Quantum Lazy Sampling and Game-Playing Proofs for Quantum Indifferentiability

Jan Czajkowski[*1], Christian Majenz[†1], Christian Schaffner[‡1], and Sebastian Zur[§1]

[1]QuSoft, University of Amsterdam

April 26, 2019

Abstract

Game-playing proofs constitute a powerful framework for classical cryptographic security arguments, most notably applied in the context of indifferentiability. An essential ingredient in such proofs is lazy sampling of random primitives. We develop a quantum game-playing proof framework by generalizing two recently developed proof techniques. First, we describe how Zhandry's compressed quantum oracles [Zha18] can be used to do quantum lazy sampling from non-uniform function distributions. Second, we observe how Unruh's one-way-to-hiding lemma [Unr14] can also be applied to compressed oracles, providing a quantum counterpart to the fundamental lemma of game-playing.

Subsequently, we use our game-playing framework to prove quantum indifferentiability of the sponge construction, assuming a random internal function or a random permutation. Our results upgrade post-quantum security of SHA-3 to the same level that is proven against classical adversaries.

[*]j.czajkowski@uva.nl
[†]c.majenz@uva.nl
[‡]c.schaffner@uva.nl
[§]zursebastian@gmail.com

Contents

1 Introduction

The modern approach to cryptography relies on mathematical rigor: Trust in a given cryptosystem is mainly established by proving that, given a set of assumptions, it fulfills a security definition formalizing real-world security needs. Apart from the definition of security, the mentioned assumptions include the threat model, specifying the type of adversaries we want to be protected against. One way of formalizing the above notions is via *games*, i.e. programs interacting with the adversaries and outputting a result signifying whether there has been a breach of security or not. Adversaries in this picture are also modeled as programs, or more formally Turing machines.

The framework of game-playing proofs introduced by Bellare and Rogaway in [BR06]—modeling security arguments as games, played by the adversaries—is especially useful because it makes proofs easier to verify. Probabilistic considerations might become quite involved when talking about complex systems and their interactions; the structure imposed by games, however, simplifies them. In the game-playing framework, randomness can be, for example, considered to be sampled on the fly, making conditional events easier to analyze. A great example of that technique is given in the proof of the PRP/PRF switching lemma in [BR06].

In this work we focus on idealized security notions; In the Random Oracle Model (ROM) one assumes that the publicly accessible hash functions are in fact random [BR93]. This is a very useful assumption as it makes the proof easier, but also cryptographic constructions designed with ROM in mind are more efficient. A modification of the original idea—that we rely on—is to assume the primitives used to construct larger objects are ideal, i.e. random.

We are interested in the post-quantum threat model, which is motivated by the present strong efforts to build a quantum computer. It has been shown that quantum computers can efficiently solve problems that are considered hard for classical machines. Hardness of the factoring and discrete-logarithm problems is, e.g., important for public-key cryptography, but these problems can be solved efficiently on a quantum computer using Shor's algorithm [Sho94]. The obvious formalization of the threat model is to include adversaries operating a fault-tolerant quantum computer, which is in particular capable of running the mentioned attacks. This model is the basis of the field of post-quantum cryptography [BBD09].

While the attacks based on Shor's algorithm are the most well-known ones, public-key cryptography may not be the only area with quantum vulnerabilities. Many cryptographic hash functions are based on publicly available compression functions [Mer90; Dam90; Ber+07] and as such they could be run on a quantum machine. This fact motivates us to analyze adversaries that have quantum access to the public building blocks of the cryptosystem. Therefore, the quantum threat model takes us from the Random-Oracle Model [BR93]—often used in the context of hash functions—to the Quantum Random-Oracle Model [Bon+11] (QROM), where the random oracle can be accessed in superposition.

Having highlighted a desirable proof structure—fitting the clear and easy-to-verify game-playing framework—and the need of including fully quantum adversaries with quantum access to random oracles into the threat model, we encounter an obvious challenge: defining a *quantum* game-playing framework. In this article, we resolve that challenge and apply the resulting framework to the setting of hash functions. In the following paragraphs we are going to describe our results and point to the main proof techniques we used to achieve them.

Our Results. We define a quantum game-playing framework for security proofs that involve fully quantum adversaries. Our framework generalizes two recently developed proof techniques: *compressed* quantum random oracles by Zhandry [Zha18] and the One-Way to Hiding (O2H) lemma by Unruh [Unr14; AHU18]. The former provides a way to lazy-sample a quantum-accessible random oracle, and the latter is a quantum counterpart of the Fundamental Game-Playing lemma—a key ingredient in the original game-playing framework. In this paper, we show how to generalize the two methods and merge them to form a clear and powerful tool for proofs in post-quantum cryptography. In particular, we generalize the compressed-oracle technique of [Zha18] to non-uniform distributions over functions, allowing a much more general form of (quantum) lazy sampling. Subsequently, we observe that the techniques of ”puncturing oracles” proposed in [AHU18] can also be applied to compressed oracles, yielding a more general version of the O2H lemma which forms the quantum counterpart of the fundamental game-playing lemma.

We go on to present a use case for our quantum game-playing framework by proving quantum indifferentiability of the sponge construction [Ber+07] used in SHA3. More precisely, we show that the sponge construction is indifferentiable from a random oracle in case the internal function is i) a random function, or ii) a random permutation. Above we call our approach a framework, that is because of the wide range of possible applications and modular structure of the method. While compressed-oracle method from [Zha18] is already widely applicable, the easy translation of classical proofs to the quantum setting is the added value of our framework.

Additionally we prove that quantum indifferentiability implies collapsingness [Unr16b; Unr16a], a quantum security notion strengthening collision resistance. Given our indifferentiability results concerning the sponge construction, that gives a proof of collapsingness of the sponge construction with random permutations, closing the security gap left in [Cza+18]. Note that our result concerns only the QROM and in [Cza+18] the authors discuss standard security notions: collision resistance and collapsingness.

Related Works. Indifferentiability is a security notion developed by Maurer, Renner, and Holenstein [MRH04] commonly used for hash-function domain-extension schemes [Cor+05; Ber+08]. Here, it captures the adversary's access to both the construction and the internal function.

The subject of quantum indifferentiability, addressed in our work, has been recently analyzed in two articles. Carstens, Ebrahimi, Tabia, and Unruh make a case in [Car+18] against the possibility of fulfilling the definition of indifferentiability for quantum adversaries. Assuming a technical conjecture, they prove a theorem stating that if two systems are perfectly (with zero advantage) quantumly indifferentiable then there is a stateless classical indifferentiability simulator. In the last part of their work they show that there cannot be a stateless simulator for domain-decreasing constructions—i.e. most constructions for hash functions. Zhandry on the other hand [Zha18] develops a technique that allows to prove indifferentiability for the Merkle-Damgård construction. His result does not contradict the result of [Car+18], as it is *imperfect*, albeit with a negligible error. The technique of that paper, compressed quantum oracles, is one of the two main ingredients of our framework. Recent work by Unruh and by Ambainis, Hamburg, and Unruh [Unr14; AHU18] form the second main ingredient of our result. They show the One-Way to Hiding (O2H) Lemma, which is the quantum counterpart of the Fundamental Game-Playing lemma—a key ingredient in the original game-playing framework. The O2H lemma provides a way to ”reprogram” quantum accessible oracles on some set of inputs, formalized as ”punctured” oracles in the latter paper.

The quantum security of domain-extension schemes has been the topic of several recent works. [SY17; CHS19] study domain extension for message authentication codes and pseudorandom functions. For random inner function, [Zha18] has proven indifferentiability of the Merkle-Damgård construction which hence has strong security in the QROM. For hash functions in the standard model, quantum generalizations of collision resistance were defined in [Unr16b; Ala+18]. For one of them, collapsingness, some domain extension schemes including the Merkle-Damgård and sponge constructions, have been shown secure [Cza+18; Feh18; Unr16a].

In a recent article [Unr19a] Unruh developed quantum Relational Hoare Logic for computer verification of proofs in (post-)quantum cryptography. There he also uses the approach of game-playing, but in general focuses on formal definitions of quantum programs and predicates. To investigate the relation between [Unr19a] and our work in more detail one would have to express our results in the language of the new logic. We leave it as an interesting direction for the future. The proof techniques of [Zha18] and [AHU18] have been recently used to show security of the 4-Round Luby-Rackoff construction in [HI19] and of generic key-encapsulation mechanisms in [JZM19] respectively.

Organization. In Section 2 we introduce the crucial classical notions we use. We provide the necessary definitions of the classical game-playing framework and indifferentiability needed in the remainder of the paper. In Section 3 we generalize the compressed-oracle technique of [Zha18] to non-uniform distributions over functions. In Section 4 we prove a generalization of the O2H lemma of [Unr14], adapted to the use with compressed oracles for non-uniform distributions. The quantum game-playing framework is defined via the general compressed quantum oracles that appear in security games, and a way to use hybrid arguments between different games coming from our version of the O2H lemma. In Section 5 we use these results to prove quantum indifferentiability of the sponge construction, thereby also proving collapsingness.

2 Preliminaries

We write $[N] := \{0, 1, \ldots, N-1\}$ for the set of size N. We denote the Euclidean norm of a vector $|\psi\rangle \in \mathbb{C}^d$ by $\||\psi\rangle\|$. By $x \leftarrow \mathsf{A}$ we denote sampling x from a distribution or getting the output of a randomized algorithm. A summary of symbols used throughout the paper can be found in the Symbol Index.

2.1 Classical Game-Playing Proofs

Many proofs of security in cryptography follow the Game-Playing framework, proposed in [BR06]. It is a very powerful technique as cryptographic security proofs tend to be simpler to follow and formulate in this framework. The central idea of this approach are *identical-until-bad* games. Say games G and H are two programs that are syntactically identical except for code that follows after setting a flag Bad to one, then we call those games identical-until-bad. Usually in cryptographic proofs G and H will represent two functions that an adversary A will have oracle access to. In the following we denote the situation when A interacts with H by A^{H}. Then we can say the following about the adversary's view.

Lemma 1 (Fundamental lemma of game-playing, Lemma 2 of [BR06])**.** *Let* G *and* H *be identical-until-bad games and let* A *be an adversary that outputs a bit* b*. Then*

$$\left|\mathbb{P}[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{H}}] - \mathbb{P}[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{G}}]\right| \leq \mathbb{P}[\mathrm{Bad} = 1 : \mathsf{A}^{\mathsf{G}}]. \tag{1}$$

2.2 Indifferentiability

In the Random-Oracle Model (ROM) we assume the hash function used in a cryptosystem to be a random function [BR93]. This model is very useful in cryptographic proofs but might not be applicable if the discussed hash function is constructed using some internal function. The ROM can still be used in this setting but by assuming the internal function is random. The notion of security is then *indistinguishability* of the constructed functions from a random oracle. In most constructions however (such as in SHA-2 [NIS15] and SHA-3 [NIS14]), the internal function is publicly known, rendering the security notion of indistinguishability too weak. A notion of security dealing with this issue is *indifferentiability* introduced by Maurer, Renner, and Holenstein [MRH04].

Access to the publicly known internal function and the hash function constructed from it is handled by *interfaces*. An interface to a system is an access structure defined by the format

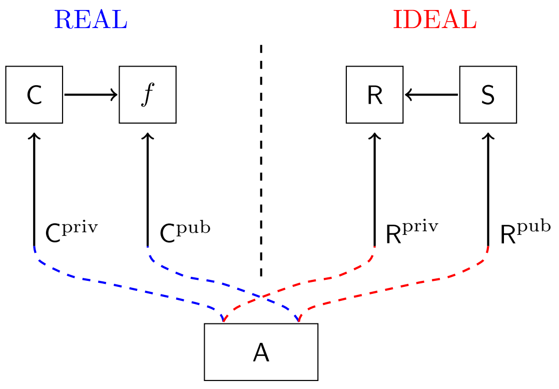

Figure 1: A schematic representation of the notion of indifferentiability, Def. 2. Arrows denote "access to" the pointed system.

of inputs and expected outputs. Let us illustrate this definition by an example, let the system C under consideration be a hash function $\mathsf{H}_f : \{0,1\}^* \to \{0,1\}^n$, constructed using a function $f : \{0,1\}^n \to \{0,1\}^n$. Then the *private* interface of the system accepts finite-length strings as inputs and outputs n-bit long strings. Outputs from the private interface are generated by the hash function, so we can write (slightly abusing notation) $\mathsf{C}^{\mathrm{priv}} = \mathsf{H}_f$. The public interface accepts n-bit long strings and outputs n-bit strings as well. We have that $\mathsf{C}^{\mathrm{pub}} = f$. Often we consider one of the analyzed systems, R, to be a random oracle. Then both interfaces are the same and output random outputs of appropriate given length.

The following definitions and Theorem 4 are the rephrased versions of definitions and theorems from [MRH04; Cor+05]. We also make explicit the fact that the definitions are independent of the threat model we consider—whether it is the classical model or the quantum model. To expose those two cases we write "classical or quantum" next to algorithms that can be classical or quantum machines; Communication between algorithms (systems, adversaries, and environments) can also be of two types, where quantum communication will involve quantum states (consisting of superpositions of inputs)—explained in more detail in the remainder of the paper.

Definition 2 (Indifferentiability [MRH04]). *A cryptographic (classical or quantum) system* C *is* (q,ε)-indifferentiable *from* R*, if there is an efficient (classical or quantum)* simulator S *and a negligible function* ε *such that for any efficient (classical or quantum)* distinguisher D *with binary output (0 or 1) the advantage*

$$\left|\mathbb{P}\left[b=1 : b \leftarrow \mathsf{D}[\mathsf{C}_k^{\mathrm{priv}}, \mathsf{C}_k^{\mathrm{pub}}]\right] - \mathbb{P}\left[b=1 : b \leftarrow \mathsf{D}[\mathsf{R}_k^{\mathrm{priv}}, \mathsf{S}[\mathsf{R}_k^{\mathrm{pub}}]]\right]\right| \leq \varepsilon(k)\,, \qquad (2)$$

where k *is the security parameter. The distinguisher makes at most* q *(classical or quantum) queries to* C.

By efficient we mean with runtime that is polynomial in the security parameter k. The definitions are still valid and the theorem below holds also if we interpret efficiency in terms of queries made by the algorithms. Note that then we can allow the algorithms to be unbounded with respect to runtime, the distinction between quantum and classical queries is still of crucial importance though. By square brackets we denote (classical or quantum) oracle access to some algorithm, we also use A^{H} if the oracle is denoted by a more confined symbol. In Fig. 1 we present a a scheme of the situation captured by Def. 2.

Definition 3 (As secure as [MRH04]). *A cryptographic (classical or quantum) system* C *is said to be* as secure as C' *if for all efficient (classical or quantum) environments* Env *the following holds: For* *any efficient (classical or quantum) attacker* A *accessing* C *there exists another (classical or quantum) attacker* A' *accessing* C' *such that the difference between the probability distributions of the binary outputs of* $\mathsf{Env}[\mathsf{C},\mathsf{A}]$ *and* $\mathsf{Env}[\mathsf{C}',\mathsf{A}']$ *is negligible, i.e.*

$$\left|\mathbb{P}\left[b=1 : b \leftarrow \mathsf{Env}[\mathsf{C},\mathsf{A}]\right] - \mathbb{P}\left[b=1 : b \leftarrow \mathsf{Env}[\mathsf{C}',\mathsf{A}']\right]\right| \leq \varepsilon(k)\,, \tag{3}$$

where ε *is a negligible function.*

Indifferentiability is a strong notion of security mainly because if fulfilled it guarantees composability of the secure cryptosystem. In the following we say that a cryptosystem T is *compatible* with C if the interfaces for interacting of T with C are matching.

Theorem 4 (Composability [MRH04])**.** *Let* T *range over (classical or quantum) cryptosystems compatible with* C *and* R*, then* C *is* (q,ε)*-indifferentiable from* R *if and only if for all* T*,* $\mathsf{T}[\mathsf{C}]$ *is as secure as* $\mathsf{T}[\mathsf{R}]$*.*

Note that composability that is guaranteed by the above theorem holds only for *single-stage* games [RSS11].

Indifferentiability is a strong security notion guaranteeing that a lower-level function (e.g. a random permutation) can be used to construct a higher-level object (e.g. a variable input-length random function) that is "equivalent" to the ideal one—in the sense of Thm. 4. Here, an adversary's complexity is measured in terms of the number of queries to the oracles only, not in terms of their time complexity. In quantum indifferentiability adversaries are allowed to access the oracles in superposition. This is necessary in the post-quantum setting, as the building blocks of many hash functions—like e.g those of SHA3 [NIS14]—are publicly specified and can be implemented on a quantum computer.

2.3 Quantum Computing

The model of quantum adversaries we use is quantum algorithms making q queries to an oracle. Each query is intertwined by a unitary operation acting on the adversary's state and all her auxiliary states. A general introduction to quantum computing can be found in [NC11]. Here we will only introduce specific operations important to understand the paper.

Let us define the *Quantum Fourier Transform* (QFT), a unitary change of basis that we will make heavy use of. For $N \in \mathbb{N}_{>0}$ and $x, \xi \in [N] = \mathbb{Z}_N$ the transform is defined as

$$\mathsf{QFT}_N|x\rangle := \frac{1}{\sqrt{N}} \sum_{\xi\in[N]} \omega_N^{\xi\cdot x}|\xi\rangle, \tag{4}$$

where $\omega_N := e^{\frac{2\pi i}{N}}$ is the N-th root of unity. An important identity for some calculations is

$$\sum_{\xi\in[N]} \omega_N^{x\cdot\xi} \cdot \bar{\omega}_N^{x'\cdot\xi} = N\delta_{x,x'}, \tag{5}$$

where $\bar{\omega}_N = e^{-\frac{2\pi i}{N}}$ is the complex conjugate of ω_N and $\delta_{x,x'}$ is the Kronecker delta function.

If we talk about n qubits an identity on their Hilbert space is denoted by $\mathbb{1}_n$, we also use this notation to denote the dimension of the identity operator, the actual meaning will be clear from the context. We write U^A to denote that we act with U on register A.

3 Quantum-Accessible Oracles

In the Quantum-Random-Oracle Model (QROM) [Bon+11], one assumes that the random oracle can be accessed in superposition. Quantum-accessible random oracles are motivated by the possibility of running an actual instantiation of the oracle as function on a quantum computer, which would allow for superposition access. In this section, oracles implement a function $f : \mathcal{X} \to \mathcal{Y}$ distributed according to some probability distribution $\mathfrak{D}$ on the set $\mathcal{F}$ of functions from $\mathcal{X}$ to $\mathcal{Y}$. Without loss of generality we set $\mathcal{X} = \mathbb{Z}_M$ and $\mathcal{Y} = \mathbb{Z}_N$ for some integers $M, N > 0$.

Classically, an oracle for a function f is modeled via a tape with the queried input x written on it, the tape is then overwritten with $f(x)$. The usual way of translating this functionality to the quantum circuit model is by introducing a special gate that implements the unitary $\mathsf{U}_f|x, y\rangle = |x, y + f(x)\rangle$. In the literature $+$ is usually the bitwise addition modulo 2, but in general it can be any group operation. We are going to use addition in $\mathbb{Z}_N$.[1]

In the case where the function f is a random variable, so is the unitary U_f. Sometimes this is not, however, the best way to think of a quantum random oracle, as the randomness of f is accounted for using classical probability theory, yielding a hybrid description. To capture the adversary's point of view more explicitly, it is necessary to switch to the *mixed-state formalism*. A mixed quantum state, or *density matrix*, is obtained by considering the projector onto the one-dimensional subspace spanned by a pure state, and then taking the expectation over any classical randomness. Say that the adversary sends the query state $|\Psi_0\rangle = \sum_{x,y} \alpha_{x,y}|x, y\rangle$ to the oracle, the output state is then

$$\begin{aligned}&\sum_f \mathbb{P}[f : f \leftarrow \mathfrak{D}]\ \mathsf{U}_f|\Psi_0\rangle\langle\Psi_0|\mathsf{U}_f^\dagger\ \otimes\ |f\rangle\langle f|_F \\ &= \sum_f \mathbb{P}[f : f \leftarrow \mathfrak{D}] \sum_{x,x',y,y'} \alpha_{x,y}\bar{\alpha}_{x',y'}|x, y + f(x)\rangle\langle x', y' + f(x')| \otimes |f\rangle\langle f|_F,\end{aligned} \tag{6}$$

where by $\bar{\alpha}$ we denote the complex conjugate of α and we have recorded the random function choice in a classical register F holding the full function table of f.

In quantum information science, a general recipe for simplifying the picture and to gain additional insight is to *purify* mixed states, i.e. to consider a pure quantum state on a system augmented by an additional register E, such that discarding E recovers the original mixed state. In [Zha18] Zhandry applies this recipe to this quantum-random-oracle formalism.

In the resulting representation of a random oracle, the classical register F is replaced by a quantum register holding a superposition of functions from $\mathfrak{D}$. The joint state before an adversary makes the first query with a state $|\Psi_0\rangle_{XY}$ is $|\Psi_0\rangle_{XY} \sum_{f\in\mathcal{F}} \sqrt{\mathbb{P}[f : f \leftarrow \mathfrak{D}]}\, |f\rangle_F$. The unitary that corresponds to U_f after purification will be called the *Standard Oracle* StO and works by reading the appropriate output of f from F and adding it to the algorithm's output register,

$$\mathsf{StO}|x, y\rangle_{XY}|f\rangle_F := |x, y + f(x)\rangle_{XY}|f\rangle_F. \tag{7}$$

Applied to a superposition of functions as intended, StO will entangle the adversary's registers XY with the oracle register F.

The main observation of [Zha18] is that if we change the basis of the initial state of the oracle register F, the redundancy of this initial state becomes apparent. If we are interested in, e.g., an oracle for a uniformly random function, the Fourier transform changes the initial oracle state $\sum_f \frac{1}{\sqrt{|\mathcal{F}|}}|f\rangle$ to a state holding only zeros $|0^M\rangle$, where $0 \in \mathcal{Y}$. The uniform case is treated in great detail in [Unr19b].

Let us start by presenting the interaction of the adversary viewed in the same basis, called the Fourier basis. The unitary operation acting in the Fourier basis is called the *Fourier Oracle* FO. Another important insight from [Zha18] is that the Fourier Oracle, instead of adding the

[1]Note that introducing the formalism using the group $\mathbb{Z}_N$ for some $N \in \mathbb{N}$ is quite general in the following sense: Any finite Abelian group G is isomorphic to a product of cyclic groups, and the (quantum) Fourier transform with respect to such a group is the tensor product of the Fourier transforms on the cyclic groups, given the natural tensor product structure of $\mathbb{C}^G$.

output of the oracle to the adversary's output register, does the opposite: It adds the value of the adversary's *output* register to the (Fourier-)transformed truth table

$$\mathsf{FO}|x,\eta\rangle_{XY}|\phi\rangle_F := |x,\eta\rangle_{XY}|\phi-\chi_{x,\eta}\rangle_F, \tag{8}$$

where ϕ is the transformed truth table f and $\chi_{x,\eta} := (0,\ldots,0,\eta,0,\ldots,0)$ is a transformed truth table equal to 0 in all rows except for row x, where it has the value η. Note that we subtract $\chi_{x,\eta}$ so that the reverse of QFT returns addition of $f(x)$.

Classically, a (uniformly) random oracle can be "compressed" by lazy-sampling the responses, i.e. by answering with previous answers if there are any, and with a fresh random value otherwise. Is lazy-sampling possible for quantum accessible oracles? Surprisingly, the answer is yes. Thanks to the groundbreaking ideas presented in [Zha18] we know that there exists a representation of a quantum random oracle that is efficiently implementable.

In the remainder of this section we present an efficient representation of oracles for functions f sampled from an arbitrary distribution that fulfills the quantum analogue of the classical condition of efficiently sampleable conditional distributions. In the first part we introduce a general structure of quantum-accessible oracles. In the second part we generalize the idea of compressed random oracles to deal with non-uniform distributions of functions. We discuss random permutations in detail. In Appendix A, we provide additional details on the implementation of the procedures introduced in this section and step-by-step calculations of important identities and facts concerning compressed oracles. In Appendix A.1 we recall in detail the compressed oracle introduced in [Zha18], where the distribution of functions is uniform and the functions map bitstrings to bitstrings. We show the oracle in different bases and present calculations that might be useful for developing intuition for working with the new view on quantum random oracles.

3.1 General Structure of the Oracles

In this subsection we describe the general structure of quantum-accessible oracles that will give us a high-level description of all the oracles we define in this paper. A quantum-accessible random oracle consists of

1. Hilbert spaces for the input $\mathcal{H}_{\mathcal{X}}$, output $\mathcal{H}_{\mathcal{Y}}$, and state registers $\mathcal{H}_{\mathcal{F}}$,
2. a procedure $\mathsf{Samp}_{\mathfrak{D}}$ that, on input a subset of the input space of the functions in $\mathfrak{D}$, prepares a superposition of partial functions on that subset of inputs with weights according to the respective marginal of the distribution $\mathfrak{D}$,
3. an update unitary $\mathsf{FO}_{\mathfrak{D}}$ that might depend on $\mathfrak{D}$ (in the case of compressed oracles) or not (e.g. in the definition from Eq.(8)).

First of all, let us note that we use the Fourier picture of the oracle as the basis for our discussion. This picture, even though less intuitive at first sight, is simpler to handle mathematically. The distribution of the functions we model by the quantum oracle are implicitly given by the procedure $\mathsf{Samp}_{\mathfrak{D}}$ that when acting on the $|0\rangle$ state generates a superposition of values consistent with outputs of a function f sampled from $\mathfrak{D}$.

In the above structure the way we implement the oracle—in a compressed way, or acting on full function tables—depends on the way we define $\mathsf{FO}_{\mathfrak{D}}$.

In full generality, $\mathsf{Samp}_{\mathfrak{D}}$ can depend on both the input to f and previous queries made by the adversary. Accordingly, $\mathsf{FO}_{\mathfrak{D}}$ might behave differently depending to the distribution $\mathfrak{D}$. The definition of $\mathsf{Samp}_{\mathfrak{D}}$ is such that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X})|0^N\rangle = \sum_{f\in\mathcal{F}}\sqrt{\mathbb{P}[f\leftarrow\mathfrak{D}]}|f\rangle$ and is a unitary operator. Whether $\mathsf{Samp}_{\mathfrak{D}}$ is defined on smaller inputs might depend on the distribution.

Quantum-accessible oracles work as follows. First the oracle state is prepared in an all-zero state. Then at every query by the adversary we run $\mathsf{FO}_{\mathfrak{D}}$ which updates the state of the database. Further details are provided in the following sections.

3.2 Non-uniform Oracles

One of the main results of this paper is generalizing the idea of purification and compression of quantum random oracles to non-uniform function distributions. Examples of such functions are random permutations or Boolean functions that output one with a given probability.

We aim at the following functionality

$$\mathsf{StO}|x,y\rangle_{XY}\sum_{f\in\mathcal{F}}\sqrt{\mathbb{P}[f:f\leftarrow\mathfrak{D}]}\,|f\rangle_F=\sum_{f\in\mathcal{F}}\sqrt{\mathbb{P}[f:f\leftarrow\mathfrak{D}]}\,|x,y+f(x)\mod N\rangle_{XY}\,|f\rangle_F,\tag{9}$$

where $\mathfrak{D}$ is a distribution on the set of functions $\mathcal{F}=\{f:\mathcal{X}\to\mathcal{Y}\}$. The first ingredient we need is an operation that prepares the superposition of function truth tables according to the given distribution. More formally, we know a unitary that for all $\mathcal{S}\subseteq\mathcal{X}$

$$\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S})|0^{|\mathcal{S}|}\rangle_{F(\mathcal{S})}=\sum_{f(x):f\in\mathcal{F},x\in\mathcal{S}}\sqrt{\mathbb{P}[f(\mathcal{S}):f\leftarrow\mathfrak{D}]}\bigotimes_{x\in\mathcal{S}}|f(x)\rangle_{F(x)},\tag{10}$$

where by $f(\mathcal{S})$ we denote the part of the full truth table of f corresponding to inputs from $\mathcal{S}$ and by $F(x)$ register corresponding to x. Later we give explicit examples of $\mathsf{Samp}_{\mathfrak{D}}$ for different $\mathfrak{D}$. Applying QFT to the adversary's register gives us the *Phase Oracle* PhO that changes the phase of the state according to the output value $f(x)$. This picture is commonly used in the context of bitstrings but is not very useful in our context. Additionally transforming the oracle register brings us to the Fourier Oracle, that we will focus on. This series of transformations can be depicted as a chain of oracles:

$$\mathsf{StO}\xleftrightarrow{\mathsf{QFT}_N^Y}\mathsf{PhO}\xleftrightarrow{\mathsf{QFT}_N^F}\mathsf{FO},\tag{11}$$

going "to the right" is done by applying QFT_N and "to the left" by applying the adjoint. The non-uniform Fourier Oracle is just $\mathsf{FO}=\mathsf{QFT}_N^{YF}\circ\mathsf{StO}\circ\mathsf{QFT}_N^{\dagger\,YF}$,

$$\begin{aligned}\mathsf{FO}|x,\eta\rangle_{XY}&\sum_{\phi}\frac{1}{\sqrt{N^M}}\sum_{f\in\mathcal{F}}\sqrt{\mathbb{P}[f:f\leftarrow\mathfrak{D}]}\,\omega_N^{\phi\cdot f}\,|\phi\rangle_F\\&=|x,\eta\rangle_{XY}\sum_{\phi}\frac{1}{\sqrt{N^M}}\sum_{f\in\mathcal{F}}\sqrt{\mathbb{P}[f:f\leftarrow\mathfrak{D}]}\,\omega_N^{\phi\cdot f}\,|\phi-\chi_{x,\eta}\mod N\rangle_F.\end{aligned}\tag{12}$$

The main difference between uniform oracles and non-uniform oracles is that in the latter, the initial state of the oracle in the Fourier basis is not necessarily an all-zero state. That is because the unitary $\mathsf{Samp}_{\mathfrak{D}}$—that is used to prepare the initial state—is not the adjoint of the transformation between oracle pictures, like it is the case for the uniform distribution.

Before defining compressed oracles for non-uniform function distributions, let us take a step back and think about classical lazy sampling for such a distribution. Let f be a random function from a distribution $\mathfrak{D}$. In principle, lazy sampling is always possible as follows. When the first input x_1 is queried, just sample from the marginal distribution for $f(x_1)$. Say the outcome is y_1 for the next query with x_2, we sample from the *conditional distribution* of $f(x_2)$ given that $f(x_1)=y_1$, etc.

Whether actual lazy sampling is feasible depends on the complexity of sampling from the conditional distributions of function values given that a polynomial number of other function values are already fixed.

The situation when constructing compressed superposition oracles for non-uniformly distributed random functions is very similar. In this case we need the operations $\mathsf{Samp}_{\mathfrak{D}|f(x_1)=y_1,\ldots,f(x_s)=y_s}$ to be efficiently implementable for the compressed oracle to be efficient. Here, $\mathfrak{D}|f(x_1)=y_1,\ldots,f(x_s)=y_s$ denotes the function distribution on $\mathcal{X}\setminus\mathcal{S}$, with $\mathcal{S}=\{x_1,\ldots,x_s\}$, obtained by conditioning $\mathfrak{D}$ on the event $f(x_1)=y_1\wedge\ldots\wedge f(x_s)=y_s$. We write

$$\mathsf{Samp}_{\mathfrak{D}|f(x_1)=y_1,\ldots,f(x_s)=y_s}(\mathcal{X}\setminus\mathcal{S})=\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}\setminus\mathcal{S}\mid\{x_1,\ldots,x_s\})\tag{13}$$

where by inputting a set to $\mathsf{Samp}_{\mathfrak{D}}$ we mean that the operation will prepare a superposition of outputs to elements of the set. By conditioning on a set $\{x_1, ..., x_s\}$ we mean that pairs (x_i, y_i) are input to $\mathsf{Samp}_{\mathfrak{D}}$ so that we get a sample of the conditional distribution. Hence we get

$$\forall \mathcal{S} \subseteq \mathcal{X} : \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \mathcal{S}) \circ \mathsf{Samp}_{\mathfrak{D}}(\mathcal{S}) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}). \tag{14}$$

We additionally require that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \mathcal{S})$ does not modify the output values of $\mathcal{S}$ and is only controlled on them. Note that while we require that $\mathsf{Samp}_{\mathfrak{D}}$ is *local*, so fulfills Eq. (14), and that it prepares the correct distribution when acting on $|0^N\rangle$, Eq. (10), we also require it to be a valid unitary. In general $\mathsf{Samp}_{\mathfrak{D}}$ fulfilling both requirements can be completed to a full unitary in any way.

Note that for $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \mathcal{S})$ to be efficient, it is not sufficient that the conditional probability distributions $\mathfrak{D}|f(x_1) = y_1, ..., f(x_s) = y_s$ are classically efficiently sampleable. This is because running a reversible circuit obtained from a classical sampling algorithm on a superposition of random inputs will, in general, entangle the sample with the garbage output of the reversible circuit. The problem of efficiently creating a superposition with amplitudes $\sqrt{p(x)}$ for some probability distribution p has appeared in other contexts, e.g. in classical-client quantum fully homomorphic encryption [Mah18].

Before we state the algorithm that realizes the general *Compressed Fourier Oracle* $\mathsf{CFO}_{\mathfrak{D}}$ we provide a high-level description of the procedure. The oracle $\mathsf{CFO}_{\mathfrak{D}}$ is a unitary algorithm that performs quantum lazy sampling, maintaining a compressed database of the adversary's queries. For the algorithm to be correct—indistinguishable for all adversaries from the full oracle—it has to respect the following invariants of the database: The full oracle is oblivious to the *order* in which a set of inputs is queried. Hence the same has to hold for the compressed oracle, i.e. we cannot keep entries (x, η) in the order of queries. We ensure this property by keeping the database *sorted* according to x.

The second issue concerns the danger of storing too much information. If after the query we save (x, η) in the database but the resulting entry would map to $(x, 0)$ in the *unprepared* basis, i.e. the basis before applying Samp, then the compressed database would entangle itself with the adversary, unlike in the case of the full oracle. Hence the database cannot contain 0 in the unprepared basis.

$\mathsf{CFO}_{\mathfrak{D}}$: On input $|x, \eta\rangle$ do the following:

1. Find the index $r \in [q]$ of the register into which we should insert (x, η).
2. If $x \neq x_r$: insert x in a register after the last element of the database and shift it to position r, moving the intermediate registers backwards.
3. Change the basis to the Fourier basis (in which the adversary's η is encoded) and update register r to contain $(x_r, \eta_r - \eta)$, change the basis back to original.
4. Check if any register contains a pair of the form $(x_i, 0)$, if yes subtract x from the first part to yield $(x_i - x, 0) = (0, 0)$ and shift it back to the end of the database.

If after q queries the database has a suffix of ℓ pairs of the form $(0, 0)$, we say the database has $s = q - \ell$ non-padding entries.

Up till now we have described the compressed database only on a high-level, let us now explain the basis changes mentioned above in more detail. To deal with the difference between the initial 0 state and the initial Fourier basis truth tables we use yet another alphabet and define Д (pronounced as [dɛ]) which denotes the unprepared database. We call it like that because the initial state of Д is the all-zero state, moreover only by applying $\mathsf{QFT}_N \circ \mathsf{Samp}_{\mathfrak{D}}$ we transform it to Δ, i.e the Fourier basis database. As we will see, operations on Д are more intuitive and easier to define. We denote an unprepared database by $|Д\rangle_D = |x_1, и_1\rangle_{D_1} |x_2, и_2\rangle_{D_2} \cdots |x_q, и_q\rangle_{D_q}$

(where the cyrillic letter и is pronounced as [i]). By $\Delta^Y(x)$ we denote the η value corresponding to the pair in Δ containing x and by $Д^X$ we denote the x values kept in Д. The intuition behind the preparation procedure is to initialize the truth table of the correct distribution in the correct basis. This notion is not visible in the uniform-distribution case, because there the sampling procedure for the uniform distribution $\mathfrak{U}$ is the Hadamard transform: $\mathsf{Samp}_{\mathfrak{U}} = \mathsf{HT}_n^\dagger$, and the database pictures Δ and Д are equivalent. The following chain of databases similar to Eq. (11) represents different pictures, i.e. bases, in which the compressed database can be viewed

$$|Д\rangle \xleftrightarrow{\mathsf{Samp}_{\mathfrak{D}}} |D\rangle \xleftrightarrow{\mathsf{QFT}_N^D} |\Delta\rangle. \tag{15}$$

Using this notation, Alg. 1 defines the procedure of updates of the database of the compressed database. We refer to Appendix A.2 for the fully detailed description of $\mathsf{CFO}_{\mathfrak{D}}$. We

```
Algorithm 1: General $\mathsf{CFO}_{\mathfrak{D}}$
  Input  : Unprepared database and adversary query: $|x,\eta\rangle_{XY}|Д\rangle_D$
  Output: $|x,\eta\rangle_{XY}|Д'\rangle_D$
1  Count in register $S$ the number of non-padding ($и \neq 0$) entries $s$
2  if $x \notin Д^X$ then                                              // add
3      Copy $x$ to $Д^X$ in the right place and add $1$ to $S$        // Keeping $Д^X$ sorted
4  Apply $\mathsf{QFT}_N^{D^Y}\mathsf{Samp}_{\mathfrak{D}}^D(Д^X)$     // Prepare the database: $Д \mapsto \Delta$
5  Subtract $\eta$ from $\Delta^Y(x)$                                 // update entry with $x$
6  Apply $\mathsf{Samp}_{\mathfrak{D}}^{\dagger D}(Д^X)\mathsf{QFT}_N^{\dagger D^Y}$   // Unprepare the database: $\Delta \mapsto Д$
7  for $i = 1, 2, \ldots, s$ do
8      if $Д_i^Y = 0$ then                                            // remove or do nothing
9          Remove $x$ from $Д_i^X$
10 Update padding of Д and subtract $1$ from $S$ if necessary
11 Uncompute $s$ from register $S$
12 Output $|x,\eta\rangle_{XY}|Д'\rangle_D$                           // $Д'$ is the modified database
```

would like to stress that to keep the compressed oracle $\mathsf{CFO}_{\mathfrak{D}}$ a unitary operation we always keep the database of size q. This can be easily changed by always appending an empty register at the beginning of each query of adversary A. The current formulation of $\mathsf{CFO}_{\mathfrak{D}}$ assumes that there is a bound on the number of queries made by the adversary, this is not a fundamental requirement.

The decompression procedure for the general Compressed Fourier Oracle is given by Alg. 2. The output of the decompression procedure $\phi(Д)$ is the state holding the prepared Fourier-basis truth table of the functions from $\mathfrak{D}$, which by construction is consistent with the adversary's interaction with the compressed oracle.

The decompression can be informally described as follows. The first operation coherently counts the number of $и \neq 0$ and stores the result in a register S. Next we prepare a fresh all-zero initial state of a function from $\mathcal{X}$ to $\mathcal{Y}$, i.e. M registers of dimension N, all in the zero state. These registers will hold the final FO superposition oracle state. The next step is swapping each Y-type register of the CFO-database with the prepared zero state in the FO at the position indicated by the corresponding X-type register in the CFO database. The task left to do is deleting x's from D. It is made possible by the fact that the non-padding entries of the CFO database are nonzero and ordered. That is why we can iterate over the entries of the truth table F and, conditioned on the entry not being 0, delete the last entry of D^X and reducing S by one to update the number of remaining non-padding entries in the CFO-database. Finally, we switch to the correct basis to end up with a full oracle of Fourier type, i.e. a FO.

Algorithm 2: General Decompressing Procedure $\mathsf{Dec}_{\mathfrak{D}}$

```
Input   : Unprepared database: $|Д\rangle_D$
Output: Prepared, Fourier-basis truth table: $|\phi(Д)\rangle$
1  Count in register $S$ the number of non-padding ($и \neq 0$) entries $s$
2  Initialize a state $\bigotimes_{x\in\mathcal{X}}|0\rangle_{F(x)}$
3  for $i = 1, 2, \ldots, s$ do
4      Swap register $D_i^Y$ with $F(x_i)$
5  for $x = M-1, M-2, ..., 0$ do                    // $x \in \mathcal{X}$ in decreasing order
6      if $F(x) \neq 0$ then
7          Subtract $x$ from $D_s^X$
8          Subtract 1 from $S$
9  Discard $D$ and $S$
10 Apply $\mathsf{QFT}_N^F \mathsf{Samp}_{\mathfrak{D}}^F(\mathcal{X})$          // Prepare the database
```

Theorem 5 (Correctness of $\mathsf{CFO}_{\mathfrak{D}}$)**.** *Say $\mathfrak{D}$ is a distribution over functions, let $\mathsf{CFO}_{\mathfrak{D}}$ be as defined in Alg. 1 and FO as in Eq.(12), then for any quantum adversary A making q quantum queries we have*

$$|\Psi_{\mathsf{FO}}\rangle = \mathsf{Dec}_{\mathfrak{D}}|\Psi_{\mathsf{CFO}}\rangle, \tag{16}$$

where $|\Psi_{\mathsf{FO}}\rangle$ is the state resulting from the interaction of A with FO and $|\Psi_{\mathsf{CFO}}\rangle$ is the state resulting from the interaction of A with $\mathsf{CFO}_{\mathfrak{D}}$.

Proof. We will show that

$$\mathsf{FO} \circ \mathsf{Dec}_{\mathfrak{D}} = \mathsf{Dec}_{\mathfrak{D}} \circ \mathsf{CFO}_{\mathfrak{D}}, \tag{17}$$

this is sufficient for the proof of the theorem as $|\Psi_{\mathsf{FO}}\rangle$ is generated by a series of the adversary's unitaries intertwined with oracle calls. If we show that $\mathsf{FO} = \mathsf{Dec}_{\mathfrak{D}} \circ \mathsf{CFO}_{\mathfrak{D}} \circ \mathsf{Dec}_{\mathfrak{D}}^{\dagger}$ then everything that happens on the oracle's register side can be compressed.

Let us start with the action of $\mathsf{Dec}_{\mathfrak{D}}$ on some database state

$$|Д(\vec{x}, \vec{и})\rangle := |x, \eta\rangle_{XY}|x_1, и_1\rangle_{D_1}\cdots|x_s, и_s\rangle_{D_s}\cdots|0,0\rangle_{D_q}, \tag{18}$$

where $\vec{x} := (x_1, x_2, \ldots, x_s)$ and $\vec{и} := (и_1, и_2, \ldots, и_s)$, additionally note that no $и_i$ in $\vec{и}$ is zero. Now we study the action of $\mathsf{Dec}_{\mathfrak{D}}$ on the above state. To write the output state we need to name the matrix elements of the sampling unitary: $(\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}))_{f\vec{и}} = a_{f\vec{и}}(\mathcal{X})$, the column index consists of a vector of size M with exactly s non-zero entries: $\vec{и} = (0, \ldots, 0, и_1, 0\ldots, 0, и_2, 0, \ldots)$. The decompressed state is

$$|\Upsilon(\vec{x}, \vec{и})\rangle_F := \mathsf{Dec}_{\mathfrak{D}}|Д(\vec{x}, \vec{и})\rangle = \sum_{\phi\in\mathcal{F}} \frac{1}{\sqrt{N^M}} \sum_{f\in\mathcal{F}} \omega_N^{\phi\cdot f}\ a_{f\vec{и}}(\mathcal{X})\ |\phi_0\rangle_{F(0)}\cdots|\phi_{M-1}\rangle_{F(M-1)}, \tag{19}$$

where $\phi \cdot f = \sum_{x\in\mathcal{X}} \phi_x f(x) \mod N$ and by $f(x)$ we denote row number x of the function truth table f.

Using locality of $\mathsf{Samp}_{\mathfrak{D}}$ as defined in Eq. (14), we have that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \{x\} \mid \{x\}) \circ \mathsf{Samp}_{\mathfrak{D}}(x)$ and we can focus our attention on some fixed x: isolate register $F(x)$ with amplitudes depending only on x. Let us compute this state after application of FO, note that FO only subtracts η from $\mathsf{F}(x)$:

$$\begin{aligned}\mathsf{FO}|x,\eta\rangle_{XY}|\Upsilon(\vec{x},\vec{и})\rangle_F = |x,\eta\rangle_{XY} &\sum_{\phi',f'\in\mathcal{F}(\mathcal{X}\setminus\{x\})} \frac{1}{\sqrt{N^{M-1}}}\ \omega_N^{\phi'\cdot f'}\ a_{f'\vec{и}'}(\mathcal{X}\setminus\{x\}\mid\{x\}) \\ &\cdot|\phi_0\rangle_{F(0)}\cdots\left(\sum_{\zeta,z\in[N]} \frac{1}{\sqrt{N}}\ \omega_N^{\zeta\cdot z}\ a_{zи_x}(x)\ |\zeta-\eta\rangle_{F(x)}\right)\cdots|\phi_{M-1}\rangle_{F(M-1)},\end{aligned} \tag{20}$$

where $\vec{\text{й}}' \in \mathcal{Y}^{M-1}$ denotes the vector of и_i without the row with index x. Note that $\text{и}_x = 0$ if x was not in $\vec{x}$ before decompression and $\text{и}_x \neq 0$ otherwise.

The harder part of the proof is showing that the right hand side of Eq. (17) actually equals the left hand side that we just analyzed. Let us inspect $|\text{Д}(\vec{x}, \vec{\text{й}})\rangle$ after application of the compressed oracle

$$\begin{aligned} &\mathsf{CFO}_{\mathfrak{D}} |x, \eta\rangle_{XY} |\text{Д}(\vec{x}, \vec{\text{й}})\rangle_D = |x, \eta\rangle_{XY} \\ &\cdot \left(\sum_{\text{и} \neq 0} \alpha(x, \eta, \tilde{\text{й}}, \text{и}_x) \; |\text{Д}'_{\text{ADD/UPD}}\rangle_D + \alpha(x, \eta, \tilde{\text{й}}, 0) \; |\text{Д}'_{\text{REM/NOT}}\rangle_D \right) \end{aligned} \tag{21}$$

where by $\text{Д}'_{\text{ADD/UPD}}$ we denote the database $\text{Д}(\vec{x}, \vec{\text{й}})$ with entry $(x, \text{и}_x)$ added or updated and by $\text{Д}'_{\text{REM/NOT}}$ we denote the database where $(x, \text{и}_x)$ was removed or nothing happened. The function $\alpha(\cdot)$ denotes the corresponding amplitudes. By $\tilde{\text{й}}$ we denote the original и in entry x in the database.

Before we proceed with decompression of the above state let us calculate the amplitudes α. Again using locality of $\mathsf{Samp}_{\mathfrak{D}}$ we describe the action of the compressed oracle on a single x step by step. Below we denote by Rem removing $\text{и} = 0$ from Д and by Sub subtraction of η from database register Δ^Y. We start with a database containing $(x, \tilde{\text{й}})$, in the case x was not already in Д, then $\tilde{\text{й}} = 0$, otherwise it is the value defined in previous queries. The simplification we make is describing $\mathsf{CFO}_{\mathfrak{D}}$ acting on a single-entry database. We do not lose generality by that as the only thing that changes for q larger than one is maintaining proper sorting and padding, which can be easily done (see Appendix A.2 for details). The calculation of $\mathsf{CFO}_{\mathfrak{D}}$ on a basis state follows:

$$|x, \eta\rangle_{XY} |x, \tilde{\text{й}}_x\rangle_D \overset{\mathsf{Samp}_{\mathfrak{D}}}{\mapsto} |x, \eta\rangle_{XY} \sum_{z \in [N]} a_{z\tilde{\text{Й}}_x}(x) \; |x, z\rangle_D \tag{22}$$

$$\overset{\mathsf{QFT}_N^{D^Y}}{\mapsto} |x, \eta\rangle_{XY} \sum_{z \in [N]} a_{z\tilde{\text{Й}}_x}(x) \sum_{\zeta \in [N]} \frac{1}{\sqrt{N}} \; \omega_N^{\zeta \cdot z} \; |x, \zeta\rangle_D \tag{23}$$

$$\overset{\mathsf{Sub}}{\mapsto} |x, \eta\rangle_{XY} \sum_{z, \zeta \in [N]} a_{z\tilde{\text{Й}}_x}(x) \frac{1}{\sqrt{N}} \; \omega_N^{\zeta \cdot z} \; |x, \zeta - \eta\rangle_D \tag{24}$$

$$\overset{\mathsf{QFT}_N^{\dagger D^Y}}{\mapsto} |x, \eta\rangle_{XY} \sum_{z, \zeta \in [N]} a_{z\tilde{\text{Й}}_x}(x) \frac{1}{\sqrt{N}} \; \omega_N^{\zeta \cdot z} \sum_{z' \in [N]} \frac{1}{\sqrt{N}} \; \bar{\omega}_N^{z' \cdot (\zeta - \eta)} \; |x, z'\rangle_D \tag{25}$$

$$= |x, \eta\rangle_{XY} \sum_{z \in [N]} a_{z\tilde{\text{Й}}_x}(x) \underbrace{\sum_{z', \zeta \in [N]} \frac{1}{N} \; \omega_N^{\zeta \cdot z} \; \bar{\omega}_N^{z' \cdot (\zeta - \eta)}}_{= \bar{\omega}_N^{-z \cdot \eta} \; \delta(z', z)} \; |x, z'\rangle_D \tag{26}$$

$$\overset{\mathsf{Samp}_{\mathfrak{D}}^{\dagger D}(x)}{\mapsto} |x, \eta\rangle_{XY} \sum_{z \in [N]} a_{z\tilde{\text{Й}}_x}(x) \; \omega_N^{z \cdot \eta} \sum_{\text{и} \in [N]} \bar{a}_{z\text{И}}(x) \; |x, \text{и}\rangle_D \tag{27}$$

$$= |x, \eta\rangle_{XY} \sum_{\text{и} \in [N]} \underbrace{\sum_{z \in [N]} a_{z\tilde{\text{Й}}_x}(x) \; \omega_N^{z \cdot \eta} \; \bar{a}_{z\text{И}}(x)}_{:= \alpha(x, \eta, \tilde{\text{Й}}, \text{И})} \; |x, \text{и}\rangle_D \tag{28}$$

$$\overset{\mathsf{Rem}^D}{\mapsto} |x, \eta\rangle_{XY} \left(\sum_{\text{и} \in [N] \setminus \{0\}} \alpha(x, \eta, \tilde{\text{й}}_x, \text{и}) \; |x, \text{и}\rangle_D + \alpha(x, \eta, \tilde{\text{й}}_x, 0) \; |0, 0\rangle_D \right). \tag{29}$$

As we have already mentioned, locality is necessary for us to analyze the action of $\mathsf{CFO}_{\mathfrak{D}}$ on a basis state with a small database. Note however that it is not sufficient; We also have to argue that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \{x\} \mid \{x\})$, that is applied to the database too, commutes with subtraction of η, namely

$$\begin{aligned}\mathsf{Samp}^{\dagger D}_{\mathfrak{D}}(\mathcal{X}\setminus\{x\}\mid\{x\})\circ\mathsf{QFT}^{D^Y_x}_N\,\mathsf{Sub}^{D^Y_x}\,\mathsf{QFT}^{\dagger D^Y_x}_N\circ\mathsf{Samp}^{D}_{\mathfrak{D}}(\mathcal{X}\setminus\{x\}\mid\{x\})\\=\mathsf{QFT}^{D^Y_x}_N\,\mathsf{Sub}^{D^Y_x}\,\mathsf{QFT}^{\dagger D^Y_x}_N.\end{aligned}\tag{30}$$

To prove the above statement we first note that $\mathsf{Samp}^{D}_{\mathfrak{D}}(\mathcal{X}\setminus\{x\}\mid\{x\})$ is controlled on register D^Y_x, but does not act on it. Secondly, in Eq.(26) we see that $\mathsf{QFT}^{D^Y}_N$ Sub $\mathsf{QFT}^{\dagger D^Y}_N$ multiplies the state by phase factor $\omega_N^{z\cdot\eta}$ and also does not modify register D^Y_x, but is controlled on it though. Hence Eq.(30) holds. In the above equations we have defined α as

$$\alpha(x,\eta,\tilde{\text{й}}_x,\text{и}):=\sum_{z\in[N]}a_{z\tilde{\text{й}}_x}(x)\,\bar{a}_{z\text{и}}(x)\,\omega_N^{z\cdot\eta}.\tag{31}$$

After decompressing the state from Eq.(21), the resulting database state will be $\sum_{\text{и}\neq0}\alpha(x,\eta,\tilde{\text{й}}_x,\text{и}_x)\ |\Upsilon(\text{Д}'_{\text{ADD/UPD}})\rangle+\alpha(x,\eta,\tilde{\text{й}}'_x,0)\ |\Upsilon(\text{Д}'_{\text{REM/NOT}})\rangle_D$, where we overload notation of $|\Upsilon(\vec{x},\vec{\text{и}})\rangle$ to denote that $(\vec{x},\vec{\text{и}})$ consists of values in the respective databases. We can write down this state in more detail using Eq.(20):

$$\begin{aligned}&\mathsf{Dec}_{\mathfrak{D}}\circ\mathsf{CFO}_{\mathfrak{D}}|x,\eta\rangle_{XY}|\text{Д}(\vec{x},\vec{\text{и}})\rangle_D=\sum_{\phi',f'\in\mathcal{F}(\mathcal{X}\setminus\{x\})}\frac{1}{\sqrt{N^{M-1}}}\,\omega_N^{\phi'\cdot f'}\,a_{f'\vec{\text{й}}'}(\mathcal{X}\setminus\{x\}\mid\{x\})\,|\phi_0\rangle_{F(0)}\cdots\\&\cdot\underbrace{\left(\sum_{\text{и}_x\neq0}\alpha(x,\eta,\tilde{\text{й}},\text{и}_x)\sum_{\zeta,z\in[N]}\frac{1}{\sqrt{N}}\omega_N^{\zeta\cdot z}a_{z\text{и}_x}(x)|\zeta\rangle_{F(x)}+\alpha(x,\eta,\tilde{\text{й}},0)\sum_{\zeta,z\in[N]}\frac{1}{\sqrt{N}}\omega_N^{\zeta\cdot z}a_{z0}(x)|\zeta\rangle_{F(x)}\right)}_{=\sum_{\zeta,z\in[N]}\frac{1}{\sqrt{N}}\,\omega_N^{\zeta\cdot z}\sum_{\text{И}_x\in[N]}\alpha(x,\eta,\tilde{\text{Й}},\text{И}_x)\,a_{z\text{И}_x}(x)\,|\zeta\rangle_{F(x)}}\\&\cdots|\phi_{M-1}\rangle_{F(M-1)}=\mathsf{FO}\,|x,\eta\rangle_{XY}|\Upsilon(\vec{x},\vec{\text{и}})\rangle=\mathsf{FO}\circ\mathsf{Dec}_{\mathfrak{D}}\,|x,\eta\rangle_{XY}|\text{Д}(\vec{x},\vec{\text{и}})\rangle_D.\end{aligned}\tag{32}$$

The second to last equality comes from the fact that $\mathsf{Samp}_{\mathfrak{D}}$ is a unitary and $\sum_{j\in[N]}a_{ij}\bar{a}_{kj}=\delta_{ik}$ and therefore we have

$$\sum_{\text{и}\in[N]}\alpha(x,\eta,\tilde{\text{й}},\text{и})\,a_{z\text{и}_x}(x)=\sum_{z'\in[N]}\underbrace{\sum_{\text{и}\in[N]}\bar{a}_{z'\text{и}}(x)\,a_{z\text{и}}(x)}_{=\delta_{z',z}}a_{z'\tilde{\text{й}}}(x)\,\omega_N^{z'\cdot\eta}=a_{z\tilde{\text{й}}}(x)\,\omega_N^{z\cdot\eta}.\tag{33}$$

Together with changing the variable $\zeta\mapsto\zeta-\eta$, we have derived the claimed identity. □

3.2.1 Conditionally Uniform Distributions

Functions from conditionally uniform distributions are such that conditioned on any series of input-output pairs the probability of any output is either zero or $\frac{1}{L}$, where L is some integer. An example of such function is a random permutation: Conditioned on some queries the output of a new query must be a value different from previous outputs. Let us state the formal definition.

Definition 6 (Conditionally uniform distributions). *Let $\mathfrak{D}$ be a distribution over $\mathcal{F}:=\{f:\mathcal{X}\to\mathcal{Y}\}$. We call $\mathfrak{D}$ conditionally uniform if for all $q\in\mathbb{N}$, for any set of input-output pairs $\{(x_i,y_i)\}_{i\in[q]}$, for all (x,y) such that $\forall i\in[q]:\ x\neq x_i$, and for all $\mathcal{S}\subsetneq\mathcal{Y}$, possibly depending on $\{(x_i,y_i)\}_{i\in[q]}$ and x, the following holds*

$$\mathop{\mathbb{P}}_{f\leftarrow\mathfrak{D}}[f(x)=y\mid\forall i\in[q]:f(x_i)=y_i]=\begin{cases}0&\text{if }y\in\mathcal{S}\\\frac{1}{|\mathcal{Y}\setminus\mathcal{S}|}&\text{if }y\notin\mathcal{S}\end{cases}.\tag{34}$$

To accommodate for conditionally uniform distributions in the purified quantum oracle approach we need a quantum operation that prepares an equal superposition over a given set $\mathcal{Y} \setminus \mathcal{S}$. Say that the codomain of f is $\mathcal{Y}$ (of size $N = |\mathcal{Y}|$) and the set with probability zero is $\mathcal{S}$. We propose that this is done first by applying the Quantum Fourier Transform over the set $[L]$, where $L = |\mathcal{Y} \setminus \mathcal{S}|$, to the state $|0\rangle$. After that we spread the values in $[L]$ in a way that values from $\mathcal{S}$ no longer appear in the set. The last operation may be realized by repeating the following unitary

$$\mathsf{V}^{AB}|a\rangle_A|b\rangle_B = \begin{cases} |a\rangle_A|b\rangle_B & \text{if } a > b \\ |a\rangle_A|b+1 \mod N\rangle_B & \text{if } a \leq b \end{cases}, \tag{35}$$

where by V^{AB} we denote applying V to registers A, B. Details about the efficient implementation of V are given in Appendix A.3. The definitions of the sampling operation for a conditionally uniform distribution is given by: $\forall \mathcal{S} \subseteq \mathcal{X}$

$$\mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}(x) := \prod_{i=1}^{|\mathcal{S}|} \mathsf{V}^{S_i D^Y(x)} \mathsf{QFT}_L^{D^Y(x)\dagger}, \tag{36}$$

where S_i are the quantum registers holding elements of $\mathcal{S}$. Note that we apply $\mathsf{QFT}_L^\dagger$ to a register of dimension N, we overload the notation but mean $\mathsf{QFT}_L^\dagger \bigoplus \mathbb{1}_{N-L}$. The sampling procedure from Eq. (36) is used to implement $\mathsf{CStO}_{\mathcal{Y}\setminus\mathcal{S}}$.

An example of a conditionally uniform distribution of this form is the uniform distribution over the set of permutations, denoted by $\mathfrak{P}$. The sampling function is defined using Eq. (36) with $\mathcal{S}$ defined as previous outputs, held in registers D_i^Y. The definitions reads as follows:

$$\mathsf{Samp}_{\mathfrak{P}}(\mathcal{S}) := \prod_{i=1}^{|\mathcal{S}|} \mathsf{Samp}_{\mathcal{Y}\setminus D^Y(\{x_1,\cdots,x_i\})} = \prod_{i=1}^{|\mathcal{S}|} \left(\prod_{j=1}^{i} \mathsf{V}^{D_j^Y D_i^Y} \right) \mathsf{QFT}_{N-(i-1)}^{D_i^Y\dagger}, \tag{37}$$

where $D^Y(\{x_1, \cdots, x_i\}$ denotes the set of outputs y in entries of D corresponding to listed x. We denote the compressed oracle returning outputs of random permutations by $\mathsf{CPerO}_{\mathcal{X}}$.

For the inverse permutation the oracle should return the x such that the output is y. For the full oracle we can implement this by rewriting the function table to contain the x values explicitly and then treating the F^X register as holding output values and F^Y register as the input register. In the case of compressed oracles we do a similar thing, applying sampling operations $\mathsf{Samp}_{\mathfrak{P}}$ with roles of registers D^X and D^Y swapped.

4 One-way to Hiding Lemma for Compressed Oracles

The fundamental game-playing lemma, Lemma 1, is a very powerful tool in proofs that include a random oracle. By designing the games for a proof we can reprogram the random oracle in a useful way. The fundamental lemma gives us a simple way of calculating how much the reprogramming costs in terms of the adversary's advantage—the difference between probabilities of A outputting 1 when interacting with one game or the other. The lemma that provides a counterpart to Lemma 1 valid for quantum accessible oracles is the *One-Way to Hiding* (O2H) Lemma first introduced by Unruh in [Unr14].

In the original statement of the O2H lemma, the main idea is that there is a marked subset of inputs to the random oracle H and an adversary tries to distinguish the situation in which she interacts with the normal oracle from an interaction with an oracle G that differs only on this set. The lemma states a bound for the distinguishing advantage which depends on the probability of an external algorithm measuring the input register of the adversary and seeing an element of the marked set. This probability is usually small, for random marked sets.

Recently this technique was generalized by Ambainis, Hamburg, and Unruh in [AHU18]. The main technical idea introduced by the generalized O2H lemma is to exchange the oracle G with a so-called *punctured oracle* that measures the input of the adversary after every query. The bound on the adversary's advantage is given by the probability of this measurement succeeding. This technique forms the link with the classical identical-until-bad games: we perform a binary measurement on the "bad" event and bound the advantage by the probability of succeeding.

In this work we present a generalization of this lemma that involves the use of compressed oracles. Our idea is to measure the database of the compressed oracle, which makes the lemma more versatile and easier to use for more general quantum oracles.

Below we state our generalized O2H lemmas. Most proofs of [AHU18] apply almost word by word so we just describe the differences and refer the reader to the original work.

The key notion we use is a relation on the database of the compressed oracle.

Definition 7 (Relation R on D). *Let D be a database of size q of pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$. We call a subset $R \subseteq \bigcup_{s\in[q+1]} (\mathcal{X} \times \mathcal{Y})^s$ a* relation R on D.

An example of such a relation is a collision, namely $R_{\text{coll}} := \{((x_1, y_1), \cdots, (x_t, y_t)) \in \bigcup_{s\in[q+1]} (\mathcal{X} \times \mathcal{Y})^s : \exists_{i,j}\ i \neq j, x_i \neq x_j, y_i = y_j\}$. Note however, that it is only reasonable to check if the non-padding entries are in R, omitting the $(0, 0)$ pairs at the end of D. We also write $\mathcal{S}$ instead of R (for some $\mathcal{S} \subseteq \mathcal{Y}$), then the relation is defined as entries of D that have $y_i \in \mathcal{S}$.

If D is held in a quantum register, R has a corresponding projective measurement J_R such that $\|\mathsf{J}_R|(x_1, y_1), \cdots, (x_q, y_q)\rangle_D\| = 1$ if and only if for some s $((x_1, y_1), \cdots, (x_s, y_s)) \in R$ holds and (x_i, y_i) are padding entries for $i > s$. We also state an explicit algorithm to implement the measurement of a relation R, given that the relation R is efficiently computable. Alg. 3 defines the measurement procedure of measuring R defined in the standard basis of the database (D).

Algorithm 3: Measurement of a relation R

```
   Input  : Unprepared database $|Д\rangle_D$
   Output: Outcome $p$ and post-measurement state $|Д'\rangle_D$
1  Count in register $S$ the number of non-padding ($и \neq 0$) entries $s$
2  Apply $\mathsf{Samp}^D_{\mathfrak{D}}(\mathcal{S})$                 // Prepare the database: $Д \mapsto D$
3  Apply $\mathsf{U}_R$ that saves a bit $j := \|\mathsf{J}_R|D\rangle_D\|$ in register $J$     // $j = 1$ iff $D$ in $R$
4  Apply $\mathsf{Samp}^{D\dagger}_{\mathfrak{D}}(\mathcal{S})$                // Unprepare the database $D \mapsto Д$
5  Uncompute register $S$, measure register $J$, output the outcome $j$
```

While not directly relevant to our applications, we keep the generality of [AHU18] by introducing the notion of *query depth* as the number of sets of parallel queries an algorithm makes. We usually assume quantum algorithms make q quantum queries in total and d (as in "query depth") sequentially, but those queries in sequence may involve a number of parallel queries. A parallel query of width p to an oracle H involves p applications of H to p query registers. Note that if H is considered to be a compressed oracle all parallel queries are held in a single database register.

First we define a compressed oracle H punctured on relation R, denoted by $\mathsf{H} \setminus R$.

Definition 8 (Punctured compressed oracle $\mathsf{H} \setminus R$). *Let* H *be a compressed oracle and R a relation on its database. The punctured compressed oracle* $\mathsf{H} \setminus R$ *is equal to* H*, except that R is measured after every query as described in Alg. 3. By* Find *we denote the event that R outputs* 1 *at least once.*

Using this definition we can prove a theorem similar to Theorem 1 of [AHU18]:

Theorem 9 (Compressed oracle O2H)**.** *Let R be a relation on the database of a quantum compressed oracle H. Let z be a random string. R, H, and z may have arbitrary joint distribution. Let A be an oracle algorithm of query depth d. Let $\mathsf{H} \setminus R$ be a punctured oracle defined in Def. 8, then*

$$\left|\mathbb{P}[b=1 : b \leftarrow \mathsf{A}^{\mathsf{H}}(z)] - \mathbb{P}[b=1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R}(z)\right| \leq 2\sqrt{(d+1)\mathbb{P}[\mathsf{Find} : \mathsf{A}^{\mathsf{H}\setminus R}(z)]}, \textit{and} \tag{38}$$

$$\left|\sqrt{\mathbb{P}[b=1 : b \leftarrow \mathsf{A}^{\mathsf{H}}(z)]} - \sqrt{\mathbb{P}[b=1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R}(z)}\right| \leq 2\sqrt{(d+1)\mathbb{P}[\mathsf{Find} : \mathsf{A}^{\mathsf{H}\setminus R}(z)]}. \tag{39}$$

Proof. The proof works almost the same as the proof of Theorem 1 of [AHU18]. Let us state the analog of Lemma 5 from [AHU18].

For the following lemma let us first define two algorithms. Let $\mathsf{A}^{\mathsf{H}}(z)$ be a unitary quantum algorithm with oracle access to H with query depth d. Let Q denote the quantum register of A and D the database of the compressed oracle H. We also need a "query log" register L consisting of d qubits.

Let $\mathsf{B}^{\mathsf{H},R}(z)$ be a unitary quantum algorithm acting on registers Q and L and having oracle access to H. First we define the following unitary

$$\mathsf{U}_R|D\rangle_D|l_1, l_2, \dots, l_d\rangle_L := \begin{cases} |D\rangle_D|l_1, l_2, \dots, l_d\rangle_L & \text{if } R(|D\rangle_D) = 0 \\ |D\rangle_D|l_1, \dots, l_i \oplus 1, \dots, l_d\rangle_L & \text{if } R(|D\rangle_D) = 1 \end{cases}, \tag{40}$$

where $R(|D\rangle_D)$ denotes the outcome of the projective binary measurement on D. The unitary exists for all relations. One can just coherently compute $R(D)$ into an auxiliary register, apply CNOT from that register to L_i and then uncompute $R(D)$. If the relation is efficiently computable, then so is the unitary. We define $\mathsf{B}^{\mathsf{H},R}(z)$ as:

- Initialize the register L with $|0^d\rangle$.
- Perform all operations that $\mathsf{A}^{\mathsf{H}}(z)$ does.
- For all i, after the i-th query of A apply the unitary U_R to registers D, L.

Let $|\Psi_{\mathsf{A}}\rangle$ denote the final state of $\mathsf{A}^{\mathsf{H}}(z)$, and $|\Psi_{\mathsf{B}}\rangle$ the final state of $\mathsf{B}^{\mathsf{H},R}(z)$. Let $\tilde{P}_{\text{find}}$ be the probability that a measurement of L in the computational basis in the state $|\Psi_{\mathsf{B}}\rangle$ returns $\neq 0^d$, i.e. $\tilde{P}_{\text{find}} := \left\|\mathbb{1}^{Q,D} \otimes (\mathbb{1}^L - |0^d\rangle_L\langle 0^d|)|\Psi_{\mathsf{B}}\rangle\right\|^2$.

Lemma 10 (Compressed oracle O2H for pure states)**.** *Fix a joint distribution for H, R, z. Consider the definitions of algorithms A and B and their quantum states, then*

$$\left\||\Psi_{\mathsf{A}}\rangle \otimes |0^d\rangle_L - |\Psi_{\mathsf{B}}\rangle\right\|^2 \leq (d+1)\tilde{P}_{\text{find}}. \tag{41}$$

Proof. This lemma can be proved in the same way as Lemma 5 of [AHU18]. Here we omit some details and highlight the most important observation of the proof.

First define $\mathsf{B}_{\text{count}}$ that works in the same way as B but instead of storing L, the log of queries with D in relation, it keeps *count*—in register C—of how many times a query resulted in $R(|D\rangle_D) = 1$. The state that results from running $\mathsf{B}_{\text{count}}$ is $|\Psi_{\mathsf{B}_{\text{count}}}\rangle = \sum_{i=0}^{d}|\Psi^i_{\mathsf{B}_{\text{count}}})|i\rangle_C$ and similarly $|\Psi_{\mathsf{B}}\rangle = \sum_{l\in\{0,1\}^d}|\Psi^l_{\mathsf{B}})|l\rangle_L$, where $|\Psi)$ denotes a not normalized state. We can observe that $|\Psi_{\mathsf{A}}\rangle = \sum_{i=0}^{d}|\Psi^i_{\mathsf{B}_{\text{count}}})$. As $\tilde{P}_{\text{find}}$ is the probability of measuring at least one bit in the register L of B, or counting at least one fulfilling of R in C, we have that $|\Psi^{0^d}_{\mathsf{B}}) = |\Psi^0_{\mathsf{B}_{\text{count}}})$. From the definition we also have $\tilde{P}_{\text{find}} = 1 - \left\||\Psi^0_{\mathsf{B}_{\text{count}}})\right\|^2$. Using the above identities we can calculate the bound

$$\begin{aligned}\left\| |\Psi_{\mathsf{B}}\rangle - |\Psi_{\mathsf{A}}\rangle \otimes |0^d\rangle_L \right\|^2 &= \left\| \sum_{i=1}^{d} |\Psi^i_{\mathsf{B}_{\text{count}}}\rangle \right\|^2 + \tilde{P}_{\text{find}} \overset{\triangle}{\leq} \left(\sum_{i=1}^{d} \left\| |\Psi^i_{\mathsf{B}_{\text{count}}}\rangle \right\| \right)^2 + \tilde{P}_{\text{find}} \\ &\overset{\text{J-I}}{\leq} d \underbrace{\sum_{i=1}^{d} \left\| |\Psi^i_{\mathsf{B}_{\text{count}}}\rangle \right\|^2}_{=\tilde{P}_{\text{find}}} + \tilde{P}_{\text{find}} = (d+1)\tilde{P}_{\text{find}}, \end{aligned} \tag{42}$$

where $\triangle$ denotes the triangle inequality and J-I denotes the Jensen's inequality. Now it is apparent that introducing $\mathsf{B}_{\text{count}}$ gave us a more coarse-grained look at the initial algorithm B, resulting in a tighter bound. □

The rest of the proof of the theorem follows the same reasoning as the proof of Lemma 6 in [AHU18] with the modifications shown in the above lemma. Using bounds on fidelity (Lemma 3 and Lemma 4 of [AHU18]) and monotonicity and joint concavity of fidelity (from Thm. 9.6 and Eq. 9.95 of [NC11]) one can generalize the results to the case of arbitrary mixed states. □

We continue by deriving an explicit formula for $\mathbb{P}[\text{Find}]$. Let A be a quantum algorithm with oracle access to H, making at most q quantum queries with depth d. Let R be a relation on the database of H and z an input to A. R and z can have any joint distribution. J_R is the projector from the measurement of R on D, $\mathsf{U}_i^{\mathsf{H}}$ is the i-th unitary performed by $\mathsf{A}^{\mathsf{H}\setminus R}$ together with a query to H, and $|\Psi_0\rangle$ is the initial state of A. Then we have the formula

$$\mathbb{P}[\text{Find} : \mathsf{A}^{\mathsf{H}\setminus R}(z)] = 1 - \left\| \prod_{i=1}^{d} (\mathbb{1} - \mathsf{J}_R)\mathsf{U}_j^{\mathsf{H}} |\Psi_0\rangle \right\|^2 . \tag{43}$$

Let us now discuss the notion of "identical until bad" games in the case of compressed oracles. For random oracles, the notion was introduced in [AHU18]. The definition is rather straightforward as H and G are considered identical until bad if they had the same outputs except for some marked set. When using compressed oracles, the outputs of H and G are quantum lazy-sampled, making the definition of what it means for two oracles to be identical until bad require more care. Here we state a definition that captures useful notions of identical-until-bad punctured oracles.

Definition 11 (Almost identical oracles). *Let* H *and* G *be compressed oracles and* R_i, $i = 1, 2$ *relations on their databases. We call the oracles* $\mathsf{H} \setminus R_1$ *and* $\mathsf{G} \setminus R_2$ almost identical *if they are equal conditioned on the event* $\neg$Find, *i.e. for any event* E, *any strings* y, z, *and any quantum algorithm* A

$$\mathbb{P}[E : y \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \neg\text{Find}] = \mathbb{P}[E : y \leftarrow A^{\mathsf{G}\setminus R_2}(z) \mid \neg\text{Find}]. \tag{44}$$

Note that unpunctured compressed oracles are a special case of punctured ones (for $R = \emptyset$), so the above definition can be applied to a pair of oracles where one is punctured and one is not. We can prove the following bound on the adversary's advantage in distinguishing almost identical punctured oracles.

Lemma 12 (Distinguishing almost identical punctured oracles). *If* $\mathsf{H} \setminus R_1$ *and* $\mathsf{G} \setminus R_2$ *are almost identical according to Def.11 then*

$$\left| \mathbb{P}[y \leftarrow A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[y \leftarrow A^{\mathsf{G}\setminus R_2}(z)] \right| \leq 2\mathbb{P}[\text{Find} : A^{\mathsf{H}\setminus R_1}(z)] + 2\mathbb{P}[\text{Find} : A^{\mathsf{G}\setminus R_2}(z)]. \tag{45}$$

Proof. We bound

$$
\begin{aligned}
&\Big|\mathbb{P}[y \leftarrow A^{\mathsf{H}\backslash R_1}(z)] - \mathbb{P}[y \leftarrow A^{\mathsf{G}\backslash R_2}(z)]\Big| \\
&\overset{\text{Def. 11}}{=} \Big|\mathbb{P}[y \leftarrow A^{\mathsf{H}\backslash R_1}(z) \mid \neg\text{Find}]\Big(\mathbb{P}[\neg\text{Find} : A^{\mathsf{H}\backslash R_1}(z)] - \mathbb{P}[\neg\text{Find} : A^{\mathsf{G}\backslash R_2}(z)]\Big) \\
&+ \mathbb{P}[y \leftarrow A^{\mathsf{H}\backslash R_1}(z) \mid \text{Find}]\mathbb{P}[\text{Find} : A^{\mathsf{H}\backslash R_1}(z)] \\
&- \mathbb{P}[y \leftarrow A^{\mathsf{G}\backslash R_2}(z) \mid \text{Find}]\mathbb{P}[\text{Find} : A^{\mathsf{G}\backslash R_2}(z)]\Big| \qquad (46) \\
&\overset{\triangle}{\leq} \left|\underbrace{\mathbb{P}[y \leftarrow A^{\mathsf{H}\backslash R_1}(z) \mid \neg\text{Find}]}_{\leq 1}\underbrace{\Big(\mathbb{P}[\neg\text{Find} : A^{\mathsf{H}\backslash R_1}(z)] - \mathbb{P}[\neg\text{Find} : A^{\mathsf{G}\backslash R_2}(z)]\Big)}_{=\mathbb{P}[\text{Find}:A^{\mathsf{G}\backslash R_2}(z)]-\mathbb{P}[\text{Find}:A^{\mathsf{H}\backslash R_1}(z)]}\right| \\
&+ \left|\underbrace{\mathbb{P}[y \leftarrow A^{\mathsf{H}\backslash R_1}(z) \mid \text{Find}]}_{\leq 1}\mathbb{P}[\text{Find} : A^{\mathsf{H}\backslash R_1}(z)]\right| \\
&+ \left|\underbrace{\mathbb{P}[y \leftarrow A^{\mathsf{G}\backslash R_2}(z) \mid \text{Find}]}_{\leq 1}\mathbb{P}[\text{Find} : A^{\mathsf{G}\backslash R_2}(z)]\right| \qquad (47) \\
&\overset{\triangle}{\leq} 2\mathbb{P}[\text{Find} : A^{\mathsf{H}\backslash R_1}(z)] + 2\mathbb{P}[\text{Find} : A^{\mathsf{G}\backslash R_2}(z)], \qquad (48)
\end{aligned}
$$

where by $\triangle$ we denote the triangle inequality. □

Note that for $R_2 = \emptyset$, the above lemma is essentially a special case of the well known Gentle-Measurement Lemma of [Win99]. Some additional details on the compressed oracle O2H lemma can be found in Appendix B.

In the following section we are going to use punctured compressed oracles to prove indifferentiability. There is one more aspect of these oracles that we need to address: A punctured oracle acts on a database that has been modified by the measurement of the relation. To not induce more errors when querying such oracle we need to apply the $\mathsf{Samp}_{\mathfrak{D}}$ procedure with respect to the database not being in relation.

For completeness we provide a definition of the general compressed standard oracle punctured on R. We highlight the "lazy" nature of this formulation but also note that the definition here is equivalent to the one provided in Alg. 1. The earlier definition is more suitable for the correctness proof and the following one might give a bit more insight and explain better why we call the approach the quantum lazy-sampling technique. We define a compressed oracle for any distribution $\mathfrak{D}$ that has a Samp procedure fulfilling the requirement of locality and preparing the initial state, Eq. (14) and Eq. (10) respectively. Note that we also make explicit the fact that the distribution of a new sample might crucially depend on the set of previous queries $\mathcal{P}$. Especially, we would like to address distributions with weights of outputs depending on the previous queries. At this point it is also important to note that $\mathsf{Samp}_{\mathfrak{D}}(x \mid \mathcal{S})$ is defined to sample x conditioned on $\mathcal{S}$ that is in the prepared basis. By the locality requirement from Eq. (14) we can always prepare or unprepare the database, by proceeding sample-by-sample.

The initial steps of Alg. 4 (lines 1 to 5) work under the condition of $\neg$Find. If the event Find occured, we need to modify the distribution to $\mathfrak{D}(R)$, with Samp conditioned on the structure of the database D corresponding to previous measurement outcomes. This information is classical though, hence it introduces no additional problems with quantum controls.

In Alg. 4 we write $D\backslash\{x\}$ to denote the content of database D without the entry that includes x and D^X (the x values in D). By $\mathsf{CStO}_{\mathfrak{D}(\mathcal{P})}$ we denote $\mathsf{QFT}_N^{YD(x)\dagger} \circ \mathsf{CFO}_{\mathfrak{D}(\mathcal{P})} \circ \mathsf{QFT}_N^{YD(x)}$ acting on registers $XYD(x)D_{s+1}S$—adversary registers XY, content of D corresponding to entry with x, register holding the first padding entry D_{s+1}, and S holding the size of D. The oracle uses

Algorithm 4: General $\mathsf{CStO}_{\mathfrak{D}} \setminus R$

```
Input  : Standard database and adversary query: $|x, y\rangle_{XY}|D\rangle_D$
Output: $|x, y + f(x)\rangle_{XY}|D'\rangle_D$
1 Unprepare $D$ by applying $\mathsf{Samp}^{\dagger XYD}_{\mathfrak{D}\setminus R}(D^X)$    // $D \to Д$
2 Count in register $S$ the number of non-padding ($и \neq 0$) entries $s$
3 if $x \notin Д^X$ then                                                  // locate $x$
4   Copy $(0, 0)$ to $Д$ in the location $x$ should be in and add $1$ to $S$
5 Prepare $D$ by applying $\mathsf{Samp}^{XYD}_{\mathfrak{D}\setminus R}(D^X)$    // If $x \in Д^X$ nothing changed, else new
   entry $(0, 0)$
6 Save the set of previous queries $\mathcal{P}$ in a fresh register $P$, controlled on $D \setminus \{x\}$
7 Apply $\mathsf{CStO}_{\mathfrak{D}(\mathcal{P})} \setminus R$ to registers $XYD(x)D_{s+1}S$ controlled on registers $PD \setminus \{x\}$
   // Lazy-sampling $f(x) \leftarrow \mathfrak{D}(\mathcal{P})$
8 Uncompute $s$ from register $S$ and $\mathcal{P}$ from $P$      // With the necessary basis switch
9 Output $|x, y + f(x)\rangle_{XY}|D'\rangle_D$                       // $D$ is the modified database
```

the sampling function $\mathsf{Samp}_{\mathfrak{D}(\mathcal{R})}$ controlled on register P. Acting with this unitary on registers $D_{s+1}S$ gives the oracle space to update the padding. By $\mathfrak{D} \setminus R$ we denote a distribution on outputs that are never in relation R. Oracle ($\mathsf{CStO}_{\mathfrak{D}(\mathcal{P})} \setminus R$) performs a measurement of R after the last step of Alg. 1 and a basis transform of register D. We would like to highlight that except for the initial preparation (from line 1 to 5) and the final clearing of the auxiliary registers (in line 8) the compressed oracle updates only one entry of the database. Moreover Alg. 4 clearly shows that any dependence on the previous queries can be added to the new queries—a fact that will be crucial in the proof of quantum indifferentiability.

5 Quantum Security of the Sponge Construction

We use our methods to show a detailed proof of quantum indifferentiability of the sponge construction used with a random transformation as the internal function. In Thm. 16 we state indifferentiability of Sponges with random permutations, in Appendix C we show the proof of this theorem.

At the end of this section we prove that quantum indifferentiability implies collapsingness. As a corollary we show that the sponge construction with random permutations is collapsing.

5.1 Sponge Construction

The sponge construction is used to design variable-input-length and variable-output-length functions. It works by applying the *internal function* φ multiple times on the *state* of the function. In Algorithm 5 we present the definition of the sponge construction, which we denote with Sponge [Ber+07]. The state of Sponge consists of two parts: one in set $\mathcal{A}$, called the *rate*, and the other in set $\mathcal{C}$, called the *capacity*. In the following we denote the part of the entire state $s \in \mathcal{A} \times \mathcal{C}$ that is $\mathcal{A}$ by $\bar{s}$ and call it the *outer* part and the part in $\mathcal{C}$ by $\hat{s}$, we will refer to it as the *inner* part of a state. Naturally the internal function is a map $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$. To denote the internal function with output limited to the part in $\mathcal{A}$ and $\mathcal{C}$ we use the same notation as for states, $\bar{\varphi}$ and $\hat{\varphi}$ respectively. Note that we use a general formulation of the construction, using any finite sets as rate and capacity. All our results also work for Sponge defined with bit-strings and addition modulo 2, as specified in [NIS14]. By pad we denote a padding function: an efficiently computable bijection mapping an arbitrary message set to strings p of elements of $\mathcal{A}$. By $|p|$ we denote the number of characters in $\mathcal{A}$ in p. A constructed function behaves as

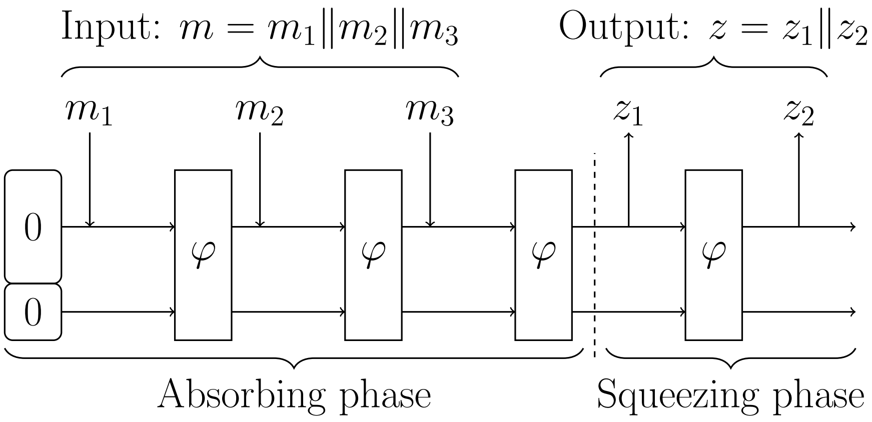

Figure 2: A schematic representation of the sponge construction: $\textsc{Sponge}_\varphi(m_1\|m_2\|m_3) = z_1\|z_2$.

follows $\textsc{Sponge}_\varphi : \mathcal{A}^* \times \mathbb{N} \to \mathcal{A}^*$, where $\mathcal{A}^* := \bigcup_{n=0}^{\infty} \mathcal{A}^n$. In Fig. 2 we present a scheme of the sponge construction evaluated on input m.

For a set $\mathcal{S} \subseteq \mathcal{A} \times \mathcal{C}$, by $\overline{\mathcal{S}}$ we denote the outer part of the set: a set of outer parts of elements of $\mathcal{S}$. Similarly by $\widehat{\mathcal{S}}$ we denote the inner part of the set. We use similar notation for quantum registers holding quantum state in $\mathcal{H}_{\mathcal{A}\times\mathcal{C}}$: $\overline{Y}$ is the part of the register holding elements of $\mathcal{A}$ and $\widehat{Y}$ holds the inner parts.

Algorithm 5: $\textsc{Sponge}_\varphi[\textsc{pad}, \mathcal{A}, \mathcal{C}]$

```
Input  : $m \in \mathcal{A}^*$, $\ell \geq 0$.
Output: $z \in \mathcal{A}^\ell$
1  $p := \textsc{pad}(m)$
2  $s := (0,0) \in \mathcal{A} \times \mathcal{C}$.
3  for $i = 1$ to $|p|$ do                          // Absorbing phase
4      $s := (\bar{s} + p_i, \hat{s})$
5      $s := \varphi(s)$
6  $z := \bar{s}$                                    // Squeezing phase
7  while $|z| < \ell$ do
8      $s := \varphi(s)$
9      $z := z\|\bar{s}$
10 Output $z$
```

An important feature of the sponge construction that was introduced in [Ber+07] is the fact that interaction with it can be represented on a graph $G = (\mathcal{V}, \mathcal{E})$. The set of vertices $\mathcal{V}$ corresponds to all possible states of the sponge, namely $\mathcal{V} := \mathcal{A} \times \mathcal{C}$. The outer part is controlled by the user, meaning that she can output that part and modify to any value in a future evaluation by querying an appropriate message. For that reason we group the nodes with the same inner-part value into *supernodes*, so that we have $|\mathcal{C}|$ supernodes and every of those consists of $|\mathcal{A}|$ nodes. A directed edge $(s,t) \in \mathcal{E}$ from a node s to a node t exists if $\varphi(s) = t$. From every node there is exactly one edge, if φ is a permutation, then there is also exactly one edge arriving at every node. Note that query algorithms add edges to $\mathcal{E}$ query by query. Then graph G reflects the current knowledge of this algorithm about φ.

In the sponge graph G a *sponge path* is a path between supernodes that starts at 0-supernode—called the *root*. A sponge path can represented by a string consisting of some number of characters from $\mathcal{A}$: following the rules of evaluating Sponge we feed those characters to the construction as inputs, every next character shifts us in a single supernode, evaluation of φ can create an edge between any two nodes (also with different inner parts, so in different supernodes). If the string representing a sponge path is a padding of some message m, a path corresponds to an input to Sponge. In the following proofs we are going to construct the input to Sponge leading to a given node s, with a given sponge graph G. Our definition works under the assumption that there is a series of edges $((v_i, w_i))_{i\in[\ell]}$ of G (so a "regular" path) that leads to s, meaning $w_\ell = s$. We define the sponge path construction operation as follows

$$\mathsf{SpPath}(s, G) := \bar{v}_1 \| (\bar{v}_2 - \bar{w}_1) \| \cdots \| (\bar{v}_\ell - \bar{w}_{\ell-1}) \| 0. \tag{49}$$

Output of the above function is the input to the construction $\text{Sponge}_\varphi(., \ell = 1)$ that yields the output $\bar{s}$.

A supernode is called *rooted* if there is a path leading to it that starts in the root (the 0-supernode). The set $\mathcal{R}$ is the set of all rooted supernodes in G. By $\mathcal{U}$ we denote the set of supernodes with a node with an outgoing edge.

In the case of an adversary querying a random function φ we are going to treat the graph as being created one edge per query. Graph G then symbolizes the current state of knowledge of the adversary of the internal function. Note that this dynamical graph can be created efficiently by focusing solely on nodes that appear in the queried edges.

A sponge graph is called *saturated* if $\mathcal{R} \cup \mathcal{U} = \mathcal{C}$. It means that for every inner state in $\mathcal{C}$ there is an edge in G that leads to it from 0 (the root) or leads from it to another node. Saturation will be important in the proof of indifferentiability as the simulator wants to pick outputs of φ without colliding inner parts (so not in $\mathcal{R}$) and making the path leading from 0 to the output longer by just one edge (so not in $\mathcal{U}$).

The simulators defined in the proofs in this section are implicitly stateful. They maintain a classical or quantum state containing a database of the adversary's queries and the simulator's outputs. Basing on that database the simulator can always construct a sponge graph containing all the current knowledge of φ.

For the proof of indifferentiability we also need an upper bound on the probability of finding a collision in the inner part of outputs of a uniformly random function $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$. Note that by such collision we also consider inputs that map to $0 \in \mathcal{C}$. We define the bound as a function of the number of queries q to φ:

$$f_{\text{coll}}(q) := \frac{q(q+1)}{2\,|\mathcal{C}|}, \tag{50}$$

the bound can be derived by bounding the probability of finding a collision and bounds on the natural logarithm: $\ln(1 - x) \le -x$ and $\ln(1 - \frac{x}{2}) \ge -\frac{x}{2}$ for $0 \le x < 1$.

As the sponge construction is used to design variable-input and variable-output functions we define the random oracle $\mathsf{H} : \mathcal{A}^* \times \mathbb{N} \to \mathcal{A}^*$ accordingly:

$$\mathsf{H}(x, \ell) := \begin{cases} \lfloor y' \rfloor_\ell & \text{if } (x, \ell' \ge \ell) \in D \\ y' \| \left(y \xleftarrow{\$} \mathcal{A}^{\ell-\ell'} \right) & \text{if } (x, \ell' < \ell) \in D \\ y \xleftarrow{\$} \mathcal{A}^\ell & \text{otherwise} \end{cases}, \tag{51}$$

where by D we denote the database of previous queries, by primes we denote the contents of entries of D, and $\lfloor y' \rfloor_\ell$ denotes the first ℓ letters (in $\mathcal{A}$) of y'. Note that such description can be easily used to define a quantum accessible oracle for H. In the following section, we omit the second input and we mean that we ask for a single letter $\mathsf{H}(x) = y \in \mathcal{A}$.

5.2 Classical Indifferentiability of Sponges with Random Functions

In the game-playing proofs and Algorithms 6 and 7 described in this section we use the following convention: every version of the algorithm executes the part of the code that is **not boxed** and among the boxed statements only the part that is inside the box in the color corresponding to the color of the name in the definition.

First we present a slightly modified proof of indifferentiability from [Ber+08]. We modify the proof to better fit the framework of game-playing proofs. It is not our goal to obtain the tightest bounds nor the simplest (classical) proof. Instead, our classical game-playing proof paves the way to the quantum security proof which is presented in the next section.

Theorem 13 (SPONGE with functions, classical indifferentiability)**.** SPONGE$_\varphi$[PAD, $\mathcal{A}, \mathcal{C}$] *calling a random function* φ *is* (q, ε)*-indifferentiable from a random oracle, Eq.* (51)*, for* classical *adversaries for any* $q < |\mathcal{C}|$ *and* $\varepsilon = 8\frac{q(q+1)}{2|\mathcal{C}|}$.

Proof. The proof proceeds in six games that we show to be indistinguishable. We start with the real world: the public interface corresponding to the internal function φ is a random transformation and the private interface is SPONGE$_\varphi$. Then in a series of games we gradually change the environment of the adversary to finally reach the ideal world, where the public interface is simulated by the simulator and the private interface is a random oracle H. The simulators used in different games of the proof are defined in Alg. 6, the index of the simulator corresponds to the game in which the simulator is used. Explanations of the simulators follow.

Algorithm 6: Classical S_2, $\boxed{\mathsf{S}_3}$, $\boxed{\mathsf{S}_4}$, $\boxed{\mathsf{S}_6}$, functions

```
State  : current sponge graph $G$
Input  : $s \in \mathcal{A} \times \mathcal{C}$
Output: $\varphi(s)$
if $s$ has no outgoing edge then                                   // new query
    if $\hat{s} \in \mathcal{R} \wedge \mathcal{R} \cup \mathcal{U} \neq \mathcal{C}$ then      // $\hat{s}$-rooted, no saturation
        $\hat{t} \xleftarrow{\$} \mathcal{C}$, $\boxed{\boxed{\text{if } \hat{t} \in \mathcal{R} \cup \mathcal{U}, \text{ set Bad} = 1}}$, $\boxed{\hat{t} \xleftarrow{\$} \mathcal{C} \setminus (\mathcal{R} \cup \mathcal{U})}$
        Construct a path to $s$: $p :=$ SpPath$(s, G)$
        if $\exists x : p = \text{PAD}(x)$ then
            $\bar{t} \xleftarrow{\$} \mathcal{A}$
            $\boxed{\boxed{\bar{t} := \mathsf{H}(x)}}$
        else
            $\bar{t} \xleftarrow{\$} \mathcal{A}$
        $t := (\bar{t}, \hat{t})$
    else
        $t \xleftarrow{\$} \mathcal{A} \times \mathcal{C}$
    Add an edge $(s, t)$ to $\mathcal{E}$.
Set $t$ to the vertex at the end of the edge starting at $s$
Output $t$
```

Game 1 We start with the real world where the distinguisher A has access to a random function $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$ and SPONGE$_\varphi$ using this random function. The formal definition of the first game is the event

$$\textbf{Game 1} := (b = 1 : b \leftarrow \mathsf{A}[\text{SPONGE}_\varphi, \varphi]) . \tag{52}$$

Game 2 In the second game we introduce the simulator S_2—defined in Alg. 6—that lazy-samples the random function φ. In Alg. 6 we define all simulators of this proof at once, but note that the behavior of S_2 is not influenced by any of the conditional "if" statements (in lines 1, 2, and 5), because in the end, the output state t is picked uniformly from $\mathcal{A} \times \mathcal{C}$ anyway. The definition of the second game is

$$\textbf{Game 2} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_2}, \mathsf{S}_2]) . \tag{53}$$

Because the simulator S_2 perfectly models a random function and we use the same function for the private interface we have

$$|\mathbb{P}[\textbf{Game 2}] - \mathbb{P}[\textbf{Game 1}]| = 0. \tag{54}$$

Game 3 In the next step we modify S_2 to S_3. The game is then

$$\textbf{Game 3} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]) . \tag{55}$$

We made a single change in S_3 compared to S_2, we introduce the "bad" event Bad that marks the difference between algorithms. We use this event as the bad event in Lemma 1. With such a change of the simulators we can use Lemma 1 to bound the difference of probabilities:

$$|\mathbb{P}[\textbf{Game 3}] - \mathbb{P}[\textbf{Game 2}]| \leq \mathbb{P}[\text{Bad} = 1]. \tag{56}$$

It is now quite easy to bound $\mathbb{P}[\text{Bad} = 1]$ as it is the probability of finding a collision or preimage of the root in the set $\mathcal{C}$ having made q random samples. Then we have that

$$\mathbb{P}[\text{Bad} = 1] \leq f_{\text{coll}}(q), \tag{57}$$

where f_{coll} is defined in Eq. (50). The bound is not necessarily tight as not all queries are made to rooted nodes.

Game 4 In this step we introduce the random oracle H but only to generate the outer part of the output of φ. The game is defined as

$$\textbf{Game 4} := \left(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, \mathsf{S}_4^{\mathsf{H}}]\right) . \tag{58}$$

Now we observe that if $\text{Bad} = 0$ the outputs are identically distributed.

Claim 14. *Given that* $\text{Bad} = 0$ *the mentioned games are the same:*

$$|\mathbb{P}[\textbf{Game 4} \mid \text{Bad} = 0] - \mathbb{P}[\textbf{Game 3} \mid \text{Bad} = 0]| = 0. \tag{59}$$

Proof. Note that the inner part is distributed in the same way in both games if $\text{Bad} = 0$, so we only need to take care of the outer part of the output. The problem might lie in the outer part, as we modify the output from a random sample to $\mathsf{H}(x)$. If $\text{Bad} = 0$ then $\hat{t}$ is not rooted and has no outgoing edge, also the whole graph G does not contain two paths leading to the same supernode. Hence, x was not queried before and is uniformly random. This reasoning is made more formal in Lemma 1 and Lemma 2 of [Ber+07]. □

The two games are identical-until-bad, this implies that the probability of setting Bad to one in both games is the same $\mathbb{P}[\text{Bad} = 1 : \textbf{Game 3}] = \mathbb{P}[\text{Bad} = 1 : \textbf{Game 4}]$. Together with the above claim we can derive the advantage:

$$\begin{aligned}
|\mathbb{P}[\textbf{Game 4}] - \mathbb{P}[\textbf{Game 3}]| \overset{\text{Claim 14}}{=} \Big| &\mathbb{P}[\textbf{Game 4} \mid \text{Bad} = 0] \\
&\cdot \underbrace{(\mathbb{P}[\text{Bad} = 1 : \textbf{Game 3}] - \mathbb{P}[\text{Bad} = 1 : \textbf{Game 4}])}_{=0} \\
&+ \underbrace{\mathbb{P}[\textbf{Game 3} \mid \text{Bad} = 1]}_{\leq 1}\mathbb{P}[\text{Bad} = 1] + \underbrace{\mathbb{P}[\textbf{Game 4} \mid \text{Bad} = 1]}_{\leq 1}\mathbb{P}[\text{Bad} = 1]\Big| \qquad (60) \\
&\leq 2\mathbb{P}[\text{Bad} = 1]. \qquad (61)
\end{aligned}$$

Game 5 In this stage of the proof we change the private interface to contain the actual random oracle. The simulator is the same as before and the game is

$$\textbf{Game 5} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{S}_4^{\mathsf{H}}]\right). \tag{62}$$

Conditioned on Bad $= 0$, the outputs of the simulator in Games 4 and 5 act in the same way and are consistent with H. To calculate the adversary's advantage in distinguishing between the two games we can follow the proof of Lemma 12. We change $\mathsf{H}\backslash R_1$ to **Game 5**, $\mathsf{G}\backslash R_2$ to **Game 4**, and event Find to Bad $= 1$. As the derivation of Lemma 12 uses no quantum mechanical arguments and the assumption holds—the games are identical conditioned on Bad $= 0$—the bound holds:

$$|\mathbb{P}[\textbf{Game 5}] - \mathbb{P}[\textbf{Game 4}]| \leq 4\mathbb{P}[\text{Bad} = 1] \leq 4f_{\text{coll}}(q). \tag{63}$$

Game 6 In the last game we use S_6, a simulator that does not check for bad events and samples from the "good" subset of $\mathcal{C}$. The game is

$$\textbf{Game 6} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{S}_6^{\mathsf{H}}]\right) \tag{64}$$

and the advantage is

$$|\mathbb{P}[\textbf{Game 6}] - \mathbb{P}[\textbf{Game 5}]| \leq \mathbb{P}[\text{Bad} = 1] \leq f_{\text{coll}}(q). \tag{65}$$

following Lemma 1. as the only difference is in code but not outputs. We included this last game in the proof because S_6 is clearly a simulator that might fail only if G is saturated but this does not happen if $q < |\mathcal{C}|$. Collecting and adding all the differences yields the claimed $\varepsilon = 8f_{\text{coll}}(q)$. □

5.3 Quantum Indifferentiability of Sponges with Random Functions

In this subsection we prove quantum indifferentiability of the sponge construction with a uniformly random internal function.

In the quantum indifferentiability simulator we want to sample the outer part of inputs of φ and the inner part separately, similarly to the classical one. To do that correctly in the quantum case though we need to maintain two databases: one responsible for the outer part and the other for the inner part. We denote them by $\overline{D}$ and $\widehat{D}$ respectively. Note that this division makes our formulation fit the definition from Alg. 4 but in principle could be avoided by a different formulation The crucial point here is maintaining the databases with the invariants we specified in mind; If we add x to D when sampling $\hat{t}$, the database is not in the correct form from the point of sampling $\bar{t}$.

At line 7 of the classical simulator we swap lazy sampled outer state to the output of the random oracle. In the quantum case we want to do the same but still save the output in the database. To do that we use CStO described in Alg. 4 with H instead of the sampling procedure Samp. Another modification is that instead of feeding the adversary's query s to H we use the sponge path x, kept in register X_H. When acting on registers $XX_H\overline{YD}(s)$, we denote this oracle by

$$\mathsf{CH}^{XX_H\overline{YD}(s)}. \tag{66}$$

Theorem 15 (Sponge with functions, quantum indifferentiability). *$\text{Sponge}_\varphi[\text{pad}, \mathcal{A}, \mathcal{C}]$ calling a random function φ is (q, ε)-indifferentiable from a random oracle, Eq.* (51), *for* quantum *adversaries for any $q < |\mathcal{C}|$ and $\varepsilon = 2\sqrt{\frac{q(q+1)^2}{2|\mathcal{C}|}} + \frac{10q(q+1)}{2|\mathcal{C}|}$.*

Proof. Even though we allow for quantum accessible oracles, the proof we present is very similar to the classical case. The proof follows the same structure, the biggest difference is in the simulators that now use the compressed oracle to lazy-sample appropriate answers.

We denote by U_G the unitary that acting on $|0\rangle$ constructs G including edges consistent with queries held by the quantum compressed database from register D. Similarly we define $\mathsf{U}_{\mathcal{R}\cup\mathcal{U}}$ to temporarily create a description of the set of supernodes that are rooted or have an outgoing edge.

In Alg. 7 we describe the simulators we use in this proof. In the quantum simulators we also make use of the graph representation of sponges. Note however that in a single query we only care about the graph before the query. Due to that fact we can apply the compressed oracle defined in Alg. 4 and additionally analyzed in Lemma 21. Lemma 21 provides a bound of the probability of Find in the case of compressed oracles and relations relevant for the sponge construction.

Line 1 of Alg. 7 corresponds to lines 1 to 5 of Alg. 4, together with appropriate Samp procedure, as discussed below Alg. 4. For clarity of presentation we omit some of the details but the procedure follows the mentioned part of the general algorithm for $\mathsf{CStO}_{\mathfrak{D}}$ and uses Samp corresponding to the correct version of the simulator, without the measurement in the case of S_4. In lines 3–13 of Alg. 7 we expand line 7 of Alg. 4 to accommodate for the changes in different games of the proof. It is important to note that the "IF" statements are in fact quantum controlled operations.

```
Algorithm 7: Quantum $\boxed{\mathsf{S}_2}$, $\boxed{\mathsf{S}_3}$, $\boxed{\mathsf{S}_4}$, $\boxed{\mathsf{S}_6}$, functions
    State  : Quantum compressed database register $D$
    Input  : $|s,v\rangle \in \mathcal{H}^{\otimes 2}_{\mathcal{A}\times\mathcal{C}}$
    Output: $|s, v+\varphi(s)\rangle$
 1  Locate input $s$ in $\overline{D}$ and $\widehat{D}$                    // Using the correct Samp
 2  Apply $\mathsf{U}_{\mathcal{R}\cup\mathcal{U}} \circ \mathsf{U}_G$ to register $\widehat{D}$ and two fresh registers
 3  if $\hat{s} \in \mathcal{R} \wedge \mathcal{R}\cup\mathcal{U} \neq \mathcal{C}$ then        // $\hat{s}$-rooted, no saturation
 4      Apply $\boxed{\mathsf{CStO}_{\mathcal{C}}^{X\widehat{Y}\widehat{D}(s)}}$, $\boxed{\boxed{(\mathsf{CStO}_{\mathcal{C}} \setminus (\mathcal{R}\cup\mathcal{U}))^{X\widehat{Y}\widehat{D}(s)}}}$, $\boxed{\mathsf{CStO}_{\mathcal{C}\setminus(\mathcal{R}\cup\mathcal{U})}^{X\widehat{Y}\widehat{D}(s)}}$, result: $\hat{t}$      // The
        red oracle is punctured!
 5      Construct a path to $s$: $p := \mathsf{SpPath}(s, G)$
 6      if $\exists x : p = \textsc{pad}(x)$ then
 7          $\boxed{\boxed{\text{Apply } \mathsf{CStO}_{\mathcal{A}}^{X\overline{Y}\overline{D}(s)}}}$, result: $\bar{t}$
 8          Write $x$ in a fresh register $X_H$, $\boxed{\boxed{\text{apply } \mathsf{CH}^{XX_H\overline{Y}\overline{D}(s)}}}$, uncompute $x$ from $X_H$,
             result: $\bar{t}$                                   // Random oracle, Eq. (66)
 9      else
10          Apply $\mathsf{CStO}_{\mathcal{A}}^{X\overline{Y}\overline{D}(s)}$, result: $\bar{t}$
11      $t := (\bar{t}, \hat{t})$, the value of registers $(\overline{D}^{Y}(s), \widehat{D}^{Y}(s))$
12  else
13      Apply $\mathsf{CStO}_{\mathcal{A}\times\mathcal{C}}^{XY\overline{D}(s)\widehat{D}(s)}$, result: $t$
14  Uncompute $G$ and $\mathcal{R}\cup\mathcal{U}$
15  Output $|s, v+t\rangle$
```

An illustration of the simulators in the quantum case is depicted in Fig.3.

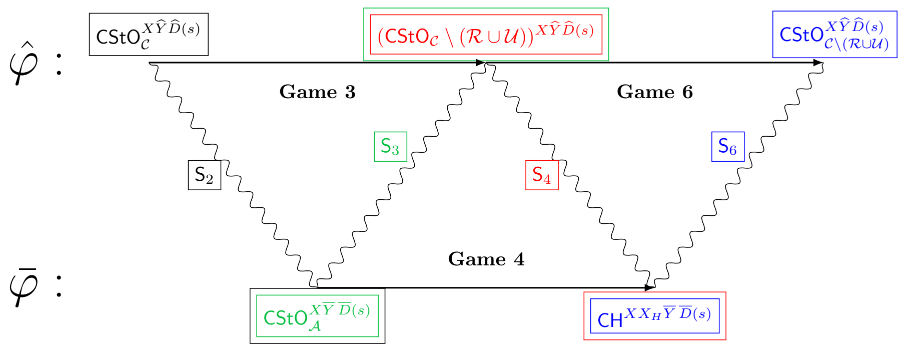
$\hat{\varphi}$:
$\mathsf{CStO}_{\mathcal{C}}^{X\widehat{Y}\widehat{D}(s)}$
$(\mathsf{CStO}_{\mathcal{C}} \setminus (\mathcal{R}\cup\mathcal{U}))^{X\widehat{Y}\widehat{D}(s)}$
$\mathsf{CStO}_{\mathcal{C}\setminus(\mathcal{R}\cup\mathcal{U})}^{X\widehat{Y}\widehat{D}(s)}$
Game 3
Game 6
S_3
S_6
S_2
S_4
Game 4
$\bar{\varphi}$:
$\mathsf{CStO}_{\mathcal{A}}^{X\overline{Y}\,\overline{D}(s)}$
$\mathsf{CH}^{XX_H\overline{Y}\,\overline{D}(s)}$

Figure 3: Schematics of the simulators defined in Alg. 7, horizontal arrows signify the change introduced in the labeled game. Note that despite the syntactic similarity of the red and blue oracle, it is important to notice that the red oracle is punctured as in Definition 8, while the blue oracle simply samples from the "good set" $\mathcal{C} \setminus (\mathcal{R} \cup \mathcal{U})$.

Game 1 We start with the real world where the distinguisher A has quantum access to a random function $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$ and the $\textsc{Sponge}_\varphi$ construction using this random function. The definition of the first game is

$$\textbf{Game 1} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_\varphi, \varphi]) \,. \tag{67}$$

Game 2 In the second game we introduce the simulator S_2, defined in Alg. 7. This algorithm is essentially a compressed random oracle, the only difference are the if statements, note that the behavior of S_2 is not influenced by any of the conditional "if" statements (in lines 3, and 6), because in the end, the output state t is picked uniformly from $\mathcal{A} \times \mathcal{C}$ anyway. The game is defined as:

$$\textbf{Game 2} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_2}, \mathsf{S}_2]) \,. \tag{68}$$

Because the simulator S_2 perfectly models a quantum random function and we use the same function for the private interface we have

$$|\mathbb{P}[\textbf{Game 2}] - \mathbb{P}[\textbf{Game 1}]| = 0. \tag{69}$$

Game 3 In the next step we modify S_2 to S_3. The game is then

$$\textbf{Game 3} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]) \,. \tag{70}$$

With such a change of the simulators we can use Thm. 9 to bound the difference of probabilities:

$$|\mathbb{P}[\textbf{Game 3}] - \mathbb{P}[\textbf{Game 2}]| \leq 2\sqrt{(q+1)\mathbb{P}[\mathrm{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]]}. \tag{71}$$

S_3 measures the relation of being an element of $\mathcal{R} \cup \mathcal{U}$. Being in this relation is equivalent to finding a random sample that is an element of this set. Note that in the punctured oracle $(\mathsf{CStO}_{\mathcal{C}} \setminus (\mathcal{R}\cup\mathcal{U}))^{XYD^{\hat{Y}}}$ we measure the relation after each query of the adversary and her best chance of finding an element in $\mathcal{R} \cup \mathcal{U}$ is querying a fresh input. The output of a fresh input is an equal superposition of all elements of $\mathcal{C}$; the following measurement of the relation has probability of succeeding of at most $|\mathcal{R} \cup \mathcal{U}| \,/\, |\mathcal{C}|$. It follows that $\mathbb{P}[\mathrm{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]]$ is upper bounded by the probability of finding a collision or a preimage of the root in the set $\mathcal{C}$ having made q random samples. A formal statement of this claim is shown in Lemma 21, for $\mathcal{S} = \emptyset$ and $\mathcal{Z} = \{0\}$. We have that

$$\mathbb{P}[\mathrm{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]] \leq f_{\mathrm{coll}}(q). \tag{72}$$

Game 4 In this step we introduce the random oracle H but only to generate the outer part of the output of φ. The game is defined as

$$\textbf{Game 4} := \left(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, \mathsf{S}_4^{\mathsf{H}}]\right). \tag{73}$$

Thanks to the classical argument we have that S_4 and S_3 are identical until bad, as in Def. 11. Then we can use Lemma 12 to bound the advantage of the adversary

$$|\mathbb{P}[\textbf{Game 4}] - \mathbb{P}[\textbf{Game 3}]| \leq 4\mathbb{P}[\text{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]] \leq 4f_{\text{coll}}(q). \tag{74}$$

Game 5 In this stage of the proof we change the private interface to contain the actual random oracle. In this game the simulator is still S_4, the definition is as follows:

$$\textbf{Game 5} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{S}_4^{\mathsf{H}}]\right) \tag{75}$$

and the advantage is

$$|\mathbb{P}[\textbf{Game 5}] - \mathbb{P}[\textbf{Game 4}]| \leq 4\mathbb{P}[\text{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, \mathsf{S}_4^{\mathsf{H}}]] \leq 4f_{\text{coll}}(q). \tag{76}$$

Conditioned on $\neg$Find, the outputs of the private interface are the same, then the games are identical-until-bad and we can use Lemma 12 to bound the advantage of the adversary.

Game 6 In the last game we use S_6, a simulator that uses a non-uniform compressed oracle. The game is

$$\textbf{Game 6} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{S}_6^{\mathsf{H}}]\right) \tag{77}$$

and the advantage is again

$$|\mathbb{P}[\textbf{Game 6}] - \mathbb{P}[\textbf{Game 5}]| \leq 2\mathbb{P}[\text{Find} : \mathsf{A}[\mathsf{H}, \mathsf{S}_4^{\mathsf{H}}]] \leq 2f_{\text{coll}}(q), \tag{78}$$

now by Lemma 12 with $R_2 = \emptyset$. The last simulator S_6 works perfectly until the sponge graph is saturated. Saturation certainly does not occur for $q < |\mathcal{C}|$ as the database in every branch of the superposition increases by at most one in every query. Collecting the differences between games yields the claimed ε. □

5.4 Quantum Indifferentiability of Sponges with Random Permutations

In the same way as for a random internal function, we can prove quantum indifferentiability of the sponge construction with the internal function instantiated with a random permutation.

Theorem 16 (Sponge with permutations, quantum indifferentiability). $\textsc{Sponge}_{\varphi}[\textsc{pad}, \mathcal{A}, \mathcal{C}]$ *calling a random permutation φ is (q, ε)-indifferentiable from a random oracle, Eq.* (51), *for* quantum *adversaries for any $q < |\mathcal{C}|$ and $\varepsilon = 4\sqrt{(q+1)f_{\text{coll}}(q)} + 12f_{\text{coll}}$.*

The proof is identical in spirit to the proof of Thm.15 and can be found in Appendix C. The main difference between the proofs for random functions and random permutations lies in the fact that the simulator has to provide interfaces for both φ and φ^{-1}. Nonetheless our framework, making heavy use of the classical proof—also presented in Appendix C—translates the arguments to the quantum threat model.

5.5 Collapsingness of Sponges with Random Permutations

Collapsingness is a security notion defined in [Unr16b]; It is a purely quantum notion strengthening collision resistance. It was developed to capture the required feature of hash functions used in cryptographic commitment protocols.

In this section we prove that quantum indifferentiability implies collapsingness, from what follows that the sponge construction instantiated with a random permutation is collapsing. We derive this result from Thm. 16, and previously proven results concerning random oracles. We begin by introducing the notion of *collapsing* functions.

For quantum algorithms A, B with quantum access to H, consider the following games:

$$\textbf{Collapse 1}: \quad (S, M, h) \leftarrow \mathsf{A}^{\mathsf{H}}(),\ m \leftarrow \mathsf{M}(M),\ b \leftarrow \mathsf{B}^{\mathsf{H}}(S, M), \tag{79}$$

$$\textbf{Collapse 2}: \quad (S, M, h) \leftarrow \mathsf{A}^{\mathsf{H}}(), \qquad\qquad b \leftarrow \mathsf{B}^{\mathsf{H}}(S, M). \tag{80}$$

Here S, M are quantum registers. $\mathsf{M}(M)$ is a measurement of M in the computational basis. The intuitive meaning of the above games is that part A of the adversary prepares a quantum register M that holds a superposition of inputs to H that all map to h. Then she sends M along with the side information S to B. The task of the second part of the adversary is to decide whether measurement M of the register M occurred or not.

We call an adversary (A, B) *valid* if and only if $\mathbb{P}[\mathsf{H}(m) = h] = 1$ when we run $(S, M, h) \leftarrow \mathsf{A}^{\mathsf{H}}()$ in **Collapse 1** from Eq.(79) and measure M in the computational basis as m.

Definition 17 (Collapsing [Unr16b])**.** *A function* H *is* collapsing *if for any valid quantum-polynomial-time adversary* (A, B)

$$|\mathbb{P}[b = 1 : \textbf{Collapse 1}] - \mathbb{P}[b = 1 : \textbf{Collapse 2}]| < \varepsilon, \tag{81}$$

where the collapsing-advantage ε *is negligible.*

A more in-depth analysis of this security notion can be found in [Unr16b; Unr16a; Cza+18; Feh18].

It was shown in [Unr16b] that if H is a random oracle then is it collapsing:

Lemma 18 (Lemma 37 [Unr16b])**.** *Let* $\mathsf{H} : \mathcal{X} \to \mathcal{Y}$ *be a random oracle, then any valid adversary* $(\mathsf{A}^{\mathsf{H}}, \mathsf{B}^{\mathsf{H}})$ *making q quantum queries to* H *has collapsing-advantage* $\varepsilon \in O\left(\sqrt{\frac{q^3}{|\mathcal{Y}|}}\right)$.

In the rest of this section we state and prove that Sponge with a random permutation is collapsing. Through the strong notion of indifferentiability we manage to overcome the limitation of [Cza+18]. While there, the authors prove that sponges are collapsing if the adversary is given one-way access to the internal function, we show that the sponge construction is collapsing even given inverse access to the internal function. We begin by proving that any function that is indifferentiable from a collapsing function is itself collapsing.

Theorem 19 (Quantum indifferentiability preserves collapsingness)**.** *Let* C *be a construction based on an internal function f, and let* C *be* $(q, \varepsilon_I(q))$*-indifferentiable from an ideal function* $\mathsf{C}_{\text{ideal}}$ *with simulator S. Assume further that* $\mathsf{C}_{\text{ideal}}$ *allows for a collapsingness advantage at most* $\varepsilon_{\text{coll}}(q)$ *for a q-query adversary. Then* C *is collapsing with advantage* $\varepsilon_{\text{coll}}(q_{\mathsf{C}}, q_f) = 2\,\varepsilon_I(q_{\mathsf{C}} + q_f) + \varepsilon_{\text{coll}}(q_{\mathsf{C}} + \alpha q_f)$*, where* q_{C} *and* q_f *are the number of queries to* C *and f, respectively, and α is the number of queries simulator* S *makes (at most) to* $\mathsf{C}_{\text{ideal}}$ *for each time it is queried.*

Proof. Given a collapsingness distinguisher $\tilde{\mathsf{D}}$ against C with advantage $\varepsilon \geq \varepsilon_{\text{coll}}(q_{\mathsf{C}} + \alpha q_f)$ that makes q_{C} queries to C and q_f queries to f, we build an indifferentiability distinguisher D as follows. Chose $b \in \{0, 1\}$ at random. Running $\tilde{\mathsf{D}}$, if $b = 0$ simulate **Collapse 1**, if $b = 1$ simulate **Collapse 2**. Output 1 if $\tilde{\mathsf{D}}$ outputs b, and 0 else.

In the real world, we have that

$$\begin{aligned}\mathbb{P}[1 \leftarrow \mathsf{D} : \mathbf{Real}] &= \frac{1}{2}\left(\mathbb{P}[0 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C},f} : \mathbf{Collapse\ 1}] + \mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C},f} : \mathbf{Collapse\ 2}]\right) \\ &= \frac{1}{2} + \frac{1}{2}\left(\mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C},f} : \mathbf{Collapse\ 2}] - \mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C},f} : \mathbf{Collapse\ 1}]\right).\end{aligned}$$

In the ideal world, the distinguisher together with the simulator S can be seen as a collapsingness distinguisher for $\mathsf{C}_{\text{ideal}}$. Therefore we get

$$\mathbb{P}[1 \leftarrow \mathsf{D} : \mathbf{Ideal}] = \frac{1}{2} + \frac{1}{2}\left(\mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C}_{\text{ideal}},\mathsf{S}} : \mathbf{Collapse\ 2}] - \mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C}_{\text{ideal}},\mathsf{S}} : \mathbf{Collapse\ 1}]\right)$$

and hence

$$\begin{aligned}\Big|\mathbb{P}[1 \leftarrow \mathsf{D} : \mathbf{Real}] - \mathbb{P}[1 \leftarrow \mathsf{D} : \mathbf{Ideal}]\Big| &= \frac{1}{2}\Big|\mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C},f} : \mathbf{Collapse\ 2}] - \mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C},f} : \mathbf{Collapse\ 1}] \\ &\quad - \mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C}_{\text{ideal}},\mathsf{S}} : \mathbf{Collapse\ 2}] + \mathbb{P}[1 \leftarrow \tilde{\mathsf{D}}^{\mathsf{C}_{\text{ideal}},\mathsf{S}} : \mathbf{Collapse\ 1}]\Big| \\ &\geq \frac{1}{2}\left(\varepsilon - \varepsilon_{\text{coll}}(q_{\mathsf{C}} + \alpha q_f)\right).\end{aligned}$$

□

As a corollary we get the collapsingness of the sponge construction.

Corollary 20 (Sponge with random permutations, collapsingness). *Fix the number of output characters in $\mathcal{A}$ to ℓ, and let* H *be a random oracle defined in Eq.*(51).*Then the sponge construction is collapsing with advantage*

$$\varepsilon = 2\,\varepsilon_I + O\left(\sqrt{\frac{q^3}{|\mathcal{A}|^{\ell}}}\right), \tag{82}$$

where ε_I is the indifferentiability error of the sponge constructions with random functions, or random permutations, according to Theorem 15, or Theorem 16, repsectively. This holds against adversaries that have oracle access to the internal function, or the internal permutation and its inverse, respectively.

Proof. follows directly from Theorem 19. □

Given that the adversary picks the shortest output, $\ell = 1$, the collapsing advantage we get is of the same order as in [Cza+18].

6 Conclusions

We develop a tool that allows for easier translation of classical security proofs to the quantum setting. Our technique shows that given the right proof structure it is relatively easy to prove stronger security notions valid in the quantum world.

It remains open to what degree classical security implies quantum security. An important open problem is specifying features of classical cryptographic constructions that allows constructions to retain their security properties in the quantum world. More concretely, tackling the problem of indifferentiability of other constructions will provide more evidence and possibly lead towards a general answer.

7 Acknowledgments

The authors thank Gorjan Alagic, Andreas Hülsing and Dominique Unruh for enlightening discussions about the superposition oracle technique. Furthermore, the authors thank Dominique Unruh for sharing a draft of [Unr19b]. The authors were supported by a NWO VIDI grant (Project No. 639.022.519). We would also like to thank the anonymous reviewers for their insightful comments.

References

[Ala+18] Gorjan Alagic, Christian Majenz, Alexander Russell, and Fang Song. "Quantum-secure message authentication via blind-unforgeability". Cryptology ePrint Archive, Report 2018/1150. https://eprint.iacr.org/2018/1150. 2018 (cit. on p. 4).

[AHU18] Andris Ambainis, Mike Hamburg, and Dominique Unruh. "Quantum security proofs using semi-classical oracles". Cryptology ePrint Archive, Report 2018/904. https://eprint.iacr.org/2018/904. 2018 (cit. on pp. 3, 4, 5, 17, 18, 19).

[BR93] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In: *Proceedings of the 1st ACM conference on Computer and communications security*. ACM. 1993, pp. 62–73. DOI: 10.1145/168588.168596 (cit. on pp. 3, 5).

[BR06] Mihir Bellare and Phillip Rogaway. "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs". In: *Advances in Cryptology - EUROCRYPT 2006*. https://eprint.iacr.org/2004/331. Springer Berlin Heidelberg, 2006, pp. 409–426. DOI: 10.1007/11761679_25 (cit. on pp. 3, 5).

[BBD09] D.J. Bernstein, J. Buchmann, and E. Dahmen. *Post-Quantum Cryptography*. Springer Berlin Heidelberg, 2009 (cit. on p. 3).

[Ber+07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Sponge functions". In: *ECRYPT hash workshop*. Vol. 2007. 9. https://keccak.team/files/SpongeFunctions.pdf. Citeseer. 2007 (cit. on pp. 3, 4, 21, 22, 25).

[Ber+08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction". In: *Advances in Cryptology – EUROCRYPT 2008*. Springer Berlin Heidelberg, 2008, pp. 181–197. DOI: 10.1007/978-3-540-78967-3_11 (cit. on pp. 4, 24, 48).

[Bon+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *Advances in Cryptology – ASIACRYPT 2011*. LNCS 7073. 2011, pp. 41–69. DOI: 10.1007/978-3-642-25385-0_3 (cit. on pp. 3, 7).

[Car+18] Tore Vincent Carstens, Ehsan Ebrahimi, Gelo Noel Tabia, and Dominique Unruh. "On Quantum Indifferentiability". Cryptology ePrint Archive, Report 2018/257. https://eprint.iacr.org/2018/257. 2018 (cit. on p. 4).

[Cor+05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. "Merkle-Damgård Revisited: How to Construct a Hash Function". In: *Advances in Cryptology – CRYPTO 2005*. Springer Berlin Heidelberg, 2005, pp. 430–448. DOI: 10.1007/11535218_26 (cit. on pp. 4, 6).

[Cza+18] Jan Czajkowski, Leon Groot Bruinderink, Andreas Hülsing, Christian Schaffner, and Dominique Unruh. "Post-quantum Security of the Sponge Construction". In: *Post-Quantum Cryptography*. Springer International Publishing, 2018, pp. 185–204. DOI: 10.1007/978-3-319-79063-3_9 (cit. on pp. 4, 30, 31).

[CHS19] Jan Czajkowski, Andreas Hülsing, and Christian Schaffner. "Quantum Indistinguishability of Random Sponges". Cryptology ePrint Archive, Report 2019/069. https://eprint.iacr.org/2019/069. 2019 (cit. on p. 4).

[Dam90] Ivan Bjerre Damgård. "A Design Principle for Hash Functions". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Springer New York, 1990, pp. 416–427. DOI: 10.1007/0-387-34805-0_39 (cit. on p. 3).

[Feh18] Serge Fehr. "Classical Proofs for the Quantum Collapsing Property of Classical Hash Functions". In: *Theory of Cryptography*. Springer International Publishing, 2018, pp. 315–338. DOI: 10.1007/978-3-030-03810-6_12 (cit. on pp. 4, 30).

[HI19] Akinori Hosoyamada and Tetsu Iwata. "Tight Quantum Security Bound of the 4-Round Luby-Rackoff Construction". Cryptology ePrint Archive, Report 2019/243. https://eprint.iacr.org/2019/243. 2019 (cit. on pp. 5, 40).

[JZM19] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. "Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model". Cryptology ePrint Archive, Report 2019/134. https://eprint.iacr.org/2019/134. 2019 (cit. on p. 5).

[Mah18] U. Mahadev. "Classical Homomorphic Encryption for Quantum Circuits". In: *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. 2018, pp. 332–338. DOI: 10.1109/FOCS.2018.00039 (cit. on p. 11).

[MRH04] Ueli Maurer, Renato Renner, and Clemens Holenstein. "Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology". In: *Theory of Cryptography*. Springer Berlin Heidelberg, 2004, pp. 21–39. DOI: 10.1007/978-3-540-24638-1_2 (cit. on pp. 4, 5, 6, 7).

[Mer90] Ralph C. Merkle. "A Certified Digital Signature". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Springer New York, 1990, pp. 218–238. DOI: 10.1007/0-387-34805-0_21 (cit. on p. 3).

[NC11] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th. Cambridge University Press, 2011 (cit. on pp. 7, 19).

[NIS14] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Draft FIPS 202. 2014. URL: http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf (cit. on pp. 5, 7, 21).

[NIS15] NIST. *Secure Hash Standard (SHS)*. Draft FIPS 180-4. 2015. DOI: 10.6028/NIST.FIPS.180-4 (cit. on p. 5).

[OR07] David Sena Oliveira and Rubens Viana Ramos. "Quantum bit string comparator: circuits and applications". In: *Quantum Computers and Computing* 7.1 (2007), pp. 17–26 (cit. on p. 40).

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. "Careful with Composition: Limitations of the Indifferentiability Framework". In: *Advances in Cryptology – EUROCRYPT 2011*. Springer Berlin Heidelberg, 2011, pp. 487–506. DOI: 10.1007/978-3-642-20465-4_27 (cit. on p. 7).

[Sho94] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700 (cit. on p. 3).

[SY17] Fang Song and Aaram Yun. "Quantum Security of NMAC and Related Constructions - PRF Domain Extension Against Quantum attacks". In: *CRYPTO*. Springer, 2017, pp. 283–309. DOI: 10.1007/978-3-319-63715-0_10 (cit. on p. 4).

[Unr14] Dominique Unruh. "Revocable Quantum Timed-Release Encryption". In: *Advances in Cryptology – EUROCRYPT 2014*. Springer Berlin Heidelberg, 2014, pp. 129–146. DOI: 10.1007/978-3-642-55220-5_8 (cit. on pp. 1, 3, 4, 5, 16).

[Unr16a] Dominique Unruh. "Collapse-Binding Quantum Commitments Without Random Oracles". In: *Advances in Cryptology – ASIACRYPT 2016*. Springer Berlin Heidelberg, 2016, pp. 166–195. DOI: 10.1007/978-3-662-53890-6_6 (cit. on pp. 4, 30).

[Unr16b] Dominique Unruh. “Computationally Binding Quantum Commitments”. In: *Advances in Cryptology – EUROCRYPT 2016*. Springer Berlin Heidelberg, 2016, pp. 497–527. DOI: 10.1007/978-3-662-49896-5_18 (cit. on pp. 4, 30).

[Unr19a] Dominique Unruh. “Quantum Relational Hoare Logic”. In: *Proc. ACM Program. Lang.* POPL (2019), 33:1–33:31. DOI: 10.1145/3290346 (cit. on p. 4).

[Unr19b] Dominique Unruh. *Recording quantum queries – explained*. In preparation. 2019 (cit. on pp. 8, 31, 36).

[Win99] Andreas Winter. “Coding theorem and strong converse for quantum channels”. In: *IEEE Transactions on Information Theory* 45.7 (1999), pp. 2481–2485. DOI: 10.1109/18.796385 (cit. on p. 20).

[Zha18] Mark Zhandry. “How to Record Quantum Queries, and Applications to Quantum Indifferentiability”. Cryptology ePrint Archive, Report 2018/276. https://eprint.iacr.org/2018/276. 2018 (cit. on pp. 1, 3, 4, 5, 8, 9, 36).

Symbol Index

J_R	Projector on relation R.	17
QFT_N	The Quantum Fourier Transform	7
Rem	Removing $\text{и} = 0$ from the database	43
$\mathcal{R}$	The set of rooted supernodes	23
$\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S})$	Algorithm preparing a superposition of samples of outputs of $f \leftarrow \mathfrak{D}$ on inputs from $\mathcal{S}$.	10, 16, 16
S	Classical and quantum simulators.	24, 27, 49, 52
$\textsc{Sponge}_{\varphi}[\textsc{pad}, \mathcal{A}, \mathcal{C}]$	Sponge construction with the internal function φ, capacity set $\mathcal{C}$, and alphabet $\mathcal{A}$	22
StO	Standard Oracle	8
$\mathfrak{U}$	The uniform distribution.	36
Upd	Updating η in the database	42
$\mathcal{U}$	The set of supernodes with outgoing edges	23
V	The "set spreading" operation.	16
$\mathcal{V}$	The set of vertices of a sponge graph	22
$\oplus$	Bitwise XOR	37
$y, \eta, \text{и}$	Values in the Y register of a database in different bases	11

A Additional Details on Quantum-Accessible Oracles

A.1 Uniform Oracles

For ease of exposition, and to highlight the connection to the formalism in [Zha18], we present a discussion of compressed oracles with *uniform oracles* that model functions sampled uniformly at random from $\mathcal{F} := \{f : \{0,1\}^m \to \{0,1\}^n\}$. A complete formal treatment of the uniform case, including applications, can be found in [Unr19b].

We denote the uniform distribution over $\mathcal{F}$ by $\mathfrak{U}$. The cardinality of the set of functions is $|\mathcal{F}| = 2^{n2^m}$ and the truth table of any $f \in \mathcal{F}$ can be represented by 2^m rows of n bits each. Uniform oracles are the most studied in the random-oracle model and are also analyzed in [Zha18].

The transformation we use in the case of uniformly sampled functions is the Hadamard transform. The unitary operation to change between types of oracles is defined as

$$\mathsf{HT}_n|x\rangle := \frac{1}{\sqrt{2^n}} \sum_{\xi \in \{0,1\}^n} (-1)^{\xi \cdot x}|\xi\rangle, \tag{83}$$

where $\xi \cdot x$ is the inner product modulo two between the n-bit strings ξ and x viewed as vectors. In this section the registers X, Y are vectors in the n-qubit Hilbert space

$$\mathcal{B} := (\mathbb{C}^2)^{\otimes n}. \tag{84}$$

In what follows we first focus on *full* oracles, i.e. not compressed ones. We analyze in detail the relations between different pictures of the oracles: the Standard Oracle, the Fourier Oracle, and the intermediate Phase Oracle. Next we provide an explicit algorithmic description of the compressed oracle and discuss the behavior of the compressed oracle in different pictures.

For the QROM, usually the Standard Oracle is the oracle used. The initial state of the oracle is the uniform superposition of truth tables f representing functions $f : \{0,1\}^m \to \{0,1\}^n$. The Standard Oracle acts as follows

$$\mathsf{StO}_{\mathfrak{U}}|x, y\rangle_{XY} \frac{1}{\sqrt{|\mathcal{F}|}} \sum_{f \in \mathcal{F}} |f\rangle_F = \frac{1}{\sqrt{|\mathcal{F}|}} \sum_{f \in \mathcal{F}} |x, y \oplus f(x)\rangle_{XY} \otimes |f\rangle_F, \tag{85}$$

where instead of modular addition we use bitwise XOR denoted by $\oplus$. Note that in the above formulation $\mathsf{StO}_{\mathfrak{U}}$ is just a controlled XOR operation from the x-th row of the truth table to the output register Y. We add the subscript $\mathfrak{U}$ to denote that in the case of uniform distribution we also fix the input and output sets to bit-strings and the operation the oracle performs is not addition modulo N like we introduced it in the main body. The register F contains vectors in $\mathcal{B}^{\otimes 2^m}$, where $\mathcal{B}$ is defined in Eq. (84).

The Fourier Oracle that stores the queries of the adversary is defined as

$$\mathsf{FO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|\phi\rangle_F := |x,\eta\rangle_{XY}|\phi \oplus \chi_{x,\eta}\rangle_F, \tag{86}$$

where $\chi_{x,\eta} := (0^n, \ldots, 0^n, \eta, 0^n, \ldots, 0^n)$ is a table with 2^m rows, among which only the x-th row equals η and the rest are filled with zeros. Note that initially the Y register is in the Hadamard basis, for that reason we use Greek letters to denote its value.

To model the random oracle we initialize the oracle register F in the Hadamard basis in the all 0 state $|\phi\rangle = |0^{n2^m}\rangle$.

If we take the Standard Oracle again and transform the adversary's Y register instead, again using HT, we recover the commonly used Phase Oracle. More formally, the phase oracle is defined as

$$\mathsf{PhO}_{\mathfrak{U}} := (\mathbb{1}_m^X \otimes \mathsf{HT}_n^Y) \otimes \mathbb{1}_{n2^m}^F \circ \mathsf{StO}_{\mathfrak{U}} \circ (\mathbb{1}_m^X \otimes \mathsf{HT}_n^Y) \otimes \mathbb{1}_{n2^m}^F, \tag{87}$$

where $\mathbb{1}_n$ is the identity operator acting on n qubits.

Applying the Hadamard transform also to register F will give us the Fourier Oracle

$$\mathsf{FO}_{\mathfrak{U}} = (\mathbb{1}^{XY}) \otimes \mathsf{HT}_n^F \circ \mathsf{PhO}_{\mathfrak{U}} \circ (\mathbb{1}^{XY}) \otimes \mathsf{HT}_n^F. \tag{88}$$

The above relations show that we have a chain of oracles, similar to Eq. (11):

$$\mathsf{StO}_{\mathfrak{U}} \xleftrightarrow{\mathsf{HT}_n^Y} \mathsf{PhO}_{\mathfrak{U}} \xleftrightarrow{\mathsf{HT}_n^F} \mathsf{FO}_{\mathfrak{U}}. \tag{89}$$

In the following paragraphs we present some calculations explicitly showing how to use the technique and helping understanding why it is correct.

A.1.1 Full Oracles, Additional Details

In this section we show detailed calculations of identities claimed in Section A.1. First we analyze the Phase Oracle, introduced in Eq. (87). We can check by direct calculation that this yields the standard Phase Oracle,

$$\mathsf{PhO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|f\rangle_F = (-1)^{\eta\cdot f(x)}|x,\eta\rangle_{XY}|f\rangle_F. \tag{90}$$

Including the full initial state of the oracle register, we calculate

$$\begin{aligned}
&\mathsf{PhO}_{\mathfrak{U}}|x,\eta\rangle_{XY}\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|f\rangle_F \\
&= (\mathbb{1}_m^X \otimes \mathsf{HT}_n^Y) \otimes \mathbb{1}_{n2^m}^F \mathsf{StO}_{\mathfrak{U}}|x\rangle_X \frac{1}{\sqrt{2^n}}\sum_y (-1)^{\eta\cdot y}|y\rangle_Y \frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|f\rangle_F && (91)\\
&= (\mathbb{1}_m^X \otimes \mathsf{HT}_n^Y) \otimes \mathbb{1}_{n2^m}^F |x\rangle_X \frac{1}{\sqrt{2^n}}\sum_y\sum_{f\in\mathcal{F}} (-1)^{\eta\cdot y}|y\oplus f(x)\rangle_Y \frac{1}{\sqrt{|\mathcal{F}|}}|f\rangle_F && (92)\\
&= \frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|x\rangle_X \sum_\zeta \underbrace{\frac{1}{2^n}\sum_y (-1)^{\eta\cdot y}(-1)^{(y\oplus f(x))\cdot\zeta}}_{=\delta(\eta,\zeta)(-1)^{\zeta\cdot f(x)}}|\zeta\rangle_Y|f\rangle_F && (93)\\
&= \frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}(-1)^{\eta\cdot f(x)}|x\rangle_X|\eta\rangle_Y|f\rangle_F. && (94)
\end{aligned}$$

Applying the Hadamard transform also to register F will give us the Fourier Oracle. In the following calculation we denote acting on register F with $\mathsf{HT}_n^{\otimes 2^m}$ by HT_n^F.

$$\mathsf{HT}_n^F \circ \mathsf{PhO}_{\mathfrak{U}} \circ \mathsf{HT}_n^F |x,\eta\rangle_{XY}|0^{2^m n}\rangle_F = \mathsf{HT}_n^F \frac{1}{\sqrt{|\mathcal{F}|}} \sum_{f\in\mathcal{F}} (-1)^{\eta\cdot f(x)}|x,\eta\rangle|f\rangle_F$$
$$= \frac{1}{|\mathcal{F}|}\sum_{\phi,f}(-1)^{\phi\cdot f}(-1)^{\eta\cdot f(x)}|x,\eta\rangle|\phi\rangle_F$$
$$= \sum_{\phi} \underbrace{\frac{1}{2^{n(2^m-1)}} \sum_{f(x'\neq x)} (-1)^{\phi_{x'}\cdot f(x')}}_{=\delta(\phi_{x'},0^n)} \underbrace{\frac{1}{2^n}\sum_{f(x)}(-1)^{\phi_x\cdot f(x)}(-1)^{\eta\cdot f(x)}}_{=\delta(\phi_x,\eta)}|x,\eta\rangle|\phi\rangle_F$$
$$= |x,\eta\rangle|0^{2^m n}\oplus \chi_{x,\eta}\rangle \tag{95}$$

where we write $f(x)$ and ϕ_x to denote the x-th row of the truth table f and ϕ respectively.

A.1.2 Compressed Oracles, Additional Details

Let us state the input-output behavior of the compressed oracle $\mathsf{CFO}_{\mathfrak{U}}$ for uniform distributions. The input-output behavior of $\mathsf{CFO}_{\mathfrak{U}}$ is given by the following equation:

$$\mathsf{CFO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|x_1,\eta_1\rangle_{D_1}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|0^m,0^n\rangle_{D_q} = |x,\eta\rangle_{XY}|\psi_{r-1}\rangle$$
$$\otimes\begin{cases} |x_r,\eta_r\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|0^m,0^n\rangle_{D_q} & \text{if } \eta=0^n,\\ |x,\eta\rangle_{D_r}|x_r,\eta_r\rangle_{D_{r+1}}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_q} & \text{if } \eta\neq 0^n, x\neq x_r,\\ |x_r,\eta_r\oplus\eta\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|0^m,0^n\rangle_{D_q} & \text{if } \eta\neq 0^n, x=x_r,\\ & \eta\neq\eta_r,\\ |x_{r+1},\eta_{r+1}\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-2}}|0^m,0^n\rangle_{F_{q-1}}|0^m,0^n\rangle_{D_q} & \text{if } \eta\neq 0^n, x=x_r,\\ & \eta=\eta_r, \end{cases} \tag{96}$$

where $|\psi_{r-1}\rangle := |x_1,\eta_1\rangle_{D_1}\cdots|x_{r-1},\eta_{r-1}\rangle_{D_{r-1}}$.

In the following let us change the picture of the compressed oracle to see how the Compressed Standard Oracle and Compressed Phase Oracle act on basis states. Let us begin with the Phase Oracle, given by the Hadamard transform of the oracle database

$$\mathsf{CPhO}_{\mathfrak{U}} := \mathbb{1}_{n+m}\otimes\mathsf{HT}_n^{D^Y}\circ\mathsf{CFO}_{\mathfrak{U}}\circ\mathbb{1}_{n+m}\otimes\mathsf{HT}_n^{D^Y}, \tag{97}$$

where by $\mathsf{HT}_n^{D^Y}$ we denote transforming just the Y registers of the database: $\mathsf{HT}_n^{D^Y} := (\mathbb{1}_m\otimes\mathsf{HT}_n)^{\otimes q}$. Let us calculate the outcome of applying CPhO to a state for the first time, for simplicity we omit all but the first register of D

$$\mathsf{CPhO}_{\mathfrak{U}}|x,\eta\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}|0^m,z\rangle_D = \mathbb{1}_{n+m}\circ\mathsf{HT}_n^{D^Y}\circ\mathsf{CFO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|0^m,0^n\rangle_D \tag{98}$$
$$= \mathbb{1}_{n+m}\circ\mathsf{HT}_n^{D^Y}\left((1-\delta(\eta,0^n))|x,\eta\rangle_{XY}|x,\eta\rangle_D + \delta(\eta,0^n)|x,\eta\rangle_{XY}|0^m,0^n\rangle_D\right) \tag{99}$$
$$= \frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}\left((1-\delta(\eta,0^n))(-1)^{\eta\cdot z}|x,\eta\rangle_{XY}|x,z\rangle_D + \delta(\eta,0^n)|x,0^n\rangle_{XY}|0^m,z\rangle_D\right). \tag{100}$$

If we defined the Compressed Phase Oracle form the scratch we might be tempted to omit the coherent deletion of $\eta = 0^n$. The following attack shows that this would brake the correctness of the compressed oracles: The adversary inputs the equal superposition in the X register $\frac{1}{\sqrt{2^m}}\sum_x|x,0^n\rangle_{XY}$, after interacting with the regular $\mathsf{CPhO}_{\mathfrak{U}}$ the state after a single query is

$$\frac{1}{\sqrt{2^m}}\sum_x|x,0^n\rangle_{XY} \overset{\mathsf{CPhO}_{\mathfrak{U}}}{\mapsto} \frac{1}{\sqrt{2^m}}\sum_x|x,0^n\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_z|0^m,z\rangle_D, \tag{101}$$

but with a modified oracle that does not take care of this deleting, simply omits the term with $\delta(\eta, 0^n)$, let us call it $\mathsf{CPhO}'_{\mathfrak{U}}$, the resulting state is

$$\frac{1}{\sqrt{2^m}}\sum_x |x, 0^n\rangle_{XY} \overset{\mathsf{CPhO}'_{\mathfrak{U}}}{\mapsto} \frac{1}{\sqrt{2^m}}\sum_x |x, 0^n\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_z |x, z\rangle_D. \tag{102}$$

Performing a measurement of the X register in the Hadamard basis distinguishes the two states with probability $1 - \frac{1}{2^m}$.

Let us inspect the state after making two queries to the Compressed Phase Oracle

$$\begin{aligned}
&\mathsf{CPhO}_{\mathfrak{U}}|x_2, \eta_2\rangle_{X_2Y_2}\mathsf{CPhO}_{\mathfrak{U}}|x_1, \eta_1\rangle_{X_1Y_1}\frac{1}{2^n}\sum_{z_1,z_2\in\{0,1\}^n}|0^m, z_1\rangle_{D_1}|0^m, z_2\rangle_{D_2}\\
&= |x_2, \eta_2\rangle|x_1, \eta_1\rangle\frac{1}{2^n}\sum_{z_1,z_2}\Bigg((-1)^{\eta_1\cdot z_1}\delta(\eta_2, 0^n)(1-\delta(\eta_1, 0^n))\underbrace{|x_1, z_1\rangle_{F_1}|0^m, z_2\rangle_{F_2}}_{=|\psi^{\mathrm{NOT}})}\\
&+ \delta(\eta_2, 0^n)\delta(\eta_1, 0^n)\underbrace{|0^m, z_1\rangle_{F_1}|0^m, z_2\rangle_{F_2}}_{=|\psi^{\mathrm{NOT}})}\\
&+ (-1)^{\eta_2\cdot z_1}(1-\delta(\eta_2, 0^n))\delta(\eta_1, 0^n)\underbrace{|x_2, z_1\rangle_{F_1}|0^m, z_2\rangle_{F_2}}_{=|\psi^{\mathrm{ADD}})}\\
&+(-1)^{\eta_1\cdot z_1}(-1)^{\eta_2\cdot z_2}(1-\delta(\eta_2, 0^n))(1-\delta(x_1, x_2))(1-\delta(\eta_1, 0^n))\underbrace{|x_1, z_1\rangle_{F_1}|x_2, z_2\rangle_{F_2}}_{=|\psi^{\mathrm{ADD}})}\\
&+(1-\delta(\eta_2, 0^n))\delta(x_1, x_2)\delta(\eta_1, \eta_2)(1-\delta(\eta_1, 0^n))\underbrace{|0^m, z_1\rangle_{F_1}|0^m, z_2\rangle_{F_2}}_{=|\psi^{\mathrm{REM}})}\\
&+ (1-\delta(\eta_2, 0^n))\delta(x_1, x_2)(1-\delta(\eta_1, \eta_2))(1-\delta(\eta_1, 0^n))\\
&\cdot(-1)^{(\eta_1\oplus\eta_2)\cdot z_1}\underbrace{|x_1, z_1\rangle_{F_1}|0^m, z_2\rangle_{F_2}}_{=|\psi^{\mathrm{UPD}})}\Bigg),
\end{aligned} \tag{103}$$

where by the superscripts we denote the operation performed by $\mathsf{CPhO}_{\mathfrak{U}}$ on the compressed database. By ADD we denote adding a new pair (x, η), by UPD changing the Y register of an already stored database entry, REM signifies removal of a database entry, and NOT stands for doing nothing, that happens if the queried $\eta = 0^n$.

Let us now discuss the Compressed Standard Oracle. We know that it is the Hadamard transform of the adversary's register followed by $\mathsf{CPhO}_{\mathfrak{U}}$

$$\mathsf{CStO}_{\mathfrak{U}} = \mathbb{1}_m \otimes \mathsf{HT}_n^Y \circ \mathsf{CPhO}_{\mathfrak{U}} \circ \mathbb{1}_m \otimes \mathsf{HT}_n^Y. \tag{104}$$

Let us now present the action of CStO in the first query of the adversary

$$\begin{aligned}
&\mathsf{CStO}_{\mathfrak{U}}|x, y\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}|0^m, z\rangle_D\\
&= \mathbb{1}_m \otimes \mathsf{HT}_n^Y \circ \mathsf{CPhO}_{\mathfrak{U}}\frac{1}{\sqrt{2^n}}\sum_{\eta\in\{0,1\}^n}(-1)^{\eta\cdot y}|x, \eta\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}|0^m, z\rangle_D \qquad (105)\\
&= \mathbb{1}_m \otimes \mathsf{HT}_n^Y\frac{1}{\sqrt{2^n}}\sum_{\eta\in\{0,1\}^n}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}(-1)^{\eta\cdot y}\Bigg((1-\delta(\eta, 0^n))(-1)^{\eta\cdot z}|x, \eta\rangle_{XY}|x, z\rangle_D\\
&+ \delta(\eta, 0^n)|x, 0^n\rangle_{XY}|0^m, z\rangle_D\Bigg) \qquad (106)
\end{aligned}$$

$$= \frac{1}{2^n}\sum_{y',\eta}\frac{1}{\sqrt{2^n}}\sum_z(-1)^{\eta\cdot y}(-1)^{y'\cdot\eta}\Bigg((1-\delta(\eta,0^n))(-1)^{\eta\cdot z}|x,y'\rangle_{XY}|x,z\rangle_D$$

$$+\,\delta(\eta,0^n)|x,y'\rangle_{XY}|0^m,z\rangle_D\Bigg) \tag{107}$$

$$= \sum_{y'}\frac{1}{\sqrt{2^n}}\sum_z\underbrace{\frac{1}{2^n}\sum_{\eta\neq 0}(-1)^{\eta\cdot y}(-1)^{y'\cdot\eta}(-1)^{\eta\cdot z}}_{=\delta(y',y\oplus z)-\frac{1}{2^n}}|x,y'\rangle_{XY}|x,z\rangle_D$$

$$+\sum_{y'}\frac{1}{\sqrt{2^n}}\sum_z\frac{1}{2^n}|x,y'\rangle_{XY}|0^m,z\rangle_D \tag{108}$$

$$= \frac{1}{\sqrt{2^n}}\sum_z\left(|x,y\oplus z\rangle_{XY}|x,z\rangle_D-\frac{1}{2^n}\sum_{y'}|x,y'\rangle_{XY}|x,z\rangle_D+\frac{1}{2^n}\sum_{y'}|x,y'\rangle_{XY}|0^m,z\rangle_D\right). \tag{109}$$

We would like to note that a similar calculation and resulting state is presented in [HI19].

A.2 Detailed Algorithm for Alg. 1: $\mathsf{CFO}_{\mathfrak{D}}$

In Algorithm 8 we present the fully-detailed version of Algorithm 1. This algorithm runs the following subroutines:

- Locate, Function 9: This subroutine locates the positions in Д where the $x-$entry coincides with the $x-$entry of the query. The result is represented as q bits, where $q_i = 1 \iff Д_i^X = x$. This result is then bitwise XOR'd into an auxilliary register L.
- Add, Function 10: This subroutine adds queried x to the database and take care of appropriate padding.
- Upd, Function 11: This subroutine updates the database by subtracting η after a suitable basis transformation.
- Rem, Function 12: This subroutine removes $(0,0)$ entries from the database and puts them to the back in the form of padding.
- Clean, Function 13: This subroutine cleans the auxiliary registers setting them back to initial values.
- Larger: This subroutine determines whether one value is larger than a second value, it works on three registers, say $D^X XA$ and flips the bit in A if the value of D^X is larger than the value in X, so $\mathsf{Larger}^{D^X XA}|u\rangle_{D^X}|v\rangle_X|a\rangle_A = |u\rangle_{D^X}|v\rangle_X\begin{cases}|a\oplus 1\rangle_A \text{ if } u>v\\ |a\rangle_A \text{ otherwise}\end{cases}$. In [OR07] an efficient implementation of Larger for u,v being bitstrings can be found.

In the Add and Rem subroutine the unitary P can be found. P permutes the database such that a recently removed entry in the database is moved to the end of the database. Conversely P^{-1} permutes the database such that an empty entry is created in the database as to ensure the correct ordering of the $x-$entries after adding the query into this newly created empty entry:

$$\mathsf{P}|x_1,...,x_q\rangle\otimes|y_1,...,y_n\rangle := |\sigma_n\circ...\circ\sigma_1(x_1,...,x_q)\rangle\otimes|y_1,...,y_n\rangle\,, \tag{110}$$

where σ_i is applied conditioned on $y_i = 1$ and $\sigma_i(x_1,...,x_n) := (x_1,...,x_{i-2},x_{i-1},x_{i+1},x_{i+2},...,x_q,x_i)$.

Algorithm 8: Detailed $\mathsf{CFO}_{\mathfrak{D}}$

```
Input  : Unprepared database and adversary query: $|x,\eta\rangle_{XY}|Д\rangle_D$
Output: $|x,\eta\rangle_{XY}|Д'\rangle_D$
1  $|a\rangle_A = |0 \in \{0,1\}\rangle_A$                       // initialize auxiliary register $A$
2  $|l\rangle_L = |0^q \in \{0,1\}^q\rangle_L$                   // initialize auxiliary register $L$
3  $|l\rangle_L \mapsto \mathsf{Locate}(|x\rangle_X|Д\rangle_D|l\rangle_L)$      // locate $x$ in the database
4  if $l = 0^q$ then                                            // if not located
5      $|a\rangle_A \mapsto |a \oplus 1\rangle_A$                 // save result to register A
6  if $a = 1$ then                                              // if not located
7      $|Д\rangle_D|l\rangle_L \mapsto \mathsf{Add}(|x\rangle_X|Д\rangle_D)$     // add $x-$entry to the database
8  $|Д^Y\rangle_{D^Y} \mapsto \mathsf{Upd}(|\eta\rangle_Y|Д^Y\rangle_{D^Y}|l\rangle_L)$   // update register $D^Y$
9  $|Д\rangle_D|l\rangle_L \mapsto \mathsf{Rem}(|x\rangle_X|Д\rangle_D|l\rangle_L)$   // remove a database entry if $и = 0$
10 $|a\rangle_A \mapsto \mathsf{Clean}(|y\rangle_Y|Д^Y\rangle_{D^Y}|l\rangle_L)$   // uncompute register $A$
11 $|l\rangle_L \mapsto \mathsf{Locate}(|x\rangle_X|Д\rangle_D|l\rangle_L)$     // uncompute register $L$
12 return $|x,\eta\rangle_{XY}|Д'\rangle_D$                      // $Д'$ is the modified database
```

Function 9: Locate

```
Input  : $|x\rangle_X|Д\rangle_D|l\rangle_L$
Output: $|x\rangle_X|Д\rangle_D|l'\rangle_L$
1 Set $|a\rangle_A = |0 \in \mathcal{X}\rangle_A$                // initialize auxiliary register $A$
2 for $i = 1, ..., q$ do
3     if $и_i \neq 0$ then                                      // locate entries in the database
4         $|a\rangle_A \mapsto |a + (Д_i^X - x)\rangle_A$          // database entry − query
5         if $a_i \neq 0$ then                                  // locate matches in the database
6             $|l_i\rangle_{L_i} \mapsto |l_i \oplus 1\rangle_{L_i}$   // save the corresponding positions
7         $|a\rangle_A \mapsto |a - (Д_i^X - x)\rangle_A$          // uncompute register $A$
8 return $|x\rangle_X|Д\rangle_D|l'\rangle_R$                    // $l'$ contains the position of $x$ in $Д$
```

Function 10: Add

```
   Input  : $|x\rangle_X |Д\rangle_D |l\rangle_L$
   Output: $|x\rangle_X |Д'\rangle_D |l'\rangle_L$
1  Set $|a\rangle_A = |0^q \in \{0,1\}^q\rangle_A$          // initialize auxiliary register A
2  for $i = 1, ..., q$ do
3      $|a_i\rangle_{A_i} \mapsto \mathsf{Larger}(|Д_i^X\rangle_{D_i^X} |x\rangle_X |a_i\rangle_{A_i})$   // check if database entry $>$ query
4      if $Д_i^Y \neq 0$ then                              // correct for empty entries
5          $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$
6      for $j = i+1, ..., q$ do                           // flip all higher entries
7          $|a_j\rangle_{A_j} \mapsto |a_j \oplus a_i\rangle_{A_j}$          // so we're left with one position
8  $|Д\rangle_D \mapsto \mathsf{P}^{-1}(|Д\rangle_D \otimes |a\rangle_A)$          // permute D to create empty entry
                                                          // P is defined in (110)
9  for $i = 1, ..., q$ do
10     if $a_i = 1$ then                                  // look for this empty entry
11         $|Д_i^X\rangle_{D_i^X} \mapsto |Д_i^X - x\rangle_{D_i^X}$          // add $x-$entry to the database
12         $|l_i\rangle_{L_i} \mapsto |l_i \oplus 1\rangle_{L_i}$              // update location register
13 if $x \neq 0$ then                                     // (Non zero $x$ implies non zero $a$
14     for $i = 1, ..., q$ do
15         if $l_i = 1$ then                              // if located
16             $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$          // uncompute register $A$
17 return $|x\rangle_X |Д'\rangle_D |l'\rangle_L$              // $Д'$ is the modified database
                                                          // $l'$ is modified $l$
```

Function 11: Upd

```
   Input  : $|\eta\rangle_Y |Д^Y\rangle_{D^Y} |l\rangle_L$
   Output: $|\eta\rangle_Y |Д'^Y\rangle_{D^Y} |l\rangle_L$
1  Apply $\mathsf{QFT}_N^{D^Y} \mathsf{Samp}_{\mathfrak{D}}$          // transform to the Fourier basis
2  for $i = 1, ..., q$ do
3      if $l_i = 1$ then                                  // if located
4          $|\Delta_i^Y\rangle_{D_i^Y} \mapsto |\Delta_i^Y - \eta\rangle_{D_i^Y}$   // Update the Y register of entry
5  Apply $\mathsf{Samp}_{\mathfrak{D}}^\dagger \mathsf{QFT}_N^{\dagger D^Y}$     // transform back to the unprepared database
6  return $|\eta\rangle_Y |Д'^Y\rangle_{D^Y} |l\rangle_L$   // $Д'^Y$ is modified $Y$ register of the database
```

Function 12: Rem

```
Input  : $|x\rangle_X |Д\rangle_D |l\rangle_L$
Output: $|x\rangle_X |Д'\rangle_D |l'\rangle_L$
Set $|a\rangle_A = |0^q\rangle_A$                                   // initialize auxiliary register $A$
Set $|b\rangle_B = |0\rangle_B$                                     // initialize auxiliary register $B$
for $i = 1, ..., q$ do
    if $l_i = 1$ then
        if $и_i = 0$ then                                               // if entry is incorrect
            $|Д_i^X\rangle_{D_i^X} \mapsto |Д_i^X - x\rangle_{D_i^X}$                          // remove the entry
            $|b\rangle_B \mapsto |b \oplus 1\rangle_B$                    // save that we have removed an entry
if $b = 1$ then                                                         // if we removed an entry
    for $i = 1, ..., q$ do
        $|a_i\rangle_{A_i} \mapsto$ Larger$(|x\rangle_X, |Д_i^X\rangle_{D_i^X}, |a_i\rangle_{A_i})$    // check if query $>$ database entry
        if $x = 0$ then                                                   // Correct for x = 0
            if $Д_i^Y \neq 0$ then                                      // correct for empty entries
                $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$
        for $j = i - 1, ..., 1$ do                                      // flip all lower entries
            $|a_j\rangle_{A_j} \mapsto |a_j \oplus a_i\rangle_{A_j}$   // so we're left with only the removed position
        $|l_i\rangle_{L_i} \mapsto |l_i \oplus a_i\rangle_{L_i}$                    // correct for the removed entry
    $|Д\rangle_D \mapsto P(|Д\rangle_D \otimes |a\rangle_A)$                 // permute D to move the empty entry
    for $i = q, ..., 1$ do                                                // uncompute register A
        for $j = q, ..., i + 1$ do                           // by calculating the first position
            $|a_j\rangle_{A_j} \mapsto |a_j \oplus a_i\rangle_{A_j}$            // such that database entry > query
        if $Д_i^Y \neq 0$ then                                          // as in the Add subroutine
            $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$
        $|a_i\rangle_{A_i} \mapsto$ Larger$(|Д_i^X\rangle_{D_i^X}, |x\rangle_X, |a_i\rangle_{A_i})$
$|a\rangle_A \mapsto$ Locate$(|x\rangle_X |Д\rangle_D |l\rangle_A)$
if $A = 0^q$ then                                                     // check if we have removed
    $|b\rangle_B \mapsto |b \oplus 1\rangle_B$                                      // Uncompute register $B$
$|a\rangle_A \mapsto$ Locate$(|x\rangle_X |Д\rangle_D |l\rangle_A)$                  // uncompute register $A$
return $|x\rangle_X |Д'\rangle_D |l'\rangle_L$                                 // $Д'$ is modified database
                                                                      // $l'$ is modified $l$
```

```
Function 13: Clean
   Input  : $|\eta\rangle_Y|Д^Y\rangle_D|l\rangle_L|a\rangle_A$
   Output: $|\eta\rangle_Y|Д^Y\rangle_D|l\rangle_L|a'\rangle_A$
1  Set $|b\rangle_B = |0 \in \mathcal{Y}\rangle_B$            // initialize auxiliary register $B$
2  Apply $\mathsf{QFT}_N^{D^Y}\,\mathsf{Samp}_{\mathfrak{D}}$           // transform to the Fourier basis
3  for $i = 1, ..., q$ do
4      if $l_i = 1$ then
5          $|b\rangle_B \mapsto |b + (\Delta_i^Y - \eta)\rangle_B$            // database entry − query
6          if $b = 0$ then                      // locate matches in the database
7              if $\eta \neq 0$ then                               // if we added
8                  $|a\rangle_A \to |a \oplus 1\rangle_A$
9          $|b\rangle_B \mapsto |b - (\Delta_i^Y - \eta)\rangle_B$            // uncompute register $B$
10 Apply $\mathsf{Samp}_{\mathfrak{D}}^{\dagger}\mathsf{QFT}_N^{\dagger D^Y}$     // transform back to the unprepared database
11 return $|\eta\rangle_Y|Д^Y\rangle_D|l\rangle_L|a'\rangle_A$               // $a'$ is modified register $A$
```

A.3 Implementation of V for $\mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}$

The "spreading" unitary is defined as follows:

$$\mathsf{V}^{AB}|a\rangle_A|b\rangle_B = \begin{cases} |a\rangle_A|b\rangle_B & \text{if } a > b \\ |a\rangle_A|b+1 \mod N\rangle_B & \text{if } a \le b \end{cases}. \tag{111}$$

The operation V as stated above is defined only on $a \in [N]$ and $b \in [N-1]$ (counted from 0), to extend it to the full domain we need to add that $\mathsf{V}|a\rangle_A|N-1\rangle_B := |a\rangle_A|N-1\rangle_B$. Action of V is shifting the states in register B in a way that possible states skip a. Let us show that V is in fact a unitary that is easy to construct. For that we need three unitary sub-routines,

$$\mathsf{V}_{\le}|a\rangle_A|b\rangle_B = \begin{cases} |a\rangle_A|b\rangle_B|0\rangle_C & \text{if } a > b \\ |a\rangle_A|b\rangle_B|1\rangle_C & \text{if } a \le b \end{cases}, \tag{112}$$

$$\mathsf{V}_{+}|a\rangle_A|b\rangle_B|c\rangle_C = |a\rangle_A|b + c \mod N\rangle_B|c\rangle_C, \tag{113}$$

$$\mathsf{V}_{-\le}|a\rangle_A|b\rangle_B|c\rangle_C = \begin{cases} |a\rangle_A|b\rangle_B|c \oplus 0\rangle_C & \text{if } a > b-1 \mod N \\ |a\rangle_A|b\rangle_B|c \oplus 1\rangle_C & \text{if } a \le b-1 \mod N \end{cases}, \tag{114}$$

where we additionally need to specify that $\mathsf{V}_{-\le}|a > 0\rangle_A|0\rangle_B|c\rangle = |a > 0\rangle_A|0\rangle_B|c\rangle$. Now we define $\mathsf{V} = \mathsf{V}_{-\le}\mathsf{V}_{+}\mathsf{V}_{\le}$, we also discard register C after the three operations. In this approach we need a register holding a description of the set $\mathcal{S}$ (that would be registers A) but as long as we do, applying V to all registers describing $\mathcal{S}$ will give us the expected result.

A.4 $\mathsf{Samp}_{\mathfrak{D}}$ for Random Boolean Functions

Let us say we want to efficiently simulate a quantum oracle oracle for a random function $h : \{0,1\}^m \to \{0,1\}$, such that $h(x) = 1$ with probability λ. Then the adding function of the corresponding compressed oracle is $\forall x \in \{0,1\}^m$:

$$\mathsf{Samp}_{\lambda}(x) := \begin{pmatrix} \sqrt{1-\lambda} & \sqrt{\lambda} \\ \sqrt{\lambda} & -\sqrt{1-\lambda} \end{pmatrix}, \tag{115}$$

independent from any previous queries. This might come useful in tasks like search in a sparse database.

B Additional Details on O2H Lemma for Compressed Oracles

We state a lemma giving a bound on the probability of Find is conditionally uniform distributions. Below we write $\mathsf{CStO}_{\mathcal{Y}\setminus\mathcal{S}}$ to denote the compressed oracle sampling from a conditionally uniform distribution, note that $\mathcal{S}$ can depend on the previous queries. Let us first properly define the relation stated in Lemma 21 below:

$$R_{\mathcal{D}} := \left\{(x, y) \in \mathcal{X} \times \mathcal{Y} \setminus \mathcal{S} : x \notin D^X, y \in \mathcal{D}\right\}, \tag{116}$$

by D^X we denote the $\mathcal{X}$ part of the entries of the database D prior to the last oracle query. In what follows we refer to this relation as simply $\mathcal{D}$ and by that mean that the database is in relation if the new output of the new query is in the set $\mathcal{D} \subseteq \mathcal{Y}$.

Lemma 21. *For any quantum adversary* A *interacting with a punctured oracle* $\mathsf{CStO}_{\mathcal{Y}\setminus\mathcal{S}} \setminus \mathcal{D}$*—where* $\mathsf{CStO}_{\mathcal{Y}\setminus\mathcal{S}}$ *is an oracle with the sampling operation defined in Eq.* (36), $\mathcal{S}$ *is some subset of* $\mathcal{Y}$, *and* $\mathcal{D}$ *is a subset of outputs to previous queries together with* $\mathcal{Z} \subseteq \mathcal{Y} \setminus \mathcal{S}$*—the probability of* Find *is bounded by:*

$$\mathbb{P}[\mathsf{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}\setminus\mathcal{S}} \setminus \mathcal{D}]] \leq \frac{q(q-1) + 2q\,|\mathcal{Z}|}{2\,|\mathcal{Y} \setminus \mathcal{S}|}, \tag{117}$$

where q is the number of queries made by A.

Proof. $\mathcal{D}$ defines a relation $R_{\mathcal{D}}$ as shown in Eq. (116). The crucial aspect is how we define the set $\mathcal{D}$: As noted in the statement of the lemma, the set consists of y-values stored in D^Y before the query: $\mathcal{D} \subseteq D^Y$. Throughout the proof we just write $\mathsf{J}_{\mathcal{D}}$ when talking about measuring the relation, but the full description of the measurement involves preparing an auxiliary register storing a description of $\mathcal{D}$ and then conditioned on this register measuring the database.

Let us start with the analysis of the probability of Find, following the Eq. (43). As described above, $\mathsf{J}_{\mathcal{D}}$ denotes the projector measuring the database for elements in $\mathcal{D}$. In the calculation below we write H instead of $\mathsf{CStO}_{\mathcal{Y}\setminus\mathcal{S}}$ to have more concise formulas:

$$\mathbb{P}[\mathsf{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}\setminus\mathcal{S}} \setminus \mathcal{D}]] = 1 - \left\| \prod_{i=q}^{1} (\mathbb{1} - \mathsf{J}_{\mathcal{D}}) \mathsf{U}_i^{\mathsf{H}} |\Psi_0\rangle \right\|^2 \tag{118}$$

$$= 1 - \left\| \prod_{i=q-1}^{1} (\mathbb{1} - \mathsf{J}_{\mathcal{D}}) \mathsf{U}_j^{\mathsf{H}} |\Psi_0\rangle \right\|^2 + \left\| \mathsf{J}_{\mathcal{D}} \mathsf{U}_q^{\mathsf{H}} \prod_{i=q-1}^{1} (\mathbb{1} - \mathsf{J}_{\mathcal{D}}) \mathsf{U}_j^{\mathsf{H}} |\Psi_0\rangle \right\|^2 \tag{119}$$

$$= \sum_{i=1}^{q} \left\| \mathsf{J}_{\mathcal{D}} \mathsf{U}_i^{\mathsf{H}} \prod_{j=i-1}^{1} (\mathbb{1} - \mathsf{J}_{\mathcal{D}}) \mathsf{U}_j^{\mathsf{H}} |\Psi_0\rangle \right\|^2 \tag{120}$$

$$= \sum_{i=1}^{q} \left\| \underbrace{\mathsf{J}_{\mathcal{D}} \mathsf{U}_i^{\mathsf{H}} \frac{1}{\left\| \prod_j (\mathbb{1} - \mathsf{J}_{\mathcal{D}}) \mathsf{U}_j^{\mathsf{H}} |\Psi_0\rangle \right\|} \prod_{j=i-1}^{1} (\mathbb{1} - \mathsf{J}_{\mathcal{D}}) \mathsf{U}_j^{\mathsf{H}} |\Psi_0\rangle}_{:=|\Phi_i\rangle} \right\|^2 \left\| \prod_{j=i-1}^{1} (\mathbb{1} - \mathsf{J}_{\mathcal{D}}) \mathsf{U}_j^{\mathsf{H}} |\Psi_0\rangle \right\|^2 \tag{121}$$

$$\leq \sum_{i=1}^{q} \left\| \mathsf{J}_{\mathcal{D}} |\Phi_i\rangle \right\|^2. \tag{122}$$

Here, the second equation follows from the fact that $\||v\rangle\|^2 = \|\Pi|v\rangle\|^2 + \|(\mathbb{1} - \Pi)|v\rangle\|^2$ for all $|v\rangle$ and projectors Π.

Hence, we have that the probability of Find occurring is upper-bounded by the sum of the probabilities of finding the relation after query i conditioned on not finding the relation in all previous queries. Now let us inspect the state just after the query, conditioned on ¬Find. Let us focus on a single query state in the adversary's full state $|\Phi_i\rangle = \sum_{x,y,w} \beta_{x,y,w} \, |\Phi_i(x,y)\rangle \, |w\rangle_W$, where W is the adversary's work register. By $\mathcal{P}$ we denote the set of x-values in entries prior to the query, the basis state is then:

$$|\Phi_i(x,y)\rangle := |x,y\rangle_{XY} \Bigg(\mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}(\mathcal{P}\cup\{x\}) \sum_{\text{и}\neq 0} \alpha(x,y,\text{и}) \, |\text{Д}'_{\text{ADD/UPD}}(\text{и})\rangle_D \\ + \mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}(\mathcal{P}) \alpha(x,y,0) \, |\text{Д}'_{\text{REM/NOT}}(0)\rangle_D \Bigg), \tag{123}$$

where by $\text{Д}'_{\text{ADD/UPD}}$ we denote the database with entry x being added or updated. By $\text{Д}'_{\text{REM/NOT}}$ we denote the database where x entry has been removed or nothing happened . The function $\alpha(\cdot)$ denotes amplitudes of the corresponding databases.

We use the unprepared basis as it provides the clearest picture with respect to the possible action of the oracle. Applying $\mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}$ transforms the database to the basis compatible with the relation.

To calculate the bound from Eq. (122), note that $\text{Д}'_{\text{REM/NOT}}$ did not have an element in $\mathcal{D}$ before the query and so there is no chance it has one now, as the database has either shrunk or not changed. Hence $\left\| \mathsf{J}_{\mathcal{D}} |D'_{\text{REM/NOT}}\rangle_D \right\| = 0$. Similarly with updating the database, for the relation to hold, a query has to increase the number of non-padding entries in Д. To highlight this fact in what follows we omit the subscript UPD. The modified database on the other hand increases by a new entry. For any $\text{и} \neq 0$ we have:

$$\mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}(\mathcal{P}\cup\{x\}) \, |\text{Д}'_{\text{ADD}}\rangle_D = \mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}(\mathcal{P}) |\text{Д}\rangle_{D\setminus\{x\}} \otimes \mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}(x \mid \mathcal{P}) |x,\text{и}\rangle_{D(x)} \tag{124}$$

$$= |D\rangle_{D\setminus\{x\}} \otimes \sum_{z\in\mathcal{Y}\setminus\mathcal{S}} \frac{1}{\sqrt{|\mathcal{Y}\setminus\mathcal{S}|}} \, \bar{\omega}_{|\mathcal{Y}\setminus\mathcal{S}|}^{z'\cdot\text{и}} \, |x,z\rangle_{D(x)}, \tag{125}$$

where by $\setminus\{x\}$ we denote omitting the register containing x in the database registers (note however that this notation is only symbolic as the database is always kept in a sorted way), by writing D instead of Д we mark the basis change, and by z' we denote the fact that there might be some discrepancy between z and the power of $\bar{\omega}_{|\mathcal{Y}\setminus\mathcal{S}|}^{\text{и}}$ coming from the action of V from Eq.(35).

From this calculation we can bound the probability of finding an element in $\mathcal{D}$. In the following, by putting a prime over α: $\alpha'(x,y,\text{и})$, we signify the fact that amplitudes corresponding to database Д_{UPD} are set to zero by $\mathsf{J}_{\mathcal{D}}$,

$$\|\mathsf{J}_{\mathcal{D}}|\Phi_i\rangle\|^2 = \left\| \mathsf{J}_{\mathcal{D}} \sum_{x,y,w} \beta_{x,y,w} |\Phi_i(x,y)\rangle |w\rangle_W \right\|^2 \tag{126}$$

$$= \left\| \mathsf{J}_{\mathcal{D}} \sum_{x,y,w} \beta_{x,y,w} |x,y\rangle_{XY} \sum_{\text{и}\neq 0} \alpha'(x,y,\text{и}) \mathsf{Samp}_{\mathcal{Y}\setminus\mathcal{S}}(\mathcal{P}\cup\{x\}) |\text{Д}'_{\text{ADD}}\rangle_D |w\rangle_W \right\|^2 \tag{127}$$

$$= \sum_{x,y,w} |\beta_{x,y,w}|^2 \left\| \mathsf{J}_{\mathcal{D}} \sum_{\text{и}\neq 0} \alpha'(x,y,\text{и}) |D\rangle_{D\setminus\{x\}} \otimes \sum_{z\in\mathcal{Y}\setminus\mathcal{S}} \frac{1}{\sqrt{|\mathcal{Y}\setminus\mathcal{S}|}} \bar{\omega}_{|\mathcal{Y}\setminus\mathcal{S}|}^{z'\cdot\text{и}} |x,z\rangle_{D(x)} \right\|^2 \tag{128}$$

$$= \sum_{x,y,w} |\beta_{x,y,w}|^2 \left\| \sum_{\text{и}\neq 0} \alpha'(x,y,\text{и}) \sum_{z\in\mathcal{D}} \frac{1}{\sqrt{|\mathcal{Y}\setminus\mathcal{S}|}} \bar{\omega}_{|\mathcal{Y}\setminus\mathcal{S}|}^{z'\cdot\text{и}} |z\rangle_{D^Y(x)} \right\|^2 \tag{129}$$

$$\leq \sum_{x,y,w} |\beta_{x,y,w}|^2 \sum_{\text{и}\neq 0} |\alpha'(x,y,\text{и})|^2 \sum_{z\in\mathcal{D}} \frac{1}{|\mathcal{Y}\setminus\mathcal{S}|} \tag{130}$$

$$\leq \sum_{z \in \mathcal{D}} \frac{1}{|\mathcal{Y} \setminus \mathcal{S}|} \leq \frac{i-1+|\mathcal{Z}|}{|\mathcal{Y} \setminus \mathcal{S}|}. \tag{131}$$

As we stated in the beginning the set $\mathcal{D}$ consists of y-values in D^Y prior to $\mathsf{U}_i^{\mathsf{H}}$ and $\mathcal{Z}$, hence its cardinality is at most $i-1+|\mathcal{Z}|$. Note that $\mathsf{J}_{\mathcal{D}}$ collapsed only the x-entry in the updated database, that is because the discussed relation is defined on the new entry relative to previous entries.

The sum $\sum_{i=1}^{q} \frac{i-1+|\mathcal{Z}|}{|\mathcal{Y} \setminus \mathcal{S}|}$ evaluates to $\frac{q(q-1)+2q|\mathcal{Z}|}{2|\mathcal{Y} \setminus \mathcal{S}|}$ as claimed. □

Note that if we omit previous outputs in $\mathcal{D}$, then we recover a classical bound on finding an element of the preimage of $\mathcal{Z}$.

Corollary 22. *For any quantum adversary* A *interacting with a punctured oracle* $\mathsf{CStO}_{\mathcal{Y} \setminus \mathcal{S}} \setminus \mathcal{Z}$*—where* $\mathcal{Z} \subseteq \mathcal{Y} \setminus \mathcal{S}$*—the probability of* Find *is bounded by:*

$$\mathbb{P}[\mathsf{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y} \setminus \mathcal{S}} \setminus \mathcal{Z}]] \leq \frac{q\,|\mathcal{Z}|}{|\mathcal{Y} \setminus \mathcal{S}|}, \tag{132}$$

where q is the number of queries made by A.

Proof. Follows from Lemma 21 without puncturing on outputs of previous queries. □

C Quantum Indifferentiability of Sponges with Random Permutations

In the beginning of this section we focus on lazy sampling of a random permutation. Specifically for permutations $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$, sampling done in two stages, first sampling the inner part of the output and then the outer part.

By $\mathcal{D}$ we denote the set of outputs of previous queries. In the language of the sponge graph $\mathcal{D}^{-1}$ is the set of nodes with outgoing edges and $\mathcal{D}$ is the set of nodes with incoming edges. By $\widehat{\mathcal{D}}$ we denote the set of supernodes with all nodes having an incoming edge. By $\overline{\mathcal{D}}(\hat{t})$ we denote the set of nodes in the supernode $\hat{t}$ with an incoming edge.

We need to define a procedure to lazy-sample outputs of a random permutation. The obvious solution of sampling uniformly from $\mathcal{A} \times \mathcal{C} \setminus \mathcal{D}$ is not good enough as we want to sample the inner part before the outer part and retain the step-by-step structure of our proof, similarly to the proof of Thm. 13.

Classically we are going to first sample uniformly from $\mathcal{A} \times \mathcal{C} \setminus \mathcal{D}$ but then discard the outer state. The value of the inner state $\hat{t}$ is then effectively sampled from $\mathcal{C}$ with weights $\frac{|\mathcal{A} \setminus \overline{\mathcal{D}}(\hat{t})|}{|\mathcal{A} \times \mathcal{C} \setminus \mathcal{D}|}$. We call this distribution $\mathfrak{C}$. At this point we will be introducing bad events concerning the inner part of the sampled state. To sample the outer state we just sample uniformly from $\mathcal{A} \setminus \overline{\mathcal{D}}(\hat{t})$. We denote this distribution by $\mathfrak{A}(\hat{t})$.

Quantumlly the situation is a bit more involved, so we are going to present the sampling procedure in more detail. First we sample pairs from $\mathcal{A} \times \mathcal{C} \setminus \mathcal{D}$ using $\mathsf{Samp}_{\mathcal{A} \times \mathcal{C} \setminus \mathcal{D}}$, defined similarly to the procedure in Eq.(36). Then we un-sample the outer part, by applying the Hermitian conjugate of $\mathsf{Samp}_{\mathcal{A} \setminus \overline{\mathcal{D}}(\hat{t})}$ to the resulting state. Note that we control the un-sample operation on the inner-part of the initial sample. We start from the following state

$$\mathsf{Samp}_{\mathcal{A} \times \mathcal{C} \setminus \mathcal{D}}|0\rangle = \sum_{t \in \mathcal{A} \times \mathcal{C} \setminus \mathcal{D}} \frac{1}{\sqrt{|\mathcal{A} \times \mathcal{C} \setminus \mathcal{D}|}}|t\rangle = \sum_{\hat{t}} \sqrt{\frac{\left|\mathcal{A} \setminus \overline{\mathcal{D}}(\hat{t})\right|}{|\mathcal{A} \times \mathcal{C} \setminus \mathcal{D}|}} \sum_{\bar{t} \in \mathcal{A} \setminus \overline{\mathcal{D}}(\hat{t})} \frac{1}{\sqrt{\left|\mathcal{A} \setminus \overline{\mathcal{D}}(\hat{t})\right|}}|\bar{t}, \hat{t}\rangle, \tag{133}$$

where the right hand side of the equation follows by just rearranging the sums and noticing that given some $\hat{t} \in \mathcal{C}$ we have $\bar{t} \in \mathcal{A}$ that have no incoming edges in the supernode $\hat{t}$. Now the definition of the second sampling procedure reads

$$\mathsf{Samp}_{\mathcal{A}\setminus\overline{\mathcal{D}}(\hat{t})}|0, \hat{t}\rangle = \sum_{\bar{t}\in\mathcal{A}\setminus\overline{\mathcal{D}}(\hat{t})} \frac{1}{\sqrt{\left|\mathcal{A} \setminus \overline{\mathcal{D}}(\hat{t})\right|}}|\bar{t}, \hat{t}\rangle \tag{134}$$

and is completed to a unitary acting on every other $\bar{s} \in \mathcal{A}$ under the constraints of Eq.(14). Note that we use the second register to control the unitary (by providing the set $\overline{D}(\hat{t})$ we exclude from $\mathcal{A}$). By applying $\mathsf{Samp}^{\dagger}_{\mathcal{A}\setminus\bar{\mathcal{D}}(\hat{t})}$ to both sides of Eq.(134) we see that we can un-compute the outer part $\bar{t}$ from the initial superposition from Eq.(133) and sample $\hat{t}$ with probability $\frac{|\mathcal{A}\setminus\overline{\mathcal{D}}(\hat{t})|}{|\mathcal{A}\times\mathcal{C}\setminus\mathcal{D}|}$. The sampling procedure $\mathsf{Samp}^{\dagger}_{\mathcal{A}\setminus\overline{\mathcal{D}}(\hat{t})} \circ \mathsf{Samp}_{\mathcal{A}\times\mathcal{C}\setminus\mathcal{D}}$ is used to define $\mathsf{CStO}_{\mathfrak{C}}$. As in the classical case in simulators after S_2 will have this part modified. The outer part is then sampled using $\mathsf{Samp}_{\mathcal{A}\setminus\overline{\mathcal{D}}(\hat{t})}$, with which we define $\mathsf{CStO}_{\mathfrak{A}(\hat{t})}$.

In the following we denote sampling from the distribution $\mathfrak{C}$ but with the set $\mathcal{C}$ changed to $\mathcal{C} \setminus (\mathcal{R} \cup \mathcal{U})$ by $\mathfrak{C} \setminus (\mathcal{R} \cup \mathcal{U})$. In the case of the inverse of the random permutation we use a similar distribution but in the above definitions we take $\mathcal{D}^{-1}$—i.e. the set of nodes with outgoing edges—in both $\mathfrak{C}$ and $\mathfrak{A}$. We denote those distributions by $\mathfrak{C}^{-1}$ and $\mathfrak{A}^{-1}(\hat{t})$ respectively.

The bound on $\mathbb{P}[\mathrm{Bad}]$ in our proof for random permutations is the same as in the case of random functions. We achieve a slightly weaker bound than in [Ber+08] because in our gradual argument we first sample from the set of inner states excluding only "full" supernodes $\widehat{\mathcal{D}}$. We can bound the size of this set by $\left|\widehat{\mathcal{D}}\right| \geq 0$ and following the same strategy as for derivation of Eq. (50) achieve a bound of f_{coll}.

C.1 Classical Indifferentiability of Sponges with Random Permutations

First we will present a slightly modified proof of indifferentiability from [Ber+08]. We modify the proof to better fit the framework of game-playing proofs.

Theorem 23 (Sponge with permutations, classical indifferentiability). *$\textsc{Sponge}_{\varphi}[\textsc{pad}, \mathcal{A}, \mathcal{C}]$ calling a random permutation φ is (q, ε)-indifferentiable from a random oracle, Eq.* (51)*, for* classical *adversaries for any $q < |\mathcal{C}|$ and $\varepsilon = 10\frac{q(q+1)}{2|\mathcal{C}|}$.*

Proof. The proof is constructed in a similar way to the proof of Thm. 13. We will introduce more steps in the proof though, as we want to keep a slow paste of statements so that each transition between games is clear.

In Alg. 14 we present the indifferentiability simulators for the case of random permutations.

Game 1 We start with the real world where the distinguisher A has access to a random permutation $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$ and the construction $\textsc{Sponge}_{\varphi}$ using this function. The formal definition of the first game is

$$\textbf{Game 1} := \left(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\varphi}, (\varphi, \varphi^{-1})]\right). \tag{135}$$

Game 2 In the second game we introduce the simulator S_2—defined in Alg. 14—that lazy-samples the random permutation φ. In Alg 14 we define all simulators of this proof at once, but note that all conditional statements lead to the same behavior of S_2. The definition of the second game is

$$\textbf{Game 2} := \left(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_2}, (\mathsf{S}_2, \mathsf{S}_2^{-1})]\right), \tag{136}$$

```
Algorithm 14: Classical S_2, \boxed{S_{3a}}, \boxed{S_{3b}}, \boxed{S_4}, \boxed{S_{6a}}, \boxed{S_{6b}}, permutations
State      : Current sponge graph $G$

Interface: $\varphi$, forward queries
Input      : $s \in \mathcal{A} \times \mathcal{C}$
Output    : $\varphi(s)$

if $s$ has no outgoing edge then                                    // New query
    if $\hat{s} \in \mathcal{R} \wedge \mathcal{R} \cup \mathcal{U} \neq \mathcal{C}$ then        // $\hat{s}$-rooted, no saturation
        $\hat{t} \leftarrow \mathfrak{C}$, $\boxed{\boxed{\boxed{\text{if } \hat{t} \in \mathcal{R} \cup \mathcal{U}\text{, set Bad}_1 = 1}}}$, $\boxed{\boxed{\hat{t} \leftarrow \mathfrak{C} \setminus (\mathcal{R} \cup \mathcal{U})}}$
        Construct a path to $s$: $p := \mathsf{SpPath}(s, G)$
        if $\exists x : p = \textsc{pad}(x)$ then
            $\bar{t} \leftarrow \mathfrak{A}(\hat{t})$
            $\boxed{\boxed{\boxed{\bar{t} := H(x)}}}$
        else
            $\bar{t} \leftarrow \mathfrak{A}(\hat{t})$
        $t := (\bar{t}, \hat{t})$
    else
        $t \xleftarrow{\$} (\mathcal{A} \times \mathcal{C}) \setminus \mathcal{D}$
    Add an edge $(s, t)$ to $\mathcal{E}$.
Set $t$ to the vertex at the end of the edge starting at $s$
Output $t$
```

```
Interface: $\varphi^{-1}$, backward queries
Input      : $s \in \mathcal{A} \times \mathcal{C}$
Output    : $\varphi^{-1}(s)$

Construct the sponge graph $G$
if $s$ has no incoming edge then                                    // New query
    $\hat{t} \leftarrow \mathfrak{C}^{-1}$, $\boxed{\boxed{\boxed{\text{if } \hat{t} \in \mathcal{R}\text{, set Bad}_2 = 1}}}$, $\boxed{\hat{t} \leftarrow \mathfrak{C}^{-1} \setminus \mathcal{R}}$
    $\bar{t} \leftarrow \mathfrak{A}^{-1}(\hat{t})$
    $t := (\bar{t}, \hat{t})$
    Add an edge $(t, s)$ to $\mathcal{E}$.
Set $t$ to the vertex at the beginning of the edge ending at $s$
Output $t$
```

where by S^{-1} we denote the backward interface of S. Because the simulator S_2 perfectly models a random permutation and we use the same function for the private interface we have

$$|\mathbb{P}[\textbf{Game 2}] - \mathbb{P}[\textbf{Game 1}]| = 0. \tag{137}$$

Game 3a In the next step we modify S_2 to S_{3a}. The game is then

$$\textbf{Game 3a} := \left(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3a}}, (\mathsf{S}_{3a}, \mathsf{S}_{3a}^{-1})]\right). \tag{138}$$

We made a single change in S_{3a} compared to S_2, we introduce the "bad" event Bad_1 that marks the difference between algorithms. We use this event as the bad event in Lemma 1. With such a change of the simulators we can use Lemma 1 to bound the difference of probabilities:

$$|\mathbb{P}[\textbf{Game 3a}] - \mathbb{P}[\textbf{Game 2}]| \leq \mathbb{P}[\mathrm{Bad}_1 = 1]. \tag{139}$$

It is now quite easy to bound $\mathbb{P}[\mathrm{Bad}_1 = 1]$ as this is the probability of finding a collision or a preimage of the root in the set $\mathcal{C}$ having made q random samples. Then we have that

$$\mathbb{P}[\mathrm{Bad}_1 = 1] \leq f_{\mathrm{coll}}(q) \tag{140}$$

where the inequality comes from the fact that not all queries are made to rooted nodes.

Game 3b In the next step we modify S_{3a} to S_{3b}. The game is then

$$\textbf{Game 3b} := \left(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3b}}, (\mathsf{S}_{3b}, \mathsf{S}_{3b}^{-1})]\right). \tag{141}$$

We made a single change in S_{3b} compared to S_{3a}, we introduce the "bad" event Bad_2 that marks the difference between algorithms. We use this event as the bad event in Lemma 1. With such a change of the simulators we can use Lemma 1 to bound the difference of probabilities:

$$|\mathbb{P}[\textbf{Game 3b}] - \mathbb{P}[\textbf{Game 3a}]| \leq \mathbb{P}[\mathrm{Bad}_2 = 1]. \tag{142}$$

We bound bound $\mathbb{P}[\mathrm{Bad}_2 = 1]$ similarly as before as again this is the probability of finding a collision or a preimage of the root in the set $\mathcal{C}$ having made q random samples. Then we have that

$$\mathbb{P}[\mathrm{Bad}_2 = 1] \leq f_{\mathrm{coll}}(q) \tag{143}$$

Game 4 In this step we introduce the random oracle H but only to generate the outer part of the output of φ. The game is defined as

$$\textbf{Game 4} := \left(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, (\mathsf{S}_4^{\mathsf{H}}, \mathsf{S}_4^{-1})]\right). \tag{144}$$

Now we need to observe that if $\mathrm{Bad}_1 = 0$ the outputs are identically distributed. Following Claim 14 and the derivation of Eq.(61) we get

$$|\mathbb{P}[\textbf{Game 4}] - \mathbb{P}[\textbf{Game 3b}]| \leq 2\mathbb{P}[\mathrm{Bad}_1 = 1] \leq 2f_{\mathrm{coll}}(q). \tag{145}$$

Game 5 In this stage of the proof we change the private interface to contain the actual random oracle. The simulator is the same and the game is

$$\textbf{Game 5} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, (\mathsf{S}_4^{\mathsf{H}}, \mathsf{S}_4^{-1})]\right). \tag{146}$$

Conditioned on $\mathrm{Bad}_1 = 0$, the outputs of the simulator in Games 4 and 5 are the same and consistent with H. To calculate the adversary's advantage in distinguishing between the two games we can follow the proof of Lemma 12. We change $\mathsf{H} \setminus R_1$ to **Game 5**, $\mathsf{G} \setminus R_2$ to **Game 4**, and event Find to $\mathrm{Bad}_1 = 1$. As the derivation of Lemma 12 uses no quantum mechanical arguments and the assumption holds—the games are identical conditioned on $\mathrm{Bad}_1 = 0$—the bound holds:

$$|\mathbb{P}[\textbf{Game 5}] - \mathbb{P}[\textbf{Game 4}]| \le 4\mathbb{P}[\mathrm{Bad}_1 = 1] \le 4 f_{\mathrm{coll}}(q). \tag{147}$$

Game 6a In this game we use S_{6a}, a simulator that does not check for bad events. The game is

$$\textbf{Game 6a} := \Big(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, (\mathsf{S}_{6a}^{\mathsf{H}}, \mathsf{S}_{6a}^{-1})]\Big) \tag{148}$$

and the advantage is

$$|\mathbb{P}[\textbf{Game 6a}] - \mathbb{P}[\textbf{Game 5}]| \le \mathbb{P}[\mathrm{Bad}_1 = 1] \tag{149}$$

Game 6b In the last game we use S_{6b}, a simulator that does not check for bad events. The game is

$$\textbf{Game 6b} := \Big(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, (\mathsf{S}_{6b}^{\mathsf{H}}, \mathsf{S}_{6b}^{-1})]\Big) \tag{150}$$

and the advantage is

$$|\mathbb{P}[\textbf{Game 6b}] - \mathbb{P}[\textbf{Game 6a}]| \le \mathbb{P}[\mathrm{Bad}_2 = 1]. \tag{151}$$

We included this last game in the proof because S_6 is clearly a simulator that might fail only if G is saturated but this does not happen if $q < |\mathcal{C}|$. Collecting and adding all the differences yields the claimed $\varepsilon = 10 f_{\mathrm{coll}}(q)$.

□

C.2 Quantum Indifferentiability of Sponges with Random Permutations

Theorem 24 (Sponge with permutations, quantum indifferentiability, restated)**.** *$\textsc{Sponge}_{\varphi}[\textsc{pad}, \mathcal{A}, \mathcal{C}]$ calling a random permutation φ is (q, ε)-indifferentiable from a random oracle, Eq.* (51)*, for* quantum *adversaries for any $q < |\mathcal{C}|$ and $\varepsilon = 4\sqrt{(q+1) f_{\mathrm{coll}}(q)} + 12 f_{\mathrm{coll}}$.*

Proof. In the quantum case the operations are modified similarly to the proof of Thm. 15. The main differences are the use of a different distribution, we apply Samp procedures defined in the beginning of this section. In Alg. 15 we describe the simulators we use in this proof.

Game 1 We start with the real world where the distinguisher A has quantum access to a random permutation $\varphi : \mathcal{R} \times \mathcal{C} \to \mathcal{R} \times \mathcal{C}$ and the construction $\textsc{Sponge}_{\varphi}$ using this function. The formal definition of the first game is

$$\textbf{Game 1} := \Big(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\varphi}, (\varphi, \varphi^{-1})]\Big) . \tag{152}$$

Game 2 In the second game we introduce the simulator S_2—defined in Alg. 15—that lazy-samples the random permutation φ. In Alg 15 we define all simulators of this proof at once, but note that all conditional statements lead to the same behavior of S_2. The definition of the second game is

$$\textbf{Game 2} := \Big(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_2}, (\mathsf{S}_2, \mathsf{S}_2^{-1})]\Big) , \tag{153}$$

where by S^{-1} we denote the backward interface of S. Because the simulator S_2 perfectly models a quantum random permutation and we use the same function for the private interface we have

$$|\mathbb{P}[\textbf{Game 2}] - \mathbb{P}[\textbf{Game 1}]| = 0. \tag{154}$$

```
Algorithm 15: Quantum $\boxed{\mathsf{S}_2}$, $\boxed{\mathsf{S}_{3a}}$, $\boxed{\mathsf{S}_{3b}}$, $\boxed{\mathsf{S}_4}$, $\boxed{\mathsf{S}_{6a}}$, $\boxed{\mathsf{S}_{6b}}$, permutations
    State     : Quantum compressed database register $D$
    Interface: $\varphi$, forward queries
    Input     : $|s, v\rangle \in \mathcal{H}_{\mathcal{A}\times\mathcal{C}}^{\otimes 2}$
    Output    : $|s, v + \varphi(s)\rangle$
 1  Locate input $s$ in $\overline{D}$ and $\widehat{D}$                          // Using the correct Samp
 2  Apply $\mathsf{U}_{\mathcal{R}\cup\mathcal{U}}\mathsf{U}_G$ to register $\widehat{D}$ and two fresh registers
 3  if $s \in \mathcal{R} \wedge \mathcal{R}\cup\mathcal{U} \neq \mathcal{C}$ then                 // $s$-rooted, no saturation
 4      Apply $\boxed{\mathsf{CStO}_{\mathfrak{C}}^{X\widehat{Y}\widehat{D}(s)}}$, $\boxed{\boxed{\boxed{(\mathsf{CStO}_{\mathfrak{C}} \setminus (\mathcal{R}\cup\mathcal{U}))^{X\widehat{Y}\widehat{D}(s)}}}}$, $\boxed{\boxed{\mathsf{CStO}_{\mathfrak{C}\setminus(\mathcal{R}\cup\mathcal{U})}^{X\widehat{Y}\widehat{D}(s)}}}$, result: $\hat{t}$
         // Find$_1$ in $\mathsf{S}_{3a}$, $\mathsf{S}_{3b}$, and $\mathsf{S}_4$
 5      Construct a path to $s$: $p := \mathsf{SpPath}(s, G)$
 6      if $\exists x : p = \textsc{pad}(x)$ then
 7          $\boxed{\boxed{\boxed{\text{Apply } \mathsf{CStO}_{\mathfrak{A}(\hat{t})}^{X\overline{Y}\,\overline{D}(s)}}}}$, result: $\bar{t}$
 8          Write $x$ in a fresh register $X_H$, $\boxed{\boxed{\boxed{\text{apply } \mathsf{CH}^{XX_H\overline{Y}\,\overline{D}(s)}}}}$, uncompute $x$ from $X_H$,
             result: $\bar{t}$                                   // Random oracle, Eq. (66)
 9      else
10          Apply $\mathsf{CStO}_{\mathfrak{A}(\hat{t})}^{X\overline{Y}\,\overline{D}(s)}$, result: $\bar{t}$
11      $t := (\bar{t}, \hat{t})$, the value of registers $(\overline{D}^{Y}(s), \widehat{D}^{Y}(s))$
12  else
13      Apply $\mathsf{CPerO}_{\mathcal{A}\times\mathcal{C}}^{XY\overline{D}(s)\widehat{D}(s)}$, result $t$
14  Uncompute $G$ and $\mathcal{R}\cup\mathcal{U}$
15  Output $|s, v + t\rangle$
    ----------------------------------------------------------------
    Interface: $\varphi^{-1}$, backward queries
    Input     : $|s, v\rangle \in \mathcal{H}_{\mathcal{A}\times\mathcal{C}}^{\otimes 2}$
    Output    : $|s, v + \varphi^{-1}(s)_{\mathcal{T}}\rangle$
16  Locate input $s$ in $D$                                        // Using the correct Samp
17  Apply $\mathsf{U}_{\mathcal{R}\cup\mathcal{U}} \circ \mathsf{U}_G$ to registers $D$ and two fresh registers
18  Apply $\boxed{\boxed{\mathsf{CStO}_{\mathfrak{C}^{-1}}^{X\widehat{Y}\widehat{D}(s)}}}$, $\boxed{\boxed{\boxed{(\mathsf{CStO}_{\mathfrak{C}^{-1}} \setminus \mathcal{R})^{X\widehat{Y}\widehat{D}(s)}}}}$, $\boxed{\mathsf{CStO}_{\mathfrak{C}^{-1}\setminus\mathcal{R}}^{X\widehat{Y}\widehat{D}(s)}}$, result $\hat{t}$        // Find$_2$ in
     $\mathsf{S}_{3b}$, $\mathsf{S}_4$, and $\mathsf{S}_{6a}$
19  Apply $\mathsf{CStO}_{\mathfrak{A}^{-1}(\hat{t})}^{X\overline{Y}\,\overline{D}(s)}$, result: $\bar{t}$
20  $t := (\bar{t}, \hat{t})$, the value of registers $(\overline{D}^{Y}(s), \widehat{D}^{Y}(s))$
21  Uncompute $G$ and $\mathcal{R}\cup\mathcal{U}$
22  Output $|s, v + t\rangle$
```

Game 3a In the next step we modify S_2 to S_{3a}. The game is then

$$\textbf{Game 3a} := \Big(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3a}}, (\mathsf{S}_{3a}, \mathsf{S}_{3a}^{-1})]\Big)\,. \tag{155}$$

We made a single change in S_{3a} compared to S_2, we introduce a punctured oracle in the forward interface. With such a change of the simulators we can use Thm. 9 to bound the difference of probabilities:

$$|\mathbb{P}[\textbf{Game 3a}] - \mathbb{P}[\textbf{Game 2}]| \leq 2\sqrt{(q+1)\mathbb{P}[\mathrm{Find}_1 : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3a}}, \mathsf{S}_{3a}]]}, \tag{156}$$

where Find_1 denotes the success of the measurement in the punctured oracle of the forward interface. We can bound $\mathbb{P}[\mathrm{Find}_1 : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3a}}, \mathsf{S}_{3a}]]$ using Lemma 21 with $\mathcal{S} = \widehat{\mathcal{D}}$ and $\mathcal{Z} = \{0\}$. Note that we bound the size of this set by zero: $\left|\widehat{\mathcal{D}}\right| \geq 0$, as discussed in the beginning of this section, giving us the same bound as in the case of random functions. Then we have that

$$\mathbb{P}[\mathrm{Find}_1 : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3a}}, \mathsf{S}_{3a}]] \leq f_{\mathrm{coll}}(q). \tag{157}$$

Game 3b In the next step we modify S_{3a} to S_{3b}. The game is then

$$\textbf{Game 3b} := \Big(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3b}}, (\mathsf{S}_{3b}, \mathsf{S}_{3b}^{-1})]\Big)\,. \tag{158}$$

We made a single change in S_{3b} compared to S_{3a}, we introduce a punctured oracle in the backward simulator. With such a change of the simulators we can use Thm. 9 to bound the difference of probabilities:

$$|\mathbb{P}[\textbf{Game 3b}] - \mathbb{P}[\textbf{Game 3a}]| \leq 2\sqrt{(q+1)\mathbb{P}[\mathrm{Find}_2 : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3b}}, \mathsf{S}_{3b}]]}, \tag{159}$$

where Find_2 denotes the success of the measurement in the punctured oracle of the backward interface. It is now quite easy to bound $\mathbb{P}[\mathrm{Find}_2 : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3a}}, \mathsf{S}_{3a}]]$ as this is the probability of finding a collision in the set $\mathcal{C}$ having made q random samples. Then we have that

$$\mathbb{P}[\mathrm{Find}_2 : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3b}}, \mathsf{S}_{3b}]] \leq f_{\mathrm{coll}}(q). \tag{160}$$

Game 4 In this step we introduce the random oracle H but only to generate the outer part of the output of φ. The game is defined as

$$\textbf{Game 4} := \Big(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, (\mathsf{S}_4^{\mathsf{H}}, \mathsf{S}_4^{-1})]\Big)\,. \tag{161}$$

Thanks to the classical argument, Claim 14 we have that S_4 and S_{3a} are identical until bad, as in Def. 11. Then we can use Lemma 12 to bound the advantage of the adversary

$$|\mathbb{P}[\textbf{Game 4}] - \mathbb{P}[\textbf{Game 3b}]| \leq 4\mathbb{P}[\mathrm{Find}_1 : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_{3b}}, (\mathsf{S}_{3b}, \mathsf{S}_{3b}^{-1})]] \leq 4f_{\mathrm{coll}}(q) \tag{162}$$

Game 5 In this stage of the proof we change the private interface to contain the actual random oracle. The simulator is the same and the game is

$$\textbf{Game 5} := \Big(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, (\mathsf{S}_4^{\mathsf{H}}, \mathsf{S}_4^{-1})]\Big)\,. \tag{163}$$

The advantage is

$$|\mathbb{P}[\textbf{Game 5}] - \mathbb{P}[\textbf{Game 4}]| \leq 4\mathbb{P}[\mathrm{Find}_1 : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, (\mathsf{S}_4^{\mathsf{H}}, \mathsf{S}_4^{-1})]] \leq 4f_{\mathrm{coll}}(q), \tag{164}$$

conditioned on $\neg$Find, outputs of the private interface are the same, then the games are identical until bad and we can use Lemma 12 to bound the advantage of the adversary.

Game 6a In the last game we use S_{6a}, a simulator that uses a non-uniform compressed oracle. The game is

$$\textbf{Game 6a} := \Big(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, (\mathsf{S}_{6a}^{\mathsf{H}}, \mathsf{S}_{6a}^{-1})]\Big) \tag{165}$$

and the advantage is again

$$|\mathbb{P}[\textbf{Game 6a}] - \mathbb{P}[\textbf{Game 5}]| \leq 2\mathbb{P}[\mathrm{Find}_1 : \mathsf{A}[\mathsf{H}, (\mathsf{S}_{6a}^{\mathsf{H}}, \mathsf{S}_{6a}^{-1})]] \leq 2f_{\mathrm{coll}}(q), \tag{166}$$

by Lemma 12 with one relation being $\emptyset$.

Game 6b In the last game we use S_{6b}, a simulator that uses a non-uniform compressed oracle. The game is

$$\textbf{Game 6b} := \Big(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, (\mathsf{S}_{6b}^{\mathsf{H}}, \mathsf{S}_{6b}^{-1})]\Big) \tag{167}$$

and the advantage is again

$$|\mathbb{P}[\textbf{Game 6a}] - \mathbb{P}[\textbf{Game 5}]| \leq 2\mathbb{P}[\mathrm{Find}_2 : \mathsf{A}[\mathsf{H}, (\mathsf{S}_{6b}^{\mathsf{H}}, \mathsf{S}_{6b}^{-1})]] \leq 2f_{\mathrm{coll}}(q), \tag{168}$$

by Lemma 12 with one relation being $\emptyset$.

The last simulator S_{6b} works perfectly until the sponge graph is saturated. Saturation certainly does not occur for $q < |\mathcal{C}|$ as the database in every branch of the superposition increases by at most one in every query. Collecting the differences between games yields the claimed ε. □