Quantum Lazy Sampling and Game-Playing Proofs for Quantum Indifferentiability

Jan Czajkowski*[1], Christian Majenz†[2], Christian Schaffner‡[1], and Sebastian Zur§[2]

[1]QuSoft, University of Amsterdam
[2]QuSoft, CWI

February 21, 2021

Abstract

Game-playing proofs constitute a powerful framework for non-quantum 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 (Crypto'19) can be used to do quantum lazy sampling of a class of non-uniform function distributions. Second, we observe how Unruh's one-way-to-hiding lemma (Eurocrypt'14) 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.

*j.czajkowski@uva.nl
†christian.majenz@gmail.com
‡c.schaffner@uva.nl
§sebastian.zur@cwi.nl

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 simplifies proofs, but also cryptographic constructions designed with the ROM in mind are more efficient.

We are interested in the post-quantum threat model, which is motivated by the present worldwide 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 describe our results and the main proof techniques we used to achieve them.

Our Results. We devise a quantum game-playing framework for security proofs that involve fully quantum adversaries. Our framework is based on a combination of two recently developed proof techniques: compressed quantum random oracles by Zhandry [Zha19] and the One-Way to Hiding (O2H) lemma by Unruh [Unr14; AHU19]. 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. As our first main result we obtain a clean and powerful tool for proofs in post-quantum cryptography. The main advantage of the framework is the fact that it allows the translation of

certain classical security proofs to the quantum setting, in a way that is arguably more straightforward than for previously available proof techniques.

On the technical side, we begin by re-formalizing Zhandry's compressed oracle technique, which, as a by-product, makes a generalization to some non-uniform distributions of oracles relatively straightforward. In particular, we generalize the compressed-oracle technique of [Zha19] to a class of non-uniform distributions over functions, allowing a more general form of (quantum) lazy sampling. Our result allows to treat distributions with outputs that are independent for distinct inputs. Subsequently, we observe that the techniques of "puncturing oracles" proposed in [AHU19] 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.

Let us comment on the generalized compressed oracle for non-uniform distributions. There are already some examples in the literature where such has been used, e.g./ [Ala+20] (superposition oracle without compression that outputs 1 with probability ϵ, we define the sampling procedure for such distribution in Appendix D) and [HM20] (a generalization similar to ours but presented after our paper was posted online). We believe that the generalized formalism developed here will continue to be useful.

Punctured oracles are quantum oracles measured after every adversarial query. An important lemma that we prove is a bound on the probability that any of these measurements succeeds. Our proof of this bound is general enough so that it can be applied with little changes to many different scenarios. Bound on the probability of any of the measurements in a punctured oracle succeeding, together with the O2H lemma for compressed oracles provides a bound on the distinguishing advantage between a regular compressed oracle and a punctured one. In Lemma 9 in [Zha19] indistinguishability of a compressed oracle and a punctured compressed oracle is also proven. The method, however, is different from ours and much fewer details are shown. A crucial difference though is that there are two nontrivial technical claims left implicit. According to [Zha20], however, there is a proof that maintains the claimed bound. As that proof is not publicly available at this point, we state and prove our indistinguishability bound for punctured oracles with almost the same bound. As far as we can tell, our bounds seem tight.

We go on to apply 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 a random function. We leave it as an interesting open question to extend our results to the setting of SHA3 which uses a permutation as internal function. A reader mostly interested in the main result of this paper can go directly to section 5. In the introduction of that section we give a high level explanation of the main concepts used in the proof of quantum indifferentiability.

Related Work. 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 [Zha19] 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 handles the *imperfect case*, 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; AHU19] 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, [Zha19] 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+20]. 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 [Unr19] 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 [Unr19] 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 [Zha19] and [AHU19] have been recently used to show security of the 4-Round Feistel construction in [HI19] and of generic key-encapsulation mechanisms in [JZM19] respectively. In [CEV20] the authors use compressed oracles for randomness in an encryption scheme using a random tweakable permutation (that is given to the algorithm externally). In [Chu+20] the prove quantum query complexity results using the compressed oracles technique and provide a framework that simplifies such tasks.

Note. A previous version of this paper contained an additional set of results about quantum lazy-sampling of random permutations and indifferentiability of SHA-3. Unfortunately there was a flaw in the argument and the technique for quantum lazy sampling random permutations presented there does not work as claimed. The difficulty lies in the fact that that permutations do not have independent outputs, which seems to require a completely different approach.

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 [Zha19] 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 we derive an upper bound on the probability of the Find event for the case of puncturing a uniform oracle on collisions. In Section 5 we use these results to prove quantum indifferentiability of the sponge construction.

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*

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

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 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) $\mathsf{C}^{\text{priv}} = \mathsf{H}_f$. The public interface accepts n-bit long strings and outputs n-bit strings as well. We have that $\mathsf{C}^{\text{pub}} = f$. 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*

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

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

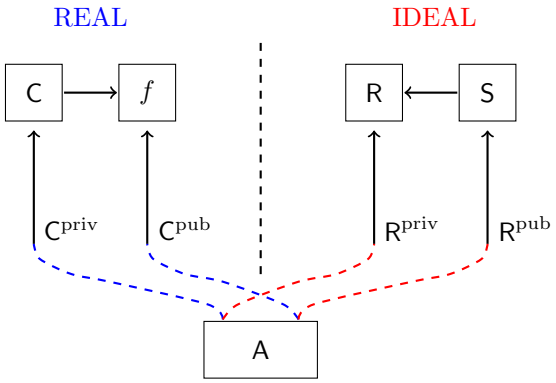

Figure 1: A schematic representation of the notion of indifferentiability, Def. 2. Arrows denote "access to" the pointed system.

It is important to note that if R is the random oracle (which is often the case), then both interfaces are the same. 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* Env[C, A] *and* Env[C′, 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, T[C] *is as secure as* T[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 $M, N > 0$.

In this section we give a formal treatment of quantum accessible oracles. We explain with special care the compressed-oracle technique of Zhandry [Zha19]. A quantum oracle can be viewed as a purification (extension to a higher dimensional Hilbert space) of the adversary's quantum state. The simplest purification extends the state to include a superposition of all full function tables from the set $\mathcal{F}$. Note that the oracle gives access to a random function from the set $\mathcal{F}$. The purification we talk about is called the oracle register. A quantum algorithm could simulate the access to the quantum oracle by preparing the oracle register and performing the correct update procedures every time the adversary makes a query. Such a simulator would not be efficient though, as the oracle register we just defined holds M entries (so one for each element of the domain) of the table of values in $[N]$. The brilliant idea of Zhandry was to propose a procedure to lazy-sample a uniformly random function. By lazy-sampling we mean here to store just the queries asked by the adversary, not the whole function table. By doing that we limit the number of entries held by the simulator to q (the bound on the number of queries performed by the adversary). Our result in this section is generalizing Zhandry's technique to independent distributions on functions: Such that outputs are distributed independently for any distinct inputs.

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 [Zha19] 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 [Zha19] 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 [Unr21], there the case of random (invertible) permutations is also analyzed.

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 [Zha19] is that the Fourier Oracle, instead of adding the 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

[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 $\mathbb{C}^G$.

value otherwise. Is lazy-sampling possible for quantum accessible oracles? Surprisingly, the answer is yes. Thanks to the groundbreaking ideas presented in [Zha19] 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 product 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. In Appendix D, 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 D.2 we recall in detail the compressed oracle introduced in [Zha19], 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 (in the case of full oracles, 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 $\mathsf{FO}_{\mathfrak{D}}$.

The definition of $\mathsf{Samp}_{\mathfrak{D}}$ is such that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X})|0^M\rangle = \sum_{f\in\mathcal{F}} \sqrt{\mathbb{P}[f \leftarrow \mathfrak{D}]}|f\rangle$ and is a unitary operator.

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 a class of non-uniform function distributions. We show that the compressed oracle technique can be used to deal with distributions over functions with outputs independent of any prior interactions. Examples of such functions are random Boolean functions that output one with a given probability.

We want to compress the following oracle

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

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 $\mathcal{S}\subseteq\mathcal{X}$

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

where by $F(x)$ we denote the 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. Also note that since register Y holds a single value in $\mathcal{Y}$ and register F holds values in $\mathcal{Y}^M$, the transform above is an appropriate tensor product of QFT_N. The non-uniform Fourier Oracle is defined as $\mathsf{FO}=\mathsf{QFT}_N^{YF}\circ\mathsf{StO}\circ\mathsf{QFT}_N^{\dagger\,YF}$, as a consequence of that definition we have

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

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 we give all details of $\mathsf{Samp}_{\mathfrak{D}}$ let us discuss the two bases: the Fourier basis and the prepared basis. 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 $\mathsf{QFT}_N^D\circ\mathsf{Samp}_{\mathfrak{D}}^D$ 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 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 Fourier transform: $\mathsf{Samp}_{\mathfrak{U}}=\mathsf{QFT}_N^\dagger$, 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

$$|Д\rangle\xleftrightarrow{\mathsf{Samp}_{\mathfrak{D}}}|D\rangle\xleftrightarrow{\mathsf{QFT}_N^{D^Y}}|\Delta\rangle.\tag{13}$$

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 method for quantum lazy sampling that we generalize in this paper is applicable only to a certain class of distributions. The distributions that we analyze must be independent for every input. By $f(\mathcal{S})$ we denote the part of the full truth table of f corresponding to inputs from $\mathcal{S}$. Below we provide a definition of *product* distributions:

Definition 5 (Product distribution)**.** *A distribution $\mathfrak{D}$ on a set of functions $\mathcal{F} \subseteq \{f : \mathcal{X} \to \mathcal{Y}\}$ is called* product *if for all disjoint $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{X}$, $f(\mathcal{S}_1)$ and $f(\mathcal{S}_2)$ are independently distributed when $f \leftarrow \mathfrak{D}$.*

The situation when constructing compressed superposition oracles for non-uniformly distributed random functions is very similar. In this case we need the operations $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S})$ to be efficiently implementable for the compressed oracle to be efficient. Here, $\mathcal{S} \subseteq \mathcal{X}$. 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.

Let us now come back to Definition 5, we want to translate the constraint on distributions to constraints on the quantum sampling procedure. The definition requires that the distribution is independent for any $\mathcal{S}_1$ and $\mathcal{S}_2$, this leads to the following requirement on sampling procedures:

$$\forall \mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{X} : \ \mathsf{Samp}_{\mathfrak{D}}(\mathcal{S}_1 \cup \mathcal{S}_2) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{S}_1) \circ \mathsf{Samp}_{\mathfrak{D}}(\mathcal{S}_2). \tag{14}$$

Let us present a detailed definition of sampling procedures for product distributions.

Definition 6 (Sampling procedure for a product $\mathfrak{D}$)**.** *A sampling procedure $\mathsf{Samp}_{\mathfrak{D}}$ for a product distribution $\mathfrak{D}$ (as defined in Def. 5) is a family of unitary operators*

$$\{\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S}_1) : \mathcal{S}_1 \subseteq \mathcal{X}\}, \tag{15}$$

where each operator fulfills the following conditions:

(*i*) *It is efficiently implementable in the number of inputs $|\mathcal{S}_1|$.*

(*ii*) *It prepares the appropriate superposition on the zero state:*

$$\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S}_1)|0^{|\mathcal{S}_1|}\rangle_{F_1} = \sum_{\vec{y}_1 \in \mathcal{Y}^{|\mathcal{S}_1|}} \sqrt{\mathop{\mathbb{P}}_{f \leftarrow \mathfrak{D}}[f(\mathcal{S}_1) = \vec{y}_1]}|\vec{y}_1\rangle_{F_1}. \tag{16}$$

(*iii*) *The operators are independent, so for $\mathcal{S}_2 \subseteq \mathcal{X}$*

$$\mathsf{Samp}^{F_1F_2}_{\mathfrak{D}}(\mathcal{S}_1 \cup \mathcal{S}_2) = \mathsf{Samp}^{F_1}_{\mathfrak{D}}(\mathcal{S}_1) \circ \mathsf{Samp}^{F_2}_{\mathfrak{D}}(\mathcal{S}_2). \tag{17}$$

Note that for $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S})$ to be efficient, it is not sufficient that the probability distributions $\mathfrak{D}$ are classically efficiently samplable. 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].

An interesting example of a distribution that is not product but which we can quantumly lazy-sample is the following: It is uniform for inputs in $\{0,1\}^n \setminus \{x\}$ for any x and is fully determined on the "last" input: $f(x) = \bigoplus_{x' \neq x} f(x')$.

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 mapped 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.

In the following we sketch the workings of the quantum algorithm $\mathsf{CFO}_{\mathfrak{D}}$ responsible fo updating the oracle register. The set of inputs $\mathcal{X}$ is expanded by the symbol $\perp$, denoting an empty entry in the quantum database.

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

1. Find the index $l \in [q]$ of the register holding the first x_l from the right that is $x_l < x$, we should insert (x, η) into this register.
2. If $x \neq x_l$: insert x in a register after the last element of the database and shift it to position l, moving the intermediate registers backwards.
3. Apply $\mathsf{QFT}_N^{D_l^Y} \circ \mathsf{Samp}_{\mathfrak{D}}^{D_l}(x)$ to change the basis to the Fourier basis (in which the adversary's η is encoded) and update register D_l to contain $(x_l, \eta_l - \eta)$, change the basis back to original by applying $\mathsf{Samp}_{\mathfrak{D}}^{\dagger D_l}(x) \circ \mathsf{QFT}_N^{\dagger D_l^Y}$.
4. Check if register l contains a pair of the form $(x_l, 0)$, if yes subtract x from the first part to yield $(\perp, 0)$ and shift it back to the end of the database.
5. Uncompute[2] l.

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

Using this notation, Alg. 1 defines the procedure of updates of the database of the compressed database. We refer to Appendix D.3 for the fully detailed description of $\mathsf{CFO}_{\mathfrak{D}}$.

Below in Alg. 2 we explain how to uncompute l in line 12 of Algorithm 1.

In Alg. 1 we use the fact that $\mathsf{Samp}_{\mathfrak{D}}$ is a local sampling procedure, Def. 6; Note that we write $\mathsf{Samp}_{\mathfrak{D}}^{D(x)}(x)$, so the sampling is independent from all queries that are already in the database.

We 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 $(\perp, 0)$ at the beginning of each query of adversary A. The current formulation of $\mathsf{CFO}_{\mathfrak{D}}$ assumes that there is an upper bound on the number of queries made by the adversary, this is not a fundamental requirement.

The interface corresponding to the compressed Fourier oracle $\mathsf{CFO}_{\mathfrak{D}}$ interprets the adversary's output register in the Fourier basis. When we want to change the basis to the standard

[2]Uncomputing a function means in the context of quantum computing applying the conjugate of the unitary calculating this function

Algorithm 1: General $\mathsf{CFO}_{\mathfrak{D}}$

Input : Unprepared database and adversary query: $|x,\eta\rangle_{XY}|Д\rangle_D$
Output: $|x,\eta\rangle_{XY}|Д'\rangle_D$

Count in register S the number of non-padding ($Д^X \neq \perp$) entries s in D
if $x \notin Д^X$ **then** `// add`
  Insert x to $Д^X$ in the right place and add 1 to S `// Keeping` $Д^X$ `sorted`
Apply $\mathsf{QFT}_N^{D^Y(x)}\mathsf{Samp}_{\mathfrak{D}}^{D^Y(x)}(x)$ `// Prepare the database:` $Д(x) \mapsto \Delta(x)$
Subtract η from $\Delta^Y(x)$ `// update entry with` x
Apply $\mathsf{Samp}_{\mathfrak{D}}^{\dagger D(x)}(x)\mathsf{QFT}_N^{\dagger D^Y(x)}$ `// Unprepare the database:` $\Delta(x) \mapsto Д(x)$
In register L save location l of x in $Д$
if $Д_l^Y = 0$ **then** `// remove or do nothing`
  Remove x from D_l^X and shift register D_l^X to the back `//` $Д_l^X \mapsto \perp$
if $Д_l^X \neq x$ **then**
  Shift D_l^Y to the back and subtract 1 from S
Uncompute l from register L `// Algorithm 2`
Uncompute s from register S
Return $|x,\eta\rangle_{XY}|Д'\rangle_D$ `//` $Д'$ `is the modified database`

Algorithm 2: Uncompute L in line 12 of Alg. 1

Control on registers X and D^X
for $i = 1\ldots, s-1$ **do**
  if $Д_i^X = x$ **then**
    Subtract i from L
  else if $Д_i^X < x$ *and* $x < Д_{i+1}^X$ **then**
    Subtract $i+1$ from L

one, we apply $\mathsf{QFT}_N^{D^Y}$ to the database register and QFT_N^Y to the adversary's output register. These basis changes give rise to the versions of oracle analogous to the full-oracle case:

$$\mathsf{CStO} \xleftrightarrow{\mathsf{QFT}_N^Y} \mathsf{CPhO} \xleftrightarrow{\mathsf{QFT}_N^{D^Y}} \mathsf{CFO}. \tag{18}$$

The intermediate oracle is the compressed phase oracle.

The decompression procedure for the general Compressed Fourier Oracle is given by Alg. 3. 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 $Д^X \neq \perp$ 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. $\mathcal{X}$ 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. This FOR loop is controlled on register S. Note that after preparing S we do not modify S anymore in this step. 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

Algorithm 3: General Decompressing Procedure $\mathsf{Dec}_{\mathfrak{D}}$

Input : Unprepared database: $|Д\rangle_D$
Output: Prepared, Fourier-basis truth table: $|\phi(Д)\rangle$

Count in register S the number of non-padding ($x \neq \perp$) entries s
Initialize register F in a state $\bigotimes_{x\in\mathcal{X}}|0\rangle$
for $i = 1, 2, \ldots, s$ **do** `// Controlled on` S
    Swap register D_i^Y with $F(x_i)$
for $x \in \mathcal{X}$ *in descending order* **do**
    if $F(x)$ *holds a value* $\neq 0$ **then**
        Subtract x from register D_s^X
        Subtract 1 from register S
Discard D and S
Apply $\mathsf{QFT}_N^F\mathsf{Samp}_{\mathfrak{D}}^F(\mathcal{X})$ `// Prepare the database`

by one to update the number of remaining non-padding entries in the CFO-database. Here the loop range does not depend on the size of the database, just the size of the domain. Finally, we switch to the correct basis to end up with a full oracle of Fourier type, i.e. a FO.

Theorem 7 (Correctness of $\mathsf{CFO}_{\mathfrak{D}}$)**.** *Say $\mathfrak{D}$ is a product distribution (Def. 5) over functions, let $\mathsf{CFO}_{\mathfrak{D}}$ be defined as in Alg. 1 and FO as in Eq.(12). Let z be a random arbitrarily distributed string. Then for any quantum adversary A making q quantum queries we have*

$$\left|\mathbb{P}\left[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{FO}}(z)\right] - \mathbb{P}\left[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{CFO}}(z)\right]\right| = 0. \tag{19}$$

Proof Proof sketch. We will show that

$$|\Psi_{\mathsf{FO}}\rangle_{AF} = \mathsf{Dec}_{\mathfrak{D}}^D|\Psi_{\mathsf{CFO}}\rangle_{AD}, \tag{20}$$

where $|\Psi_{\mathsf{FO}}\rangle_{AF}$ is the joint state of the adversary and the oracle resulting from the interaction of A with FO and $|\Psi_{\mathsf{CFO}}\rangle_{AD}$ is the state resulting from the interaction of A with $\mathsf{CFO}_{\mathfrak{D}}$. The state $|\Psi_{\mathsf{FO}}\rangle_{AF}$ is generated by applying $\prod_{i=1}^q \mathsf{U}_i \circ \mathsf{FO}$ to the $|\psi_0\rangle_A|0^M\rangle_F$, where the $|\psi_0\rangle_A$ is the initial state of the adversary. In the case of the compressed oracle the state $|\Psi_{\mathsf{CFO}}\rangle_{AD}$ is generated by applying $\prod_{i=1}^q \mathsf{U}_i \circ \mathsf{CFO}$ to the $|\Psi_0\rangle_A|(\perp, 0)^q\rangle_D$, where $(\perp, 0)^q$ denotes q pairs $(\perp, 0)$.

We can focus on the state equality from Eq. (20) because if they are indeed equal, then any adversary's measurement on $|\Psi_{\mathsf{FO}}\rangle_{AF}$ will yield the output $b = 1$ with the same probability as on $\mathsf{Dec}_{\mathfrak{D}}^D|\Psi_{\mathsf{CFO}}\rangle_{AD}$.

To prove that Eq. (20) indeed holds we calculate a single query made to the compressed oracle. We can really perform a detailed calculation of that procedure thanks to the assumption put on the distribution, $\mathfrak{D}$ is a product distribution (Def. 5) and the sampling procedure in constructed accordingly (Def. 6).

After we calculated the updated compressed database we can easily decompress it and compare with the corresponding updated full oracle register. All the details of this proof can be found in Appendix A. □

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. A common use of the framework is to 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 this section we generalize the O2H lemma to work with the compressed oracle technique. The oracle register in this technique is a superposition over databases of input-output pairs. A relation on a database is a specific set of databases that fulfill some requirement, e.g., contains a collision (two entries with distinct inputs and the same output). The O2H lemma, as stated in [AHU19], works with punctured oracles, these are quantum oracles that include a binary measurement after every query. After introducing the notion of relations on databases we bring the concept of punctured oracles to the compressed oracles technique. Punctured compressed oracles involve measurements on the superposition of databases. These measurements allow to analyze adversaries that had access to oracles that e.g. never output colliding outputs, this is a very useful situation, considering how often we lazy-sample functions is cryptographic proofs and then want to focus on some transcripts of input-output pairs. Our version of the O2H lemma provides a bound on the distinguishing advantage between an oracles that is not punctured and an oracle that is. The bound in the O2H lemma is stated in terms of the probability of any measurement in the punctured oracle succeeding, i.e., finding a database in the oracle register that fulfills the relation we discuss. The strength of our result lies in how versatile the new O2H lemma is, moreover the proof of the lemma is almost the same as the one in [AHU19].

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 [AHU19]. 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 observing this bad event.

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 [AHU19] apply almost word by word so we just describe the differences and refer the reader to the original work.

4.1 Relations on databases

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

Definition 8 (Classical relation R on D)**.** *Let D be a database of size at most q pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$. We call a subset $R \subseteq \bigcup_{t\in[q+1]} (\mathcal{X} \times \mathcal{Y})^t$*[3] *a* classical relation R on D.

An example of such a relation is a collision, namely

$$R_{\text{coll}} := \{((x_1, y_1), \cdots, (x_s, y_s)) \in \bigcup_{t\in[q+1]} (\mathcal{X} \times \mathcal{Y})^t : s \leq q, \exists_{i,j}\ i \neq j, x_i \neq x_j, y_i = y_j\}. \quad (21)$$

[3]Note that $[q + 1] = \{0, 1, \ldots, q\}$

Note however, that it is only reasonable to check if the *non-padding entries* are in R, omitting the $(\perp, 0)$ pairs at the end of D. If D is held in a quantum register, the classical relation 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 it holds that $((x_1, y_1), \cdots, (x_s, y_s)) \in R$ and for the remaining $i > s$, the (x_i, y_i) are padding entries.

We also state an explicit algorithm to implement the measurement of a relation R, given that membership in R is efficiently decidable. To denote the single-bit membership decision by $D \in R$, the bit is 1 if and only if database D is in R. To measure the relation we define a unitary V_R^{SDJ} that XORs a bit $D \in R$ to register J; This unitary is controlled on registers S and D, the former holds the information about the size of the database and the latter the database itself. Alg. 4 defines the measurement procedure of measuring R on quantum databases in the standard basis.

Algorithm 4: Measurement of a relation R

Input : Database $|D\rangle_D$ in the standard basis
Output: Outcome p and post-measurement state $|D'\rangle_D$

1 Count in register S the number of non-padding ($D^X \neq \perp$) entries s in D
2 Initialize a new qubit register $|0\rangle_J$
3 Apply V_R^{SJD} that XORs a bit $j := D \in R$ to register J
4 Uncompute register S, measure register J, output the outcome j

An important issue concerning measuring relations is the basis in which we store the quantum database. For the measurement to be meaningful it has to be done in the standard basis, so it is easiest to analyze $\mathsf{CStO}_{\mathfrak{D}}$ or $\mathsf{CPhO}_{\mathfrak{D}}$, defined by Eq. (18).

While not directly relevant to our applications, we keep the generality of [AHU19] 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, p-parallel queries are processed by sequentially applying the compressed oracle unitary p times.

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

Definition 9 (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. 4. By* Find *we denote the event that* R *outputs* 1 *at least once among all queries.*

Full oracles can be punctured as well, the relation is then checked only on the queried entries of the function table—those queried entries need to be identified (like in $\mathsf{Dec}_{\mathfrak{D}}$ from Alg. 3) prior to the measurement of R.

In many applications of punctured oracles we might want to apply $\mathsf{H} \setminus R$ only if some condition is fulfilled. Moreover, this condition might be quantum—in other words we control $\mathsf{H} \setminus R$ on some quantum register. To avoid the situation of a measurement being performed or not depending on a state of a quantum register—which is not permitted by quantum mechanics—we propose the following solution: We postpone the measurement to the end of the quantum algorithm. Namely, we omit the measurement of register J in Alg. 4 and perform it at the very end of the overarching algorithm. After the measurement we can uncompute the outcome register J. We are not changing notation and implicitly assume the postponement of puncturing—e.g. in Alg. 7.

4.2 One-way to Hiding Lemma

Using the definitions from the previous sections we can prove a theorem similar to Theorem 1 of [AHU19].

Let us also comment on the differences of the O2H lemma in [AHU19] and our paper. The main difference is that in our generalization we no longer focus solely (we can recover the original O2H lemma though) on adversary's inputs but also treat the outputs of the oracle. Function outputs are also important in [AHU19], but the oracle is not lazy sampled, there they pick a subset of the domain such that e.g. the output is 0 and then puncture on inputs in this random set. We use lazy sampled functions and puncture on databases, so functions defined only on the queried inputs. In addition, defining the puncturing operation on the compressed oracle database is more expressive, as it allows puncturing conditions depending on more than one input-output pair.

Theorem 10 (Compressed oracle O2H)**.** *Let R_1 and R_2 be relations on the database of a quantum oracle* H*. Let z be a random string. R and z may have arbitrary joint distribution. Let* A *be an oracle algorithm of query depth d, then*

$$\begin{aligned}
&\left|\mathbb{P}[b=1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[b=1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R_1\cup R_2}(z)]\right| \\
&\leq \sqrt{(d+1)\mathbb{P}[\mathrm{Find}_2 : \mathsf{A}^{\mathsf{H}\setminus R_1\cup R_2}(z)]}, \qquad (22)\\
&\left|\sqrt{\mathbb{P}[b=1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R_1}(z)]} - \sqrt{\mathbb{P}[b=1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R_1\cup R_2}(z)}\right| \\
&\leq \sqrt{(d+1)\mathbb{P}[\mathrm{Find}_2 : \mathsf{A}^{\mathsf{H}\setminus R_1\cup R_2}(z)]}, \qquad (23)
\end{aligned}$$

where Find_2 *is the event that measuring R_2 succeeds.*

Proof Proof sketch. The proof works almost the same as the proof of Theorem 1 of [AHU19]. Instead of checking register X for the success of the puncturing measurement we analyze the oracle register. The rest follows exactly the same reasoning. All the details of the full proof can be found in Appendix B □

We continue by deriving an explicit formula for $\mathbb{P}[\mathrm{Find}]$. 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

$$\mathbb{P}[\mathrm{Find} : \mathsf{A}^{\mathsf{H}\setminus R}(z)] = 1 - \left\|\prod_{i=1}^{d}(\mathbb{1}-\mathsf{J}_R)\mathsf{U}_j^{\mathsf{H}}|\Psi_0\rangle\right\|^2. \qquad (24)$$

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 [AHU19]. 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 events* $\neg\mathrm{Find}_1$ *and* $\neg\mathrm{Find}_2$ *respectively, i.e. for any string z and any quantum algorithm* A

$$\mathbb{P}[b=1 : b \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \neg\mathrm{Find}_1] = \mathbb{P}[b=1 : b \leftarrow A^{\mathsf{G}\setminus R_2}(z) \mid \neg\mathrm{Find}_2]. \qquad (25)$$

Note that not punctured 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 for any $b \in \{0,1\}$*

$$\left|\mathbb{P}[b \leftarrow A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[b \leftarrow A^{\mathsf{G}\setminus R_2}(z)]\right| \leq 2\mathbb{P}[\text{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] + 2\mathbb{P}[\text{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]. \quad (26)$$

Proof. We bound

$$\begin{aligned}
&\left|\mathbb{P}[b \leftarrow A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[b \leftarrow A^{\mathsf{G}\setminus R_2}(z)]\right| \\
&\overset{\text{Def. 11}}{=} \Big|\mathbb{P}[b \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \neg\text{Find}_1]\Big(\mathbb{P}[\neg\text{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[\neg\text{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]\Big) \\
&+ \mathbb{P}[b \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \text{Find}_1]\mathbb{P}[\text{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] \\
&-\mathbb{P}[b \leftarrow A^{\mathsf{G}\setminus R_2}(z) \mid \text{Find}_2]\mathbb{P}[\text{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]\Big| \qquad (27) \\
&\overset{\triangle}{\leq} \left|\underbrace{\mathbb{P}[b \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \neg\text{Find}_1]}_{\leq 1}\underbrace{\Big(\mathbb{P}[\neg\text{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[\neg\text{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]\Big)}_{=\mathbb{P}[\text{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)] - \mathbb{P}[\text{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)]}\right| \\
&+ \left|\underbrace{\mathbb{P}[b \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \text{Find}_1]}_{\leq 1}\mathbb{P}[\text{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)]\right| \\
&+ \left|\underbrace{\mathbb{P}[b \leftarrow A^{\mathsf{G}\setminus R_2}(z) \mid \text{Find}_2]}_{\leq 1}\mathbb{P}[\text{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]\right| \qquad (28) \\
&\overset{\triangle}{\leq} 2\mathbb{P}[\text{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] + 2\mathbb{P}[\text{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)], \qquad (29)
\end{aligned}$$

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].

It is a fact of quantum mechanics that measurements disturb the state. Considering that, one might be curious if measuring the database does not disturb it too much. As an example, note that after a measurement of the collision relation, eq. (21), the database does not necessarily consist of only non-Fourier-0 entries. Even though this is true, if the disturbance of the oracle is low enough, then the adversary will not notice it. This is exactly the case of the O2H lemma, the disturbance is low enough so the adversary does not notice any difference in the content of the oracle's output.

4.3 Calculating Find for the Collision and Preimage Relations

We state a lemma giving a bound on the probability of Find for the uniform distribution over the set $\{f : \mathcal{X} \to \mathcal{Y}\}$, and for the union of the collision and preimage relations. The preimage relation is satisfied when the output of the oracle is 0:

$$R_{\text{preim}} := \{((x_1, y_1), \cdots, (x_t, y_t)) \in \bigcup_{s\in[q+1]} (\mathcal{X} \times \mathcal{Y})^s : \exists i : y_i = 0\}. \quad (30)$$

In the following we assume $\mathcal{Y} = [N]$.

Lemma 13. *For any quantum adversary* A *interacting with a punctured oracle* $\mathsf{CStO}_{\mathcal{Y}} \setminus (R_{\text{preim}} \cup R_{\text{coll}})$*—where* R_{coll} *is defined in Eq.* (21) *and* R_{preim} *in Eq.* (30)*—the probability of* Find *is bounded by:*

$$\mathbb{P}[\mathsf{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}} \setminus (R_{\text{preim}} \cup R_{\text{coll}})]] \leq \frac{2q(q+1)}{N} + \frac{3q^2(q+1)^2}{N\sqrt{N-q}} + \frac{2q^3(q+1)^3}{N(N-q)}, \tag{31}$$

where q *is the maximal number of queries made by* A *and* $N = |\mathcal{Y}|$*.*

The full proof of Lemma 13 is located in Appendix C. The above bound for $q \in \left(\sqrt[4]{N}\right)$ simplifies to $\frac{7q(q+1)}{N}$, so just the classical collision (and preimage) finding bound. Intuitively, this is justified by the fact that the coherence needed by the optimal quantum search algorithms (e.g. the Grover algorithm [Gro96]) is broken by the repeated measurement.

Here we give a simple proof for a slightly weaker bound:

$$\mathbb{P}[\mathsf{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}} \setminus (R_{\text{preim}} \cup R_{\text{coll}})]] \leq 4\sqrt{\frac{q^5}{|\mathcal{Y}|}}. \tag{32}$$

To prove Equation (32), we need two lemmas. The first lemma is a bound on the probability of finding a preimage of 0, or a collision, when interacting with the un-punctured CStO.

Lemma 14 (Theorems 1 and 2 in [Zha19])**.** *After* q *queries to* CStO*, a measurement of* R_{preim}*, respectively* R_{coll}*, returns* Find *with probability at most* $\frac{q^2}{|\mathcal{Y}|}$*, respectively* $3\frac{q^3}{|\mathcal{Y}|}$*.*

The second lemma is a variant of the gentle measurement lemma [Win99].

Lemma 15 (Gentle measurement lemma for multiple measurements, Lemma 17 in [CMP20])**.** *Let* $|\psi\rangle$ *be a quantum state and* Π_i*,* $i = 1, ..., r$ *projectors such that*

$$\|\Pi_i|\psi\rangle\|_2^2 \geq 1 - \varepsilon$$

for all i*. Then*

$$\|\Pi_r \dots \Pi_2\Pi_1|\psi\rangle\|_2^2 \geq 1 - 2r\sqrt{\varepsilon}.$$

Proof Proof of Equation (32)*.* Without loss of generality, we can assume A to be unitary (except for a final measurement.) Let U_i be the unitary that implements A up to right before the i-th query (including previous query unitaries), and let $\Pi = \mathbb{1} - \mathsf{J}_R$ for $R = R_{\text{coll}} \cup R_{\text{preim}}$. Defining the projectors $\Pi_i = \mathsf{U}_i^\dagger \Pi \mathsf{U}_i$, note that

$$\mathbb{P}[\mathsf{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}} \setminus (R_{\text{preim}} \cup R_{\text{coll}})]] = 1 - \|\Pi_q \dots \Pi_2\Pi_1|0\rangle\|_2^2.$$

Now observe that by Lemma 14 and a union bound we have

$$1 - \|\Pi_i|\psi\rangle\|_2^2 \leq \frac{i^2 + 3i^3}{|\mathcal{Y}|} \leq 4\frac{q^3}{|\mathcal{Y}|}.$$

An application of Lemma 15 finishes the proof of Equation (32). □

Finally let us provide a clearer explanation for how to use our technique. Whenever we lazy sample a uniform function in the (classical) game-playing framework we have some bad events, for example the newest output collides with some previous one. To translate the proof to the quantum case we reformulate the bad events to the language of relations and use a punctured compressed oracle. Hybrid jumps are bounded with the O2H lemma and $\mathbb{P}[\mathsf{Find}]$ with (a version of) Lemma 13. Note that only this technique allows us to deal with collisions in quantumly lazy sampled functions. The only other paper that considers this problem is [Zha19] but there are some things that are a bit unclear in the proof of the important lemma there.

5 Quantum Security of the Sponge Construction

We use our methods to show a detailed proof of quantum indifferentiability of the sponge construction when used with a random function as the internal function. In Appendix E we prove that quantum indifferentiability implies collapsingness.

After introducing the sponge construction in the next section, we present two proofs of indifferentiability of the sponge construction. The first proves classical security and the second quantum. We present two proofs to simplify reading the latter proof, it follows the same reasoning as the former one. We also want ot highlight how similar these proofs are, this is what we consider to be one of the main advantages of our quantum game-playing framework. In our framework all proofs of quantum indifferentiability can follow the same reasoning and very similar steps as the classical version.

Before we proceed let us remind the reader the main concepts introduced in this paper, that are necessary to follow the proof of quantum indifferentiability. The central object of the proof are punctured oracles, defined in Def. 9. They play the role of subroutines that lazy-sample functions and output "True" when a bad event occurs. Readers familiar with the original game-playing framework [BR06] will recognize the crucial subroutines of the classical games. Additionally, punctured oracles are objects that allow to condition probabilistic events on some aspects of quantum queries done by the adversary. This useful feature allows us to sometimes use arguments from the classical proof in the quantum one.

A punctured oracle is built using the compressed oracle framework and formally includes a quantum database register, as described in detail in section 3. Nonetheless these details are not necessary to follow the contents of this section. The only two things to keep in mind are that in general the adversary can make quantum queries to the primitives and that the responses of queries are saved in the adversary's quantum register $|s, v\rangle$, where s is the query and v is any value in the codomain of the queried function.

The reason we use punctured oracles is that they allow to use the One-way To Hiding (O2H) lemma. This is an extremely useful tool for bounding the distinguishability advantage of two quantum games. We cover this lemma in details in section 4. Technically the most demanding part of using the O2H lemma is bounding the probability of any pucturing measurement succeeding (we call this event Find). We compute a bound on $\mathbb{P}\left[\text{Find}\right]$ useful in the quantum indifferentiability proof for sponges in section 4.3.

The second distinguishability bound that we use is shown in Lemma 12. This is a relatively simple statement, that is true for games that are almost identical (Def.11).

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 internal state $s = (\bar{s}, \hat{s}) \in \{0,1\}^r \times \{0,1\}^{c}$[4] of Sponge consists of two parts: the *outer* part $\bar{s} \in \{0,1\}^r$ and the *inner part* $\hat{s} \in \{0,1\}^c$. The logarithm of the number of possible outer parts r is called the *rate* of the sponge, and c is called *capacity*. Naturally the internal function is a map $\varphi : \{0,1\}^r \times \{0,1\}^c \to \{0,1\}^r \times \{0,1\}^c$. To denote the internal function with output limited to the first r bits and the last c bits we use the same notation as for states, $\bar{\varphi}$ and $\hat{\varphi}$ respectively. By $(\{0,1\}^r)^*$ we denote the strings consisting of an arbitrary number of r-bit blocks. By $\text{pad} : \{0,1\}^* \to (\{0,1\}^r)^*$ we denote a padding function: an efficiently computable injection such that $|\text{pad}(m)| \geq r$ and that the last bit of $\text{pad}(m)$ is never 0 (this ensures injectivity for inputs of different lengths). By $|p|_r$ we denote the number of r-bit blocks in p and by $\lfloor p \rfloor_i$ we denote the i-th r-bit block of p. The function constructed in that way behaves as follows,

[4]Our result also holds for arbitrary finite sets $\mathcal{A} \times \mathcal{C}$, where additionally $\mathcal{A}$ is an Abelian group.

$\textsc{Sponge}_\varphi : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^*$, where $\{0,1\}^* := \bigcup_{n=0}^{\infty}\{0,1\}^*$. In Fig. 2 we present a scheme of the sponge construction evaluated on input m.

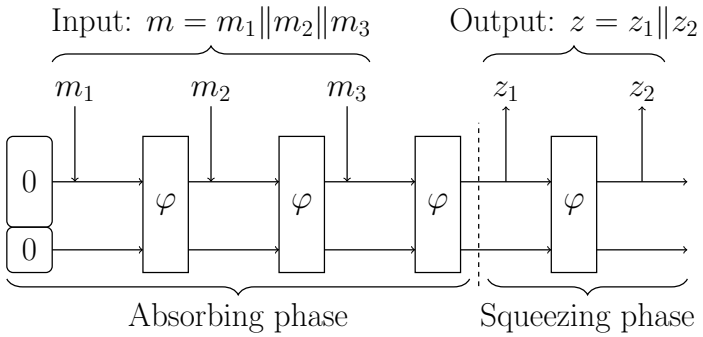

Figure 2: A schematic representation of the sponge construction: $\textsc{Sponge}_\varphi(m_1\|m_2\|m_3) = z_1\|z_2$.

For a set $\mathcal{S} \subseteq \{0,1\}^{r+c}$, 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 $\mathcal{H}_{\{0,1\}^{r+c}}$: $\overline{Y}$ is the part of the register holding the outer parts and $\widehat{Y}$ holds the inner parts.

Algorithm 5: $\textsc{Sponge}_\varphi[\textsc{pad}, r, c]$

Input : $m \in \{0,1\}^*$, $\ell \geq 0$.
Output: $z \in \{0,1\}^\ell$

$p := \textsc{pad}(m)$
$s := (0,0) \in \{0,1\}^r \times \{0,1\}^c$.
for $i = 1$ to $|p|_r$ **do** `// Absorbing phase`
  $s := (\bar{s} \oplus \lfloor p \rfloor_i, \hat{s})$
  $s := \varphi(s)$
$z := \bar{s}$ `// Squeezing phase`
while $|z| < \ell$ **do**
  $s := \varphi(s)$
  $z := z\|\bar{s}$
Output z

An important feature of the sponge construction is that one can associate to the internal function φ a graph $G = (\mathcal{V}, \mathcal{E})$ [Ber+07]. It is called the *sponge graph*; The set of nodes $\mathcal{V} := \{0,1\}^{r+c}$ corresponds to all possible states of the sponge. A directed edge connects any two nodes (s,t) whenever $\varphi(s) = t$, hence there are 2^{r+c} edges in $\mathcal{E}$. From each node starts exactly one edge. We group the nodes with the same inner-part value into *supernodes*, so that we have 2^c supernodes and each such supernode consists of 2^r nodes. Edges between nodes are also edges between supernodes.

Whenever the adversary queries $\textsc{Sponge}$, she starts at the $(0^r, 0^c)$ node. This node is called the *root*. Next the first r-bit block $\lfloor p \rfloor_1$ in the padded message $p = \textsc{pad}(m)$ is added to the outer part of the state and queried to the internal function $\varphi(\lfloor p \rfloor_1, 0^c) = s_2$. The node s_2 is the node in the edge $((\lfloor p \rfloor_1, 0^c), s_2) \in \mathcal{E}$. The same situation repeats for all blocks in p, namely the absorbing phase. When $\textsc{Sponge}$ starts generating output, we no longer modify the state, or just add 0^r to the outer part. Note that knowing just p and G we can get to the last node traversed by $\textsc{Sponge}_\varphi(m)$. This leads us to the definition of a *sponge path*.

Definition 16 (Sponge Path, Definition 3 in [Ber+08]). *First, the empty string is a sponge path to the node $(0^r, 0^c)$. Then, if p is a sponge path to node $s = (\bar{s}, \hat{s})$ and there is an edge $(\bar{s} \oplus a \| \hat{s}, t)$ in the sponge graph G, $p' = p \| a$ is a sponge path to node t.*

Given the above definition, let us say that p forms a sponge path to s, then we define a function

$$\mathsf{SpPath}(s, G) := p. \tag{33}$$

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

When we talk about the simulator in a proof of indifferentiability, we define the *simulator graph*. The graph kept by the simulator differs from the sponge graph discussed above by the number of edges in it. As the simulator lazy samples the internal function φ the set of edges $\mathcal{E}$ grows by at most one edge per one adversary's query. Other than that, everything that has been said above holds. We refer to the simulator graph G as just the (sponge) graph whenever it is clear from context.

A supernode is called *rooted* if there is a path (a regular path that is just a set of edges connected by the end-start nodes) 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.

A simulator graph is called *saturated* if $\mathcal{R} \cup \mathcal{U} = \{0,1\}^c$. It means that for every inner state in $\{0,1\}^c$ there is an edge in G that leads to it from 0^c (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^c 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. Using 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 $f : \{0,1\}^{r+c} \to \{0,1\}^{r+c}$. Considering how Sponge is defined we want a bound on finding collisions and zero-preimages. We define the bound as a function of the number of queries q to f:

$$f_{\text{coll}}(q) := \frac{q(q+1)}{2^{c+1}}, \tag{34}$$

the bound can be derived by in the standard way. Probability that any classical algorithm finds a collision or a preimage of zero in $[N]$ after q queries is:

$$\mathbb{P}\left[\text{coll} \cup \text{preim} \leftarrow \mathsf{A}\right] \leq \sum_{i=1}^{q} \frac{i}{N} = \frac{q(q+1)}{2N}, \tag{35}$$

where we use the union bound and note that after i queries the adversary can either find the preimeage of zero or hit any of the previous outputs, producing a collision. For a more detailed derivation please go to Appendix A.4 in [KL14].

As the sponge construction is used to design variable-input and variable-output functions we define the random oracle

$$\mathsf{H} : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^* \tag{36}$$

accordingly. A random oracle grants access to a function sampled from distribution $\mathfrak{R}$ on functions $\{0,1\}^* \times \mathbb{N} \to \{0,1\}^*$, that is defined as follows: To sample a function $h \leftarrow \mathfrak{R}$ we

- choose g uniformly at random from $\{g : \{0,1\}^* \to \{0,1\}^\infty\}$, where by $\{0,1\}^\infty$ we denote the set of infinitely long bitstrings,
- for each $(x, \ell) \in \{0,1\}^* \times \mathbb{N}$ set $h(x,\ell) := \lfloor g(x) \rfloor_\ell$, that is output the first ℓ bits of the output of g.

In the following section, we omit the second input and we mean that we ask for a single letter $\mathsf{H}(x) = y \in \{0,1\}^r$.

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 17 (Sponge with functions, classical indifferentiability)**.** $\textsc{Sponge}_\varphi[\textsc{pad}, r, c]$ *calling a random function* φ *is* (q,ε)*-indifferentiable from a random oracle, Eq.* (36)*, for* classical *adversaries for any* $q < 2^c$ *and* $\varepsilon = 8\frac{q(q+1)}{2^{c+1}}$.

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 $\textsc{Sponge}_\varphi$. 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.

Game 1 We start with the real world where the distinguisher A has access to a random function $\varphi : \{0,1\}^{r+c} \to \{0,1\}^{r+c}$ and $\textsc{Sponge}_\varphi$ using this random function. The formal definition of the first game is the event

$$\textbf{Game 1} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_\varphi, \varphi])\,. \tag{37}$$

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 $\{0,1\}^{r+c}$ anyway. The definition of the second game is

$$\textbf{Game 2} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_2}, \mathsf{S}_2])\,. \tag{38}$$

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

$$|\mathbb{P}[\textbf{Game 2}] - \mathbb{P}[\textbf{Game 1}]| = 0. \tag{39}$$

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

$$\textbf{Game 3} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3])\,. \tag{40}$$

Algorithm 6: Classical S_2, S_3, S_4, I_6, functions

State : current sponge graph G
Input : $s \in \{0,1\}^{r+c}$
Output: $\varphi(s)$

if s has no outgoing edge **then** `// new query`
  if $\hat{s} \in \mathcal{R} \wedge \mathcal{R} \cup \mathcal{U} \neq \{0,1\}^c$ **then** `// ŝ-rooted, no saturation`
    $\hat{t} \xleftarrow{\$} \{0,1\}^c$, **if** $\hat{t} \in \mathcal{R} \cup \mathcal{U}$, set Bad $= 1$, $\hat{t} \xleftarrow{\$} \{0,1\}^c \setminus (\mathcal{R} \cup \mathcal{U})$
    Construct a path to s: $p := \mathsf{SpPath}(s, G)$
    if $\exists x : p = \text{PAD}(x)$ **then**
      $\bar{t} \xleftarrow{\$} \{0,1\}^r$
      $\bar{t} := \mathsf{H}(x)$
    else
      $\bar{t} \xleftarrow{\$} \{0,1\}^r$
    $t := \bar{t} \| \hat{t}$
  else
    $t \xleftarrow{\$} \{0,1\}^{r+c}$
  Add an edge (s, t) to $\mathcal{E}$.
Set t to the vertex at the end of the edge starting at s
Output t

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:

$$|\mathbb{P}[\textbf{Game 3}] - \mathbb{P}[\textbf{Game 2}]| \leq \mathbb{P}[\text{Bad} = 1]. \tag{41}$$

It is 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 $\{0,1\}^c$ having made q random samples. Then we have that

$$\mathbb{P}[\text{Bad} = 1] \leq f_{\text{coll}}(q), \tag{42}$$

where f_{coll} is defined in Eq. (34). 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

$$\textbf{Game 4} := \left(b = 1 : b \leftarrow \mathsf{A}[\text{SPONGE}_{\mathsf{S}_4}, \mathsf{S}_4^{\mathsf{H}}]\right). \tag{43}$$

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

Claim 18. *Given that* $\text{Bad} = 0$ *the mentioned games are the same:*

$$|\mathbb{P}[\textbf{Game 4} \mid \text{Bad} = 0] - \mathbb{P}[\textbf{Game 3} \mid \text{Bad} = 0]| = 0. \tag{44}$$

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:

$$
\begin{aligned}
&|\mathbb{P}[\textbf{Game 4}] - \mathbb{P}[\textbf{Game 3}]| \overset{\text{Claim 18}}{=} \Big| \mathbb{P}[\textbf{Game 4} \mid \text{Bad} = 0] \\
&\cdot \underbrace{(\mathbb{P}[\text{Bad} = 1 : \textbf{Game 3}] - \mathbb{P}[\text{Bad} = 1 : \textbf{Game 4}])}_{=0} \\
&+ \underbrace{\mathbb{P}[\textbf{Game 3} \mid \text{Bad} = 1]}_{\leq 1}\mathbb{P}[\text{Bad} = 1] + \underbrace{\mathbb{P}[\textbf{Game 4} \mid \text{Bad} = 1]}_{\leq 1}\mathbb{P}[\text{Bad} = 1] \Big| \qquad (45) \\
&\leq 2\mathbb{P}[\text{Bad} = 1]. \qquad (46)
\end{aligned}
$$

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

$$\textbf{Game 5} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{S}_4^{\mathsf{H}}]\right). \qquad (47)$$

Conditioned on $\text{Bad} = 0$, 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, with $\mathsf{H} \setminus R_1$ replaced by **Game 5**, $\mathsf{G} \setminus R_2$ replaced by **Game 4**, and event Find replaced by $\text{Bad} = 1$. As the derivation of Lemma 12 uses no quantum mechanical arguments and the assumption holds—the games are identical conditioned on $\text{Bad} = 0$—the bound holds:

$$|\mathbb{P}[\textbf{Game 5}] - \mathbb{P}[\textbf{Game 4}]| \leq 4\mathbb{P}[\text{Bad} = 1] \leq 4f_{\text{coll}}(q). \qquad (48)$$

Game 6 In the last game we use I_6 (we call it I for *ideal*, that is the world we arrive in the last step of the proof), a simulator that does not check for bad events and samples from the "good" subset of $\{0,1\}^c$. The game is

$$\textbf{Game 6} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{I}_6^{\mathsf{H}}]\right) \qquad (49)$$

and the advantage is

$$|\mathbb{P}[\textbf{Game 6}] - \mathbb{P}[\textbf{Game 5}]| \leq \mathbb{P}[\text{Bad} = 1] \leq f_{\text{coll}}(q). \qquad (50)$$

following Lemma 1. as the only difference is in code but not outputs. We included this last game in the proof because I_6 is clearly a simulator that might fail only if G is saturated but this does not happen if $q < 2^c$. 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.

At line 7 of the classical simulator we replace the lazy sampled outer state by the output of the random oracle. In the quantum case we want to do the same. Unlike in the classical case we cannot, however, save the input-output pairs of the random oracle H that were sampled to generate the sponge graph, as they contain information about the adversary's query input. An attempt to store this data would effectively measure the adversary's state and render our simulation distinguishable from the real world. To get around this issue we reprepare the sponge graph at the beginning of each run of the simulator. To prepare the sponge graph we query H on all necessary inputs to $\hat{\varphi}$, i.e. on the inputs that are consistent with a path from the root to a rooted node. This is done gradually by iterating over the length of the paths. We begin with the length-0 paths, i.e. with all inputs in the database $\widehat{D}$ where the inner part is the all zero string. If the outer part of such an input (which is not changed by the application of SpPath) is equal to a padding of an input, that input is queried to determine the outer part of the output of φ, creating an edge in the sponge graph. We can continue with length-1 paths. For each entry of the database $\widehat{D}$, check whether the input register is equal to a node in the current partial sponge graph. If so, the entry corresponds to a rooted node. Using the entry and the edge connecting its input to the root, a possible padded input to $\textsc{Sponge}$ is created using SpPath. If it is a valid padding, H is queried to determine the outer part of the output of φ, etc.

In the proof we will make use of the result from Lemma 13. Let us denote the bound on inner collisions by

$$f^{Q}_{\text{coll}}(q) := \frac{7q(q+1)}{2^c}, \tag{51}$$

which is valid for $q \in \left(2^{c/4}\right)$.

The main statement of this section is stated below. Noting the distinguishing bound that we prove, we would like to highlight that our result is most probably tight. Roughly, a quantum algorithm for finding inner-collisions in a sponge construction (such a collision would allow to distinguish a sponge from a random oracle) with a random internal function uses $O(|\mathcal{C}|^{1/3})$ queries. The distinguishing complexity coming from our bounds, stated without limiting the range of q for them to apply in Lemma 13, is the matching $\Omega(|\mathcal{C}|^{1/3})$.

Theorem 19 ($\textsc{Sponge}$ with functions, quantum indifferentiability)**.** $\textsc{Sponge}_{\varphi}[\textsc{pad}, r, c]$ *calling a random function φ is (q, ε)-indifferentiable from a random oracle, Eq.* (36)*, for* quantum *adversaries for any $q < 2^c$ and $\varepsilon = 56\frac{q(q+1)}{2^c} + \sqrt{7\frac{q(q+1)^2}{2^c}}$.*

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 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. 1 and additionally analyzed in Lemma 13. Eq. (137) provides a bound of the probability of Find (as defined in Section 4) in the case of compressed oracles and relations relevant for the sponge construction.

It is important to note that the "IF" statements are in fact quantum controlled operations. In line 4 we apply a punctured compressed oracle controlled on the input and the database; To correctly perform this operation we postpone the measurement to after uncomputing of G and $\mathcal{R} \cup \mathcal{U}$ in line 14. This procedure is also discussed in the end of Section 4.

Algorithm 7: Quantum $\boxed{\mathsf{S}_2}$, $\boxed{\mathsf{S}_3}$, $\boxed{\mathsf{S}_4}$, functions

State : Quantum compressed database register D
Input : $|s, v\rangle \in \mathcal{H}^{\otimes 2}_{\{0,1\}^{r+c}}$
Output: $|s, v \oplus \varphi(s)\rangle$

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** $\hat{s} \in \mathcal{R} \ \wedge \ \mathcal{R} \cup \mathcal{U} \neq \{0,1\}^c$ **then** `// ŝ-rooted, no saturation`
4 Apply $\boxed{\mathsf{CStO}^{X\widehat{Y}\widehat{D}(s)}_{\{0,1\}^c}}$, $\boxed{(\mathsf{CStO}_{\{0,1\}^c \setminus (\mathcal{R}\cup\mathcal{U})})^{X\widehat{Y}\widehat{D}(s)}}$, 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 $\boxed{\text{Apply } \mathsf{CStO}^{X\overline{Y}\,\overline{D}(s)}_{\{0,1\}^r}}$, result: $\bar{t}$
8 Write x in a fresh register X_H, $\boxed{\text{apply } \mathsf{H}^{XX_H\overline{Y}\,\overline{D}(s)}}$, uncompute x from X_H, result: $\bar{t}$
9 **else**
10 Apply $\mathsf{CStO}^{X\overline{Y}\,\overline{D}(s)}_{\{0,1\}^r}$, 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 $\mathsf{CStO}^{XY\overline{D}(s)\widehat{D}(s)}_{\{0,1\}^{r+c}}$, result: t
14 Uncompute G and $\mathcal{R} \cup \mathcal{U}$
15 Output $|s, v \oplus t\rangle$

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

Game 1 We start with the real world where the distinguisher A has quantum access to a random function $\varphi : \{0,1\}^r \times \{0,1\}^c \to \{0,1\}^r \times \{0,1\}^c$ and the $\textsc{Sponge}_\varphi$ construction using this random function. The definition of the first game is

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

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 $\{0,1\}^{r+c}$ anyway. The game is defined as:

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

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

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

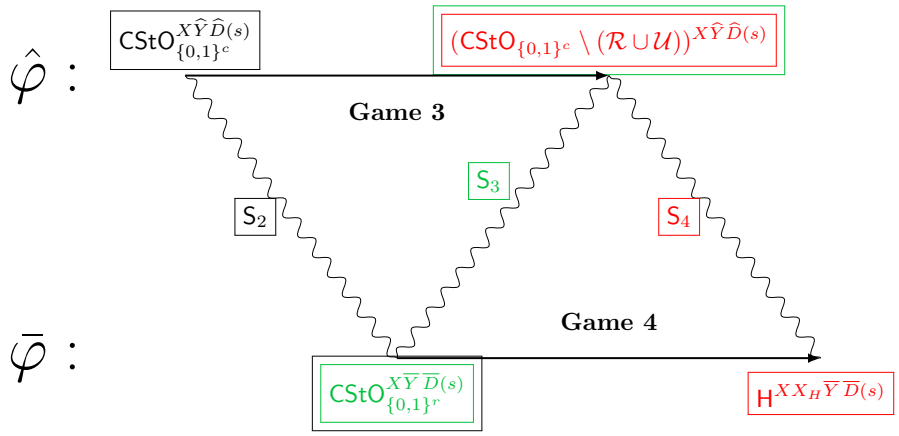

Figure 3: Schematics of the simulators defined in Alg. 7, horizontal arrows signify the change introduced in the labeled game.

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

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

With such a change of the simulators we can use Thm. 10 to bound the difference of probabilities. S_3 measures the relation of being an element of $\mathcal{R} \cup \mathcal{U}$. This relation is equivalent to $R_{\text{preim}} \cup R_{\text{coll}}$. The distinguishing advantage is

$$|\mathbb{P}[\textbf{Game 3}] - \mathbb{P}[\textbf{Game 2}]| \leq \sqrt{(q+1)\mathbb{P}[\text{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]]}. \tag{56}$$

Using Lemma 13 we have that

$$\mathbb{P}[\text{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]] \leq f^Q_{\text{coll}}(q). \tag{57}$$

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

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

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

$$|\mathbb{P}[\textbf{Game 4}] - \mathbb{P}[\textbf{Game 3}]| \leq 4\mathbb{P}[\text{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]] \leq 4f^Q_{\text{coll}}(q). \tag{59}$$

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:

$$\textbf{Game 5} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{S}_4^{\mathsf{H}}]\right) \tag{60}$$

and the advantage is

$$|\mathbb{P}[\textbf{Game 5}] - \mathbb{P}[\textbf{Game 4}]| \leq 4\mathbb{P}[\text{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, \mathsf{S}_4^{\mathsf{H}}]] \leq 4f^Q_{\text{coll}}(q). \tag{61}$$

Conditioned on ¬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.

As long as Find does not occur and the graph is not saturated the adversary cannot distinguish the simulator from a random function except for the distinguishing advantage that we calculated. Saturation certainly does not occur for $q < 2^c$ 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 $\varepsilon = 8f^Q_{\text{coll}}(q) + \sqrt{(q+1)f^Q_{\text{coll}}(q)}$. □

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.

Another open problem is to find a way to quantum lazy sample random permutations. An almost completely new approach has to be devised to tackle this problem as our correctness theorem only applies to local distributions.

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 [Unr21]. 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+20] Gorjan Alagic, Christian Majenz, Alexander Russell, and Fang Song. "Quantum-Access-Secure Message Authentication via Blind-Unforgeability". In: *Advances in Cryptology – EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Cham: Springer International Publishing, 2020, pp. 788–817. ISBN: 978-3-030-45727-3 (cit. on pp. 4, 5).

[AHU19] Andris Ambainis, Mike Hamburg, and Dominique Unruh. "Quantum Security Proofs Using Semi-classical Oracles". In: *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*. 2019, pp. 269–295. DOI: 10.1007/978-3-030-26951-7_10. URL: https://doi.org/10.1007/978-3-030-26951-7%5C_10 (cit. on pp. 3, 4, 5, 16, 17, 18, 38, 39).

[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, 6).

[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, 6, 21).

[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, 26).

[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, 23, 24).

[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, 8).

[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).

[CEV20] Céline Chevalier, Ehsan Ebrahimi, and Quoc Huy Vu. "On the Security Notions for Encryption in a Quantum World." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 237 (cit. on p. 5).

[Chu+20] Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao. "On the Compressed-Oracle Technique, and Post-Quantum Security of Proofs of Sequential Work". In: *arXiv preprint arXiv:2010.11658* (2020) (cit. on p. 5).

[CMP20] Andrea Coladangelo, Christian Majenz, and Alexander Poremba. *Quantum copy-protection of compute-and-compare programs in the quantum random oracle model*. 2020. arXiv: 2009.13865 [quant-ph] (cit. on p. 20).

[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. 5, 57).

[CHS19] Jan Czajkowski, Andreas Hülsing, and Christian Schaffner. "Quantum Indistinguishability of Random Sponges". In: *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*. 2019, pp. 296–325. DOI: 10.1007/978-3-030-26951-7_11. URL: https://doi.org/10.1007/978-3-030-26951-7%5C_11 (cit. on p. 5).

[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. 5, 57).

[Gro96] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219 (cit. on p. 20).

[HM20] Yassine Hamoudi and Frédéric Magniez. "Quantum Time-Space Tradeoffs by Recording Queries". In: *arXiv preprint arXiv:2002.08944* (2020) (cit. on p. 4).

[HI19] Akinori Hosoyamada and Tetsu Iwata. "4-Round Luby-Rackoff Construction is a qPRP". In: *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*. 2019, pp. 145–174. DOI: 10.1007/978-3-030-34578-5_6. URL: https://doi.org/10.1007/978-3-030-34578-5%5C_6 (cit. on pp. 5, 53).

[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).

[KL14] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2014 (cit. on p. 23).

[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. 12).

[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, 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. 8, 39).

[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. 6, 7).

[NIS15] NIST. *Secure Hash Standard (SHS)*. Draft FIPS 180-4. 2015. DOI: 10.6028/NIST.FIPS.180-4 (cit. on p. 6).

[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. 54).

[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. 5).

[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. 3, 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. 5, 57).

[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. 5, 57).

[Unr19] Dominique Unruh. "Quantum Relational Hoare Logic". In: *Proc. ACM Program. Lang.* POPL (2019), 33:1–33:31. DOI: 10.1145/3290346 (cit. on p. 5).

[Unr21] Dominique Unruh. *Compressed Permutation Oracles*. Unfinished draft. 2021 (cit. on pp. 9, 30, 49).

[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 pp. 19, 20).

[Zha19] Mark Zhandry. "How to Record Quantum Queries, and Applications to Quantum Indifferentiability". In: *Advances in Cryptology – CRYPTO 2019*. Springer International Publishing, 2019, pp. 239–268. DOI: 10.1007/978-3-030-26951-7_9 (cit. on pp. 3, 4, 5, 8, 9, 10, 20, 49).

[Zha20] Mark Zhandry. *Private communication*. 2020 (cit. on p. 4).

Symbol Index

$\lvert x\rvert$	Cardinality of a set x / length of a string x/ absolute value	
Add	Function adding x to the compressed database	55
A, B	An adversary, a classical or quantum algorithm	6, 18
$\mathcal{A}$	The alphabet set of outer states, generalization of $\{0,1\}^r$, outer part of $s \in \mathcal{A} \times \mathcal{C}$ denoted by $\bar{s}$	
Bad	A "bad" event in a game.	6, 25
$\mathsf{CFO}_{\mathfrak{D}}$	Compressed Fourier Oracle for distribution $\mathfrak{D}$	14
$\mathcal{C}$	The set of inner states, generalization of $\{0,1\}^c$, inner part of $s \in \mathcal{A} \times \mathcal{C}$ denoted by $\hat{s}$	
Clean	Clean up function for auxiliary register	57
Collapse 1	Collapsing game	57
$\mathsf{CPhO}_{\mathfrak{U}}$	Compressed Phase Oracle	52
$\mathsf{CStO}_{\mathfrak{D}}, \mathsf{CStO}_{\mathcal{Y}}$	Compressed Standard Oracle, for distribution $\mathfrak{D}$ and for a conditionally uniform distribution over $\mathcal{Y}$	14
$D, \Delta, Д$	Prepared database in the standard basis (and the database register), prepared database in the Fourier basis, and the unprepared databse	11
$\mathsf{Dec}_{\mathfrak{D}}$	Decompression procedure	15
D	The distinguisher	6
$\mathfrak{D}$	A distribution.	11
$\mathcal{E}$	The set of edges of a sponge graph	22
Find	Event of measurement of the relation R returning 1	17, 18
FO	Fourier Oracle, $\mathsf{QFT}_N^{YF} \circ \mathsf{StO} \circ \mathsf{QFT}_N^{\dagger YF}$	9
H, G	Compressed Oracle	16
HT_n	The Hadamard transform	50
$\lvert\psi\rangle$	A quantum state, a normalized vector in a Hilbert space	
Larger	A unitary for comparing two bit-strings	54
Locate	Locate the position of x in the database	55
$\textsc{pad}$	Padding function	21
$\mathsf{SpPath}(s, G)$	Function constructing an input to $\textsc{Sponge}$ leading to a given node	23
φ	The map between states in $\textsc{Sponge}$.	21
$\bar{\varphi}$	The map between states with its output limited to the first r bits	21
$\hat{\varphi}$	The map between states with its output limited to the last c bits	21
PhO	Phase Oracle, $\mathsf{QFT}_N^{Y} \circ \mathsf{StO} \circ \mathsf{QFT}_N^{\dagger Y}$	51
J_R	Projector on relation R.	17
QFT_N	The Quantum Fourier Transform	8
Rem	Removing $\text{и} = 0$ from the database	56
$\mathcal{R}$	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 $\mathcal{S}$.	11
S	Classical and quantum simulators.	25, 28

$\textsc{Sponge}_\varphi[\textsc{pad}, r, c]$	Sponge construction with the internal function φ, capacity c, and rate r	22
StO	Standard Oracle	9
$\mathfrak{U}$	The uniform distribution.	49
Upd	Updating η in the database	55
$\mathcal{U}$	The set of supernodes with outgoing edges	23
V_R	The unitary outputting $D \in R$.	17
$\mathcal{V}$	The set of vertices of a sponge graph	22
$\oplus$	Bitwise XOR	50
$y, \eta, и$	Values in the Y register of a database in different bases	11

A Full Proof of Theorem 7

Proof of Theorem 7.

We will show that

$$|\Psi_{\mathsf{FO}}\rangle_{AF} = \mathsf{Dec}^D_{\mathfrak{D}} |\Psi_{\mathsf{CFO}}\rangle_{AD}, \tag{62}$$

where $|\Psi_{\mathsf{FO}}\rangle_{AF}$ is the state resulting from the interaction of A with FO and $|\Psi_{\mathsf{CFO}}\rangle_{AD}$ is the state resulting from the interaction of A with $\mathsf{CFO}_{\mathfrak{D}}$. The state $|\Psi_{\mathsf{FO}}\rangle_{AF}$ is generated by applying $\prod_{i=1}^{q} \mathsf{U}_i \circ \mathsf{FO}$ to the $|\psi_0\rangle_A |0^M\rangle_F$, where the $|\psi_0\rangle_A$ is the initial state of the adversary. In the case of the compressed oracle the state $|\Psi_{\mathsf{CFO}}\rangle_{AD}$ is generated by applying $\prod_{i=1}^{q} \mathsf{U}_i \circ \mathsf{CFO}$ to the $|\Psi_0\rangle_A |(\perp, 0)^q\rangle_D$, where $(\perp, 0)^q$ denotes q pairs $(\perp, 0)$.

We can focus on the state equality from Eq. (62) because if they are indeed equal, then any adversary's measurement on $|\Psi_{\mathsf{FO}}\rangle_{AF}$ will yield the output $b = 1$ with the same probability as on $\mathsf{Dec}^D_{\mathfrak{D}} |\Psi_{\mathsf{CFO}}\rangle_{AD}$.

Let us call a database state

$$|Д(\vec{x}, \vec{и})\rangle := |x, \eta\rangle_{XY} |x_1, и_1\rangle_{D_1} \cdots |x_s, и_s\rangle_{D_s} \cdots |\perp, 0\rangle_{D_q}, \tag{63}$$

where $\vec{x} := (x_1, x_2, \ldots, x_s)$ and $\vec{и} := (и_1, и_2, \ldots, и_s)$ well-formed, if no x_i in $\vec{x}$ is $\perp$ and no $и_i$ in $\vec{и}$ is zero.

To prove Eq. (62) we show that

$$\mathsf{FO} \circ \mathsf{Dec}_{\mathfrak{D}} |Д(\vec{x}, \vec{и})\rangle = \mathsf{Dec}_{\mathfrak{D}} \circ \mathsf{CFO}_{\mathfrak{D}} |Д(\vec{x}, \vec{и})\rangle. \tag{64}$$

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 $\mathsf{FO} = \mathsf{Dec}_{\mathfrak{D}} \circ \mathsf{CFO}_{\mathfrak{D}} \circ \mathsf{Dec}^{\dagger}_{\mathfrak{D}}$, when acting on well-formed databases, then everything that happens on the oracle's register side can be compressed. Note that as we start from the empty oracle state and only apply the oracle to the oracle register, the database will always be well-formed.

We study the action of $\mathsf{Dec}_{\mathfrak{D}}$ on the state in Eq. (63). 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

$$|\Upsilon(\vec{x}, \vec{и})\rangle_F := \mathsf{Dec}_{\mathfrak{D}} |Д(\vec{x}, \vec{и})\rangle = \sum_{\phi \in \mathcal{F}} \frac{1}{\sqrt{N^M}} \sum_{f \in \mathcal{F}} \omega_N^{\phi \cdot f} \; a_{f\vec{и}}(\mathcal{X}) \; |\phi_0\rangle_{F(0)} \cdots |\phi_{M-1}\rangle_{F(M-1)}, \tag{65}$$

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 the fact that $\mathsf{Samp}_{\mathfrak{D}}$ is defined for a product distribution, as in Def. 6, we have that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}\backslash\{x\})\circ\mathsf{Samp}_{\mathfrak{D}}(x)$ 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 $\mathsf{F}(x)$:

$$\begin{aligned}\mathsf{FO}|x,\eta\rangle_{XY}|\Upsilon(\vec{x},\vec{\text{и}})\rangle_F = |x,\eta\rangle_{XY} \sum_{\phi',f'\in\mathcal{F}(\mathcal{X}\setminus\{x\})} \frac{1}{\sqrt{N^{M-1}}}\,\omega_N^{\phi'\cdot f'}\; a_{f'\vec{\text{и}}'}(\mathcal{X}\setminus\{x\}) \\ \cdot|\phi_0\rangle_{F(0)}\cdots\left(\sum_{\zeta,z\in[N]}\frac{1}{\sqrt{N}}\,\omega_N^{\zeta\cdot z}\; a_{z\text{и}_x}(x)\;|\zeta-\eta\rangle_{F(x)}\right)\cdots|\phi_{M-1}\rangle_{F(M-1)},\end{aligned} \tag{66}$$

where $\vec{\text{и}}' \in \mathcal{Y}^{M-1}$ 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. (64) actually equals the left hand side that we just analyzed. Let us inspect $|Д(\vec{x},\vec{\text{и}})\rangle$ after application of the compressed oracle

$$\begin{aligned}&\mathsf{CFO}_{\mathfrak{D}}|x,\eta\rangle_{XY}|Д(\vec{x},\vec{\text{и}})\rangle_D = |x,\eta\rangle_{XY} \\ &\cdot\left(\sum_{\tilde{\text{и}}_x\neq 0}\alpha(x,\eta,\text{и}_x,\tilde{\text{и}}_x)\;|Д'_{\text{ADD/UPD}}\rangle_D + \alpha(x,\eta,\text{и}_x,0)\;|Д'_{\text{REM/NOT}}\rangle_D\right),\end{aligned} \tag{67}$$

where $\tilde{\text{и}}_x$ is the new value of $Д^Y(x)$ and и_x is the old content of the database. By $Д'_{\text{ADD/UPD}}$ we denote the database $Д(\vec{x},\vec{\text{и}})$ with entry $\tilde{\text{и}}_x \neq 0$, it corresponds to x being added or updated. By $Д'_{\text{REM/NOT}}$ we denote the database where $\tilde{\text{и}}_x = 0$, meaning x was removed from $Д$ or nothing happened. The function $\alpha(\cdot)$ denotes the corresponding amplitudes.

Before we proceed with decompression of the above state let us calculate the amplitudes α. Again using the definition 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 D^Y. We start with a database containing $(x,\text{и}_x)$, which we can always assume due to line 3 in Alg. 1. In the case that x was not already in $Д$ we have $\text{и}_x = 0$, otherwise it is the value defined in previous queries. The simplification we make is to describe $\mathsf{CFO}_{\mathfrak{D}}$ 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 D.3 for details). The calculation of $\mathsf{CFO}_{\mathfrak{D}}$ on a basis state follows:

$$|x,\eta\rangle_{XY}|x,\text{и}_x\rangle_D \overset{\mathsf{Samp}_{\mathfrak{D}}}{\mapsto} |x,\eta\rangle_{XY}\sum_{z\in[N]} a_{z\text{и}_x}(x)\;|x,z\rangle_D \tag{68}$$

$$\overset{\mathsf{QFT}_N^{D^Y}}{\mapsto} |x,\eta\rangle_{XY}\sum_{z\in[N]} a_{z\text{и}_x}(x)\sum_{\zeta\in[N]}\frac{1}{\sqrt{N}}\,\omega_N^{\zeta\cdot z}\;|x,\zeta\rangle_D \tag{69}$$

$$\overset{\mathsf{Sub}}{\mapsto}|x,\eta\rangle_{XY}\sum_{z,\zeta\in[N]} a_{z\text{и}_x}(x)\frac{1}{\sqrt{N}}\,\omega_N^{\zeta\cdot z}\;|x,\zeta-\eta\rangle_D \tag{70}$$

$$\overset{\mathsf{QFT}_N^{\dagger D^Y}}{\mapsto} |x,\eta\rangle_{XY}\sum_{z,\zeta\in[N]} a_{z\text{и}_x}(x)\frac{1}{\sqrt{N}}\,\omega_N^{\zeta\cdot z}\sum_{z'\in[N]}\frac{1}{\sqrt{N}}\,\bar{\omega}_N^{z'\cdot(\zeta-\eta)}\;|x,z'\rangle_D \tag{71}$$

$$=|x,\eta\rangle_{XY}\sum_{z\in[N]} a_{z\text{и}_x}(x)\underbrace{\sum_{z',\zeta\in[N]}\frac{1}{N}\,\omega_N^{\zeta\cdot z}\;\bar{\omega}_N^{z'\cdot(\zeta-\eta)}}_{=\bar{\omega}_N^{-z\cdot\eta}\;\delta(z',z)}\;|x,z'\rangle_D \tag{72}$$

$$\overset{\mathsf{Samp}_{\mathfrak{D}}^{\dagger D}(x)}{\mapsto}|x,\eta\rangle_{XY}\sum_{z\in[N]} a_{z\text{и}_x}(x)\;\omega_N^{z\cdot\eta}\sum_{\tilde{\text{и}}_x\in[N]}\bar{a}_{z\tilde{\text{и}}_x}(x)\;|x,\tilde{\text{и}}_x\rangle_D \tag{73}$$

$$= |x,\eta\rangle_{XY} \sum_{\tilde{и}_x \in [N]} \underbrace{\sum_{z\in[N]} a_{z и_x}(x)\ \omega_N^{z\cdot\eta}\ \bar{a}_{z\tilde{и}_x}(x)}_{:=\alpha(x,\eta,и_x,\tilde{и}_x)}\ |x,\tilde{и}_x\rangle_D \tag{74}$$

$$\overset{\mathsf{Rem}^D}{\mapsto} |x,\eta\rangle_{XY} \left(\sum_{и\in[N]\setminus\{0\}} \alpha(x,\eta,и_x,\tilde{и}_x)\ |x,\tilde{и}_x\rangle_D + \alpha(x,\eta,и_x,0)\ |\perp,0\rangle_D \right). \tag{75}$$

In the above equations we have defined α as

$$\alpha(x,\eta,и_x,\tilde{и}_x) := \sum_{z\in[N]} a_{z и_x}(x)\ \bar{a}_{z\tilde{и}_x}(x)\ \omega_N^{z\cdot\eta}. \tag{76}$$

After decompressing the state from Eq. (67), the resulting database state will be $\sum_{\tilde{и}_x\neq 0} \alpha(x,\eta,и_x,\tilde{и}_x)\ |\Upsilon(Д'_{\text{ADD/UPD}})\rangle + \alpha(x,\eta,и_x,0)\ |\Upsilon(Д'_{\text{REM/NOT}})\rangle_D$, where we overload notation of $|\Upsilon(\vec{x},\vec{и})\rangle$ to denote that $(\vec{x},\vec{и})$ consists of values in the respective databases. We can write down this state in more detail using Eq. (66):

$$\begin{aligned}
&\mathsf{Dec}_{\mathfrak{D}} \circ \mathsf{CFO}_{\mathfrak{D}} |x,\eta\rangle_{XY} |Д(\vec{x},\vec{и})\rangle_D \\
&= \sum_{\phi',f'\in\mathcal{F}(\mathcal{X}\setminus\{x\})} \frac{1}{\sqrt{N^{M-1}}}\ \omega_N^{\phi'\cdot f'}\ a_{f'\vec{и}'}(\mathcal{X}\setminus\{x\})\ |\phi_0\rangle_{F(0)} \cdots \\
&\cdot \left(\sum_{\tilde{и}_x\neq 0} \alpha(x,\eta,и_x,\tilde{и}_x) \sum_{\zeta,z\in[N]} \frac{1}{\sqrt{N}} \omega_N^{\zeta\cdot z} a_{z\tilde{и}_x}(x) |\zeta\rangle_{F(x)} \right. \\
&\left. + \alpha(x,\eta,и_x,0) \sum_{\zeta,z\in[N]} \frac{1}{\sqrt{N}} \omega_N^{\zeta\cdot z} a_{z0}(x) |\zeta\rangle_{F(x)} \right) \cdots |\phi_{M-1}\rangle_{F(M-1)}.
\end{aligned} \tag{77}$$

In the above equation we notice that

$$\begin{aligned}
&\sum_{\tilde{и}_x\neq 0} \alpha(x,\eta,и_x,\tilde{и}_x) \sum_{\zeta,z\in[N]} \frac{1}{\sqrt{N}} \omega_N^{\zeta\cdot z} a_{z\tilde{и}_x}(x) |\zeta\rangle_{F(x)} \\
&+ \alpha(x,\eta,и_x,0) \sum_{\zeta,z\in[N]} \frac{1}{\sqrt{N}} \omega_N^{\zeta\cdot z} a_{z0}(x) |\zeta\rangle_{F(x)} \\
&= \sum_{\zeta,z\in[N]} \frac{1}{\sqrt{N}}\ \omega_N^{\zeta\cdot z} \sum_{\tilde{и}_x\in[N]} \alpha(x,\eta,и_x,\tilde{и}_x)\ a_{z\tilde{и}_x}(x)\ |\zeta\rangle_{F(x)}
\end{aligned} \tag{78}$$

which 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

$$\begin{aligned}
&\sum_{\tilde{и}_x\in[N]} \alpha(x,\eta,и_x,\tilde{и}_x)\ a_{z\tilde{и}_x}(x) \\
&= \sum_{z'\in[N]} \underbrace{\sum_{\tilde{и}_x\in[N]} \bar{a}_{z'\tilde{и}_x}(x)\ a_{z\tilde{и}_x}(x)}_{=\delta_{z',z}} a_{z'и_x}(x)\ \omega_N^{z'\cdot\eta} = a_{z и_x}(x)\ \omega_N^{z\cdot\eta}.
\end{aligned} \tag{79}$$

Together with changing the variable $\zeta \mapsto \zeta - \eta$ and observing Eq. (66) we derive the claimed identity:

$$\begin{aligned}
&\mathsf{Dec}_{\mathfrak{D}} \circ \mathsf{CFO}_{\mathfrak{D}} |x,\eta\rangle_{XY} |Д(\vec{x},\vec{и})\rangle_D \\
&= \mathsf{FO}\ |x,\eta\rangle_{XY} |\Upsilon(\vec{x},\vec{и})\rangle = \mathsf{FO} \circ \mathsf{Dec}_{\mathfrak{D}}\ |x,\eta\rangle_{XY} |Д(\vec{x},\vec{и})\rangle_D.
\end{aligned} \tag{80}$$

□

B Full Proof of Theorem 10

Proof of Theorem 10. The proof works almost the same as the proof of Theorem 1 of [AHU19]. Let us state the analog of Lemma 5 from [AHU19].

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

$$\mathsf{V}_{R,i}|D\rangle_D|l_1,l_2,\ldots,l_d\rangle_L := \begin{cases} |D\rangle_D|l_1,l_2,\ldots,l_d\rangle_L & \text{if } R(|D\rangle_D)=0 \\ |D\rangle_D|l_1,\ldots,l_i\oplus 1,\ldots,l_d\rangle_L & \text{if } R(|D\rangle_D)=1 \end{cases}, \tag{81}$$

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 V_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 $l \neq 0^d$, i.e. $\tilde{P}_{\text{find}} := \left\| \mathbb{1}^{Q,D} \otimes (\mathbb{1}^L - |0^d\rangle_L\langle 0^d|)|\Psi_{\mathsf{B}}\rangle \right\|^2$.

To deal with relation R_1 we consider algorithms with all measurements postponed to the end of their operation; Instead of performing the actual measurement we save the outcome into a fresh quantum register—with V_R as in Alg. 4, note that prior to the measurement this fresh register can hold a superposition. Moreover we postpone the measurement of the auxiliary register until the very end of the run of the quantum algorithm. The coherent evaluation of R_1 happens in both algorithms. In addition, the proof below does not make use of the particular form of the unitaries that are applied between the measurements of R_2, so the evaluation of R_1 can be absorbed into the compressed oracle unitary.

Lemma 20 (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*

$$\left\| |\Psi_{\mathsf{A}}\rangle \otimes |0^d\rangle_L - |\Psi_{\mathsf{B}}\rangle \right\|^2 \leq (d+1)\tilde{P}_{\text{find}}. \tag{82}$$

Proof. This lemma can be proved in the same way as Lemma 5 of [AHU19]. 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 sub-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

$$\left\| |\Psi_{\mathsf{B}}\rangle - |\Psi_{\mathsf{A}}\rangle \otimes |0^d\rangle_L \right\|^2 = \left\| \sum_{i=1}^{d} |\Psi^i_{\mathsf{B}_{\text{count}}}) \right\|^2 + \tilde{P}_{\text{find}} \overset{\triangle}{\leq} \left(\sum_{i=1}^{d} \left\| |\Psi^i_{\mathsf{B}_{\text{count}}}) \right\| \right)^2 + \tilde{P}_{\text{find}}$$

$$\overset{\text{J-I}}{\leq} \underbrace{d \sum_{i=1}^{d} \left\| |\Psi^i_{\mathsf{B}_{\text{count}}}) \right\|^2}_{=\tilde{P}_{\text{find}}} + \tilde{P}_{\text{find}} = (d+1)\tilde{P}_{\text{find}}, \tag{83}$$

where $\triangle$ denotes the triangle inequality and J-I denotes the Jensen's inequality. 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 [AHU19] with the modifications shown in the above lemma. Using bounds on fidelity (Lemma 3 and Lemma 4 of [AHU19]) 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. □

C Full Proof of Lemma 13

Proof of Lemma 13. In Lemma 13 we prove a bound on the probability of finding a database fulfilling the relation of collision or a preimage of 0. This event of finding is denoted by Find. This relation is crucial in the proof of quantum indifferentiability of the sponge construction.

The first observation of the proof is that the probability of Find is the sum of probabilities that after the i'th query we find a database that fulfills the relation given that we did not find such database in any previous query. Hence, the proof focuses on calculating this probability for any i and then performing the sum. It is in general challenging to calculate such probability, and especially challenging to write out the joint state of the adversary and the oracle after i queries to the punctured oracle. Our solution to this challenge is to define an auxiliary state, called the *good* state, such that we exactly know what it looks like.

In a hybrid argument we introduce a sum over differences between the actual state and the good state. This is the focal point of our proof, if we find this difference, then we can work with the good state and calculate the bound on Find much easier. Technically the most difficult part of our proof is bounding the norm of the difference of the actual and the good states, it is the topic of the first Claim of the proof of Lemma 13.

The second important technical part is calculating the norm of finding a database that fulfills the relation in the good state after a query. Thankfully, after the analysis of the first problem we mentioned it is a relatively easy task.

Punctured oracles are defined in Definition 9. We start the proof by specifying some operations involved in that definition.

Introduction We define a "lazy" approach to calculating the number of non-empty entries in D. In this unitary we focus on using the ordered structure of D^X. We use the phase oracle instead of the standard oracle; in detailed calculations that we do later on in the proof, CPhO is easier to deal with than CStO.

Let us define $\mathsf{Queries}$, a unitary that outputs the size of a database. It acts on an auxiliary register S and is controlled on D. This unitary acts exactly like Alg. 1 in lines 1 and 13: it counts the number of non-padding ($x \neq \perp$) entries.

The full description of the measurement involves using an auxiliary register J—note Def. 4 measuring a relation—with a bit stating whether the database fulfills the relation. Then the

actual measurement is a computational basis measurement of register J. The measurement that we apply after $\mathsf{CPhO}_{\mathcal{Y}}$, in line 4 of Alg. 4 is

$$\mathsf{J}_R := \mathbb{1} \otimes |1\rangle_J\langle 1|, \tag{84}$$

$$\overline{\mathsf{J}}_R := \mathbb{1} \otimes |0\rangle_J\langle 0|. \tag{85}$$

In the following we focus on the punctured oracle just prior to measurement J_R. A unitary that omits the last step of Alg. 4 in $\mathsf{CPhO}_{\mathcal{Y}} \setminus R_{\text{preim}} \cup R_{\text{coll}}$ acts on registers ADJ, we define it as

$$\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R := \mathsf{Queries}^\dagger \circ \mathsf{V}_R \circ \mathsf{Queries} \circ \mathsf{CPhO}_{\mathcal{Y}}, \tag{86}$$

where the unitary V_R checks whether the queried values in registers D fulfill the relation R—in our case it is the collision and preimage relations from Eqs. (21), (30)—and saves the single bit answer to register J.

We proceed by rephrasing the definition of $\mathbb{P}[\text{Find} : \mathsf{A}[\mathsf{CPhO}_{\mathcal{Y}} \setminus R_{\text{preim}} \cup R_{\text{coll}}]]$, after that we treat the part specific to our relation. We follow Eq. (24) to analyze the probability of Find:

$$\mathbb{P}[\text{Find} : \mathsf{A}[\mathsf{CPhO}_{\mathcal{Y}} \setminus R_{\text{preim}} \cup R_{\text{coll}}]] = 1 - \left\| \left(\prod_{i=q}^{1} \overline{\mathsf{J}}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \right) |\Psi_0\rangle|0\rangle_J \right\|^2 \tag{87}$$

$$= 1 - \left\| \left(\prod_{i=q-1}^{1} \overline{\mathsf{J}}_R \mathsf{U}_j \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \right) |\Psi_0\rangle|0\rangle_J \right\|^2$$

$$+ \left\| \mathsf{J}_R \mathsf{U}_q \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \left(\prod_{i=q-1}^{1} \overline{\mathsf{J}}_R \mathsf{U}_j \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \right) |\Psi_0\rangle|0\rangle_J \right\|^2 = \cdots = \tag{88}$$

$$= \sum_{i=1}^{q} \left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \underbrace{\left(\prod_{j=i-1}^{1} \overline{\mathsf{J}}_R \mathsf{U}_j \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \right) |\Psi_0\rangle|0\rangle_J}_{:=\mathsf{U}_{i-1}|\Phi_{i-1}\rangle} \right\|^2 \tag{89}$$

$$= \sum_{i=1}^{q} \left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle \right\|^2, \tag{90}$$

where $|\Psi_0\rangle$ is the initial state of the adversary. Note that in the definition

$$|\Phi_{i-1}\rangle := \mathsf{U}_{i-1}^\dagger \left(\prod_{j=i-1}^{1} \overline{\mathsf{J}}_R \mathsf{U}_j \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \right) |\Psi_0\rangle|0\rangle_J \tag{91}$$

we use $[\mathsf{U}_{i-1}, \overline{\mathsf{J}}_R] = 0$[5]. Here, the second and third equations follow from the fact that $\||v\rangle\|^2 = \|\mathsf{P}|v\rangle\|^2 + \|(\mathbb{1} - \mathsf{P})|v\rangle\|^2$ for all $|v\rangle$ and projectors P.

In what follows we analyze $\|\mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle\|^2$. Our approach is to propose a state $|\Psi_{i-1}^{\text{Good}}\rangle$, close to the original $|\Phi_{i-1}\rangle$, for which bounding $\left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \mathsf{U}_{i-1} |\Psi_{i-1}^{\text{Good}}\rangle|0\rangle_J \right\|^2$ is easy. The intuition behind $|\Psi_{i-1}^{\text{Good}}\rangle$ is to have a superposition over databases that do not contain $y = 0$ and are collision free for the queried values.

To define the good state we specify the set of bad databases $D \in R$. For the relation $R_{\text{preim}} \cup R_{\text{coll}}$ we have

$$\mathcal{B}(s) := [N]^s \setminus \{(y_1, \ldots, y_s) \in [N]^s : \text{all } y_i \text{ are distinct and } \neq 0\}, \tag{92}$$

$$\mathcal{B}(1 \mid D) := \{y\}_{y \in D^Y} \cup \{0\}. \tag{93}$$

[5]The commutator of two operators (matrices) is defined as $[\mathsf{A}, \mathsf{B}] := \mathsf{AB} - \mathsf{BA}$.

The second set defined above is the subset of the codomain of the sampled function corresponding to the new value creating a collision or being a preimage of 0. To better understand $\mathcal{B}(1 \mid D)$ let us assume $D \notin R$ and x is some input $\notin D^X$. Then $\mathcal{B}(1 \mid D)$ is the set of y such that $D \cup \{(x, y)\} \in R$. We also define a coefficient $b(s)$ defined as

$$b(s) := |\mathcal{B}(1 \mid D)|\,, \text{ where } D \notin \mathcal{B}(s-1), \tag{94}$$

where we use the fact that $|\mathcal{B}(1 \mid D)|$ depends only on the size of D and not the actual contents of it. We define $\mathcal{B}(1 \mid D)$ in a way specific to $R_{\text{coll}} \cup R_{\text{preim}}$ but the definition can be easily extended to other relations. As examples consider R_{preim}, then $b(s) = 1$, there is just one value $y = 0$ that causes a fresh query to be in relation; For R_{coll} we have $b(s) = s - 1$, the new y can be any of the previously queried values to make D fulfill the relation. Finally for our relation $R_{\text{preim}} \cup R_{\text{coll}}$ we have $b(s) = s$, database D consists of $s-1$ distinct values that are distinct from 0, matching any of them or 0 causes $D^Y \cup \{y\}$ to be in $\mathcal{B}(s)$. Throughout the rest of this proof we do not evaluate $b(s)$, which makes it is easier to reuse the proof for other relations.

The good state In what follows we write $\vec{x}$ to denote all the previous inputs asked by the adversary and (x, η) is the last query. The state $|\Psi^{\text{Good}}_{i,R}\rangle_{AD}$ corresponds to the adversary's state just after the i-th query and before the application of U_i. The size of the database s depends on whether the new query x was added to, updated, or removed from the database, it equals $|\vec{x} \cup \{x\}|$, $|\vec{x}|$, or $|\vec{x} \setminus \{x\}|$ respectively. After i queries s can range from 0 to i and the joint state of A and the oracle can be a superposition over different database sizes. We denote the outputs given to A by $\vec{y} := (y_1, \ldots, y_s)$. When we use set operations on vectors we mean a set consisting of entries of $\vec{x}$, there are no repetitions in the vector as this is an invariant of the oracle. By $D(\perp)$ we denote the part of the database containing empty entries. Adversary's work register is denoted by A^W and its contents by $\psi(x, \eta, \vec{x}, \vec{\eta}, w)$, where w can be any value of finite size. We define the good state as:

$$\begin{aligned}
|\Psi^{\text{Good}}_{i,R}\rangle_{AD} := &\sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x, \eta\rangle_{A^{XY}} |\psi(x, \eta, \vec{x}, \vec{\eta}, w)\rangle_{A^W} \\
&\sum_{\vec{y} \notin \mathcal{B}(s)} \frac{1}{\sqrt{(N - b(1))(N - b(2)) \cdots (N - b(s))}} \omega_N^{\vec{\eta} \cdot \vec{y}} |(x_1, y_1), \ldots, (x_s, y_s)\rangle_{D(\vec{x})} \\
&\sum_{y_{s+1}, \ldots, y_q \in [N]} \frac{1}{\sqrt{N^{q-s}}} |(\perp, y_{s+1}), \ldots, (\perp, y_q)\rangle_{D(\perp)}.
\end{aligned} \tag{95}$$

In case we have added x to D, the database above contains (x, y_j). In the rest of the proof we omit the subscript R, however note that $|\Psi^{\text{Good}}_i\rangle$ does indeed depend on R.

In the rest of the proof we analyze how far apart the state $|\Psi^{\text{Good}}_{i-1}\rangle$ is after a query from $|\Psi^{\text{Good}}_i\rangle$. To achieve this goal we inspect in detail the state $\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \mathsf{U}_{i-1} |\Psi^{\text{Good}}_{i-1}\rangle_{AD} |0\rangle_J$. We distinguish different modes of operation: ADD when the queried x is added to D, UPD when x was already in D and is not removed from the database, REM when we remove x from D, and NOT where register A^Y is in state $|0\rangle$. These modes correspond to different branches of superposition in $\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R \mathsf{U}_{i-1} |\Psi^{\text{Good}}_{i-1}\rangle_{AD} |0\rangle_J$. We write

$$\mathsf{U}_{i-1} |\Psi^{\text{Good}}_{i-1}\rangle_{AD} |0\rangle_J = |\xi_{i-1}(\text{ADD})\rangle + |\xi_{i-1}(\text{UPD})\rangle + |\xi_{i-1}(\text{REM})\rangle + |\xi_{i-1}(\text{NOT})\rangle \tag{96}$$

and analyze the action of $\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R$ on the above states separately.

For $|\xi_{i-1}(\text{NOT})\rangle$ there is no change to the state. Adding a new entry to a database results in setting the register corresponding to x to $\sum_{y_{s+1} \in [N]} \frac{1}{\sqrt{N}} \omega_N^{\eta y_{s+1}} |x, y_{s+1}\rangle$, just like expected from

a phase oracle for the uniform distribution. After applying $\mathsf{Queries}^\dagger \circ \mathsf{V}_R \circ \mathsf{Queries}$ the state is:

$$\begin{aligned}
&\mathrm{ADD} : \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R |\xi_{i-1}(\mathrm{ADD})\rangle|0\rangle_J = \sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x,\eta\rangle_{A^{XY}} |\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W} \\
&\sum_{\vec{y}\notin\mathcal{B}(s)} \frac{1}{\sqrt{(N-b(1))\cdots(N-b(s))}} \omega_N^{\vec{\eta}\cdot\vec{y}} |(x_1,y_1),\ldots,(x_s,y_s)\rangle_{D(\vec{x})} \\
&\Bigg(\sqrt{\frac{N-b(s+1)}{N}} \underbrace{\sum_{y_{s+1}\notin\mathcal{B}(1|D(\vec{x}))} \frac{1}{\sqrt{N-b(s+1)}} \omega_N^{\eta y_{s+1}} |x,y_{s+1}\rangle|0\rangle_J}_{|\Psi_i^{\mathrm{Good}}(\mathrm{ADD},s)\rangle} \\
&+ \sqrt{\frac{b(s+1)}{N}} \sum_{y_{s+1}\in\mathcal{B}(1|D(\vec{x}))} \frac{1}{\sqrt{b(s+1)}} \omega_N^{\eta y_{s+1}} |x,y_{s+1}\rangle|1\rangle_J \Bigg) \\
&\sum_{y_{s+2},\ldots,y_q\in[N]} \frac{1}{\sqrt{N^{q-s-1}}} |(\perp,y_{s+2}),\ldots,(\perp,y_q)\rangle_{D(\perp)},
\end{aligned} \tag{97}$$

where the appropriate position of register J is after D. By $|\Psi_i^{\mathrm{Good}}(\mathrm{ADD};s)\rangle$ we mean a state equal to the above state but with just the underlined part in the parentheses. We add s as the argument to specify the size of the database.

For $|\xi_{i-1}(\mathrm{UPD})\rangle$ and $|\xi_{i-1}(\mathrm{REM})\rangle$, we treat the updated x as the last one in D, this does not have to be true but it simplifies notation. Note that we want the corresponding y_s to depend on previous queries but not the other way around, this assumption is without loss of generality as there is no fixed order for $\sum_{\vec{y}}$. The empty register is moved to the back of D, we do not write it out for simplicity but still consider it done.

$$\begin{aligned}
&\mathrm{UPD/REM} : \mathsf{CPhO}_{\mathcal{Y}} \left(|\xi_{i-1}(\mathrm{UPD})\rangle + |\xi_{i-1}(\mathrm{REM})\rangle \right) \\
&= \sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x,\eta\rangle_{A^{XY}} |\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W} \\
&\sum_{\vec{y}\notin\mathcal{B}(s-1)} \frac{1}{\sqrt{(N-b(1))(N-b(2))\cdots(N-b(s-1))}} \omega_N^{\vec{\eta}\cdot\vec{y}} |(x_1,y_1),\ldots,(x_{s-1},y_{s-1})\rangle_{D(\vec{x}\setminus\{x\})} \\
&\Bigg(\sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \frac{1}{\sqrt{N-b(s)}} \omega_N^{(\eta_s+\eta)y_s} |x,y_s\rangle_{D(x)} \\
&- \frac{1}{\sqrt{N(N-b(s))}} \sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \omega_N^{(\eta_s+\eta)y_s} \sum_{y'_s\in[N]} \frac{1}{\sqrt{N}} |x,y'_s\rangle_{D(x)} \\
&+ \frac{1}{\sqrt{N(N-b(s))}} \sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \omega_N^{(\eta_s+\eta)y_s} \sum_{y'_s\in[N]} \frac{1}{\sqrt{N}} |\perp,y'_s\rangle_{D(x)} \Bigg) \\
&\sum_{y_{s+1},\ldots,y_q\in[N]} \frac{1}{\sqrt{N^{q-s}}} |(\perp,y_{s+1}),\ldots,(\perp,y_q)\rangle_{D(\perp)}.
\end{aligned} \tag{98}$$

Whether we are in the branch UPD or REM depends on whether $\eta = -\eta_s$ or not.

When the database is updated we have the following state after the query:

$$\begin{aligned}
&\mathrm{UPD} : \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R |\xi_{i-1}(\mathrm{UPD})\rangle|0\rangle_J = \sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x,\eta\rangle_{A^{XY}} |\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W} \\
&\sum_{\vec{y}\notin\mathcal{B}(s-1)} \frac{1}{\sqrt{(N-b(1))\cdots(N-b(s-1))}} \omega_N^{\vec{\eta}\cdot\vec{y}} |(x_1,y_1),\ldots,(x_{s-1},y_{s-1})\rangle_{D(\vec{x}\setminus\{x\})}
\end{aligned}$$

$$
\Bigg(\underbrace{\sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\frac{1}{\sqrt{N-b(s)}}\omega_N^{(\eta_s+\eta)y_s}|x,y_s\rangle_{D(x)}|0\rangle_J}_{|\Psi_j^{\text{Good}}(\text{UPD};s)\rangle}
$$

$$
-\underbrace{\frac{1}{\sqrt{N(N-b(s))}}\sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\omega_N^{(\eta_s+\eta)y_s}\sum_{y'_s\in[N]}\frac{1}{\sqrt{N}}|\perp,y'_s\rangle_{D(x)}|0\rangle_J}_{|\Psi_{i,1}^{\text{Bad}}(\text{UPD};s)\rangle}
$$

$$
+\underbrace{\frac{1}{N}\sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\omega_N^{(\eta_s+\eta)y_s}\sum_{y'_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\frac{1}{\sqrt{N-b(s)}}|x,y'_s\rangle_{D(x)}|0\rangle_J}_{|\Psi_{i,2}^{\text{Bad}}(\text{UPD};s)\rangle}
$$

$$
+\sqrt{\frac{b(s)}{N^2(N-b(s))}}\sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\omega_N^{(\eta_s+\eta)y_s}\sum_{y'_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\frac{1}{\sqrt{b(s)}}|x,y'_s\rangle_{D(x)}|1\rangle_J\Bigg)
$$

$$
\sum_{y_{s+1},\ldots,y_q\in[N]}\frac{1}{\sqrt{N^{q-s}}}|(\perp,y_{s+1}),\ldots,(\perp,y_q)\rangle_{D(\perp)}. \tag{99}
$$

In the above state we have simplified the sum $\sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}=-\sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}$. Register J is supposed to be placed after D, for the sake of presentation though, we put it in the middle. By $|\Psi_i^{\text{Good}}(\text{UPD};s)\rangle$, $|\Psi_{i,1}^{\text{Bad}}(\text{UPD};s)\rangle$, and $|\Psi_{i,2}^{\text{Bad}}(\text{UPD};s)\rangle$ we mean the whole state with just the underlined states in the parentheses equals the given state. We add s as the argument to specify the size of the database.

After removing an element from the database we have:

$$
\text{REM}:\mathsf{CPhO}_{\mathcal{Y}}\setminus\mathsf{V}_R|\xi_{i-1}(\text{REM})\rangle|0\rangle_J=\sum_{x,\eta,\vec{x},\vec{\eta},w}\alpha_{x,\eta,\vec{x},\vec{\eta},w}|x,\eta\rangle_{A^{XY}}|\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W}
$$

$$
\sum_{\vec{y}\notin\mathcal{B}(s-1)}\frac{1}{\sqrt{(N-b(1))\cdots(N-b(s-1))}}\omega_N^{\vec{\eta}\cdot\vec{y}}|(x_1,y_1),\ldots,(x_{s-1},y_{s-1})\rangle_{D(\vec{x}\setminus\{x\})}
$$

$$
\Bigg(\sqrt{\frac{N-b(s)}{N}}\underbrace{\sum_{y_s\in[N]}\frac{1}{\sqrt{N}}|\perp,y_s\rangle_{D(x)}|0\rangle_J}_{|\Psi_i^{\text{Good}}(\text{REM},s)\rangle}
$$

$$
+\underbrace{\frac{b(s)}{N}\sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\frac{1}{\sqrt{N-b(s)}}|x,y_s\rangle_{D(x)}|0\rangle_J}_{|\Psi_i^{\text{Bad}}(\text{REM};s)\rangle}
$$

$$
-\frac{\sqrt{b(s)(N-b(s))}}{N}\sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\frac{1}{\sqrt{b(s)}}|x,y_s\rangle_{D(x)}|1\rangle_J\Bigg)
$$

$$
\sum_{y_{s+1},\ldots,y_q\in[N]}\frac{1}{\sqrt{N^{q-s}}}|(\perp,y_{s+1}),\ldots,(\perp,y_q)\rangle_{D(\perp)}. \tag{100}
$$

We want to show that after any query, $|\Phi_i\rangle_{ADJ}$ is close to $|\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J$:

Claim 21. *For states defined as above we have*

$$
\left\||\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J-|\Phi_i\rangle_{ADJ}\right\|\leq\frac{2i^{5/2}}{\sqrt{N(N-q)}}. \tag{101}
$$

Proof. We are going to prove the statement by recursion over the number of queries made by the adversary. For $i = 0$ the statement is true, as $|\Psi_0^{\text{Good}}\rangle|0\rangle_J = |\Phi_0\rangle = |\Psi_0\rangle|0\rangle_J$. Next we proceed as follows:

$$\left\||\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J - |\Phi_i\rangle_{ADSJ}\right\| = \left\||\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J - \overline{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{i-1}|\Phi_{i-1}\rangle_{ADJ}\right\| \tag{102}$$

$$\leq \left\||\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J - \overline{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{i-1}|\Psi_{i-1}^{\text{Good}}\rangle_{AD}|0\rangle_J\right\|$$

$$+ \left\|\overline{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{i-1}|\Psi_{i-1}^{\text{Good}}\rangle_{AD}|0\rangle_J - \overline{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{i-1}|\Phi_{i-1}\rangle_{ADJ}\right\| \tag{103}$$

$$\leq \varepsilon_{\text{step}}(i) + \left\||\Psi_{i-1}^{\text{Good}}\rangle_{AD}|0\rangle_J - |\Phi_{i-1}\rangle_{ADJ}\right\| \leq \sum_{j=1}^{i} \varepsilon_{\text{step}}(j). \tag{104}$$

We just need to calculate $\varepsilon_{\text{step}}(j) := \left\||\Psi_j^{\text{Good}}\rangle|0\rangle - \overline{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle\right\|_2$.

Introduction From Eqs. (97), (99), and (100) we know how querying works for $|\Psi_{j-1}^{\text{Good}}\rangle$, now we distinguish two types of errors compared to $|\Psi_j^{\text{Good}}\rangle$: an additive error of adding a small-weight state to the original one and a multiplicative error where one branch of the superposition is multiplied by some factor.

The additive error includes all states of small-weight states multiplied by $|0\rangle_J$: $|\Psi_{j,1}^{\text{Bad}}(\text{UPD})\rangle$ and $|\Psi_{j,2}^{\text{Bad}}(\text{UPD})\rangle$ in Eq. (99) and $|\Psi_j^{\text{Bad}}(\text{REM})\rangle$ in Eq. (100).

In the branches of the superposition where we add a new entry to the database we see that we recover $|\Psi_j^{\text{Good}}\rangle|0\rangle_J$ after multiplying a branch of $\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J$ by $\sqrt{\frac{N-b(s+1)}{N}}$ (Eq. (97)) or by $\sqrt{\frac{N-b(s)}{N}}$ (Eq. (100)).

Our approach to the rest of the proof consists of first dealing with the additive and later the multiplicative error. To this end let us define $|\psi_j^{\times}\rangle_{ADJ}$ as the state $\overline{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J$ with all branches classified as the additive error excluded. By classified as the additive error we mean states with superscript Bad and highlighted in red in Eqs. (97, 99, 100). The new state is defined as

$$|\psi_j^{\times}\rangle_{ADJ} = \left(\sum_s |\Psi_j^{\text{Good}}(\text{NOT}; s)\rangle + \sqrt{\frac{N-b(s+1)}{N}}|\Psi_j^{\text{Good}}(\text{ADD}; s)\rangle + |\Psi_j^{\text{Good}}(\text{UPD}; s)\rangle\right.$$

$$\left. + \sqrt{\frac{N-b(s)}{N}}|\Psi_j^{\text{Good}}(\text{REM}; s)\rangle\right)|0\rangle_J, \tag{105}$$

where the states above correspond to branches of superposition where we do nothing (NOT, for $\eta = 0$), add an entry, update the database, and remove an entry from D. Bounding the difference of the states is done as follows

$$\left\||\Psi_j^{\text{Good}}\rangle|0\rangle_J - \overline{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J\right\|$$

$$\leq \left\||\Psi_j^{\text{Good}}\rangle|0\rangle_J - |\psi_j^{\times}\rangle_{ADJ}\right\| + \left\||\psi_j^{\times}\rangle_{ADJ} - \overline{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J\right\|. \tag{106}$$

The second term above is just the norm of all states amplifying the additive error—we call them the bad states.

We bound the additive error $\||\psi_j^{\times}\rangle_{ADJ} - \overline{\mathsf{J}}_R\mathsf{H} \setminus \mathsf{V}_R\mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J\|$ by first splitting the three cases underlined above:

$$\left\||\Psi_j^{\text{Bad}}\rangle\right\| \leq \left\||\Psi_{j,1}^{\text{Bad}}(\text{UPD})\rangle\right\| + \left\||\Psi_{j,2}^{\text{Bad}}(\text{UPD})\rangle\right\| + \left\||\Psi_j^{\text{Bad}}(\text{REM})\rangle\right\|, \tag{107}$$

where $|\Psi_j^{\text{Bad}}\rangle$ is the sum of all three bad states, the bound follows from the triangle inequality.

Calculating all of the three norms above is done by first focusing on particular sizes of databases:

$$\left\| |\Psi_j^{\text{Bad}}\rangle \right\| = \sqrt{\sum_{s=0}^{j} |\beta(s)|^2 \left\| |\Psi_j^{\text{Bad}}(s)\rangle \right\|^2}, \tag{108}$$

where $\beta(s)$ is the amplitude of the good state projected to states with the specified parameters: For a projector P_s to databases of size s we have $\beta(s) := \mathsf{P}_s|\Psi_j^{\text{Good}}\rangle$ and $|\Psi_j^{\text{Bad}}(s)\rangle := \mathsf{P}_s|\Psi_j^{\text{Bad}}\rangle$.

Additive errors Dealing with additive errors, we begin with the UPD branch. In the bad states in the UPD case, Eq. (99), we need to take special care of $\sum_{y_s \in \mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \omega_N^{(\eta_s+\eta)y_s}$; This is a a complex number that depends on η_s, so it enters the norm in a non-trivial way. The first step is a change of variables: Instead of summing over elements of of the bad state we sum over $y_s \in [b(s)]$ and change y_s in the expression to $\mathcal{B}(1 \mid D(\vec{x} \setminus \{x\}))(y_s)$, by which we denote the y_s-th element of $\mathcal{B}(1 \mid D(\vec{x} \setminus \{x\}))$. Note that there is a natural order in the bad set, as $\mathcal{Y} = [N]$.

Given the change of variables we can use the triangle inequality to focus on the norm of a state with a single phase factor $\omega_N^{(\eta_s+\eta)\mathcal{B}(1|D(\vec{x}\setminus\{x\}))(y_s)}$, instead of the whole sum:

$$\left\| |\Psi_j^{\text{Bad}}(\text{UPD}; s)\rangle \right\| \leq \sum_{y_s \in [b(s)]} \left\| |\Psi_j^{\text{Bad}}(\text{UPD}; s, \mathcal{B}(1 \mid D(\vec{x} \setminus \{x\}))(y_s))\rangle \right\|, \tag{109}$$

where we omit the index of the UPD errors because the techniques here work in almost the same way for both states. The input $D(\vec{x} \setminus \{x\})$ should not be treated as an actual argument of the state, we still consider the superposition over different inputs, we just mean that in the state $|\Psi_j^{\text{Bad}}(\text{UPD}; s)\rangle$ we change the variable y_s. In what follows we denote the state on the right hand side of the above equation by $|\Psi_j^{\text{Bad}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle$.

Now we focus on the state with a fixed $\mathcal{B}'(y_s)$, we bound the norm of this state.

Claim 22. *For all $y_s \in [b(s)]$*

$$\left\| |\Psi_{j,1}^{\text{Bad}}(\text{UPD}; s, \mathcal{B}(1 \mid D(\vec{x} \setminus \{x\}))(y_s))\rangle \right\| \leq \sqrt{\frac{b(s)}{N(N-b(s))}} \text{ and} \tag{110}$$

$$\left\| |\Psi_{j,2}^{\text{Bad}}(\text{UPD}; s, \mathcal{B}(1 \mid D(\vec{x} \setminus \{x\}))(y_s))\rangle \right\| \leq \frac{\sqrt{b(s)}}{N}. \tag{111}$$

Proof. Our idea for the proof is to first show that the norm of a good state in the UPD branch with a modified sum over y_s is not greater than 1. Then to prove that the norm of $|\Psi_j^{\text{Bad}}(\text{UPD}; s, \mathcal{B}(1 \mid D(\vec{x} \setminus \{x\}))(y_s))\rangle$ multiplied by the corresponding right hand side of Eq. (110) equals the norm of the good state we mentioned earlier.

We start by defining two states:

$$\begin{aligned}
&\sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x,\eta\rangle_{A^{XY}} |\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W} \\
&\sum_{\vec{y} \notin \mathcal{B}(s-1)} \frac{1}{\sqrt{(N-b(1))\cdots(N-b(s-1))}} \omega_N^{\vec{\eta}\cdot\vec{y}} |(x_1,y_1),\ldots,(x_{s-1},y_{s-1})\rangle_{D(\vec{x}\setminus\{x\})} \\
&\sum_{y_{s+1},\ldots,y_q \in [N]} \frac{1}{\sqrt{N^{q-s}}} |(\perp,y_{s+1}),\ldots,(\perp,y_q)\rangle_{D(\perp)} \\
&\otimes \begin{cases} \sum_{y_s \in \mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \frac{1}{\sqrt{b(s)}} \omega_N^{(\eta_s+\eta)y_s} |x,y_s\rangle_{D(x)} =: |\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle \\ \sum_{y_s \in [N]} \frac{1}{\sqrt{N}} \omega_N^{(\eta_s+\eta)y_s} |x,y_s\rangle_{D(x)} =: |\widetilde{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle \end{cases}.
\end{aligned} \tag{112}$$

The first one, $|\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle$ is the one that we use in the last step of the proof, as described in the previous paragraph. The second one will be used to show that the norm of $|\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle$ is bounded by 1.

One more introductory statement that we need to prove is that $\left\||\widetilde{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle\right\| \leq 1$. To this end let us remind ourselves that the good state is a state interacting with the not-punctured oracle for j queries, projected to databases that are not in R, and normalized. Let us consider a projection that just omits register $D(x)$ when bringing D to be not in R. Using this latter projection on a state interacting with the not-punctured oracle results in the state $|\widetilde{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle$. Hence $\left\||\widetilde{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle\right\| \leq 1$, just like $\left\||\Psi_j^{\text{Good}}(\text{UPD}; s)\rangle\right\| \leq 1$. The inequality comes from excluding a single branch of the superposition in $|\widetilde{\Psi}_j^{\text{Good}}(s)\rangle$.

The fact that the state with $\sum_{y_s \in [N]}$ is sub-normalized is important because now we can bound the norm of $|\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle$. Having in mind that $\sum_{y_s \in \mathcal{B}(1|D(\vec{x}\setminus\{x\}))} = \sum_{y_s \in [N]} - \sum_{y_s \notin \mathcal{B}(1|D(\vec{x}\setminus\{x\}))}$ we see that

$$\begin{aligned} & b(s) \left\||\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle\right\|^2 \\ & = N \left\||\widetilde{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle\right\|^2 - (N - b(s)) \left\||\Psi_j^{\text{Good}}(\text{UPD}; s)\rangle\right\|^2 \leq b(s), \end{aligned} \tag{113}$$

hence $\left\||\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle\right\|^2 \leq 1$.

From the definition of the states we know that

$$\sum_{y_s \in [b(s)]} \left\||\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle\right\|^2 = \left\||\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s)\rangle\right\|^2 \leq 1. \tag{114}$$

Hence for every y_s we have $\left\||\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle\right\| \leq 1$.

Now we can use the state $|\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle$ to analyze the norm of $|\Psi_j^{\text{Bad}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle$. First let us inspect the norm squared of the bad state:

$$\begin{aligned} & \left\||\Psi_j^{\text{Bad}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle\right\|^2 = \sum_{x,\eta,\vec{x},\vec{\eta}',\vec{\eta},w',w} \sum_{\eta_s',\eta_s} \bar{\alpha}'_{x,\eta,\vec{x},\vec{\eta}',\eta_s',w'} \alpha'_{x,\eta,\vec{x},\vec{\eta},\eta_s,w} \\ & \langle \psi(x,\eta,\vec{x},\vec{\eta}',\eta_s',w') | \psi(x,\eta,\vec{x},\vec{\eta},\eta_s,w)\rangle \\ & \sum_{\vec{y} \notin \mathcal{B}(s-1)} \frac{1}{(N-b(1))\cdots(N-b(s))} \bar{\omega}_N^{\vec{\eta}'\cdot\vec{y}} \omega_N^{\vec{\eta}\cdot\vec{y}} \\ & \frac{1}{N^2(N-b(s))} \bar{\omega}_N^{(\eta_s'+\eta)\mathcal{B}'(y_s)} \omega_N^{(\eta_s+\eta)\mathcal{B}'(y_s)} \underbrace{\sum_{y_s' \in [\nu]}}_{=\nu}, \end{aligned} \tag{115}$$

where $\nu = N$ for $|\Psi_{j,1}^{\text{Bad}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle$ and $\nu = N - b(s)$ for $|\Psi_{j,2}^{\text{Bad}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle$ (in the second case the sum goes over $y_s' \notin \mathcal{B}(1 \mid D(\vec{x} \setminus \{x\}))$ instead of $[\nu]$). It is easy to notice, that the only difference between Eq. (115) and norm squared of $|\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle$ lies in the factor $\frac{\nu}{N^2(N-b(s))}$. This factor in the modified good state equals $\frac{1}{b(s)}$. This observation implies that

$$\left\||\Psi_j^{\text{Bad}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle\right\| = \sqrt{\frac{b(s)\cdot\nu}{N^2(N-b(s))}} \left\||\overline{\Psi}_j^{\text{Good}}(\text{UPD}; s, \mathcal{B}'(y_s))\rangle\right\|. \tag{116}$$

Together with the bound on the norm in the left hand side this proves the claimed bounds. □

Claim 22, together with the bound from Eq. (109) gives us:

$$\left\| |\Psi^{\text{Bad}}_{j,1}(\text{UPD}; s)\rangle \right\| \leq \frac{b(s)^{3/2}}{\sqrt{N(N-b(s))}}, \tag{117}$$

$$\left\| |\Psi^{\text{Bad}}_{j,2}(\text{UPD}; s)\rangle \right\| \leq \frac{b(s)^{3/2}}{N}. \tag{118}$$

The bounds from Eq. (117) in Eq. (108) give us the bound on the additive error in the UPD branch. The additive error for the REM branch ($|\Psi^{\text{Bad}}_j(\text{REM})\rangle$ in Eq. (100)) is much easier to calculate: As register $D(x)$ is normalized and all the rest of the state is the same as $|\Psi^{\text{Good}}_j(\text{REM})\rangle$, the only error comes from the factor $\frac{b(s)}{N}$. To calculate the norm of the state we can follow the analysis of Eq. (115). Finally we get:

$$\left\| |\Psi^{\text{Bad}}_{j,1}(\text{UPD})\rangle \right\| \leq \max_s \left\{ \frac{b(s)^{3/2}}{\sqrt{N(N-b(s))}} \right\}, \tag{119}$$

$$\left\| |\Psi^{\text{Bad}}_{j,2}(\text{UPD})\rangle \right\| \leq \max_s \left\{ \frac{b(s)^{3/2}}{N} \right\}, \tag{120}$$

$$\left\| |\Psi^{\text{Bad}}_{j}(\text{REM})\rangle \right\| \leq \max_s \left\{ \frac{b(s)}{N} \right\}, \tag{121}$$

where $s \leq j-1$.

Multiplicative errors The multiplicative error is a factor that multiplies a part of the state $|\psi^{\times}_j\rangle_{ADJ}$. Similarly as before we need to take care of the fact that the joint state of the adversary and the oracle is a sum over databases of different sizes and queries to different interfaces:

$$|\psi^{\times}_j\rangle = \sum_s |\psi^{\times}_j(s)\rangle, \tag{122}$$

where the states $|\psi^{\times}_j(s)\rangle$ are orthogonal. The above is also true for $|\Psi^{\text{Good}}_j\rangle = \sum_s |\Psi^{\text{Good}}_j(s)\rangle$.

There are two sources of multiplicative errors, ADD from Eq. (97) and REM from Eq. (100), we split the two sources with the triangle inequality. We deal with both in the same way, just the final bound is different.

Let us write down the two parts, one affected by the error and the second not:

$$|\Psi^{\text{Good}}_j\rangle_{AD}|0\rangle_J = \sum_s \alpha(s)|\varphi_1(s)\rangle + \beta(s)|\varphi_2(s)\rangle, \tag{123}$$

$$|\psi^{\times}_j\rangle_{ADJ} = \sum_s \alpha(s)|\varphi_1(s)\rangle + \sqrt{1-e}\beta(s)|\varphi_2(s)\rangle, \tag{124}$$

where $\sqrt{1-e}$ is the multiplicative error, in the case ADD the error is $e = \frac{b(s+1)}{N}$ and $e = \frac{b(s)}{N}$ in the case REM. We know that $\sum_s |\alpha(s)|^2 + |\beta(s)|^2 \leq 1$, because we excluded a single branch of the superposition, for ADD and REM. This inequality implies $\sum_s |\beta(s)|^2 \leq 1$. We continue with the bound

$$\left\| |\psi^{\times}_j\rangle_{ADJ} - |\Psi^{\text{Good}}_j\rangle_{AD}|0\rangle_J \right\| = \left\| \sum_s (1-\sqrt{1-e})\beta(s)|\varphi_2(s)\rangle \right\| \tag{125}$$

$$= \sqrt{\sum_s (1-\sqrt{1-e})^2 |\beta(s)|^2} \leq \max_s\{1-\sqrt{1-e}\} \leq \max_s\{e\}, \tag{126}$$

Maximization is done over $s \leq j-1$.

Bound on one step From Eqs. (106), (119), and (126) (for the two sources of error) the bound on the single step is

$$\varepsilon_{\text{step}}(j) \leq \max_{s\leq j-1}\left\{\frac{b(s)^{3/2}}{\sqrt{N(N-b(s))}}+\frac{b(s)^{3/2}}{N}+2\frac{b(s)}{N}+\frac{b(s+1)}{N}\right\} \tag{127}$$

and the final bound is

$$\begin{aligned}
&\left\||\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J - |\Phi_i\rangle_{ADJ}\right\| \\
&\leq \sum_{j=1}^{i}\max_{s\leq j-1}\left\{\frac{b(s)^{3/2}}{\sqrt{N(N-b(s))}}+\frac{b(s)^{3/2}}{N}+2\frac{b(s)}{N}+\frac{b(s+1)}{N}\right\} &(128)\\
&\leq 5\sum_{j=1}^{i}\max_{s\leq j-1}\left\{\frac{b(s+1)^{3/2}}{\sqrt{N(N-b(q))}}\right\}. &(129)
\end{aligned}$$

For our relation $R_{\text{coll}} \cup R_{\text{preim}}$ we know that $b(s) = s$. To get the claimed bound we note that $\sum_{j=1}^{i} j^{3/2} \leq \int_1^i \mathrm{d}j j^{3/2} \leq \int_0^i \mathrm{d}j j^{3/2} = \frac{2}{5}i^{5/2}$. Moreover $b(s) \leq b(q)$, which we use in the denominator. □

To calculate the probability of measuring $R = R_{\text{coll}} \cup R_{\text{preim}}$, Eq. (90) implies

$$\mathbb{P}[\text{Find}] \leq \sum_{i=1}^{q} \|\mathsf{J}_R\mathsf{U}_i\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{i-1}|\Phi_{i-1}\rangle\|^2, \tag{130}$$

and we can use the bound between $|\Phi_{i-1}\rangle$ and $|\Psi_{i-1}^{\text{Good}}\rangle$ using

$$\begin{aligned}
&\|\mathsf{J}_R\mathsf{U}_i\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{i-1}|\Phi_{i-1}\rangle\| \\
&\leq \left\||\Phi_{i-1}\rangle - |\Psi_{i-1}^{\text{Good}}\rangle\right\| + \left\|\mathsf{J}_R\mathsf{U}_i\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{i-1}|\Psi_{i-1}^{\text{Good}}\rangle\right\|, &(131)
\end{aligned}$$

For calculating the norm of the state in R we want to take care of the norm of states multiplied by $|1\rangle_J$. For UPD we use the same reasoning as in achieving the bound from Eq. (119) that uses Claim 22. The norm of the state with $|1\rangle_J$ in Eq. (99) is $\sqrt{\frac{b^3(s)}{N^2(N-b(s))}}$. For REM, the bound coming from Eq. (100) is $\frac{\sqrt{b(s)(N-b(s))}}{N}$. And for ADD the bound from Eq.(97) is $\sqrt{\frac{b(s+1)}{N}}$. We use these bounds and the triangle inequality to bound the second term in Eq. (131):

$$\begin{aligned}
&\left\|\mathsf{J}_R\mathsf{U}_i\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{V}_R\mathsf{U}_{i-1}|\Psi_{i-1}^{\text{Good}}\rangle\right\| \\
&\leq \max_{s\leq i-1}\left\{\sqrt{\frac{b^3(s)}{N^2(N-b(s))}}+\frac{\sqrt{b(s)(N-b(s))}}{N}+\sqrt{\frac{b(s+1)}{N}}\right\} \\
&\leq \max_{s\leq i-1}\left\{\sqrt{\frac{b^3(s)}{N^2(N-b(s))}}+2\sqrt{\frac{b(s+1)}{N}}\right\}. &(132)
\end{aligned}$$

The bound on Eq. (131) is the sum of Eq. (132) and Eq. (128) with $i-1$ instead of i. Summing the square of the above bound over $1 \leq i \leq q$ gives us the final bound:

$$\begin{aligned}
&\mathbb{P}\left[\text{Find} : \mathsf{A}[\mathsf{CPhO}_{\mathcal{Y}} \setminus R]\right] \\
&\leq \sum_{i=1}^{q}\left(5\sum_{j=1}^{i-1}\max_{s\leq j-1}\left\{\frac{b(s)^{3/2}}{\sqrt{N(N-b(q))}}\right\}+\max_{s\leq i-1}\left\{\sqrt{\frac{b^3(s+1)}{N^2(N-b(q))}}+2\sqrt{\frac{b(s+1)}{N}}\right\}\right)^2 &(133)
\end{aligned}$$

The claimed bound is achieved by simplifying the bound and then substituting $b(s) = s$:

$$\begin{aligned}
&\mathbb{P}\left[\mathsf{Find} : \mathsf{A}[\mathsf{CPhO}_{\mathcal{Y}} \setminus R_{\text{coll}} \cup R_{\text{preim}}]\right] \\
&\leq \sum_{i=1}^{q} \left(\frac{2i^{5/2}}{\sqrt{N(N-q)}} + \sqrt{\frac{i^3}{N^2(N-q)}} + 2\sqrt{\frac{i}{N}} \right)^2 && (134) \\
&\leq \sum_{i=1}^{q} \left(\frac{3i^{5/2}}{\sqrt{N(N-q)}} + 2\sqrt{\frac{i}{N}} \right)^2 && (135) \\
&\leq \frac{2q(q+1)}{N} + \frac{3q^2(q+1)^2}{N\sqrt{N-q}} + \frac{\frac{9}{12}q^2(q+1)^2\left(2q^2+2q-1\right)}{N(N-q)}. && (136)
\end{aligned}$$

Simplifying the above bound yields the claimed result of Lemma 13. □

For R_{coll} we use eq. (127) with $b(i) = i - 1$ instead of $b(i) = i$. The bound on the probability of the event Find is

$$\mathbb{P}[\mathsf{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}} \setminus R_{\text{coll}}]] \leq \frac{2q(1+q)}{N} + \frac{3q^2(1+q)^2}{N\sqrt{N-q}} + \frac{6q^3(1+q)^2(1+q^2)}{4N(N-q)}. \tag{137}$$

For R_{preim} in eq. (127) we set a constant $b(j) = 1$. The bound on the probability of Find is then

$$\mathbb{P}[\mathsf{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}} \setminus R_{\text{preim}}]] \leq \frac{9q}{N} + \frac{15q(q+1)}{N\sqrt{N-1}} + \frac{52(q+1)(2q+1)}{N(N-1)}. \tag{138}$$

D Additional Details on Quantum-Accessible Oracles

D.1 Example Non-Uniform Distributions

The most important distribution that can be quantumly lazy sampled is the uniform distribution. It was first shown in [Zha19] how to do that. We present a lot of details and intuitions on this matter in the rest of this section.

Let us say we want to efficiently simulate a quantum 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 $\forall x \in \{0,1\}^m$:

$$\mathsf{Samp}_\lambda(x) := \begin{pmatrix} \sqrt{1-\lambda} & \sqrt{\lambda} \\ \sqrt{\lambda} & -\sqrt{1-\lambda} \end{pmatrix}, \tag{139}$$

independent from any previous queries. This observation comes in useful in tasks like search in a sparse database.

D.2 Uniform Oracles

For ease of exposition, and to highlight the connection to the formalism in [Zha19], 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 [Unr21].

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 [Zha19].

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

$$\mathsf{HT}_n|x\rangle := \frac{1}{\sqrt{2^n}} \sum_{\xi\in\{0,1\}^n} (-1)^{\xi\cdot x}|\xi\rangle, \tag{140}$$

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 $(\mathbb{C}^2)^{\otimes n}$.

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

$$\mathsf{StO}_{\mathfrak{U}}|x,y\rangle_{XY} \frac{1}{\sqrt{|\mathcal{F}|}} \sum_{f\in\mathcal{F}} |f\rangle_F = \frac{1}{\sqrt{|\mathcal{F}|}} \sum_{f\in\mathcal{F}} |x, y\oplus f(x)\rangle_{XY} \otimes |f\rangle_F, \tag{141}$$

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 $(\mathbb{C}^2)^{\otimes n2^m}$.

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

$$\mathsf{FO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|\phi\rangle_F := |x,\eta\rangle_{XY}|\phi\oplus\chi_{x,\eta}\rangle_F, \tag{142}$$

where $\chi_{x,\eta} := (0^n,\dots,0^n,\eta,0^n,\dots,0^n)$ 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

$$\mathsf{PhO}_{\mathfrak{U}} := (\mathbb{1}_m^X \otimes \mathsf{HT}_n^Y) \otimes \mathbb{1}_{n2^m}^F \circ \mathsf{StO}_{\mathfrak{U}} \circ (\mathbb{1}_m^X \otimes \mathsf{HT}_n^Y) \otimes \mathbb{1}_{n2^m}^F, \tag{143}$$

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

$$\mathsf{FO}_{\mathfrak{U}} = (\mathbb{1}^{XY}) \otimes \mathsf{HT}_{n2^m}^F \circ \mathsf{PhO}_{\mathfrak{U}} \circ (\mathbb{1}^{XY}) \otimes \mathsf{HT}_{n2^m}^F. \tag{144}$$

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

$$\mathsf{StO}_{\mathfrak{U}} \xleftrightarrow{\mathsf{HT}_n^Y} \mathsf{PhO}_{\mathfrak{U}} \xleftrightarrow{\mathsf{HT}_{n2^m}^F} \mathsf{FO}_{\mathfrak{U}}. \tag{145}$$

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

D.2.1 Full Oracles, Additional Details

In this section we show detailed calculations of identities claimed in Section D.2. First we analyze the Phase Oracle, introduced in Eq. (143). We can check by direct calculation that this yields the standard Phase Oracle,

$$\mathsf{PhO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|f\rangle_F = (-1)^{\eta\cdot f(x)}|x,\eta\rangle_{XY}|f\rangle_F. \tag{146}$$

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

$$\begin{aligned}
&\mathsf{PhO}_{\mathfrak{U}}|x,\eta\rangle_{XY}\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|f\rangle_F \\
&= (\mathbb{1}_m^X\otimes\mathsf{HT}_n^Y)\otimes\mathbb{1}_{n2^m}^F\mathsf{StO}_{\mathfrak{U}}|x\rangle_X\frac{1}{\sqrt{2^n}}\sum_y(-1)^{\eta\cdot y}|y\rangle_Y\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|f\rangle_F &(147)\\
&= (\mathbb{1}_m^X\otimes\mathsf{HT}_n^Y)\otimes\mathbb{1}_{n2^m}^F|x\rangle_X\frac{1}{\sqrt{2^n}}\sum_y\sum_{f\in\mathcal{F}}(-1)^{\eta\cdot y}|y\oplus f(x)\rangle_Y\frac{1}{\sqrt{|\mathcal{F}|}}|f\rangle_F &(148)\\
&= \frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|x\rangle_X\sum_\zeta\underbrace{\frac{1}{2^n}\sum_y(-1)^{\eta\cdot y}(-1)^{(y\oplus f(x))\cdot\zeta}}_{=\delta(\eta,\zeta)(-1)^{\zeta\cdot f(x)}}|\zeta\rangle_Y|f\rangle_F &(149)\\
&= \frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}(-1)^{\eta\cdot f(x)}|x\rangle_X|\eta\rangle_Y|f\rangle_F. &(150)
\end{aligned}$$

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 $\mathsf{HT}_{n2^m}^{\otimes 2^m}$ by $\mathsf{HT}_{n2^m}^F$.

$$\begin{aligned}
&\mathsf{HT}_{n2^m}^F\circ\mathsf{PhO}_{\mathfrak{U}}\circ\mathsf{HT}_{n2^m}^F|x,\eta\rangle_{XY}|0^{2^m n}\rangle_F = \mathsf{HT}_{n2^m}^F\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}(-1)^{\eta\cdot f(x)}|x,\eta\rangle|f\rangle_F\\
&= \frac{1}{|\mathcal{F}|}\sum_{\phi,f}(-1)^{\phi\cdot f}(-1)^{\eta\cdot f(x)}|x,\eta\rangle|\phi\rangle_F\\
&= \sum_\phi\underbrace{\frac{1}{2^{n(2^m-1)}}\sum_{f(x'\neq x)}(-1)^{\phi_{x'}\cdot f(x')}}_{=\delta(\phi_{x'},0^n)}\underbrace{\frac{1}{2^n}\sum_{f(x)}(-1)^{\phi_x\cdot f(x)}(-1)^{\eta\cdot f(x)}}_{=\delta(\phi_x,\eta)}|x,\eta\rangle|\phi\rangle_F\\
&= |x,\eta\rangle|0^{2^m n}\oplus\chi_{x,\eta}\rangle
\end{aligned} \tag{151}$$

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

D.2.2 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, x_r is the smallest $x_i\in D^X$ such that $x_r\geq x$:

$$\begin{aligned}
&\mathsf{CFO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|x_1,\eta_1\rangle_{D_1}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|\perp,0^n\rangle_{D_q} = |x,\eta\rangle_{XY}|\psi_{r-1}\rangle\\
&\otimes\begin{cases}
|x_r,\eta_r\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|\perp,0^n\rangle_{D_q} & \text{if } \eta=0^n,\\
|x,\eta\rangle_{D_r}|x_r,\eta_r\rangle_{D_{r+1}}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_q} & \text{if } \eta\neq 0^n, x\neq x_r,\\
|x_r,\eta_r\oplus\eta\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|\perp,0^n\rangle_{D_q} & \text{if } \eta\neq 0^n, x=x_r,\\
& \quad \eta\neq\eta_r,\\
|x_{r+1},\eta_{r+1}\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-2}}|\perp,0^n\rangle_{D_{q-1}}|\perp,0^n\rangle_{D_q} & \text{if } \eta\neq 0^n, x=x_r,\\
& \quad \eta=\eta_r,
\end{cases}
\end{aligned} \tag{152}$$

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

$$\mathsf{CPhO}_{\mathfrak{U}} := \mathbb{1}_{n+m} \otimes \mathsf{HT}_n^{D^Y} \circ \mathsf{CFO}_{\mathfrak{U}} \circ \mathbb{1}_{n+m} \otimes \mathsf{HT}_n^{D^Y}, \tag{153}$$

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

$$\mathsf{CPhO}_{\mathfrak{U}}|x,\eta\rangle_{XY} \frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} |\perp, z\rangle_D = \mathbb{1}_{n+m} \circ \mathsf{HT}_n^{D^Y} \circ \mathsf{CFO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|\perp, 0^n\rangle_D \tag{154}$$

$$= \mathbb{1}_{n+m} \circ \mathsf{HT}_n^{D^Y} \left((1-\delta(\eta,0^n))|x,\eta\rangle_{XY}|x,\eta\rangle_D + \delta(\eta,0^n)|x,\eta\rangle_{XY}|\perp,0^n\rangle_D\right) \tag{155}$$

$$= \frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} \left((1-\delta(\eta,0^n))(-1)^{\eta\cdot z}|x,\eta\rangle_{XY}|x,z\rangle_D + \delta(\eta,0^n)|x,0^n\rangle_{XY}|\perp,z\rangle_D\right). \tag{156}$$

If we defined the Compressed Phase Oracle from 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

$$\frac{1}{\sqrt{2^m}}\sum_x |x,0^n\rangle_{XY} \overset{\mathsf{CPhO}_{\mathfrak{U}}}{\mapsto} \frac{1}{\sqrt{2^m}}\sum_x |x,0^n\rangle_{XY} \frac{1}{\sqrt{2^n}}\sum_z |\perp,z\rangle_D, \tag{157}$$

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

$$\frac{1}{\sqrt{2^m}}\sum_x |x,0^n\rangle_{XY} \overset{\mathsf{CPhO}'_{\mathfrak{U}}}{\mapsto} \frac{1}{\sqrt{2^m}}\sum_x |x,0^n\rangle_{XY} \frac{1}{\sqrt{2^n}}\sum_z |x,z\rangle_D. \tag{158}$$

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

$$\mathsf{CPhO}_{\mathfrak{U}}|x_2,\eta_2\rangle_{X_2Y_2}\mathsf{CPhO}_{\mathfrak{U}}|x_1,\eta_1\rangle_{X_1Y_1} \frac{1}{2^n} \sum_{z_1,z_2\in\{0,1\}^n} |\perp,z_1\rangle_{D_1}|\perp,z_2\rangle_{D_2}$$

$$= |x_2,\eta_2\rangle|x_1,\eta_1\rangle \frac{1}{2^n}\sum_{z_1,z_2} \Bigg((-1)^{\eta_1\cdot z_1}\delta(\eta_2,0^n)(1-\delta(\eta_1,0^n)) \underbrace{|x_1,z_1\rangle_{F_1}|\perp,z_2\rangle_{F_2}}_{=|\psi^{\text{NOT}}\rangle}$$

$$+ \delta(\eta_2,0^n)\delta(\eta_1,0^n)\underbrace{|\perp,z_1\rangle_{F_1}|\perp,z_2\rangle_{F_2}}_{=|\psi^{\text{NOT}}\rangle}$$

$$+ (-1)^{\eta_2\cdot z_1}(1-\delta(\eta_2,0^n))\delta(\eta_1,0^n)\underbrace{|x_2,z_1\rangle_{F_1}|\perp,z_2\rangle_{F_2}}_{=|\psi^{\text{ADD}}\rangle}$$

$$+(-1)^{\eta_1\cdot z_1}(-1)^{\eta_2\cdot z_2}(1-\delta(\eta_2,0^n))(1-\delta(x_1,x_2))(1-\delta(\eta_1,0^n))\underbrace{|x_1,z_1\rangle_{F_1}|x_2,z_2\rangle_{F_2}}_{=|\psi^{\text{ADD}}\rangle}$$

$$+(1-\delta(\eta_2,0^n))\delta(x_1,x_2)\delta(\eta_1,\eta_2)(1-\delta(\eta_1,0^n))\underbrace{|\perp,z_1\rangle_{F_1}|\perp,z_2\rangle_{F_2}}_{=|\psi^{\text{REM}}\rangle}$$

$$+ (1-\delta(\eta_2,0^n))\delta(x_1,x_2)(1-\delta(\eta_1,\eta_2))(1-\delta(\eta_1,0^n))$$

$$\cdot(-1)^{(\eta_1\oplus\eta_2)\cdot z_1}\underbrace{|x_1,z_1\rangle_{F_1}|\perp,z_2\rangle_{F_2}}_{=|\psi^{\mathrm{UPD}})}\Bigg), \tag{159}$$

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

$$\mathsf{CStO}_{\mathfrak{U}}=\mathbb{1}_m\otimes\mathsf{HT}_n^Y\circ\mathsf{CPhO}_{\mathfrak{U}}\circ\mathbb{1}_m\otimes\mathsf{HT}_n^Y. \tag{160}$$

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

$$\begin{aligned}
&\mathsf{CStO}_{\mathfrak{U}}|x,y\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}|\perp,z\rangle_D\\
&=\mathbb{1}_m\otimes\mathsf{HT}_n^Y\circ\mathsf{CPhO}_{\mathfrak{U}}\frac{1}{\sqrt{2^n}}\sum_{\eta\in\{0,1\}^n}(-1)^{\eta\cdot y}|x,\eta\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}|\perp,z\rangle_D &(161)\\
&=\mathbb{1}_m\otimes\mathsf{HT}_n^Y\frac{1}{\sqrt{2^n}}\sum_{\eta\in\{0,1\}^n}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}(-1)^{\eta\cdot y}\Bigg((1-\delta(\eta,0^n))(-1)^{\eta\cdot z}|x,\eta\rangle_{XY}|x,z\rangle_D\\
&+\delta(\eta,0^n)|x,0^n\rangle_{XY}|\perp,z\rangle_D\Bigg) &(162)\\
&=\frac{1}{2^n}\sum_{y',\eta}\frac{1}{\sqrt{2^n}}\sum_z(-1)^{\eta\cdot y}(-1)^{y'\cdot\eta}\Bigg((1-\delta(\eta,0^n))(-1)^{\eta\cdot z}|x,y'\rangle_{XY}|x,z\rangle_D\\
&+\delta(\eta,0^n)|x,y'\rangle_{XY}|\perp,z\rangle_D\Bigg) &(163)\\
&=\sum_{y'}\frac{1}{\sqrt{2^n}}\sum_z\underbrace{\frac{1}{2^n}\sum_{\eta\neq0}(-1)^{\eta\cdot y}(-1)^{y'\cdot\eta}(-1)^{\eta\cdot z}}_{=\delta(y',y\oplus z)-\frac{1}{2^n}}|x,y'\rangle_{XY}|x,z\rangle_D\\
&+\sum_{y'}\frac{1}{\sqrt{2^n}}\sum_z\frac{1}{2^n}|x,y'\rangle_{XY}|\perp,z\rangle_D &(164)\\
&=\frac{1}{\sqrt{2^n}}\sum_z\left(|x,y\oplus z\rangle_{XY}|x,z\rangle_D-\frac{1}{2^n}\sum_{y'}|x,y'\rangle_{XY}|x,z\rangle_D+\frac{1}{2^n}\sum_{y'}|x,y'\rangle_{XY}|\perp,z\rangle_D\right). &(165)
\end{aligned}$$

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

D.3 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 $q_i=1\iff \text{Д}_i^X=x$. This result is then bitwise XOR'ed into an auxiliary register L.
- Add, Function 10: This subroutine adds queried x to the database and take care of appropriate padding. Here our padding is simply $(0^m,0^n)$.

- Upd, Function 11: This subroutine updates the database by subtracting η after a suitable basis transformation.
- Rem, Function 12: This subroutine removes $(x, 0)$ 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 $D^X X A$ and flips the bit in A if the value of D^X is larger than the value in X, so

$$\mathsf{Larger}^{D^X X A}|u\rangle_{D^X}|v\rangle_X|a\rangle_A = |u\rangle_{D^X}|v\rangle_X \begin{cases} |a\oplus 1\rangle_A \text{ if } u > v \\ |a\rangle_A \text{ otherwise} \end{cases} . \tag{166}$$

  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:

$$\mathsf{P}|x_1, ..., x_q\rangle \otimes |y_1, ..., y_n\rangle := |\sigma_n \circ ... \circ \sigma_1(x_1, ..., x_q)\rangle \otimes |y_1, ..., y_n\rangle , \tag{167}$$

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 $\quad |a\rangle_A \mapsto |a \oplus 1\rangle_A$ // save result to register A
6 **if** $a = 1$ **then** // if not located
7 $\quad |Д\rangle_D|l\rangle_L \mapsto \mathsf{Add}(|x\rangle_X|Д\rangle_D)$ // 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: $|x\rangle_X|Д\rangle_D|l'\rangle_L$

Set $|a\rangle_A = |0 \in \mathcal{X}\rangle_A$ // initialize auxiliary register A
for $i = 1, ..., q$ **do**
  if $и_i \neq 0$ **then** // locate entries in the database
    $|a\rangle_A \mapsto |a + (Д_i^X - x)\rangle_A$ // database entry $-$ query
    if $a_i \neq 0$ **then** // locate matches in the database
      $|l_i\rangle_{L_i} \mapsto |l_i \oplus 1\rangle_{L_i}$ // save the corresponding positions
    $|a\rangle_A \mapsto |a - (Д_i^X - x)\rangle_A$ // uncompute register A
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: $|x\rangle_X|Д'\rangle_D|l'\rangle_L$

Set $|a\rangle_A = |0^q \in \{0,1\}^q\rangle_A$ // initialize auxiliary register A
for $i = 1, ..., q$ **do**
  $|a_i\rangle_{A_i} \mapsto \mathsf{Larger}(|Д_i^X\rangle_{D_i^X}|x\rangle_X|a_i\rangle_{A_i})$ // check if database entry $>$ query
  if $Д_i^X \neq \perp$ **then** // correct for empty entries
    $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$
  for $j = i + 1, ..., q$ **do** // flip all higher entries
    $|a_j\rangle_{A_j} \mapsto |a_j \oplus a_i\rangle_{A_j}$ // so we're left with one position
$|Д\rangle_D \mapsto \mathsf{P}^{-1}(|Д\rangle_D \otimes |a\rangle_A)$ // permute D to create empty entry
// P is defined in (167)
for $i = 1, ..., q$ **do**
  if $a_i = 1$ **then** // look for this empty entry
    $|Д_i^X\rangle_{D_i^X} \mapsto |Д_i^X - x\rangle_{D_i^X}$ // add $x-$entry to the database
    $|l_i\rangle_{L_i} \mapsto |l_i \oplus 1\rangle_{L_i}$ // update location register
if $x \neq 0$ **then** // Non zero x implies non zero a
  for $i = 1, ..., q$ **do**
    if $l_i = 1$ **then** // if located
      $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$ // uncompute register A
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: $|\eta\rangle_Y|Д'^Y\rangle_{D^Y}|l\rangle_L$

Apply $\mathsf{QFT}_N^{D^Y}\mathsf{Samp}_{\mathfrak{D}}$ // transform to the Fourier basis
for $i = 1, ..., q$ **do**
  if $l_i = 1$ **then** // if located
    $|\Delta_i^Y\rangle_{D_i^Y} \mapsto |\Delta_i^Y - \eta\rangle_{D_i^Y}$ // Update the Y register of entry
Apply $\mathsf{Samp}_{\mathfrak{D}}^{\dagger}\mathsf{QFT}_N^{\dagger D^Y}$ // transform back to the unprepared database
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**
    $|a_i\rangle_{A_i} \mapsto \mathsf{Larger}(|x\rangle_X, |Д_i^X\rangle_{D_i^X}, |a_i\rangle_{A_i})$ // 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
      $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$
    $|a_i\rangle_{A_i} \mapsto \mathsf{Larger}(|Д_i^X\rangle_{D_i^X}, |x\rangle_X, |a_i\rangle_{A_i})$
$|a\rangle_A \mapsto \mathsf{Locate}(|x\rangle_X|Д\rangle_D|l\rangle_A)$
if $A = 0^q$ **then** // check if we have removed
  $|b\rangle_B \mapsto |b \oplus 1\rangle_B$ // Uncompute register B
$|a\rangle_A \mapsto \mathsf{Locate}(|x\rangle_X|Д\rangle_D|l\rangle_A)$ // 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 : $|\eta\rangle_Y|\text{Д}^Y\rangle_D|l\rangle_L|a\rangle_A$
Output: $|\eta\rangle_Y|\text{Д}^Y\rangle_D|l\rangle_L|a'\rangle_A$

Set $|b\rangle_B = |0 \in \mathcal{Y}\rangle_B$ // initialize auxiliary register B
Apply $\mathsf{QFT}_N^{D^Y}\mathsf{Samp}_{\mathfrak{D}}$ // transform to the Fourier basis
for $i = 1, ..., q$ **do**
  if $l_i = 1$ **then**
    $|b\rangle_B \mapsto |b + (\Delta_i^Y - \eta)\rangle_B$ // database entry − query
    if $b = 0$ **then** // locate matches in the database
      if $\eta \neq 0$ **then** // if we added
        $|a\rangle_A \to |a \oplus 1\rangle_A$
    $|b\rangle_B \mapsto |b - (\Delta_i^Y - \eta)\rangle_B$ // uncompute register B
Apply $\mathsf{Samp}_{\mathfrak{D}}^{\dagger}\mathsf{QFT}_N^{\dagger D^Y}$ // transform back to the unprepared database
return $|\eta\rangle_Y|\text{Д}^Y\rangle_D|l\rangle_L|a'\rangle_A$ // a' is modified register A

E Collapsingness of Sponges

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. We begin by introducing the notion of *collapsing* functions.

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

$$\textbf{Collapse 1}: \quad (S, M, h) \leftarrow \mathsf{A}^{\mathsf{H}}(),\ m \leftarrow \mathsf{M}(M),\ b \leftarrow \mathsf{B}^{\mathsf{H}}(S, M), \tag{168}$$

$$\textbf{Collapse 2}: \quad (S, M, h) \leftarrow \mathsf{A}^{\mathsf{H}}(),\qquad b \leftarrow \mathsf{B}^{\mathsf{H}}(S, M). \tag{169}$$

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.(168) and measure M in the computational basis as m.

Definition 23 (Collapsing [Unr16b])**.** *A function* H *is* collapsing *if for any valid quantum-polynomial-time adversary* (A, B)

$$|\mathbb{P}[b = 1 : \textbf{Collapse 1}] - \mathbb{P}[b = 1 : \textbf{Collapse 2}]| < \varepsilon, \tag{170}$$

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 24 (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 any function that is indifferentiable from a collapsing function is itself collapsing. In the context of sponges, together with thm. 19, we reprove the result of [Cza+18] in a modular way that might come useful when indifferentiability of sponges with permutations is established.

Theorem 25 (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

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

□