mathsf{Decomp}_1 = Q \cdot U_\perp \cdot Q^\dagger$. Using that definition, we can compute what $\mathsf{Decomp}_1$ does to various basis states:

$$|\perp\rangle \overset{Q^\dagger}{\longmapsto} |\perp\rangle \overset{U_\perp}{\longmapsto} |0\rangle \overset{Q}{\longmapsto} \sum_z Q_{z0}|z\rangle =: |*\rangle$$

$$|y\rangle \overset{Q^\dagger}{\longmapsto} \sum_z \overline{Q_{yz}}|z\rangle \overset{U_\perp}{\longmapsto} \underbrace{\sum_z \overline{Q_{yz}}|z\rangle}_{=Q^\dagger|y\rangle} + \underbrace{\overline{Q_{y0}}}_{=\langle *|\, y\rangle} \big(|\perp\rangle - |0\rangle\big) \overset{Q}{\longmapsto} |y\rangle + \langle *|\, y\rangle \big(|\perp\rangle - |*\rangle\big).$$

Note that this calculation did not use that $Q$ is the Fourier transform, only the fact that it is unitary. And the state $|*\rangle$ is simply the first column of $Q$. Which, in case of the Fourier transform, is of course the uniform superposition $|*\rangle = \sum_z \frac{1}{\sqrt{N}}|z\rangle$. However, any other unitary with the same first column would lead to the same result – the definition of $\mathsf{Decomp}_1$ does not actually use the Fourier transform, and it only depends on the first column of $Q$! In fact, the above calculation works even if the first column is not the uniform superposition. For example, if we wish to analyze random oracles that use a random function $h$ that is not uniformly chosen, but where each $h(x)$ is independently chosen according to some distribution $\mathcal{D}$, we take a unitary $Q$ whose first column is $|*\rangle := \sum_z \frac{1}{\sqrt{\mathcal{D}(z)}}|z\rangle$, and now $\mathsf{Decomp}_1$ still maps

$$\begin{aligned} &\mathsf{Decomp}_1 : |\perp\rangle \mapsto |*\rangle \\ &\mathsf{Decomp}_1 : |y\rangle \mapsto |y\rangle + \langle *|\, y\rangle \big(|\perp\rangle - |*\rangle\big). \end{aligned} \tag{2}$$

In fact, we can just take this as the definition of $\mathsf{Decomp}_1$ (relative to a given $|*\rangle$). The operators $Q$ and $U_\perp$ are then just a technical tool to show that $\mathsf{Decomp}_1$ is indeed unitary, and one possible way of implementing $\mathsf{Decomp}_1$ efficiently, but they are not part of its definition.

We can still define an oracle $\mathsf{CFO}$ based on this new $\mathsf{Decomp}_1$ in the same way as before via (1). Except now $\mathsf{CFO}$ will be indistinguishable from the standard oracle that has the initial state $|*\rangle \otimes \cdots \otimes |*\rangle$. Which is indistinguishable from the original random oracle if $|*\rangle$ is the uniform superposition. And if $|*\rangle = \sum_z \alpha_z |z\rangle$, then it is indistinguishable from the a random oracle where each $h(x)$ is sampled to be $y$ with probability $|\alpha_y|^2$.[12] Everything discussed so far still applies. In particular, we still have that the oracle is compressed and records queries: In the compressed state, for any $x$ that has not been queried, $H_x$ will be in state $|\perp\rangle$ (with the intuitive meaning that the value of $h(x)$ is not sampled yet).

Thus, by removing the Fourier transform from the picture, we have generalized the compressed oracle technique to nonuniformly distributed oracles "for free".[13] However, we stress that this approach does not yet allow us to model random permutations because a random permutation $h$ does not have *independently* distributed $h(x)$.[14]

In our opinion, this new view of the compressed oracle has multiple advantages:

[11] [30] considers the special case of a qubit-wise Hadamard which is the Fourier transform over the abelian group $\{0,1\}^n$. [12] generalizes this to Fourier transforms over arbitrary abelian groups.

[12] We could go even farther and use a different $|*\rangle$ for every $x$. This would allow us to analyze oracles where $h(x)$ is picked from different distributions for different $x$.

[13] [15] also generalizes Zhandry's technique to non-uniformly distributed functions. (With the condition that the outputs are independently sampled, i.e., not covering permutations.) However, their presentation still involves Fourier transforms.

[14] We had one *failed* approach how to generalize this to random permutations (and possibly other function distributions). Since we believe that this approach might be natural, we shortly describe it here and why we got stuck trying to use it:

For $S \subseteq R$, we can define $\mathsf{Decomp}_1^S$ to be the $\mathsf{Decomp}_1$ operation for the uniform distribution on $S$. (I.e., $\mathsf{Decomp}_1^S$ is defined by (2) where $|*\rangle := \sum_{y\in S} \frac{1}{\sqrt{|S|}}|y\rangle$.) Then we can define $\mathsf{Decomp}_1^{@x}$ to apply $\mathsf{Decomp}_1^M$ on register $H_x$, where $M$ is the set of all values that are not yet used in other registers. Formally, if $D = \{x_1, \ldots, x_M\}$, $\mathsf{Decomp}_1^{@x_i}|y_1, \ldots, y_M\rangle := |y_1, \ldots, y_{i-1}\rangle \otimes \mathsf{Decomp}_1^{S_i}|y_i\rangle \otimes |y_{i+1}, \ldots, y_M\rangle$ where $S_i := R \setminus \{y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_M\}$. (Here all $y_i \in R \cup \{\perp\}$.) And then we can define a decompression for permutations as $\mathsf{Decomp}_1^{perm} := \mathsf{Decomp}_1^{@x_M}\, \mathsf{Decomp}_1^{@x_{M-1}} \cdots \mathsf{Decomp}_1^{@x_2}\, \mathsf{Decomp}_1^{@x_1}$.

It is reasonably easy to verify that $\mathsf{Decomp}_1|\perp \ldots \perp\rangle = \sum_{h:D\hookrightarrow R} \frac{1}{\sqrt{|D\hookrightarrow R|}}|h\rangle$. So decompressing the initial state indeed leads to a uniform superposition of permutations (more precisely, of injections).

And it is also easy to define an oracle query in this model, namely $\mathsf{CFO}^{perm} := \mathsf{Decomp}^{perm\,\dagger} \otimes \mathsf{StO} \otimes \mathsf{Decomp}^{perm}$.

However, beyond that, things become difficult. First, the definition of $\mathsf{Decomp}_1$ depends on the ordering of the domain $D$. If we would apply the $\mathsf{Decomp}_1^{@x_i}$ in a different order, we would get a different operator $\mathsf{Decomp}_1^{perm}$. Second, it becomes very difficult to understand the behavior of $\mathsf{CFO}^{perm}$. We were unable to give an explicit description of how it operates on a basis state. And it is not clear that $\mathsf{CFO}^{perm}$ maps a state $|y_1 \ldots\rangle$ where at most $\ell$ of the $y_i \neq \perp$ to a superposition of states $|\tilde{y}_1 \ldots\rangle$ where at most $\ell + 1$ of the $\tilde{y}_i \neq \perp$. But if this does not hold, then we do not have a compressed oracle because we have no upper bound on the size of the oracle state.

- It becomes clearer what the essence of the transformation $\mathsf{Decomp}_1$ is (see also the discussion below). To assume that the Fourier transform plays a relevant role in the construction may even hinder understanding of what is really happening.
- There is no need to find a group structure on the range $R$ of the function so that it matches the operation $\oplus$ in the definition of the oracle query unitary $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus h(x)\rangle$. This may lead to less requirements in proofs.
- The technique becomes more general as we are not limited to uniformly distributed functions.

For the remainder of this paper, we will not make use of this potential for generalization and assume that $Q$ is an arbitrary unitary whose first column is the uniform superposition.

**Understanding Decomp.** In order to better understand what the decompression operation does, let us have another look at the definition.

$$\begin{aligned}&\mathsf{Decomp}_1 : |\bot\rangle \mapsto |*\rangle\\ &\mathsf{Decomp}_1 : |y\rangle \mapsto |y\rangle + \underbrace{\langle *|\, y\rangle\big(|\bot\rangle - |*\rangle\big)}_{\text{correction term}}\end{aligned} \tag{3}$$

And thus $\mathsf{Decomp}$ operates as follows:

$$\mathsf{Decomp} : |y_1 y_2 y_3 \ldots\rangle \mapsto |\hat{y}_1 \hat{y}_2 \hat{y}_3 \ldots\rangle + \mathit{correction} \tag{4}$$

where $\hat{y} := y$ for $y \in R$ and $\hat{y} := *$ for $y = \bot$, and where *correction* is a sum of tensor products of "correction terms".

This means that in the compressed oracle state, $|\bot\rangle$ is used to denote the uniform superposition in $H_x$, i.e., an output that is completely undetermined so far. On the other hand, $|y\rangle$ in the compressed oracle state has a somewhat more subtle meaning. Intuitively, we might expect/want that $|y\rangle$ in the compressed oracle state means that the output is $y$. I.e., $|y\rangle$ in the compressed state should translate to $|y\rangle$ in the uncompressed state. In other words, the intuitively natural definition of $\mathsf{Decomp}_1$ would be the definition (4) with the "correction term" removed. Unfortunately, the resulting operation would not be unitary. So the purpose of the correction terms is to stay as close to mapping $|y\rangle$ to $|y\rangle$ as possible, while keeping the operation unitary. Note that the correction terms are small because $\langle *|\, y\rangle = 1/\sqrt{N}$.[15]

This leads to a different view of how $\mathsf{Decomp}_1$ could be derived: Instead of constructing it bottom-up from $Q$ and $U_\bot$, we could use the ansatz that $\mathsf{Decomp}$ is an operator defined as:

$$\begin{aligned}&\mathsf{Decomp} : |\bot \ldots \bot\rangle \mapsto |*\rangle \ldots |*\rangle\\ &\mathsf{Decomp} : |y_1 y_2 y_3 \ldots\rangle \mapsto |\hat{y}_1 \hat{y}_2 \hat{y}_3 \ldots\rangle + \mathit{correction}\end{aligned} \tag{5}$$

where *correction* must be chosen in such a way that $\mathsf{Decomp}$ becomes unitary, such that *correction* is as small as possible, and – most importantly – that the correction terms do not make the compressed oracle state bigger (i.e., when starting with $|y_1 y_2 \ldots\rangle$ where there are at most $\ell$ non-$\bot$ entries, decompressing with $\mathsf{Decomp}$, applying the oracle query operation $\mathsf{StO}$, and then recompressing with $\mathsf{Decomp}^\dagger$, we should get a superposition of states $|y'_1 y'_2 \ldots\rangle$ with at most $\ell + 1$ non-$\bot$ entries). The definitions of $\mathsf{Decomp}$ given above are then just one (although very natural) solution to this ansatz.

Mainly, we presented this approach in (5) to give a different view on the compressed oracle technique (that hopefully gives some intuition about what is going on). But maybe this approach is also one way to extend the compressed oracle technique to more complex cases such as oracles with non-independently chosen outputs or similar. We did not manage to use it for the random permutation case, but maybe future work will.

**The sanitized CFO.** There is a subtle variation of the CFO that we described above. We call it the *sanitized CFO*. Recall that we use $\mathsf{Decomp}_1$ to switch between two representations of the oracle state, the compressed representation (where the initial state is $|\varnothing\rangle$), and the uncompressed representation (where the initial state would be the superposition of all total functions). In the compressed representation,

However, we do not exclude that these problems could be solved and the approach made viable. For example, it might be possible to find some operator that approximately implements $\mathsf{CFO}^{perm}$ and that has an easy description and that does not grow the state too much during a query. But we were unable to find such an operator.

[15] However, the fact that they are small does not, unfortunately, mean that we can ignore them in calculations. They do, in many situations, add up to very relevant errors. In fact, $\mathsf{Decomp}_1$ without the correction terms has operator norm $\sqrt{2}$ which means that the errors can be almost as big as the state itself.

we make strong use of the fact that the oracle register $H$ can contain partial functions. But in the uncompressed representation, partial functions make little sense, so the initial state in that representation is the superposition of only *total* functions. And it follows directly from the definition (1) of the CFO that throughout an execution, the uncompressed state will never contain a partial function (see also footnote 9). We arbitrarily specified the behavior of $\mathsf{StO}$ when it encounters $h(x) = \bot$ to do nothing. Since this case never occurs, we can also define $\mathsf{StO}$ differently, for example:

$$\mathsf{StO_s}\,|x\rangle|y\rangle|h\rangle = \begin{cases} |x\rangle\,|y \oplus h(x)\rangle\,|h\rangle & \text{if } h(x) \neq \bot, \\ 0 & \text{if } h(x) = \bot. \end{cases}$$

That is, this oracle effectively measures whether $h(x) = \bot$ before computing the query and, if so, diverges. (We call this the *sanitized standard oracle* since it removes the case where $h(x)$ is undefined in the uncompressed representation.) Based on this, we can define the *sanitized CFO*:

$$\mathsf{CFO_s} := \mathsf{Decomp}^\dagger \cdot \mathsf{StO_s} \cdot \mathsf{Decomp}\,. \tag{6}$$

It is straightforward to verify that $\mathsf{CFO_s}$ and $\mathsf{CFO}$ are perfectly indistinguishable (by adversaries that do not touch $H$, and given initial state $|\varnothing\rangle$). In particular, $\mathsf{CFO_s}$ is also indistinguishable from a random function. Everything said above about $\mathsf{CFO}$ also applies to $\mathsf{CFO_s}$. (Only difference: $\mathsf{CFO_s}$ is not unitary because it represents a potentially diverging computation.)

If there is no difference between the CFO and the sanitized CFO, why consider the latter? This is because $\mathsf{CFO_s}$ behaves more nicely when it comes to invariant preservation. (The concept of invariant preservation will be explored in the "usage example" below.)

For example, if the state of the $X, Y, H$ registers lies in the subspace $\operatorname{span}\{|xyh\rangle : x = 0, y = 0\}$, then after a query to $\mathsf{CFO_s}$, the state is $O(1/\sqrt{N})$-close to the subspace $\operatorname{span}\{|xyh\rangle : x = 0, y = h(0)\}$. This is very natural because it says that after a evaluating the oracle at 0, the $Y$-register contains the result of that evaluation. Surprisingly, we cannot show a corresponding invariant preservation for $\mathsf{CFO}$.[16]

$\mathsf{CFO}$ still performs well with invariants that do not talk about the $Y$ register. If the CFO is invoked by the adversary, there is little to be said about the content of $Y$, anyway. But when the CFO is queried by an honest party or the challenger, for example, the oracle response in $Y$ may be very relevant. So in some proofs, we might get further using $\mathsf{CFO_s}$, while in others it may not matter.

The difference between $\mathsf{CFO_s}$ and $\mathsf{CFO}$ will also matter in the compressed permutation case in Section 4.

**Usage example.** We will now give an example how the compressed oracle is used. We do not work out all the details (in particular, we skip some calculations) but instead focus on a high level overview required for understanding the methodology. The example works both with $\mathsf{CFO}$ and $\mathsf{CFO_s}$ in the same way.

Consider the following problem:

> Given quantum (i.e., superposition) access to a random oracle $H$, find $x$ such that $H(x) = 0$.

We want to show that this problem is hard. We will use the compressed oracle to do so.[17]

Fix an adversary $A^{\mathsf{CFO}}$ making polynomially many queries. Recall that $H$ is the register of the CFO that contains the superposition of the partial functions $h$, and that those partial functions intuitively represent the knowledge what has been been queried and what the responses were.

Consider the following invariant $\mathbf{I} := \operatorname{span}\{|h\rangle : 0 \notin \operatorname{im} h\}$. This invariant contains all superpositions of partial functions $h$ where 0 is not in the outputs, i.e., where $A$ never got 0 as a result to its queries.

In an execution of $A^{\mathsf{CFO}}$, the initial state of the system has $|\varnothing\rangle$ in $H$. Since $0 \notin \operatorname{im} \varnothing$, we have $|\varnothing\rangle \in \mathbf{I}$. Thus the initial state of the overall system (containing $X$, $Y$, and $A$'s registers) is in $\mathbf{I}$ as well.[18]

Now, when $A$ performs a (w.l.o.g. unitary) operation on its registers, this operation does not touch $H$. (Recall that $H$ is inaccessible to $A$.) So if the state $\psi$ before that operation is $\psi \in \mathbf{I}$, then the state $\psi'$ after the operation is also $\psi' \in \mathbf{I}$. In fact, if $\psi \overset{\varepsilon}{\approx} \mathbf{I}$, then $\psi' \overset{\varepsilon}{\approx} \mathbf{I}$ as well. (This follows immediately from the previous fact because the unitary performed by $A$ has operator norm 1.)

[16] For example, $|0\rangle|*\rangle$ is in the first subspace, but $\mathsf{CFO}|0\rangle|*\rangle = |0\rangle|*\rangle$ is not in the second subspace.

[17] Of course, there are many results that show that this specific problem is hard, predating the compressed oracle technique, and considerably simpler mathematically (e.g., [3, 19]). This just happens to be the simplest example to demonstrate the technique.

[18] Strictly speaking, the initial state is in $\mathbf{I} \otimes \mathcal{H}_{XY\ldots}$ where $\mathcal{H}_{XY\ldots}$ is the Hilbert space of all registers besides $H$. We omit the $\otimes\mathcal{H}_{XY\ldots}$ for simplicity.

More interesting is the case when $A$ queries the CFO. In that case, it is less obvious, but we can show:

$$\psi \in \mathbf{I} \qquad \Longrightarrow \qquad \psi' \overset{O\left(\sqrt{\frac{1}{N}}\right)}{\approx} \mathbf{I}.$$

(Here $\psi, \psi'$ are the state before and after the query.) We will not do the math here, see for example Zhandry's original paper [30] for a calculation, or [12] for some generic rules for bounding such invariant preservations in the CFO.

And again using that $\mathsf{CFO}$ has operator norm 1 and the triangle inequality for the norm, we get

$$\psi \overset{\varepsilon}{\approx} \mathbf{I} \qquad \Longrightarrow \qquad \psi' \overset{\varepsilon+O\left(\sqrt{\frac{1}{N}}\right)}{\approx} \mathbf{I}.$$

So by induction, we have that if the adversary does $q$ queries, the final state is $\psi_{final} \overset{O\left(q\sqrt{\frac{1}{N}}\right)}{\approx} \mathbf{I}$. This is negligible if $q \ll \sqrt{N}$, meaning that the adversary needs around $\sqrt{N}$ queries to find a zero-preimage.[19] This concludes the analysis of zero-preimage finding in the CFO.

The advantage of the CFO is that it can be very easily adapted to a variety of different problems by changing the invariant $\mathbf{I}$. For example, if we want to show that finding a collision ($x \neq x'$ with $H(x) = H(x')$) is hard, we use the invariant

$$\mathbf{I} := \operatorname{span}\{|h\rangle : h \text{ injective}\}$$

instead. This represents the fact that the oracle as queried so far is injective, i.e., no two outputs are the same, i.e., the adversary has not found a collision. Obviously the initial state $|\varnothing\rangle \in \mathbf{I}$ again, and we can show (see, e.g., [30] again):

$$\psi \in \mathbf{I} \qquad \Longrightarrow \qquad \psi' \overset{O\left(\sqrt{\frac{i}{N}}\right)}{\approx} \mathbf{I} \qquad \text{in the } i\text{-th query.}$$

Then the proof proceeds as above by induction, giving us $\psi_{final} \overset{O\left(\sqrt{\frac{q^3}{N}}\right)}{\approx} \mathbf{I}$. So finding a collision takes $\sqrt[3]{N}$ queries.

**Technical summary.** The standard oracle $\mathsf{StO}$ operates on registers $X, Y, H$ with Hilbert spaces $\mathbb{C}^D$, $\mathbb{C}^R$, $\mathbb{C}^{D \rightharpoonup R}$. It is defined as the unitary $\mathsf{StO} : |x, y, h\rangle \mapsto |x, y \oplus h(x), h\rangle$ if $h(x) \neq \bot$ and $\mathsf{StO} : |x, y, h\rangle \mapsto |x, y, h\rangle$ otherwise. The sanitized $\mathsf{StO_s}$ is defined the same, except $\mathsf{StO} : |x, y, h\rangle \mapsto 0$ when $h(x) = \bot$. The initial state of $H$ is $\sum_{h \in D \to R} |R|^{-|D|/2} |h\rangle$ (uniform superposition of all total functions).

The compressed function oracle $\mathsf{CFO}$ operates the same registers $X, Y, H$. $H$ can be seen equivalently as a collection of registers $H_x$ with $x \in D$, each having Hilbert space $\mathbb{C}^{R \cup \{\bot\}}$. $Q$ is an arbitrary unitary with $Q|0\rangle = \sum_x \frac{1}{\sqrt{N}} |x\rangle$ (e.g., the quantum Fourier transform). $U_\bot$ is the unitary mapping $|\bot\rangle \mapsto |0\rangle$, $|0\rangle \mapsto |\bot\rangle$, $|z\rangle \mapsto |z\rangle$ otherwise. $\mathsf{Decomp}_1 := Q \cdot U_\bot \cdot Q^\dagger$ is the decompression unitary (for a single register $H_x$). $\mathsf{Decomp}$ is $\mathsf{Decomp}_1$ applied to each $H_x$ (making it an operation on $H$). Then the $\mathsf{CFO}$ is defined as $\mathsf{Decomp} \cdot \mathsf{StO} \cdot \mathsf{Decomp}^\dagger$. The sanitized $\mathsf{CFO_s}$ is defined as $\mathsf{Decomp} \cdot \mathsf{StO_s} \cdot \mathsf{Decomp}^\dagger$. The initial state of $H$ is $|\varnothing\rangle$ (the empty partial function) for both $\mathsf{CFO}$ and $\mathsf{CFO_s}$.

# 4 Compressed permutation oracles

In the preceding section, we considered compressed function oracles. These models the "normal" random oracle in which the adversary gets access to a uniformly random function. However, in many cases, we may be interested in uniformly random *permutations*, instead. For example, an ideal cipher is nothing but a family of random permutations (indexed by the key). Abstractly, random permutations are not a much more complicated concept than random functions (we simply pick the function from a smaller set)

[19] There is a technicality we gloss over here: We only have shown that the oracle state does not contain a zero-preimage. To fully finish the proof, we additionally need to show that this implies that the adversary cannot guess a zero-preimage. For example, the adversary could just output something random that was not queried before and hope that it is a zero-preimage. The probability of the latter succeeding is, of course, tiny. But in a complete analysis, this all needs to be taken into account. This is important but not relevant for *illustrating* the compressed oracle technique, nor for the purposes of our paper. We refer to existing works on the compressed oracle (e.g., [30]) for details on this.

but they can be considerably harder to analyze. The reason for this is that in a random function, all outputs are sampled independently, while in a random permutation, this is not the case. Because of this, even simple questions relating to (superposition access to) random permutations are to the best of our knowledge not in the scope of existing techniques, such as the following conjecture:

**Conjecture 1 (Double-sided zero-search)** *Let $H$ be a uniformly random permutation on $\{0,1\}^{2n}$. The following problem is hard for any adversary making polynomially many superposition queries to $H$ and $H^{-1}$:*

*Find $x \in \{0,1\}^n$ such that $H(x\|0^n) = y\|0^n$ for some $y$.*

Note that in this example, we explicitly allowed the adversary to query not only $H$ (what we call a "forward query"), but also $H^{-1}$ ("backward query"). If we model a non-invertible permutation (adversary can only make forward queries), random permutations are easy to handle, even in the quantum setting. Namely, we know that a random permutation is indistinguishable from a random function, even in the quantum setting [28]. So when analyzing a situation where the adversary has quantum access to a non-invertible function, we can simply replace it by a random function as the first step, and analyze from there using established techniques for random oracles (such as, for example, the CFO).

What we are interested in are, therefore, *invertible* random permutations (forward and backward queries).

In the classical setting, even invertible random permutations are quite easy to handle: The same way as we can model a random oracle via lazy sampling (i.e., pick all outputs of the oracle only when first accessed), we can also model a random permutation via lazy sampling: We pick the output to a forward query $x$ at random, unless that query was already made, or some backward query returned $x$. And mutatis mutandis for backward queries. This does not give us exactly the distribution of answers that a random permutation would give, but is negligibly close. And now we have all the nice benefits of the lazy sampling of the random oracle, in particular the fact that any fresh queries give (near) uniformly random independent outputs.

Since the compressed oracle technique, roughly speaking, is a quantum analogue of lazy sampling, we might wonder whether the same is possible in the quantum setting. That is, is the following possible?

> Define an oracle $\mathsf{CPO}$ (for "compressed permutation oracle") that keeps a superposition of partial functions as its internal state, that responds to forward and backward queries in some way that increases the length of those partial function only by 1 (or at least something small), and that is indistinguishable from having access to a permutation that is chosen uniformly at random.

It turns out that *defining* such a $\mathsf{CPO}$ is not too hard. What is hard (and what we will only make a step towards in this paper) is to prove that the $\mathsf{CPO}$ is indeed indistinguishable from a truly random permutation.

To define $\mathsf{CPO}$, we need to define its behavior on forward and backwards queries. Forward queries are easy: The internal state is, like in the CFO case, a superposition of partial functions. That is, a register $H$ with Hilbert space $\mathbb{C}^{D\to D}$. (Since we are in the permutation case, we have $D = R$ and thus $D \to R$ becomes $D \to D$.) A forward query to $\mathsf{CPO}$ is then just handled by the oracle $\mathsf{CFO_s}$ defined in the previous section.

(We could use the non-sanitized CFO here as well, but we use the sanitized CFO for two reasons: It tends to behave better with some invariants as explained in the previous section, and more importantly, we do not know how to prove the results in the next section with the non-sanitized one.)

Backward queries are more interesting. We could define an oracle that, given an input $|x\rangle$, searches through the partial function $|h\rangle$ in $H$, and tries to find $x$ in the output of $h$. (All of this in superposition between different $|x\rangle$ and $|h\rangle$, of course.) But there is a simple trick that makes the definition (and also the use of $\mathsf{CPO}$ in the end) much simpler without changing its substance: Instead of searching through $|h\rangle$, we simply invert $h$ in place, then evaluate it, and then invert it again.

More precisely: Let $\mathsf{Flip}$ be a linear operator such that $\mathsf{Flip}|h\rangle = |h^{-1}\rangle$ for all injective $h$. We do not specify what $\mathsf{Flip}$ does on any $|h\rangle$ that is not injective.[20] We only require that $\|\mathsf{Flip}\| \leq 1$, that is, $\|\mathsf{Flip}\,\psi\| \leq \|\psi\|$ for all $\psi$. (This is to avoid strange cases where the state of the system ends up having norm greater than 1.)

[20] For example, $\mathsf{Flip}|h\rangle$ might return an $h'$ that so that $h'(x)$ is the lexicographically smallest preimage of $x$ under $h$ if there are several. Or $\mathsf{Flip}|h\rangle = 0$ for non-injective $h$. Since a non-injective $h$ should not happen anyway when simulating a permutation, it should not matter how $\mathsf{Flip}$ is defined on these, so we make sure that our results hold independent of the design choices for that case.

Then a backward query to $\mathsf{CPO}$ is handled by invoking the operator $\mathsf{Flip} \cdot \mathsf{CFO}_\mathsf{s} \cdot \mathsf{Flip}$. (Inverting, evaluating, inverting back.)

To summarize:

**Definition 1 (Compressed permutation oracle)** $\mathsf{CPO}$ *is a pair of oracles* $\mathsf{CFO}_\mathsf{s}$ *and* $\mathsf{Flip} \cdot \mathsf{CFO}_\mathsf{s} \cdot \mathsf{Flip}$ *where* $\mathsf{CFO}_\mathsf{s}$ *is as defined in* (6) *in the previous section, and* $\mathsf{Flip}|h\rangle = |h^{-1}\rangle$ *for injective* $h$ *(and* $\|\mathsf{Flip}\| \leq 1$*), both operating on the same registers* $X, Y, H$ *where* $H$ *is private to* $\mathsf{CPO}$ *(not accessible to the querying algorithm) and initialized in the beginning with* $|\varnothing\rangle$*.*

We conjecture:

**Conjecture 2** *For any polynomial-query algorithm* $A$*,*

$$\left|\Pr[A^{\mathsf{CPO}} \Rightarrow 1] - \Pr[A^{\pi,\pi^{-1}} \Rightarrow 1 : \pi \xleftarrow{\$} (D \hookrightarrow D)]\right| \text{ is negligible.} \tag{7}$$

*(Negligible in* $\log|D|$*.)*

**Usage example.** We now illustrate how the CPO can be used by showing Conjecture 1 (double-sided zero-search) using the CPO. Of course, the validity of this example rests on Conjecture 2.

The reasoning is very similar to that done for the zero-preimage search in Section 3. We urge the reader to recap that reasoning first.

Specifically, we first come up with some invariant that describes that the adversary has not found $x, y$ with $H(x\|0^n) = y\|0^n$.

$$\mathbf{I} := \operatorname{span}\big\{|h\rangle : \nexists xy.\ h(x\|0^n) = y\|0^n\big\}.$$

Obviously, the initial state of the CPO (namely $|\varnothing\rangle$ in $H$) satisfies $\mathbf{I}$, and unitaries evaluated by the adversary on non-$H$ registers preserve $\mathbf{I}$.

Given a forward-query to $\mathsf{CPO}$ (i.e., an application of the operator $\mathsf{CFO}_\mathsf{s}$), we have

$$\psi \in \mathbf{I} \qquad \Longrightarrow \qquad \psi' \overset{O(2^{-n/2})}{\approx} \mathbf{I} \tag{8}$$

where $\psi, \psi'$ are the state before/after that query. Again, we omit the details of this computation and refer the reader to existing work.[21] We only stress that intuitively, this is what we expect, since when querying the oracle on any fresh input, the output will be of the form $y\|0^n$ only with small probability.

The only conceptually new thing in this example is when the adversary performs a backward-query: This will execute $\mathsf{Flip} \cdot \mathsf{CFO}_\mathsf{s} \cdot \mathsf{Flip}$. Note that if $h$ satisfies $h(x\|0^n) = y\|0^n$, then $h^{-1}$ satisfies it, too. So any $|h\rangle \in \mathbf{I}$ is mapped by $\mathsf{Flip}$ to $|h^{-1}\rangle \in \mathbf{I}$. Thus $\mathsf{Flip}$ preserves $\mathbf{I}$. (Meaning, if $\psi \in \mathbf{I}$, then $\mathsf{Flip}\,\psi \in \mathbf{I}$.) Furthermore, we already know from (8) that an invocation of $\mathsf{CFO}_\mathsf{s}$ only introduces an $O(2^{-n/2})$ distance from $\mathbf{I}$. So together with the fact that $\mathsf{Flip}$ preserves $\mathbf{I}$, we have that (8) also holds for backward-queries.

All in all, we then have by induction that for a $q$-query adversary, at the end it holds:

$$\psi_{final} \overset{O\left(q2^{-n/2}\right)}{\approx} \mathbf{I}.$$

So to find a "doubled-sided zero", the adversary needs $\Theta(2^{n/2})$ queries.

# 5 Towards compressed permutations

As discussed in the introduction, page 3, the main contribution of our work is the following claim:

> **Main contribution (informal):** If some construction (say based on a random oracle) is indistinguishable from a CPO, then the CPO is indistinguishable from a random invertible permutation.

To make this formal, we first need to say what exactly we mean by a construction. Specifically, we focus on constructions that implement permutations. Roughly speaking, such a construction is some deterministic algorithm $C$ that uses one or several oracles $H_1, \ldots, H_n$ and implements two functions $\pi$ and $\tau$ that are inverses of each other.

[21] Existing work on compressed oracles applies here since (8) refers specifically to the preservation of $\mathbf{I}$ under a query to $\mathsf{CFO}_\mathsf{s}$ and thus is not specific to the permutation case.

For example, in the case of three-round Luby-Rackoff,[22] the algorithm $C$ would take three oracles and implement $\pi(x_Lx_R)$ as: $t_1 := H_1(x_L)$, $t_2 := H_2(x_R \oplus t_1)$, $t_3 := H_3(x_L \oplus t_2)$, return $(x_L \oplus t_2, x_R \oplus t_1 \oplus t_3)$, and $\tau(x_Lx_R)$ as: $t_1 := H_3(x_L)$, $t_2 := H_2(x_R \oplus t_1)$, $t_3 := H_1(x_L \oplus t_2)$, return $(x_L \oplus t_2, x_R \oplus t_1 \oplus t_3)$. It is easy to see that $\pi$ and $\tau$ are permutations with $\tau = \pi^{-1}$ for any choice of $H_1, H_2, H_3$.

More abstractly, a construction is some function $C$ that takes (fixed) functions $H_1, \ldots, H_n$, and returns functions $\pi, \tau$. (Deterministically. That is, for fixed $H_1, \ldots, H_n$, $\pi, \tau$ are fixed, too.) We do not care about the algorithmic details of how $C$ transforms the oracles $H_1, \ldots, H_n$ into functions $\pi, \tau$. While it would be typical that $C$ does this by doing a few queries to $H_1, \ldots, H_n$, this is formally not required for our result.

In addition to the algorithm/function $C$ that specified how $\pi, \tau$ are implemented given the oracles, the specification of the construction also needs to tell us what kinds of oracles $H_1, \ldots, H_n$ are. E.g., in the Luby-Rackoff case, they are random functions. In other constructions, they might be functions with some other distribution (e.g., uniformly random permutations). To be as general as possible, we simply include the desired distribution $\mathcal{D}$ of the oracles in the specification of the construction.

The following definition summarizes all this:

**Definition 2** *A* permutation-construction $(C, \mathcal{D})$ *(implicitly parametrized by a security parameter $\lambda$) consists of a function $C$ that takes a tuple of functions $\underline{H} = (H_1, \ldots, H_n)$ and returns a pair of functions $\pi, \tau : D \to D$, and of a distribution $\mathcal{D}$ (for the functions $H_1, \ldots, H_n$), and satisfies the following:*

*For $\underline{H}$ distributed according to $\mathcal{D}$, with overwhelming probability, $\pi$ is a permutation and $\tau = \pi^{-1}$.*

**Remark.** We do not require that $\pi, \tau$ can be efficiently computed given oracle access to $\underline{H}$. In practice, one would of course require efficient constructions, but our result holds without this additional condition, so we do not include it in our definition.

**Remark.** Note that the definition requires that the construction produces an invertible permutation (and not, e.g., a pair of non-invertible functions) with overwhelming probability, i.e., it includes a correctness requirement. But it does not require that $\pi$ and $\tau$ look random in any way.

**Notation.** Given $\underline{H}$, $C(\underline{H})$ is a pair of functions $\pi, \tau$. If we write $A^{C(\underline{H})}$, we mean that $A$ gets superposition access to $\pi, \tau$. I.e., $A^{C(\underline{H})}$ can be read as $(\pi, \tau) := C(\underline{H}), A^{\pi,\tau}$.

We are ready to state the main result formally:

**Theorem 1** *Let $C$ be a permutation-construction. Assume that for any polynomial-query adversary $A$,*

$$\left|\Pr[A^{C(\underline{H})} \Rightarrow 1 : \underline{H} \xleftarrow{\$} \mathcal{D}] - \Pr[A^{\mathsf{CPO}} \Rightarrow 1]\right| \text{ is negligible.} \tag{9}$$

*Then for any polynomial-query adversary $A$,*

$$\left|\Pr[A^{\mathsf{CPO}} \Rightarrow 1] - \Pr[A^{\pi,\pi^{-1}} \Rightarrow 1 : \pi \xleftarrow{\$} (D \hookrightarrow D)]\right| \text{ is negligible.} \tag{10}$$

*(Recall that $D \hookrightarrow D$ is the set of permutations on $D$.)*

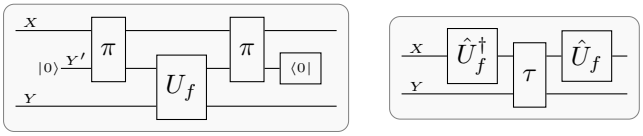
In particular, the existence of such a construction shows Conjecture 2.[23]

We present a similar result where the construction may use computational assumptions in Corollary 1 below.

The (very rough) idea of the proof is simple: For a random permutation $f$, let $f \circ \mathsf{CPO}$ denote the $\mathsf{CPO}$, but where we apply $f$ to its outputs. (Or its inputs when doing a backward query.) Then $\mathsf{CPO}$ and $f \circ \mathsf{CPO}$ are indistinguishable. (This follows from symmetries in the definition of $\mathsf{CPO}$ and does not require that $\mathsf{CPO}$ is actually indistinguishable from a permutation.) By assumption, $\mathsf{CPO}$ and the construction $C$ are indistinguishable, so $f \circ \mathsf{CPO}$ and $f \circ C$ are indistinguishable, too. (Since a distinguisher could just simulate $f$ itself.) And since $C$ implements a permutation (not necessarily random), $f \circ C$ is a random permutation composed with permutation, thus a random permutation. So $f \circ C$ is indistinguishable from $f$. Taking this all together, we have that $\mathsf{CPO}$ is indistinguishable from $f$ which shows the theorem.

[22] We use three-round Luby-Rackoff for this example just to keep the formulas short and readable. We are aware that three-round Luby-Rackoff is not even indistinguishable from an invertible random permutation in the classical setting, let alone the quantum setting.

[23] For those domains $D$ that the construction $C$ actually operates on. (E.g., if $C$ only implements permutations on domains $D = \{0,1\}^{n^2}$, then this would only prove Conjecture 2 for such domains.)




(a) (b)

Figure 2: Circuits implemented by $B$. (a) shows $\pi'$, (b) shows $\tau'$. The $\boxed{\langle 0|}$ gate denotes a multiplication with $\langle 0|$. Equivalently, this can be thought of as applying a projector onto $|0\rangle$ on $Y'$ and then removing the register $Y'$. $U_f$ is the unitary $U_f : |x, y\rangle \mapsto |x, y \oplus f(x)\rangle$. $\hat{U}_f$ is the unitary $\hat{U}_f : |x\rangle \mapsto |f(x)\rangle$. (Recall that $f$ is bijective, so $\hat{U}_f$ is a unitary.)

*Proof.* Fix a polynomial-query adversary $A^{\pi,\tau}$ (taking two oracles $\pi, \tau$). We need to show that (10) holds for this adversary.

Consider the following adversary $B^{\pi,\tau}$:

- It takes two oracles $\pi, \tau$.
- It picks a permutation $f : D \hookrightarrow D$ uniformly at random
- It runs $A^{\pi',\tau'}$ where $\pi', \tau'$ are implemented by the circuits in Figure 2.
- It returns what $A$ returns.

For the intuition: if $\pi$ and $\tau$ are simply oracles providing superposition access to some functions $\pi, \tau$, then $\pi' = f \circ \pi$ and $\tau' = \tau \circ f^{-1}$. And if moreover $\tau = \pi^{-1}$, then $\tau' = \pi'^{-1}$. However, $B$ may be invoked with stateful oracles $\pi, \tau$, so we cannot simply define $\pi', \tau'$ that way but instead need to give concrete circuits. (We also take care to ensure that $B$ does not make more queries than $A$. Otherwise the proof of (11) below would become much harder.)

We have:

$$\Pr[A^{\mathsf{CPO}} \Rightarrow 1] \approx \Pr[B^{\mathsf{CPO}} \Rightarrow 1]. \tag{11}$$

Here $\approx$ means a negligible difference. Intuitively, this follows because the definition of the $\mathsf{CPO}$ is symmetric, i.e., all inputs and outputs are treated the same, so permuting them should not make a difference. In reality, we need to be more careful because with small probability we can end up with oracle states on which $\mathsf{Flip}$ is defined arbitrarily (and possibly non-symmetrically.) We defer the proof of (11) to the auxiliary Lemma 1 below.

Since $B$ makes the same number of queries as $A$, it is a polynomial-query adversary. So by assumption (9) of the lemma, we have:

$$\Pr[B^{\mathsf{CPO}} \Rightarrow 1] \approx \Pr[B^{C(\underline{H})} \Rightarrow 1 : \underline{H} \xleftarrow{\$} \mathcal{D}]. \tag{12}$$

Since $C$ is a permutation-construction by assumption, for any fixed $\underline{H}$, $C(\underline{H}) = (\pi_H, \tau_H)$ for some functions $\pi_H, \tau_H$ that depend only on $\underline{H}$ (see Definition 2). We call $\underline{H}$ good if $\pi_H$ is a permutation and $\tau_H = \pi_H^{-1}$. Still by definition of permutation-constructions, $\underline{H}$ is good with overwhelming probability. For a fixed good $\underline{H}$, we thus have:

$$\begin{aligned}
\Pr[B^{C(\underline{H})} \Rightarrow 1] &= \Pr[B^{\pi_H, \pi_H^{-1}} \Rightarrow 1] \overset{(*)}{=} \Pr[A^{f\circ\pi_H, \pi_H^{-1}\circ f^{-1}} \Rightarrow 1 : f \xleftarrow{\$} (D \hookrightarrow D)] \\
&= \Pr[A^{f\circ\pi_H, (f\circ\pi_H)^{-1}} \Rightarrow 1 : f \xleftarrow{\$} (D \hookrightarrow D)] \\
&\overset{(**)}{=} \Pr[A^{f, f^{-1}} \Rightarrow 1 : f \xleftarrow{\$} (D \hookrightarrow D)] = \Pr[A^{\pi, \pi^{-1}} \Rightarrow 1 : \pi \xleftarrow{\$} (D \hookrightarrow D)]
\end{aligned}$$

Here $(*)$ follows since the circuits that $B$ computes $f \circ \pi, \tau \circ f^{-1}$ given oracles that implement fixed functions $\pi, \tau$. And $(**)$ follows because for fixed permutation $\pi_H$ and uniformly random permutation $f$, we have that $f \circ \pi_H$ has the same distribution as $f$.

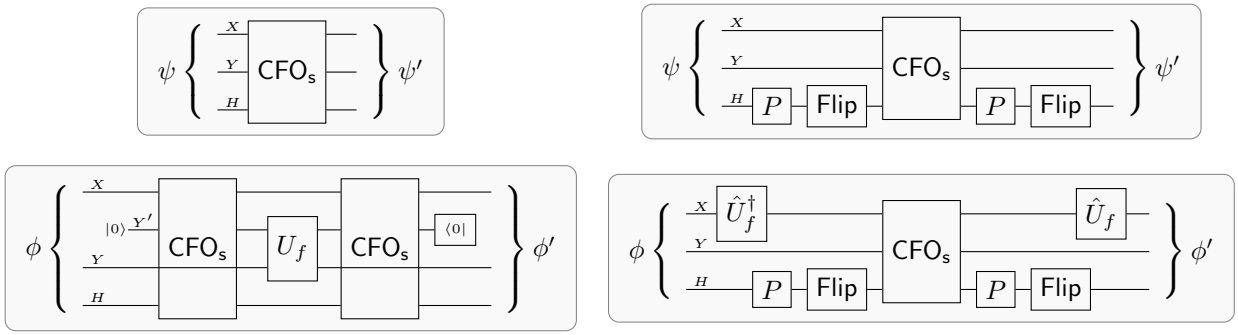
Since this holds for any good $\underline{H}$, and $\underline{H}$ is good with overwhelming probability, by averaging we have:

$$\Pr[B^{C(\underline{H})} \Rightarrow 1 : \underline{H} \xleftarrow{\$} \mathcal{D}] \approx \Pr[A^{\pi, \pi^{-1}} \Rightarrow 1 : \pi \xleftarrow{\$} (D \hookrightarrow D)]. \tag{13}$$

Equation (10) follows by (11)–(13). □

**Lemma 1** *For $B$ as defined in the proof of Theorem 1, we have*

$$\Pr[A^{\mathsf{CPO}} \Rightarrow 1] \approx \Pr[B^{\mathsf{CPO}} \Rightarrow 1].$$




(a) (b)

Figure 3: Circuits in invariant preservation proof.

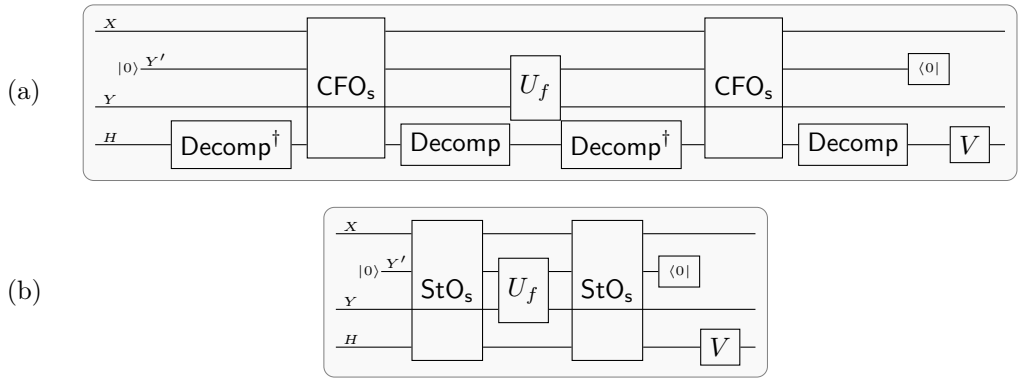



Figure 4: Circuits in invariant preservation proof, first case.

*Here ≈ means negligible difference.*

*Proof.* The basic idea in this proof is that $B$ essentially just permutes the different possible outputs of the permutation implemented by the CPO, and since the definition of the CPO does not treat any possible output differently from any other, this permutation of outputs has no observable effect. However, this is not fully true: For example, if the oracle register $H$ contains $|h\rangle$ where $h$ is a non-injective partial function, the behavior of $\mathsf{Flip}$ is unspecified and might be asymmetric. (E.g., $\mathsf{Flip}$ might always pick the lexigraphically smallest string in case of ambiguities.) To work around this, we first sanitize our CPO somewhat. Let $P$ be the projection onto $\mathrm{span}\{|h\rangle : h \text{ injective}\}$ on register $H$. We then insert $P$ before every call to $\mathsf{Flip}$. Specifically, let $\mathsf{CPO}'$ denote the two oracles $\mathsf{CFO_s}$ and $\mathsf{Flip}\cdot P\cdot\mathsf{CFO_s}\cdot\mathsf{Flip}\cdot P$. (Instead of $\mathsf{CFO_s}$ and $\mathsf{Flip}\cdot\mathsf{CFO_s}\cdot\mathsf{Flip}$ as per definition of $\mathsf{CPO}$.) That is, $\mathsf{CPO}'$ adds extra invocations of the projection $P$ before each $\mathsf{Flip}$.[a] The oracles making up $\mathsf{CPO}'$ are also depicted in the top row of Figure 3. Then we have

$$\Pr[A^{\mathsf{CPO}}\Rightarrow 1]\approx\Pr[A^{\mathsf{CPO}'}\Rightarrow 1]\qquad\text{and}\qquad\Pr[B^{\mathsf{CPO}}\Rightarrow 1]\approx\Pr[B^{\mathsf{CPO}'}\Rightarrow 1]\tag{14}$$

We see this as follows: Consider the invariant $\mathbf{I}:=\mathrm{span}\{|h\rangle : h \text{ injective}\}$. This invariant is preserved by adversary operations on registers other than $H$. Queries to $\mathsf{CFO_s}$ introduce an $O(i/N)$ error in the $i$-th query which is negligible. And since with $h$, $h^{-1}$ is also injective, we have that $\mathsf{Flip}$ preserves $\mathbf{I}$. Finally, the difference between $A^{\mathsf{CPO}}$ and $A^{\mathsf{CPO}'}$ is that the latter has additional applications of the projector $P$. Since $P$ projects onto $\mathbf{I}$ by definition, and the state is negligibly close to $\mathbf{I}$, this will change the state by a negligible amount. Therefore the final state of $A$ differs only by a negligible amount (in the norm). Thus $A$'s output probability also differs only by a negligible amount. Analogously for $B$. This shows (14).

Because of (14), to prove our lemma, it is sufficient to show:

$$\Pr[A^{\mathsf{CPO}}\Rightarrow 1]=\Pr[B^{\mathsf{CPO}}\Rightarrow 1].\tag{15}$$

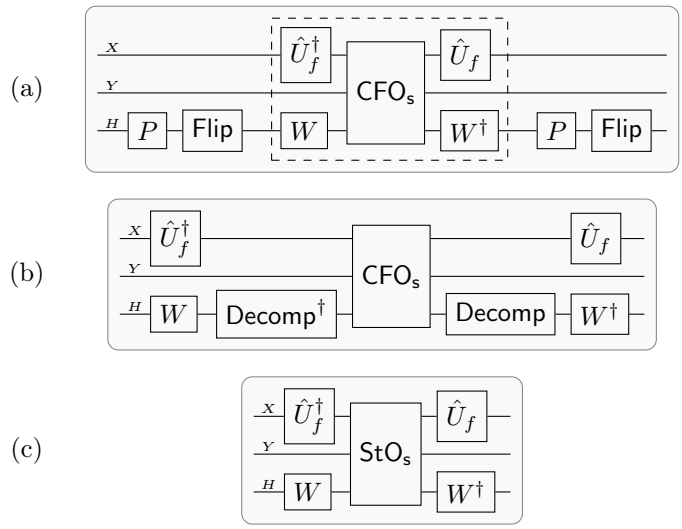



Figure 5: Circuits in invariant preservation proof, second case.

And by averaging, it is sufficient to show this for a fixed permutation $f$. (Then it will also hold when $B$ chooses $f$ at random.) Thus, for the remainder of this proof, $f$ will be an arbitrary but fixed permutation on $D$.

Let $V$ denote the unitary on $H$ mapping $|h\rangle$ to $|f \circ h\rangle$. To show (15), we will show that the states of $A^{\mathsf{CPO}'}$ and $B^{\mathsf{CPO}'}$ are always related by $V$. More precisely, when $\psi$ is the state of $A$ before or after the invocation of the $n$-th query to $\mathsf{CPO}'$ (or the initial or final state), and $\phi$ is the state of $B$ before or after the $n$-th invocation of the corresponding circuits from Figure 2 (or the initial or final state), then:

$$\psi = V\phi. \tag{16}$$

For the initial state, (16) is immediate. (Recall that in the initial state, $H$ contains $|\varnothing\rangle$, and $V|\varnothing\rangle = |f \circ \varnothing\rangle = |\varnothing\rangle$.) We then proceed inductively through the execution of $A$ and $B$.

Without loss of generality, $A$ or $B$, respectively, interleave oracle queries and applications of some unitary operation on the adversaries state. This unitary is the same for $A$ and $B$, and it trivially commutes with $V$ (since $V$ operates on $H$ and $H$ is not part of $A$'s or $B$'s state). Thus (16) is preserved under application of this unitary.

It remains to show that (16) is preserved under invocations of the oracle by $A$ and $B$, respectively. (In the case of $B$, this is meant to include the wrapper circuits from Figure 2.) Let $\psi, \phi$ denote the state of $A$ or $B$ before that invocation, and $\psi', \pi'$ the one after the invocation. We then need to show

$$\psi = V\phi \qquad \Longrightarrow \qquad \psi' = V\phi' \tag{17}$$

for $\psi'$ and $\phi'$ being computed as in Figure 3 (a) or (b). (Depending whether the current query of $A/B$ is one to its first or second oracle.)

We first show (17) for $\psi', \phi'$ as computed in Figure 3 (a). Denote the operation computed by the circuit for $\phi'$ by $C_\phi$. Then we need to show that $\psi' = \mathsf{CFO}_s\,\psi = \mathsf{CFO}_s\,V\phi \stackrel{!}{=} V\phi' = VC_\phi\phi$. So it is sufficient to prove $\mathsf{CFO}_s\,V = VC_\phi$. Since $\mathsf{Decomp}$ is unitary, this is equivalent to showing

$$\mathsf{Decomp} \cdot \mathsf{CFO}_s \cdot V \cdot \mathsf{Decomp}^\dagger = \mathsf{Decomp} \cdot V \cdot C_\phi \cdot \mathsf{Decomp}^\dagger. \tag{18}$$

Note that $V$ and $\mathsf{Decomp}_1$ commute: Both $V$ and $\mathsf{Decomp}$ operate on each $H_x$ individually, namely as $\hat{U}_f$ and $\mathsf{Decomp}_1$ where $\hat{U}_f$ is the unitary mapping $|z\rangle \mapsto |f(z)\rangle$, $|\perp\rangle \mapsto |\perp\rangle$. So we only need to check that $\hat{U}_f$ and $\mathsf{Decomp}_1$ commute. Using the formula for $\mathsf{Decomp}_1$ from (2), we compute that $\mathsf{Decomp}_1\,\hat{U}_f|\perp\rangle = |*\rangle$, $\hat{U}_f\,\mathsf{Decomp}_1|\perp\rangle = U_f|*\rangle = |*\rangle$, $\mathsf{Decomp}_1\,\hat{U}_f|z\rangle = \mathsf{Decomp}_1|f(z)\rangle = |f(z)\rangle + \frac{1}{\sqrt{N}}|\perp\rangle - \frac{1}{\sqrt{N}}|*\rangle$, $\hat{U}_f\,\mathsf{Decomp}_1|z\rangle = \hat{U}_f|z\rangle + \frac{1}{\sqrt{N}}\hat{U}_f|\perp\rangle - \frac{1}{\sqrt{N}}\hat{U}_f|*\rangle = |f(z)\rangle + \frac{1}{\sqrt{N}}|\perp\rangle - \frac{1}{\sqrt{N}}|*\rangle$. So $\hat{U}_f$, $\mathsf{Decomp}_1$ commute on all basis states, hence they commute everywhere, hence $V$ and $\mathsf{Decomp}$ commute.

Therefore the lhs of (18) equals $\mathsf{Decomp} \cdot \mathsf{CFO_s} \cdot \mathsf{Decomp}^\dagger \cdot V$ which is by definition $\mathsf{StO_s} \cdot V$. And the rhs equals what is drawn in Figure 4 (a). (Besides commuting $\mathsf{Decomp}$ and $V$, we also inserted $\mathsf{Decomp}$, $\mathsf{Decomp}^\dagger$ in the middle. These cancel out because they are unitary.) And since $\mathsf{StO_s} = \mathsf{Decomp} \cdot \mathsf{CFO_s} \cdot \mathsf{Decomp}^\dagger$, Figure 4 (a) further simplifies to what is shown in Figure 4 (b).

Finally, both $\mathsf{StO_s} \cdot V$ and the circuit from Figure 4 (b), upon input $|x, y, h\rangle$, return the state $|x, y \oplus f(h(x)), f \circ h\rangle$ when $h(x) \neq \bot$ and 0 when $h(x) = \bot$.[b] (By elementary computation using the definitions of $\mathsf{StO_s}, U_f, V$.) Since they return the same state on every basis state, they are equal (by linearity). Thus the lhs and the rhs of (18) are equal; (18) holds. Hence (17) holds for $\psi', \phi'$ as computed in Figure 3 (a).

We now show (17) for $\psi', \phi'$ as computed in Figure 3 (b). Denote the circuit defining $\psi'$ by $D_\psi$ and the circuit defining $\phi'$ by $D_\phi$. Then we need to show that $\psi' = D_\psi\psi = D_\psi V\phi \stackrel{!}{=} V\phi' = VD_\phi\phi$. So it is sufficient to prove $D_\psi V = VD_\phi$, or equivalently $D_\psi = VD_\phi V^\dagger$ (since $V$ is unitary).

Define the unitary $W : |h\rangle \mapsto |h \circ f\rangle$ on $H$. (Note: compared to $V$, here we have $h \circ f$, not $f \circ h$.)

Note that $\mathsf{Flip}\, PV^\dagger = W\, \mathsf{Flip}\, P$: For non-injective $h$, $\mathsf{Flip}\, PV^\dagger|h\rangle$ and $W\, \mathsf{Flip}\, P|h\rangle$ are both 0. ($P$ projects such $|h\rangle$ to 0, and $V^\dagger$ preserves non-injectivity.) For injective $h$, $\mathsf{Flip}\, PV^\dagger|h\rangle = \left|(f^{-1} \circ h)^{-1}\right\rangle = \left|h^{-1} \circ f\right\rangle$ and $W\, \mathsf{Flip}\, P|h\rangle = |h^{-1} \circ f\rangle$. So $\mathsf{Flip}\, PV^\dagger$ and $W\, \mathsf{Flip}\, P$ coincide on all basis states, hence $\mathsf{Flip}\, PV^\dagger = W\, \mathsf{Flip}\, P$.

Analogously, $V\, \mathsf{Flip}\, P = \mathsf{Flip}\, PW^\dagger$.

This means that $VD_\phi V^\dagger$ can be rewritten to the circuit in Figure 5 (a). To show that that circuit is equal to $D_\psi$, all we need to prove is that the dashed part in Figure 5 (a) (henceforth called $E_\phi$) is equal to $\mathsf{CFO_s}$. And since $\mathsf{Decomp}$ is unitary, this in turn is equivalent to

$$\mathsf{Decomp} \cdot \mathsf{CFO_s} \cdot \mathsf{Decomp}^\dagger = \mathsf{Decomp} \cdot E_\phi \cdot \mathsf{Decomp}^\dagger . \tag{19}$$

Note further that $W$ and $\mathsf{Decomp}^\dagger$ commute: $W$ is just a reordering of the registers $H_x$ (it moves $H_x$ into $H_{f^{-1}(x)}$), and $\mathsf{Decomp}^\dagger$ applies the same unitary to each of those registers. So it makes no difference whether we apply $\mathsf{Decomp}^\dagger$ before or after reordering. Analogously $W^\dagger$ and $\mathsf{Decomp}$ commute.

So $\mathsf{Decomp}\, E_\phi\, \mathsf{Decomp}^\dagger$ is equal to what is depicted in Figure 5 (b). And, since $\mathsf{Decomp}\, \mathsf{CFO_s}\, \mathsf{Decomp}^\dagger = \mathsf{StO_s}$, that in turn simplifies to Figure 5 (c). (Denoted $F_\phi$.) So our goal (19) becomes $\mathsf{StO_s} = F_\phi$. And by elementary calculation, we get that $F_\phi|x, y, h\rangle = |x, y \oplus h(x), h\rangle$, same as $\mathsf{StO_s}$. So they coincide on basis states, so we have $\mathsf{StO_s} = F_\phi$, hence (19) holds. So (17) holds for $\psi', \phi'$ as computed in Figure 3 (b).

So (17) holds in all cases. This implies that (16) holds for the final states of $A$ and $B$. The output bit of the adversary $A$ and $B$ is produced by the same measurement on $A$'s and $B$'s final state, respectively, and that measurement measures only registers belonging to the adversary (i.e., not $H$). So applying $V$ on $H$ does not change the distribution of that output bit. Hence (15) holds. By (14) the lemma follows. □

[a] We should clarify what we mean by invoking a projection $P$ in a quantum circuit. Mathematically, this simply means that the current state is multiplied with $P$ (analogous to applying a unitary). Of course, this means that a normalized state could become a non-normalized state. An operational interpretation of this is the following: We measure with the binary measurement $P, 1 - P$, and if the measurement fails (second outcome), we abort. The state after applying $P$ then describes the state in the non-aborting computation path, scaled with the square-root of the probability of reaching that path.

[b] This is where we need to use the sanitized CFO. $\mathsf{StO_s} \cdot V$ and the circuit from Figure 4 (b) (with $\mathsf{StO}$ instead of $\mathsf{StO_s}$) return different states.

**Computational case.** Theorem 1 assumes that the construction $C$ is indistinguishable from $\mathsf{CPO}$ for all polynomial-query adversaries, not just polynomial-time adversaries. If we have a construction $C$ that is only secure under computational assumptions, Theorem 1 cannot be applied. However, the following variant of Theorem 1 applies. That is, we can at least show that $\mathsf{CPO}$ is computationally indistinguishable from a random invertible permutation.

**Corollary 1** *Let $C$ be a permutation-construction.* ***Assume that $C$ is efficiently implementable.***[24] ***Assume that a strong qPRP with domain $D$ exists.***[25] *Assume that for any* ***polynomial-time***

[24] That is, we assume that there is a (potentially stateful) polynomial-time algorithm $\hat{C}$ such that $\hat{C}$ is indistinguishable from $C(\underline{H})$ with $\underline{H} \stackrel{\$}{\leftarrow} \mathcal{D}$ by all polynomial-query (not only polynomial-time) algorithms.

[25] See, e.g., [27] for constructions.

*adversary $A$,*

$$\left|\Pr[A^{C(\underline{H})} \Rightarrow 1 : \underline{H} \xleftarrow{\$} \mathcal{D}] - \Pr[A^{\mathsf{CPO}} \Rightarrow 1]\right| \text{ is negligible.} \tag{20}$$

*Then for any* ***polynomial-time*** *adversary $A$,*

$$\left|\Pr[A^{\mathsf{CPO}} \Rightarrow 1] - \Pr[A^{\pi,\pi^{-1}} \Rightarrow 1 : \pi \xleftarrow{\$} (D \hookrightarrow D)]\right| \text{ is negligible.} \tag{21}$$

(The differences from Theorem 1 are highlighted in boldface.)

*Proof.* The proof closely follows the lines of the one of Theorem 1. Fix a polynomial-time adversary $A^{\pi,\tau}$. Define $B^{\pi,\tau}$ as in the proof of Theorem 1. We assume that the unitaries $\hat{U}_f$ and $\hat{U}_f^\dagger$ in $B$ are implemented by the following respective subcircuits:

- Initialize an extra register $Z$ with $|0\rangle$, apply $U_f$ to $XZ$, swap $X, Z$, apply $U_{f^{-1}}$ to $XZ$, and discard $Z$.
- Initialize an extra register $Z$ with $|0\rangle$, apply $U_{f^{-1}}$ to $XZ$, swap $X, Z$, apply $U_f$ to $XZ$, and discard $Z$.

These subcircuits exactly implement $\hat{U}_f$ and $\hat{U}_f^\dagger$, so this replacement does not change the behavior of $B$.

Since $A$ is polynomial-time it is also polynomial-query, and thus we have by Lemma 1:

$$\Pr[A^{\mathsf{CPO}} \Rightarrow 1] \approx \Pr[B^{\mathsf{CPO}} \Rightarrow 1].$$

By assumption of the lemma, a strong qPRP $f_k : D \hookrightarrow D$ exists. Let $\hat{B}$ be defined like $B$, with the following differences: It initially picks a key $k$ for the strong qPRP $f_k$. And invocations to $U_f$ and $U_{f^{-1}}$ are replaced by $U_{f_k}$ and $U_{f_k^{-1}}$.

Since $\mathsf{CFO}_\mathsf{s}$ and $\mathsf{Flip}$ can be implemented efficiently (using a compact representation of the partial functions in $H$), and $A$ is polynomial-time, and $f_k$ is a strong qPRP, it follows that

$$\Pr[B^{\mathsf{CPO}} \Rightarrow 1] \approx \Pr[\hat{B}^{\mathsf{CPO}} \Rightarrow 1].$$

Also note that $\hat{B}$ is the polynomial-time. ($B$ was not because it picks a random permutation $f$. There might not be a polynomial-time implementation for that.) Then by assumption (20), we have:

$$\Pr[\hat{B}^{\mathsf{CPO}} \Rightarrow 1] \approx \Pr[\hat{B}^{C(\underline{H})} \Rightarrow 1 : \underline{H} \xleftarrow{\$} \mathcal{D}].$$

By assumption, $C$ is efficiently implementable, so there is a polynomial-time $\hat{C}$ that is indistinguishable from $C(\underline{H})$ by polynomial-query adversaries, so:

$$\Pr[\hat{B}^{C(\underline{H})} \Rightarrow 1 : \underline{H} \xleftarrow{\$} \mathcal{D}] \approx \Pr[\hat{B}^{\hat{C}} \Rightarrow 1].$$

Since $\hat{C}$ is polynomial-time, we can use the strong qPRP property of $f_k$ again and get:

$$\Pr[\hat{B}^{\hat{C}} \Rightarrow 1] \approx \Pr[B^{\hat{C}} \Rightarrow 1].$$

And since $B$ is polynomial-query (though not polynomial-time because it picks a random permutation $f$),

$$\Pr[B^{\hat{C}} \Rightarrow 1] \approx \Pr[B^{C(\underline{H})} \Rightarrow 1 : \underline{H} \xleftarrow{\$} \mathcal{D}].$$

Finally, exactly as in the proof of Theorem 1, we show:

$$\Pr[B^{C(\underline{H})} \Rightarrow 1 : \underline{H} \xleftarrow{\$} \mathcal{D}] \approx \Pr[A^{\pi,\pi^{-1}} \Rightarrow 1 : \pi \xleftarrow{\$} (D \hookrightarrow D)].$$

Taking all the equations together, we get (21). □

# References


[1] Andris Ambainis. "Quantum Lower Bounds by Quantum Arguments". In: *J. Comput. Syst. Sci.* 64.4 (June 2002), pp. 750–767. ISSN: 0022-0000. DOI: 10.1006/jcss.2002.1826.

[2] Andris Ambainis, Mike Hamburg, and Dominique Unruh. "Quantum Security Proofs Using Semi-classical Oracles". In: *Crypto 2019*. Springer, 2019, pp. 269–295.

[3] Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. "Quantum Attacks on Classical Proof Systems: The Hardness of Quantum Rewinding". In: *55th FOCS*. Philadelphia, PA, USA: IEEE Computer Society Press, 2014, pp. 474–483. DOI: 10.1109/FOCS.2014.57.

[4] Marko Balogh, Edward Eaton, and Fang Song. "Quantum Collision-Finding in Non-uniform Random Functions". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by Tanja Lange and Rainer Steinwandt. Springer, Heidelberg, Germany, 2018, pp. 467–486. DOI: 10.1007/978-3-319-79063-3_22.

[5] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. "Quantum Lower Bounds by Polynomials". In: *J. ACM* 48.4 (July 2001), pp. 778–797. ISSN: 0004-5411. DOI: 10.1145/502090.502097.

[6] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *CCS '93*. ACM, 1993, pp. 62–73.

[7] Guido Bertoni, J. Daemen, Michaël Peeters, and Gilles van Assche. *Sponge functions*. Ecrypt Hash Workshop, http://sponge.noekeon.org/SpongeFunctions.pdf. May 2007.

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Assche. "On the Indifferentiability of the Sponge Construction". In: *Eurocrypt 2008*. Vol. 4965. LNCS. Berlin, Heidelberg: Springer, 2008, pp. 181–197. ISBN: 978-3-540-78966-6 978-3-540-78967-3. DOI: 10.1007/978-3-540-78967-3_11.

[9] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Seoul, South Korea: Springer, Heidelberg, Germany, 2011, pp. 41–69. DOI: 10.1007/978-3-642-25385-0_3.

[10] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. "Tight Bounds on Quantum Searching". In: *Fortschritte der Physik* 46.4-5 (1998). Eprint is arXiv:quant-ph/9605034, pp. 493–505. ISSN: 1521-3978. DOI: 10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P.

[11] Ran Canetti, Oded Goldreich, and Shai Halevi. "The Random Oracle Methodology, Revisited". In: *STOC 1998*. ACM, 1998, pp. 209–218.

[12] Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao. *On the Compressed-Oracle Technique, and Post-Quantum Security of Proofs of Sequential Work*. arXiv:2010.11658 [quant-ph]. 2020.

[13] Jan Czajkowski, Leon Groot Bruinderink, Andreas Hülsing, Christian Schaffner, and Dominique Unruh. "Post-quantum security of the sponge construction". In: *PQCrypto 2018*. Vol. 10786. LNCS. Springer, 2018, pp. 185–204.

[14] Jan Czajkowski, Andreas Hülsing, and Christian Schaffner. "Quantum Indistinguishability of Random Sponges". In: *Advances in Cryptology – CRYPTO 2019*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Cham: Springer International Publishing, 2019, pp. 296–325. ISBN: 978-3-030-26951-7.

[15] Jan Czajkowski, Christian Majenz, Christian Schaffner, and Sebastian Zur. *Quantum Lazy Sampling and Game-Playing Proofs for Quantum Indifferentiability*. arXiv:1904.11477 [quant-ph]. 2020.

[16] E. Ehsan Ebrahimi and Dominique Unruh. "Quantum collision-resistance of non-uniformly distributed functions: upper and lower bounds". In: *Quantum Information & Computation* 18.15&16 (2018), pp. 1332–1349. DOI: 10.26421/QIC18.15-16.

[17] Akinori Hosoyamada and Tetsu Iwata. Personal communication. 2021.

[18] Akinori Hosoyamada and Tetsu Iwata. "4-Round Luby-Rackoff Construction is a qPRP". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 145–174. DOI: 10.1007/978-3-030-34578-5_6.

[19] Andreas Hülsing, Joost Rijneveld, and Fang Song. "Mitigating Multi-target Attacks in Hash-Based Signatures". In: *PKC 2016 , Part I*. Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Vol. 9614. LNCS. Taipei, Taiwan: Springer, Heidelberg, Germany, 2016, pp. 387–416. DOI: 10.1007/978-3-662-49384-7_15.

[20] Gembu Ito, Akinori Hosoyamada, Ryutaroh Matsumoto, Yu Sasaki, and Tetsu Iwata. "Quantum Chosen-Ciphertext Attacks Against Feistel Ciphers". In: *Topics in Cryptology – CT-RSA 2019*. Springer International Publishing, 2019, pp. 391–411. DOI: 10.1007/978-3-030-12612-4_20.

[21] Michael Luby and Charles Rackoff. "How to Construct Pseudorandom Permutations from Pseudorandom Functions". In: *SIAM Journal on Computing* 17.2 (Apr. 1988), pp. 373–386. DOI: 10.1137/0217022.

[22] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Draft FIPS 202. Available at http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf. 2014.

[23] Ehsan Ebrahimi Targhi, Gelo Noel Tabia, and Dominique Unruh. "Quantum Collision-Resistance of Non-uniformly Distributed Functions". In: *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*. Ed. by Tsuyoshi Takagi. Springer, Heidelberg, Germany, 2016, pp. 79–85. DOI: 10.1007/978-3-319-29360-8_6.

[24] Dominique Unruh. *Compressed Permutation Oracles (And the Collision-Resistance of Sponge/SHA3)*. IACR Cryptology ePrint Archive, 2021/062. 2021.

[25] Dominique Unruh. "Computationally Binding Quantum Commitments". In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Vienna, Austria: Springer, Heidelberg, Germany, 2016, pp. 497–527. DOI: 10.1007/978-3-662-49896-5_18.

[26] Dominique Unruh. "Non-Interactive Zero-Knowledge Proofs in the Quantum Random Oracle Model". In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Sofia, Bulgaria: Springer, Heidelberg, Germany, 2015, pp. 755–784. DOI: 10.1007/978-3-662-46803-6_25.

[27] Mark Zhandry. *A Note on Quantum-Secure PRPs*. Cryptology ePrint Archive, Report 2016/1076. http://eprint.iacr.org/2016/1076. 2016.

[28] Mark Zhandry. "A note on the quantum collision and set equality problems". In: *Quantum Inf. Comput.* 15.7&8 (2015), pp. 557–567. DOI: 10.26421/QIC15.7-8.

[29] Mark Zhandry. "How to Construct Quantum Random Functions". In: *FOCS 2013*. IEEE, 2012, pp. 679–687. DOI: 10.1109/FOCS.2012.37.

[30] Mark Zhandry. "How to Record Quantum Queries, and Applications to Quantum Indifferentiability". In: *Crypto 2019*. Vol. 11693. LNCS. Springer, 2019, pp. 239–268. DOI: 10.1007/978-3-030-26951-7_9.

[31] Mark Zhandry. "Secure Identity-Based Encryption in the Quantum Random Oracle Model". In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, pp. 758–775. DOI: 10.1007/978-3-642-32009-5_44.

[32] Mark Zhandry. "Secure Identity-Based Encryption in the Quantum Random Oracle Model". In: *Crypto 2012*. Vol. 7417. LNCS. Springer, 2012, pp. 758–775. ISBN: 978-3-642-32008-8. DOI: 10.1007/978-3-642-32009-5_44.

# Symbol index

| | | |
|---|---|---|
| $\lvert x\rvert$ | Absolute value/cardinality | |
| $\lVert\psi\rVert$ | Norm of $\psi$ | 4 |
| $\lVert\!\lVert A\rVert\!\rVert$ | Operator norm of $A$ | 4 |
| $\lvert x\rangle$ | Computational basis state | 4 |
| $\langle x\rvert$ | Adjoint of computational basis state | 4 |
| $\operatorname{span} X$ | Span of vectors $X$ | 4 |
| $\mathsf{StO}$ | Standard oracle | 5, 6 |
| $\mathsf{StO}_\mathsf{s}$ | Sanitized standard oracle | 10 |
| $\operatorname{im} f$ | Image of (partial) function $f$ | 4 |
| $\operatorname{dom} f$ | Domain of (partial) function $f$ | 4 |
| $D \rightharpoonup R$ | Partial functions from $D$ to $R$ | 4 |
| $D \to R$ | Functions from $D$ to $R$ | 4 |
| $D \hookrightarrow R$ | Injective functions from $D$ to $R$ | 4 |
| $\mathbb{C}$ | Complex numbers | |
| $\mathcal{D}$ | A distribution | |

| | | |
|---|---|---|
| $\mathcal{H}$ | A Hilbert space | |
| $\mathsf{Func}$ | Set of valid uncompressed functions in oracle | 5 |
| $\oplus$ | "Group operation" used to implement quantum queries | 5 |
| $D$ | Domain of the random oracle | 5 |
| $R$ | Range of the random oracle | 5 |
| $N$ | Size of the random oracle range ($N := \|R\|$) | 5 |
| $M$ | Size of the random oracle domain ($M := \|D\|$) | 5 |
| $\mathsf{Decomp}_1$ | Decompression operation in Zhandry's compressed oracle, one register | 6 |
| $\mathsf{Decomp}$ | Decompression operation in Zhandry's compressed oracle, all registers | 7 |
| $\|*\rangle$ | Uniform superposition, $\sum_x \frac{1}{\sqrt{N}}\|x\rangle$ | 8 |
| $\mathsf{CFO}$ | Compressed function oracle | 7 |
| $\mathsf{CFO}_\mathsf{s}$ | Compressed function oracle | 10 |
| $\mathsf{CPO}$ | Compressed function oracle, consisting of two oracles $\mathsf{CFO}$ and $\mathsf{Flip}$ | 13 |
| $\mathsf{Flip}$ | Permutation flipping oracle | 12 |
| $U_\perp$ | Part of $\mathsf{Decomp}_1$, swaps $\|\perp\rangle$ and $\|0\rangle$ | 6 |
| $Q$ | Part of $\mathsf{Decomp}_1$, quantum Fourier transform (or not) | 6 |
| $\overset{\varepsilon}{\approx}$ | $\varepsilon$-close with respect to $\|\cdot\|$ | 4, 4 |
| $\varnothing$ | Empty set / empty partial function | 4 |

# Index


CFO, *see* compressed function oracle
    sanitized, 9
closed span, 4
compressed function oracle, 5
compressed oracle
    Zhandry's, 5

database, 7
decompression operation, 6

empty partial function, 4

partial function, 4
    empty, 4
projector, 4

sanitized CFO, 9
span
    (closed), 4
standard oracle, 6

total function, 4
total injection, 4