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

October 31, 2020

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.

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 permutatation as internal function.

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 [Unr19a] Unruh developed quantum Relational Hoare Logic for computer verification of proofs in (post-)quantum cryptography. There he also uses the approach of game-playing, but in general focuses on formal definitions of quantum programs and predicates. To investigate the relation between [Unr19a] and our work in more detail one would have to express our results in the language of the new logic. We leave it as an interesting direction for the future. The proof techniques of [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).

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.

2 Detailed Summary

In the following, we give a detailed summary of our results, introducing the necessary background along the way.

2.1 Quantum Game-Playing Proofs

We begin by recalling the main ingredients for our game-playing-proofs framework, the Compressed-oracle technique and the one-way to hiding lemma.

Compressed Oracles. The compressed-oracle technique, introduced by Zhandry [Zha19], is a way of lazy sampling random functions in the quantum realm. We want to lazy sample a random function for an adversary A that can access the oracle for this function in superposition. Superposition access is usually modeled as $\mathsf{U}_f|x, y\rangle = |x, y + f(x)\rangle$. If the function f is chosen uniformly at random according from the set $\mathcal{F}$ of all functions for some fixed domain and range, it is usually treated as a random variable. Following an extremely common paradigm in quantum information science, *purification*, we can *include the function's randomness into the quantum-mechanical description* of the problem. 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,

$$\mathsf{StO}|x, y\rangle_{XY}|f\rangle_F := |x, y + f(x)\rangle_{XY}|f\rangle_F. \tag{1}$$

The initial state of the oracle for uniformly random functions is $\sum_f \frac{1}{\sqrt{|\mathcal{F}|}}|f\rangle$. 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 to a state holding only zeros $|0^M\rangle$, where $0 \in \mathcal{Y}$.

The oracle register is maintained in a specific way when the adversary makes queries. 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

$$\mathsf{FO}|x, \eta\rangle_{XY}|\phi\rangle_F := |x, \eta\rangle_{XY}|\phi - \chi_{x,\eta}\rangle_F, \tag{2}$$

where ϕ is the transformed truth table f and $\chi_{x,\eta} := (0, \dots, 0, \eta, 0, \dots, 0)$ 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 $f(x)$.

Finally, it turns out that compression of ϕ is possible and the purification can be a meaningful database of past queries. After compression, we can of course change back to the standard basis and see that the compressed database holds outputs of f on inputs queried by the adversary. We denote the oracle after this basis change by $\mathsf{CStO}_{\mathfrak{D}}$. We call this technique quantum lazy sampling: it provides a quantum analogue of freshly sampling function outputs and noting them down for future reference. In the next paragraph we describe how to apply this technique.

A new formalization. We put the compressed oracle onto a new formal footing. Along the way, we generalize the technique slightly, expanding it to the class of distributions over functions $\mathfrak{D}$ that are locally sampleable. By $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S})$ we denote the unitary that maps the oracle register to the superposition over outputs of $x \in \mathcal{S}$ with appropriate weights—for the uniform distribution it is QFT_N. In the case of non-uniform distributions we need the operations $\mathsf{Samp}_{\mathfrak{D}|f(x_1)=y_1,\ldots,f(x_s)=y_s}$ to be efficiently implementable for the compressed oracle to be efficient. Here, $\mathfrak{D}|f(x_1) = y_1, ..., f(x_s) = y_s$ denotes the function distribution on $\mathcal{X} \setminus \mathcal{S}$, with $\mathcal{S} = \{x_1, ..., x_s\}$, obtained by conditioning $\mathfrak{D}$ on the event $f(x_1) = y_1 \wedge ... \wedge f(x_s) = y_s$. We write

$$\mathsf{Samp}_{\mathfrak{D}|f(x_1)=y_1,\ldots,f(x_s)=y_s}(\mathcal{X} \setminus \mathcal{S}) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \{x_1, ..., x_s\}) \tag{3}$$

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

$$\forall \mathcal{S} \subseteq \mathcal{X} : \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \mathcal{S}) \circ \mathsf{Samp}_{\mathfrak{D}}(\mathcal{S}) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}). \tag{4}$$

We additionally require that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \mathcal{S})$ does not modify the output values of $\mathcal{S}$ and is only controlled on them. Note that while we require that $\mathsf{Samp}_{\mathfrak{D}}$ is *local*, so fulfills Eq. (4), and that it prepares the correct distribution when acting on $|0^N\rangle$, see Eq. (18). We also require it to be a valid unitary. In general $\mathsf{Samp}_{\mathfrak{D}}$ fulfilling both requirements can be completed to a full unitary in any way. Note that an interesting class of distributions that we know how to quantum lazy sample are those with outputs distributed independently for every input. This class is not the only set of distributions that are local but still the restriction of eq. (4) is pretty severe—excluding e.g. random permutations.

The most important example remains the uniform distribution. Another example are random Boolean functions, where on every input the probability of the outcome being 1 is λ (see also [Ala+20; HM20]).

One-way to Hiding lemma The One-way to Hiding (O2H) lemma [Unr14] is a very useful statement, used for reprogramming a random oracle on some inputs. The O2H lemma relates one-wayness and hiding in the following sense. Given an algorithm that distinguishes two oracles that are equal except on a certain marked set of inputs, the lemma can be used to construct an algorithm that *finds* the input where the two oracles differ. As both endeavors are doomed to fail unless the algorithms receive some information about the *outputs* that the two different oracles will return upon an input from the marked set, the resulting algorithm breaks some form of one-wayness of the function.

A new version of the lemma from [AHU19] defines puncturing: measuring the state of the adversary after every query to check if she queries values from some given set. We restate this lemma to be used with compressed oracles. This gives more freedom into the relations on inputs and outputs of the oracles that we can puncture.

One-way to Hiding lemma for compressed oracles. Combining the two described techniques, we proceed to develop a variant of the O2H lemma where the mentioned puncturing, and therefore also the extraction of a marked set of inputs, is performed on the compressed-oracle database instead of the adversary's input. This allows for more complex puncturings that *may depend on the adversary's previous queries*. Such a feature is clearly impossible when just measuring the adversary's query input, and makes crucial use of the ability of the compressed oracle to circumvent the no-cloning theorem (sometimes also called the "recording barrier" in this context) using the randomness of the random function.

This more general form of puncturing is most conveniently formalized using *relations*. A relation R on the database of a compressed oracle is a subset of $\bigcup_{s\in[|\mathcal{Y}|]}(\mathcal{X}\times\mathcal{Y})^s$. A relation defines a two-outcome quantum measurement $\mathcal{M}_R$ on the compressed oracle's database register, that measures whether the database fulfills the relation or not (note that in general, the database will be in superposition of these two options). We go on to define a *punctured compressed oracle* $\mathsf{H}\backslash R$ by adding an application of $\mathcal{M}_R$ after each call to the compressed oracle H. Our version of the O2H lemma reads:

Theorem 1 (Compressed oracle O2H, simplified)**.** *Let R be a relation 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 making at most q queries to H. Then the distinguishing advantage between the plain and punctured oracles can be bounded as*

$$\left|\mathbb{P}[b=1: b\leftarrow \mathsf{A}^{\mathsf{H}}(z)] - \mathbb{P}[b=1: b\leftarrow \mathsf{A}^{\mathsf{H}\backslash R}(z)\right| \leq \sqrt{(q+1)\mathbb{P}[\mathrm{Find}: \mathsf{A}^{\mathsf{H}\backslash R}(z)]}, \tag{5}$$

where Find *is the event that the measurement $\mathcal{M}_R$ succeeds at least once.*

The proof of this theorem follows closely the structure of the proof of the O2H lemma for punctured oracles from [AHU19]. In the main body, we give a proof that highlights the differences due to the different puncturing method.

The following bound on $\mathbb{P}[\mathrm{Find}]$ for the collision relation is important for proving indifferentiability. In the language of relations, collisions are the set of database strings (i.e. the set of strings of input/output pairs of the considered random oracle) containing at least two entries with distinct x parts and the same y part. In the following, let $\mathsf{CStO}_{\mathcal{Y}}$ denote the compressed oracle of a uniformly random function from $\mathcal{X}$ to $\mathcal{Y}$.[1]

Lemma 2. *For any quantum adversary A interacting with a punctured oracle $\mathsf{CStO}_{\mathcal{Y}}\setminus R_{\mathrm{coll}}$—where R_{coll} is the collision relation—the probability of* Find *is bounded by:*

$$\mathbb{P}[\mathrm{Find}: \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}}\setminus R_{\mathrm{coll}}]] \leq 3\frac{q^5}{|\mathcal{Y}|}, \tag{6}$$

where q is the maximal number of queries made by A.

Proof sketch. First we propose a quantum state $|\Psi^{\mathrm{Good}}\rangle$ that approximately describes the joint state of the adversary and the decompressed database conditioned on being not in relation R_{coll}. It is much easier to calculate how $\mathsf{CStO}_{\mathcal{Y}}\setminus R_{\mathrm{coll}}$ affects $|\Psi^{\mathrm{Good}}\rangle$ than the actual state. After calculating the result of a single application of $\mathsf{CStO}_{\mathcal{Y}}\setminus R_{\mathrm{coll}}$ to $|\Psi^{\mathrm{Good}}\rangle$, it is straightforward to show that this state is in fact close to the actual state. Using that approximate characterization, we can bound $\mathbb{P}[\mathrm{Find}: \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}}\setminus R_{\mathrm{coll}}]]$ with standard techniques. □

We suspect that the proof technique used above generalizes to other relations.

Formalizing the game-playing proofs framework. The above two ingredients, the O2H lemma for compressed oracles and the technique for analyzing the probability of the Find event that plays a central role in it, are the heart of the quantum game-playing proofs framework that we put forward in this paper. The main concept of the classical game-playing proof framework is the idea of two games being equal, except (and until) a certain bad event happens. This idea can already be captured in the quantum case with the O2H lemma for punctured oracles from [AHU19], but for a very limited set of "local" events that only depend on a single query to the oracle. The compressed-oracle O2H lemma introduced above can capture much more complex events: Any event that only depends on the set of all queries that the algorithm has made can be

[1]We subscript the compressed oracle for a uniformly random function with the range of that function, as this turns out to be convenient for describing the application to the indifferentiability of the sponge construction.

formalized as a relation of the type described above. The idea of objects that are identical until bad is therefore captured on the level of oracles via the following definition of almost identical oracles.

Definition 3 (Almost identical oracles)**.** *Let* H *and* G *be compressed oracles and* R_i*,* $i = 1, 2$ *relations on their databases. We call the punctured oracles* $\mathsf{H} \setminus R_1$ *and* $\mathsf{G} \setminus R_2$ almost identical *if they are equal conditioned on the events* $\neg\text{Find}_1$ *and* $\neg\text{Find}_2$ *respectively, i.e. for any event* E*, any strings* y, z*, and any quantum algorithm* A

$$\mathbb{P}[E : y \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \neg\text{Find}_1] = \mathbb{P}[E : y \leftarrow A^{\mathsf{G}\setminus R_2}(z) \mid \neg\text{Find}_2]. \tag{7}$$

The increased flexibility for defining bad events comes, however, at the cost of an increased difficulty in analyzing the probability that a bad event happens. The technique used in Lemma 2 can be used to bound that probability, completing the toolbox for quantum game-playing proofs. In the following section, we present a concrete example of our framework in action: a proof of quantum indifferentiability of the sponge construction (which is used, e.g., in the SHA3 hash function).

In the main body, we illustrate how closely quantum security proofs can follow their classical counterparts when the quantum game-playing proofs framework is used: We first present the classical proof and then the quantum one. In the next section, we summarize the application to the sponge construction.

2.2 Indifferentiability of the Sponge Construction

Indifferentiability The tools and techniques we have described above can be used for proving quantum indifferentiability [Zha19; Car+18]. For some cryptographic constructions, especially in idealized models like the random-oracle model, security is best captured by indifferentiability [MRH04]. The most prominent class of schemes where indifferentiability plays an important role as a security definition are domain-extension schemes for hash functions. A domain-extension scheme is an algorithm that uses a primitive that takes fixed-length inputs to construct a primitive with similar properties that accepts inputs of arbitrary length. In the random-oracle model, hash functions are modeled as random oracles. For a domain-extension scheme, it is then natural to ask for the result to look like a random oracle if the fixed-length hash function is modeled as a random oracle. Formally, this is exactly the property that is captured by indifferentiability. The adversary gets access to both the construction and the public fixed-length random oracle and we want to prove that this situation is indistinguishable from the adversary interacting with a random oracle and a simulator—simulating the public internal function. For post-quantum security, we need to allow the adversary to make quantum queries to all the oracles [Bon+11].

The sponge construction – introduction The construction that we focus on is the sponge construction, used to design variable-input-length and variable-output-length functions [Ber+07]. It works by applying the *internal function* φ multiple times to an internal *state*, interspersed with simple operations processing the input and generating the output. In Algorithm 1 we present the definition of the sponge construction, which we denote with SPONGE. The internal state $s = (\bar{s}, \hat{s}) \in \mathcal{A} \times \mathcal{C}$ of SPONGE consists of two parts: the *outer* part $\bar{s} \in \mathcal{A}$ and the *inner part* $\hat{s} \in \mathcal{C}$. The number of possible outer parts $|\mathcal{A}|$ is called the *rate* of the sponge, and $|\mathcal{C}|$ is called *capacity*. Naturally the internal function is a map $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$. To denote the internal function with output limited to the part in $\mathcal{A}$ and $\mathcal{C}$ we use the same notation as for states, $\bar{\varphi}$ and $\hat{\varphi}$ respectively. Note that we use a general formulation of the construction, using any finite sets for $\mathcal{A}$ and $\mathcal{C}$. All our results also work for SPONGE defined with bit-strings and addition modulo 2, as specified in [NIS14]. By PAD we denote a padding function: an efficiently computable bijection mapping an arbitrary message set to strings p of elements of $\mathcal{A}$. By $|p|$ we denote the number of characters (elements of $\mathcal{A}$) in p.

In it's most general form, the constructed function is $\textsc{Sponge}_\varphi : \mathcal{A}^* \times \mathbb{N} \to \mathcal{A}^*$, where $\mathcal{A}^* := \bigcup_{n=0}^{\infty} \mathcal{A}^n$, where the extra parameter determines the number of "squeezing rounds", i.e. the number of times φ is applied after the whole input was processed, and thus the number of times the output is appended with an additional element of $\mathcal{A}$.

Main result The second main result of this paper is the proof of quantum indifferentiability of the sponge construction.

Theorem 4 (Sponge with functions, quantum indifferentiability)**.** *$\textsc{Sponge}_\varphi[\textsc{pad}, \mathcal{A}, \mathcal{C}]$ calling a random function φ is indifferentiable from a random oracle up to error ε against* quantum *adversaries making $q < |\mathcal{C}|$ queries and*

$$\varepsilon \leq 288\frac{q^5}{|\mathcal{C}|} + 6\sqrt{\frac{(q+1)q^5}{|\mathcal{C}|}}.$$

The sponge construction – some more details. To provide a proof sketch of the above theorem, we need to introduce some more notation related to the sponge construction. For a set $\mathcal{S} \subseteq \mathcal{A} \times \mathcal{C}$, by $\overline{\mathcal{S}}$ we denote the outer part of the set: a set of outer parts of elements of $\mathcal{S}$. Similarly by $\widehat{\mathcal{S}}$ we denote the set of inner parts of the set. We use similar notation for quantum registers holding a quantum state in $\mathcal{H}_{\mathcal{A}\times\mathcal{C}}$: $\overline{Y}$ is the part of the register holding elements of $\mathcal{A}$ and $\widehat{Y}$ holds the inner parts in $\mathcal{C}$.

Algorithm 1: $\textsc{Sponge}_\varphi[\textsc{pad}, \mathcal{A}, \mathcal{C}]$

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

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

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

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

$$\mathsf{SpPath}(s, G) := \bar{v}_1 \| (\bar{v}_2 - \bar{w}_1) \| \cdots \| (\bar{v}_\ell - \bar{w}_{\ell-1}) \| 0. \tag{8}$$

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

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

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

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

Proof sketch.

We are ready to give a sketch of the proof of Theorem 4.

Proof sketch. By virtue of the quantum game-playing proofs framework introduced in Section 2.1, our quantum proof closely mirrors (a slightly modified version of) the proof of classical indifferentiability [Ber+08]. The indifferentiability simulator we construct simulates the inner function using a compressed oracle punctured on the collision relation.

The proof idea is to provide the adversary with outputs of φ that do not collide in the inner part. If this step succeeds, every path—input to sponge or the random oracle—is a unique input to the construction or the random oracle. To keep the answers from colliding, the simulator has to have access to the sponge graph, a graph of queried inputs and given outputs to φ. Every path in the graph starts at the node with the inner part equal 0, the initial value in $\textsc{Sponge}_\varphi$. The second important feature of the sponge graph that the simulator needs to take into account is whether the graph is saturated. Saturation happens when in every supernode (a set of nodes with the same outer part, defined above) there is a node in a path. Saturation does not happen before $|\mathcal{C}|$ queries.

Quantumly we avoid collisions by puncturing the compressed oracle on collisions. Conditioned on $\neg$Find we are certain that the adversary does not know about any inner collision.

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 these two sampling steps correctly in the quantum case, we, however, need to maintain *two* databases: one responsible for the outer part and another for the inner part. We denote them by $\overline{D}$ and $\widehat{D}$ respectively. The simulator is given in Algorithm 2

In 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 an 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

Algorithm 2: Quantum $\boxed{\mathsf{S}_2}$, $\boxed{\mathsf{S}_3}$, $\boxed{\mathsf{S}_4}$, functions

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

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 now 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 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. □

Organization. In Section 3 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 4 we generalize the compressed-oracle technique of [Zha19] to non-uniform distributions over functions. In Section 5 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 6 we use these results to prove quantum indifferentiability of the sponge construction.

3 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.

3.1 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 5 (Fundamental lemma of game-playing, Lemma 2 of [BR06])**.** *Let* G *and* H *be identical-until-bad games and let* A *be an adversary that outputs a bit* b*. Then*

$$\left|\mathbb{P}[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{H}}] - \mathbb{P}[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{G}}]\right| \leq \mathbb{P}[\mathrm{Bad} = 1 : \mathsf{A}^{\mathsf{G}}]. \tag{9}$$

3.2 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) $\mathsf{C}^{\mathrm{priv}} = \mathsf{H}_f$. The public interface accepts n-bit long strings and outputs n-bit strings as well. We have that $\mathsf{C}^{\mathrm{pub}} = f$. Often we consider one of the analyzed systems, R, to be a random oracle. Then both interfaces are the same and output random outputs of appropriate given length.

The following definitions and Theorem 8 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.

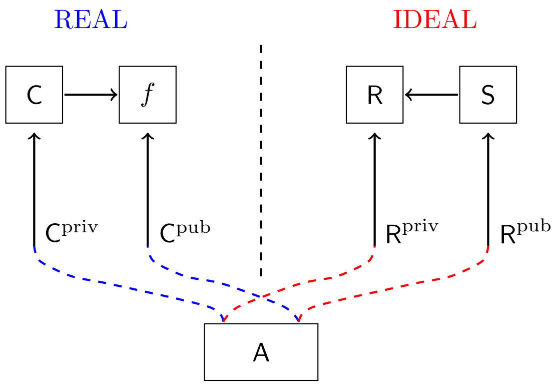


Figure 1: A schematic representation of the notion of indifferentiability, Def. 6. Arrows denote "access to" the pointed system.

Definition 6 (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*

$$\left|\mathbb{P}\left[b = 1 : b \leftarrow \mathsf{D}[\mathsf{C}_k^{\text{priv}}, \mathsf{C}_k^{\text{pub}}]\right] - \mathbb{P}\left[b = 1 : b \leftarrow \mathsf{D}[\mathsf{R}_k^{\text{priv}}, \mathsf{S}[\mathsf{R}_k^{\text{pub}}]]\right]\right| \leq \varepsilon(k)\,, \tag{10}$$

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

By efficient we mean with runtime that is polynomial in the security parameter k. The definitions are still valid and the theorem below holds also if we interpret efficiency in terms of queries made by the algorithms. Note that then we can allow the algorithms to be unbounded with respect to runtime, the distinction between quantum and classical queries is still of crucial importance though. By square brackets we denote (classical or quantum) oracle access to some algorithm, we also use A^{H} if the oracle is denoted by a more confined symbol. In Fig. 1 we present a a scheme of the situation captured by Def. 6.

Definition 7 (As secure as [MRH04]). *A cryptographic (classical or quantum) system* C *is said to be* as secure as C' *if for all efficient (classical or quantum) environments* Env *the following holds: For any efficient (classical or quantum) attacker* A *accessing* C *there exists another (classical or quantum) attacker* A' *accessing* C' *such that the difference between the probability distributions of the binary outputs of* $\mathsf{Env}[\mathsf{C}, \mathsf{A}]$ *and* $\mathsf{Env}[\mathsf{C}', \mathsf{A}']$ *is negligible, i.e.*

$$\left|\mathbb{P}\left[b = 1 : b \leftarrow \mathsf{Env}[\mathsf{C}, \mathsf{A}]\right] - \mathbb{P}\left[b = 1 : b \leftarrow \mathsf{Env}[\mathsf{C}', \mathsf{A}']\right]\right| \leq \varepsilon(k)\,, \tag{11}$$

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

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

Indifferentiability is a strong security notion guaranteeing that a lower-level function (e.g. a random permutation) can be used to construct a higher-level object (e.g. a variable input-length random function) that is "equivalent" to the ideal one—in the sense of Thm. 8. 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.

3.3 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

$$\mathsf{QFT}_N |x\rangle := \frac{1}{\sqrt{N}} \sum_{\xi \in [N]} \omega_N^{\xi \cdot x} |\xi\rangle, \tag{12}$$

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

$$\sum_{\xi \in [N]} \omega_N^{x \cdot \xi} \cdot \bar{\omega}_N^{x' \cdot \xi} = N \delta_{x,x'}, \tag{13}$$

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.

4 Quantum-Accessible Oracles

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

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

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*.

[2]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 $\mathbb{C}^G$.

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

$$\begin{aligned}&\sum_f \mathbb{P}[f : f \leftarrow \mathfrak{D}]\ \mathsf{U}_f|\Psi_0\rangle\langle\Psi_0|\mathsf{U}_f^\dagger\ \otimes\ |f\rangle\langle f|_F\\ &= \sum_f \mathbb{P}[f : f \leftarrow \mathfrak{D}] \sum_{x,x',y,y'} \alpha_{x,y}\bar{\alpha}_{x',y'}|x, y + f(x)\rangle\langle x', y' + f(x')| \otimes |f\rangle\langle f|_F,\end{aligned} \tag{14}$$

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,

$$\mathsf{StO}|x, y\rangle_{XY}|f\rangle_F := |x, y + f(x)\rangle_{XY}|f\rangle_F. \tag{15}$$

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 [Unr19b].

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

$$\mathsf{FO}|x, \eta\rangle_{XY}|\phi\rangle_F := |x, \eta\rangle_{XY}|\phi - \chi_{x,\eta}\rangle_F, \tag{16}$$

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

Classically, a (uniformly) random oracle can be "compressed" by lazy-sampling the responses, i.e. by answering with previous answers if there are any, and with a fresh random value otherwise. Is lazy-sampling possible for quantum accessible oracles? Surprisingly, the answer is yes. Thanks to the groundbreaking ideas presented in [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 an arbitrary distribution that fulfills the quantum analogue of the classical condition of efficiently samplable conditional distributions. In the first part we introduce a general structure of quantum-accessible oracles. In the second part we generalize the idea of compressed random oracles to deal with non-uniform distributions of functions. In Appendix A, we provide additional details on the implementation of the procedures introduced in this section and step-by-step calculations of important identities and facts concerning compressed oracles. In Appendix A.1 we recall in detail the compressed oracle introduced in [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.

4.1 General Structure of the Oracles

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

1. Hilbert spaces for the input $\mathcal{H}_{\mathcal{X}}$, output $\mathcal{H}_{\mathcal{Y}}$, and state registers $\mathcal{H}_{\mathcal{F}}$,
2. a procedure $\mathsf{Samp}_{\mathfrak{D}}$ that, on input a subset of the input space of the functions in $\mathfrak{D}$, prepares a superposition of partial functions on that subset of inputs with weights according to the respective marginal of the distribution $\mathfrak{D}$,
3. an update unitary $\mathsf{FO}_{\mathfrak{D}}$ that might depend on $\mathfrak{D}$ (in the case of compressed oracles) or not (e.g. in the definition from Eq.(16)).

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.

4.2 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 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 aim at the following functionality

$$\mathsf{StO}|x,y\rangle_{XY} \sum_{f\in\mathcal{F}} \sqrt{\mathbb{P}[f : f \leftarrow \mathfrak{D}]}\,|f\rangle_F = \sum_{f\in\mathcal{F}} \sqrt{\mathbb{P}[f : f \leftarrow \mathfrak{D}]}\,|x, y + f(x) \mod N\rangle_{XY}\,|f\rangle_F, \tag{17}$$

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

$$\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S})|0^{|\mathcal{S}|}\rangle_{F(\mathcal{S})} = \sum_{f(x): f\in\mathcal{F}, x\in\mathcal{S}} \sqrt{\mathbb{P}[f(\mathcal{S}) : f \leftarrow \mathfrak{D}]} \bigotimes_{x\in\mathcal{S}} |f(x)\rangle_{F(x)}, \tag{18}$$

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

$$\mathsf{StO} \xleftrightarrow{\mathsf{QFT}_N^Y} \mathsf{PhO} \xleftrightarrow{\mathsf{QFT}_N^F} \mathsf{FO}, \tag{19}$$

going "to the right" is done by applying QFT_N and "to the left" by applying the adjoint. Also note that 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 just $\mathsf{FO} = \mathsf{QFT}_N^{YF} \circ \mathsf{StO} \circ \mathsf{QFT}_N^{\dagger\, YF}$,

$$\begin{aligned}
&\mathsf{FO}|x,\eta\rangle_{XY} \sum_{\phi} \frac{1}{\sqrt{N^M}} \sum_{f\in\mathcal{F}} \sqrt{\mathbb{P}[f : f \leftarrow \mathfrak{D}]}\, \omega_N^{\phi\cdot f}\, |\phi\rangle_F \\
&= |x,\eta\rangle_{XY} \sum_{\phi} \frac{1}{\sqrt{N^M}} \sum_{f\in\mathcal{F}} \sqrt{\mathbb{P}[f : f \leftarrow \mathfrak{D}]}\, \omega_N^{\phi\cdot f}\, |\phi - \chi_{x,\eta} \mod N\rangle_F.
\end{aligned} \tag{20}$$

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

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

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

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

$$\mathsf{Samp}_{\mathfrak{D}|f(x_1)=y_1,\dots,f(x_s)=y_s}(\mathcal{X} \setminus \mathcal{S}) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \{x_1, \dots, x_s\}) \tag{21}$$

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

$$\forall \mathcal{S} \subseteq \mathcal{X} : \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \mathcal{S}) \circ \mathsf{Samp}_{\mathfrak{D}}(\mathcal{S}) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}). \tag{22}$$

We additionally require that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \mathcal{S} \mid \mathcal{S})$ does not modify the output values of $\mathcal{S}$ and is only controlled on them. Note that while we require that $\mathsf{Samp}_{\mathfrak{D}}$ is *local*, so fulfills Eq. (22), and that it prepares the correct distribution when acting on $|0^N\rangle$, Eq. (18), we also require it to be a valid unitary. In general $\mathsf{Samp}_{\mathfrak{D}}$ fulfilling both requirements can be completed to a full unitary in any way. Note that an interesting class of distributions that we know how to quantum lazy sample are those with outputs distributed independently for every input. This class is not the only set of distributions that are local but still the restriction of eq. (22) is pretty severe—excluding e.g. random permutations.

Note that for $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}\backslash\mathcal{S} \mid \mathcal{S})$ to be efficient, it is not sufficient that the conditional probability distributions $\mathfrak{D}|f(x_1) = y_1, ..., f(x_s) = y_s$ are classically efficiently 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 important example of a sampling procedure following the above requirements is an external oracle. Note that even if the oracle itself is not efficiently samplable, from the perspective of its user it outputs values do not depend on what she has queried. Also, the unitary $\mathsf{Samp} = \mathsf{H}$, where H is the external oracle, commutes for any set of inputs, just like the locality requirement from Eq. (22) demands.

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

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

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

1. Find the index $l \in [q]$ of the register into which we should insert (x, η).
2. If $x \neq x_l$: insert x in a register after the last element of the database and shift it to position r, moving the intermediate registers backwards.
3. Change the basis to the Fourier basis (in which the adversary's η is encoded) and update register l to contain $(x_l, \eta_l - \eta)$, change the basis back to original.
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. Uncompute 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.

Up till now we have described the compressed database only on a high-level, let us now explain the basis changes mentioned above in more detail. To deal with the difference between the initial 0 state and the initial Fourier basis truth tables we use yet another alphabet and define Д (pronounced as [dɛ]) which denotes the unprepared database. We call it like that because the initial state of Д is the all-zero state, moreover only by applying $\mathsf{QFT}_N \circ \mathsf{Samp}_{\mathfrak{D}}$ we transform it to Δ, i.e the Fourier basis database. As we will see, operations on Д are more intuitive and easier to define. We denote an unprepared database by $|\text{Д}\rangle_D = |x_1, \text{и}_1\rangle_{D_1}|x_2, \text{и}_2\rangle_{D_2} \cdots |x_q, \text{и}_q\rangle_{D_q}$ (where the Cyrillic letter и is pronounced as [i]). By $\Delta^Y(x)$ we denote the η value corresponding to the pair in Δ containing x and by Д^X we denote the x values kept in Д. The intuition behind the preparation procedure is to initialize the truth table of the correct distribution in the correct basis. This notion is not visible in the uniform-distribution case, because there the sampling procedure for the uniform distribution $\mathfrak{U}$ is the Hadamard transform: $\mathsf{Samp}_{\mathfrak{U}} = \mathsf{HT}^{\dagger}_{n2^m}$, and the database pictures Δ and Д are equivalent. The following chain of databases similar to Eq. (19) represents different pictures, i.e. bases, in which the compressed database can be viewed

$$|Д\rangle \xleftrightarrow{\mathsf{Samp}_{\mathfrak{D}}} |D\rangle \xleftrightarrow{\mathsf{QFT}_N^{D^Y}} |\Delta\rangle. \tag{23}$$

Using this notation, Alg. 3 defines the procedure of updates of the database of the compressed database. We refer to Appendix A.2 for the fully detailed description of $\mathsf{CFO}_{\mathfrak{D}}$.

Algorithm 3: General $\mathsf{CFO}_{\mathfrak{D}}$

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

Below in Algorithm 4 we explain how to uncompute l in Line 12 of Algorithm 3.

Algorithm 4: Uncompute L in Line 12 of Algorithm 3

```
1  Control on registers $A^X$ and $D^X$
2  for $i = 1\ldots, s-1$ do
3      if $Д_i^X = x$ then
4          Subtract $i$ from $L$
5      else if $Д_i^X < x$ and $x < Д_{i+1}^X$ then
6          Subtract $i+1$ from $L$
```

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 a bound on the number of queries made by the adversary, this is not a fundamental requirement.

The decompression procedure for the general Compressed Fourier Oracle is given by Alg. 5. The output of the decompression procedure $\phi(Д)$ is the state holding the prepared Fourier-basis

Algorithm 5: General Decompressing Procedure $\mathsf{Dec}_{\mathfrak{D}}$

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

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

Theorem 9 (Correctness of $\mathsf{CFO}_{\mathfrak{D}}$)**.** *Say $\mathfrak{D}$ is a distribution over functions, let $\mathsf{CFO}_{\mathfrak{D}}$ be as defined in Alg. 3 and* FO *as in Eq.*(20)*, then for any quantum adversary* A *making q quantum queries we have*

$$|\Psi_{\mathsf{FO}}\rangle = \mathsf{Dec}_{\mathfrak{D}}|\Psi_{\mathsf{CFO}}\rangle, \tag{24}$$

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

Proof. We will show that

$$\mathsf{FO} \circ \mathsf{Dec}_{\mathfrak{D}} = \mathsf{Dec}_{\mathfrak{D}} \circ \mathsf{CFO}_{\mathfrak{D}}, \tag{25}$$

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

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

$$|Д(\vec{x}, \vec{и})\rangle := |x, \eta\rangle_{XY}|x_1, и_1\rangle_{D_1} \cdots |x_s, и_s\rangle_{D_s} \cdots |\perp, 0\rangle_{D_q}, \tag{26}$$

where $\vec{x} := (x_1, x_2, \ldots, x_s)$ and $\vec{и} := (и_1, и_2, \ldots, и_s)$, additionally note that no x_i in $\vec{x}$ is $\perp$ and no $и_i$ in $\vec{и}$ is zero. We study the action of $\mathsf{Dec}_{\mathfrak{D}}$ on the above state. To write the output state we need to name the matrix elements of the sampling unitary: $(\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}))_{f\vec{и}} =$

$a_{f\vec{\text{и}}}(\mathcal{X})$, the column index consists of a vector of size M with exactly s non-zero entries: $\vec{\text{и}} = (0, \ldots, 0, \text{и}_1, 0 \ldots, 0, \text{и}_2, 0, \ldots)$. The decompressed state is

$$|\Upsilon(\vec{x}, \vec{\text{и}})\rangle_F := \mathsf{Dec}_{\mathfrak{D}}|\text{Д}(\vec{x}, \vec{\text{и}})\rangle = \sum_{\phi \in \mathcal{F}} \frac{1}{\sqrt{N^M}} \sum_{f \in \mathcal{F}} \omega_N^{\phi \cdot f} \; a_{f\vec{\text{и}}}(\mathcal{X}) \; |\phi_0\rangle_{F(0)} \cdots |\phi_{M-1}\rangle_{F(M-1)}, \tag{27}$$

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

Using locality of $\mathsf{Samp}_{\mathfrak{D}}$ as defined in Eq. (22), we have that $\mathsf{Samp}_{\mathfrak{D}}(\mathcal{X}) = \mathsf{Samp}_{\mathfrak{D}}(\mathcal{X} \setminus \{x\} \mid \{x\}) \circ \mathsf{Samp}_{\mathfrak{D}}(x)$ and we can focus our attention on some fixed x: isolate register $F(x)$ with amplitudes depending only on x. Let us compute this state after application of FO, note that FO only subtracts η from $\mathsf{F}(x)$:

$$\begin{aligned} \mathsf{FO}|x, \eta\rangle_{XY}|\Upsilon(\vec{x}, \vec{\text{и}})\rangle_F = |x, \eta\rangle_{XY} & \sum_{\phi', f' \in \mathcal{F}(\mathcal{X} \setminus \{x\})} \frac{1}{\sqrt{N^{M-1}}} \, \omega_N^{\phi' \cdot f'} \; a_{f'\vec{\text{и}}'}(\mathcal{X} \setminus \{x\} \mid \{x\}) \\ \cdot |\phi_0\rangle_{F(0)} \cdots & \left(\sum_{\zeta, z \in [N]} \frac{1}{\sqrt{N}} \, \omega_N^{\zeta \cdot z} \; a_{z\text{и}_x}(x) \; |\zeta - \eta\rangle_{F(x)} \right) \cdots |\phi_{M-1}\rangle_{F(M-1)}, \end{aligned} \tag{28}$$

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. (25) actually equals the left hand side that we just analyzed. Let us inspect $|\text{Д}(\vec{x}, \vec{\text{и}})\rangle$ after application of the compressed oracle

$$\begin{aligned} & \mathsf{CFO}_{\mathfrak{D}}|x, \eta\rangle_{XY}|\text{Д}(\vec{x}, \vec{\text{и}})\rangle_D = |x, \eta\rangle_{XY} \\ & \cdot \left(\sum_{\text{и} \neq 0} \alpha(x, \eta, \tilde{\text{и}}, \text{и}_x) \; |\text{Д}'_{\text{ADD/UPD}}\rangle_D + \alpha(x, \eta, \tilde{\text{и}}, 0) \; |\text{Д}'_{\text{REM/NOT}}\rangle_D \right) \end{aligned} \tag{29}$$

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

Before we proceed with decompression of the above state let us calculate the amplitudes α. Again using locality of $\mathsf{Samp}_{\mathfrak{D}}$ we describe the action of the compressed oracle on a single x step by step. Below we denote by Rem removing $\text{и} = 0$ from Д and by Sub subtraction of η from database register Δ^Y. We start with a database containing $(x, \tilde{\text{и}})$, which we can always assume due to line 3 in Alg. 3. In the case that x was not already in Д we have $\tilde{\text{и}} = 0$, otherwise it is the value defined in previous queries. The simplification we make is describing $\mathsf{CFO}_{\mathfrak{D}}$ acting on a single-entry database. We do not lose generality by that as the only thing that changes for q larger than one is maintaining proper sorting and padding, which can be easily done (see Appendix A.2 for details). The calculation of $\mathsf{CFO}_{\mathfrak{D}}$ on a basis state follows:

$$|x, \eta\rangle_{XY}|x, \tilde{\text{и}}_x\rangle_D \overset{\mathsf{Samp}_{\mathfrak{D}}}{\mapsto} |x, \eta\rangle_{XY} \sum_{z \in [N]} a_{z\tilde{\text{и}}_x}(x) \; |x, z\rangle_D \tag{30}$$

$$\overset{\mathsf{QFT}_N^{D^Y}}{\mapsto} |x, \eta\rangle_{XY} \sum_{z \in [N]} a_{z\tilde{\text{и}}_x}(x) \sum_{\zeta \in [N]} \frac{1}{\sqrt{N}} \, \omega_N^{\zeta \cdot z} \; |x, \zeta\rangle_D \tag{31}$$

$$\overset{\mathsf{Sub}}{\mapsto} |x, \eta\rangle_{XY} \sum_{z, \zeta \in [N]} a_{z\tilde{\text{и}}_x}(x) \frac{1}{\sqrt{N}} \, \omega_N^{\zeta \cdot z} \; |x, \zeta - \eta\rangle_D \tag{32}$$

$$\overset{\mathsf{QFT}_N^{\dagger D^Y}}{\mapsto} |x, \eta\rangle_{XY} \sum_{z, \zeta \in [N]} a_{z\tilde{\text{и}}_x}(x) \frac{1}{\sqrt{N}} \, \omega_N^{\zeta \cdot z} \sum_{z' \in [N]} \frac{1}{\sqrt{N}} \, \bar{\omega}_N^{z' \cdot (\zeta - \eta)} \; |x, z'\rangle_D \tag{33}$$

$$=|x,\eta\rangle_{XY}\sum_{z\in[N]} a_{z\tilde{й}_x}(x) \underbrace{\sum_{z',\zeta\in[N]} \frac{1}{N}\,\omega_N^{\zeta\cdot z}\,\bar{\omega}_N^{z'\cdot(\zeta-\eta)}}_{=\bar{\omega}_N^{-z\cdot\eta}\,\delta(z',z)} |x,z'\rangle_D \tag{34}$$

$$\overset{\mathsf{Samp}^{\dagger D}_{\mathfrak{D}}(x)}{\mapsto}|x,\eta\rangle_{XY}\sum_{z\in[N]} a_{z\tilde{й}_x}(x)\,\omega_N^{z\cdot\eta}\sum_{и\in[N]}\bar{a}_{zи}(x)\,|x,и\rangle_D \tag{35}$$

$$=|x,\eta\rangle_{XY}\sum_{и\in[N]}\underbrace{\sum_{z\in[N]} a_{z\tilde{й}_x}(x)\,\omega_N^{z\cdot\eta}\,\bar{a}_{zи}(x)}_{:=\alpha(x,\eta,\tilde{й},и)}|x,и\rangle_D \tag{36}$$

$$\overset{\mathsf{Rem}^D}{\mapsto}|x,\eta\rangle_{XY}\left(\sum_{и\in[N]\setminus\{0\}}\alpha(x,\eta,\tilde{й}_x,и)\,|x,и\rangle_D+\alpha(x,\eta,\tilde{й}_x,0)\,|\bot,0\rangle_D\right). \tag{37}$$

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

$$\begin{aligned}\mathsf{Samp}^{\dagger D}_{\mathfrak{D}}(\mathcal{X}\setminus\{x\}\mid\{x\})\circ\mathsf{QFT}_N^{D_x^Y}\,\mathsf{Sub}^{D_x^Y}\,\mathsf{QFT}_N^{\dagger D_x^Y}\circ\mathsf{Samp}^{D}_{\mathfrak{D}}(\mathcal{X}\setminus\{x\}\mid\{x\})\\ =\mathsf{QFT}_N^{D_x^Y}\,\mathsf{Sub}^{D_x^Y}\,\mathsf{QFT}_N^{\dagger D_x^Y}.\end{aligned} \tag{38}$$

To prove the above statement we first note that $\mathsf{Samp}^{D}_{\mathfrak{D}}(\mathcal{X}\setminus\{x\}\mid\{x\})$ is controlled on register D_x^Y, but does not act on it. Secondly, in Eq.(34) we see that $\mathsf{QFT}_N^{D^Y}$ Sub $\mathsf{QFT}_N^{\dagger D^Y}$ multiplies the state by phase factor $\omega_N^{z\cdot\eta}$ and also does not modify register D_x^Y, but is controlled on it though. Hence Eq.(38) holds. In the above equations we have defined α as

$$\alpha(x,\eta,\tilde{й}_x,и):=\sum_{z\in[N]} a_{z\tilde{й}_x}(x)\,\bar{a}_{zи}(x)\,\omega_N^{z\cdot\eta}. \tag{39}$$

After decompressing the state from Eq.(29), the resulting database state will be $\sum_{и\neq0}\alpha(x,\eta,\tilde{й}_x,и_x)\ |\Upsilon(Д'_{\text{ADD/UPD}})\rangle+\alpha(x,\eta,\tilde{й}'_x,0)\ |\Upsilon(Д'_{\text{REM/NOT}})\rangle_D$, 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.(28):

$$\begin{aligned}&\mathsf{Dec}_{\mathfrak{D}}\circ\mathsf{CFO}_{\mathfrak{D}}|x,\eta\rangle_{XY}|Д(\vec{x},\vec{и})\rangle_D=\sum_{\phi',f'\in\mathcal{F}(\mathcal{X}\setminus\{x\})}\frac{1}{\sqrt{N^{M-1}}}\,\omega_N^{\phi'\cdot f'}\,a_{f'\vec{й}'}(\mathcal{X}\setminus\{x\}\mid\{x\})\,|\phi_0\rangle_{F(0)}\cdots\\ &\underbrace{\cdot\left(\sum_{и_x\neq0}\alpha(x,\eta,\tilde{й},и_x)\sum_{\zeta,z\in[N]}\frac{1}{\sqrt{N}}\omega_N^{\zeta\cdot z}a_{zи_x}(x)|\zeta\rangle_{F(x)}+\alpha(x,\eta,\tilde{й},0)\sum_{\zeta,z\in[N]}\frac{1}{\sqrt{N}}\omega_N^{\zeta\cdot z}a_{z0}(x)|\zeta\rangle_{F(x)}\right)}_{=\sum_{\zeta,z\in[N]}\frac{1}{\sqrt{N}}\,\omega_N^{\zeta\cdot z}\sum_{и_x\in[N]}\alpha(x,\eta,\tilde{й},и_x)\,a_{zи_x}(x)\,|\zeta\rangle_{F(x)}}\\ &\cdots|\phi_{M-1}\rangle_{F(M-1)}=\mathsf{FO}\ |x,\eta\rangle_{XY}|\Upsilon(\vec{x},\vec{и})\rangle=\mathsf{FO}\circ\mathsf{Dec}_{\mathfrak{D}}\ |x,\eta\rangle_{XY}|Д(\vec{x},\vec{и})\rangle_D.\end{aligned} \tag{40}$$

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

$$\sum_{и\in[N]}\alpha(x,\eta,\tilde{й},и)\,a_{zи_x}(x)=\sum_{z'\in[N]}\underbrace{\sum_{и\in[N]}\bar{a}_{z'и}(x)\,a_{zи}(x)}_{=\delta_{z',z}}a_{z'\tilde{й}}(x)\,\omega_N^{z'\cdot\eta}=a_{z\tilde{й}}(x)\,\omega_N^{z\cdot\eta}. \tag{41}$$

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

4.2.1 Example 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 Appendix A.

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

$$\mathsf{Samp}_\lambda(x) := \begin{pmatrix} \sqrt{1-\lambda} & \sqrt{\lambda} \\ \sqrt{\lambda} & -\sqrt{1-\lambda} \end{pmatrix}, \tag{42}$$

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

5 One-way to Hiding Lemma for Compressed Oracles

The fundamental game-playing lemma, Lemma 5, 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 5 valid for quantum accessible oracles is the *One-Way to Hiding* (O2H) Lemma first introduced by Unruh in [Unr14].

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

Recently this technique was generalized by Ambainis, Hamburg, and Unruh in [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 succeeding.

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

Below we state our generalized O2H lemmas. Most proofs of [AHU19] apply almost word by word so we just describe the differences and refer the reader to the original work.

5.1 Relations on databases

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

Definition 10 (Classical relation R on D)**.** *Let D be a database of size q of pairs $(x,y) \in \mathcal{X} \times \mathcal{Y}$. We call a subset $R \subseteq \bigcup_{s\in[q+1]} (\mathcal{X} \times \mathcal{Y})^s$ a* classical relation R on D*.*

An example of such a relation is a collision, namely

$$R_{\text{coll}} := \{((x_1,y_1),\cdots,(x_t,y_t)) \in \bigcup_{s\in[q+1]} (\mathcal{X} \times \mathcal{Y})^s : \exists_{i,j}\; i \neq j, x_i \neq x_j, y_i = y_j\}. \tag{43}$$

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. We also write $\mathcal{B}$ instead of R (for some $\mathcal{B} \subseteq \mathcal{Y}$), then the relation is defined as entries of D that have $y_i \in \mathcal{B}$. 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.

A more general way to view a relation on D is by identifying it with any quantum measurement.

Definition 11 (Relation R on D)**.** *Let $|D\rangle$ be a quantum database in the Hilbert space $\mathcal{H}_D$. We call a binary measurement on $\mathcal{H}_D$ a* relation R on $|D\rangle$.

This way of defining relations and punctured oracles will be very useful later on. An example of a relation defined in this more general way is $R_{\xi=0}$ that is fulfilled by $|D\rangle$ with D^X containing a Fourier 0.

We also state an explicit algorithm to implement the measurement of a relation R, given that membership in R is efficiently decidable. Alg. 6 defines the measurement procedure of measuring R.

Algorithm 6: Measurement of a relation R

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

The above discussion brings some new issues though, especially when we consider measuring on two relations at the same time. From Heisenberg's uncertainty principle we know that one cannot make two non-commuting measurements at the same time. For example if the two relations are defined on D in two different bases then the measurements do not commute in general. We will not discuss the matter further, we will just limit the discussion to *commuting relations* that have commuting measurements.

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 12 (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. 6. 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—that need to be identified (like in $\mathsf{Dec}_{\mathfrak{D}}$ from Alg. 5) prior to the measurement of R.

In many applications of punctured oracles H we might want to apply different compressed oracles (also punctured) conditioned on some quantum state. Such an operation is not permitted by quantum mechanics; We can achieve the same functionality by simply postponing the measurement to the end of the quantum algorithm involving H controlled on a quantum register. Note that this change can be done by just omitting the measurement of register J in Alg. 6 and performing it at the end of the whole 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. 9.

5.2 One-way to Hiding Lemma

Using this definition we can prove a theorem similar to Theorem 1 of [AHU19]:

Theorem 13 (Compressed oracle O2H)**.** *Let R_1 and R_2 be commuting 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*

$$\left|\mathbb{P}[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R_1 \cup R_2}(z)\right| \leq \sqrt{(d+1)\mathbb{P}[\mathrm{Find}_2 : \mathsf{A}^{\mathsf{H}\setminus R_1 \cup R_2}(z)]}, \tag{44}$$

$$\left|\sqrt{\mathbb{P}[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R_1}(z)]} - \sqrt{\mathbb{P}[b = 1 : b \leftarrow \mathsf{A}^{\mathsf{H}\setminus R_1 \cup R_2}(z)}\right| \leq \sqrt{(d+1)\mathbb{P}[\mathrm{Find}_2 : \mathsf{A}^{\mathsf{H}\setminus R_1 \cup R_2}(z)]}, \tag{45}$$

where Find$_2$ *is the event that measuring R_2 succeeds.*

Proof. 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

$$\mathsf{U}_{R,i}|D\rangle_D|l_1, l_2, \ldots, l_d\rangle_L := \begin{cases} |D\rangle_D|l_1, l_2, \ldots, l_d\rangle_L & \text{if } R(|D\rangle_D) = 0 \\ |D\rangle_D|l_1, \ldots, l_i \oplus 1, \ldots, l_d\rangle_L & \text{if } R(|D\rangle_D) = 1 \end{cases}, \tag{46}$$

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

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

Let $|\Psi_{\mathsf{A}}\rangle$ denote the final state of $\mathsf{A}^{\mathsf{H}}(z)$, and $|\Psi_{\mathsf{B}}\rangle$ the final state of $\mathsf{B}^{\mathsf{H},R}(z)$. Let $\tilde{P}_{\text{find}}$ be the probability that a measurement of L in the computational basis in the state $|\Psi_{\mathsf{B}}\rangle$ returns $l \neq 0^d$, 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 U_R as in alg. 6, 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 inton the compressed oracle unitary.

Lemma 14 (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*

$$\left\| |\Psi_{\mathsf{A}}\rangle \otimes |0^d\rangle_L - |\Psi_{\mathsf{B}}\rangle \right\|^2 \leq (d+1)\tilde{P}_{\text{find}}. \tag{47}$$

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

$$\begin{aligned} \left\| |\Psi_{\mathsf{B}}\rangle - |\Psi_{\mathsf{A}}\rangle \otimes |0^d\rangle_L \right\|^2 &= \left\| \sum_{i=1}^{d} |\Psi^i_{\mathsf{B}_{\text{count}}}) \right\|^2 + \tilde{P}_{\text{find}} \overset{\triangle}{\leq} \left(\sum_{i=1}^{d} \left\| |\Psi^i_{\mathsf{B}_{\text{count}}}) \right\| \right)^2 + \tilde{P}_{\text{find}} \\ &\overset{\text{J-I}}{\leq} \underbrace{d \sum_{i=1}^{d} \left\| |\Psi^i_{\mathsf{B}_{\text{count}}}) \right\|^2}_{=\tilde{P}_{\text{find}}} + \tilde{P}_{\text{find}} = (d+1)\tilde{P}_{\text{find}}, \end{aligned} \tag{48}$$

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. □

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

$$\mathbb{P}[\text{Find} : \mathsf{A}^{\mathsf{H}\setminus R}(z)] = 1 - \left\| \prod_{i=1}^{d} (\mathbb{1} - \mathsf{J}_R) \mathsf{U}^{\mathsf{H}}_j |\Psi_0\rangle \right\|^2. \tag{49}$$

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 15 (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 event* E*, any strings* y, z*, and any quantum algorithm* A

$$\mathbb{P}[E : y \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \neg\mathrm{Find}_1] = \mathbb{P}[E : y \leftarrow A^{\mathsf{G}\setminus R_2}(z) \mid \neg\mathrm{Find}_2]. \tag{50}$$

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 16 (Distinguishing almost identical punctured oracles)**.** *If* $\mathsf{H} \setminus R_1$ *and* $\mathsf{G} \setminus R_2$ *are almost identical according to Def.15 then*

$$\left|\mathbb{P}[y \leftarrow A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[y \leftarrow A^{\mathsf{G}\setminus R_2}(z)]\right| \leq 2\mathbb{P}[\mathrm{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] + 2\mathbb{P}[\mathrm{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]. \tag{51}$$

Proof. We bound

$$\begin{aligned}
&\left|\mathbb{P}[y \leftarrow A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[y \leftarrow A^{\mathsf{G}\setminus R_2}(z)]\right| \\
&\overset{\text{Def. 15}}{=} \Big|\mathbb{P}[y \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \neg\mathrm{Find}_1]\Big(\mathbb{P}[\neg\mathrm{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[\neg\mathrm{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]\Big) \\
&+ \mathbb{P}[y \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \mathrm{Find}_1]\mathbb{P}[\mathrm{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] \\
&-\mathbb{P}[y \leftarrow A^{\mathsf{G}\setminus R_2}(z) \mid \mathrm{Find}_2]\mathbb{P}[\mathrm{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]\Big| \qquad (52) \\
&\overset{\triangle}{\leq} \left|\underbrace{\mathbb{P}[y \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \neg\mathrm{Find}_1]}_{\leq 1}\underbrace{\Big(\mathbb{P}[\neg\mathrm{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] - \mathbb{P}[\neg\mathrm{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]\Big)}_{=\mathbb{P}[\mathrm{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)] - \mathbb{P}[\mathrm{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)]}\right| \\
&+ \left|\underbrace{\mathbb{P}[y \leftarrow A^{\mathsf{H}\setminus R_1}(z) \mid \mathrm{Find}_1]}_{\leq 1}\mathbb{P}[\mathrm{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)]\right| \\
&+ \left|\underbrace{\mathbb{P}[y \leftarrow A^{\mathsf{G}\setminus R_2}(z) \mid \mathrm{Find}_2]}_{\leq 1}\mathbb{P}[\mathrm{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)]\right| \qquad (53) \\
&\overset{\triangle}{\leq} 2\mathbb{P}[\mathrm{Find}_1 : A^{\mathsf{H}\setminus R_1}(z)] + 2\mathbb{P}[\mathrm{Find}_2 : A^{\mathsf{G}\setminus R_2}(z)], \qquad (54)
\end{aligned}$$

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. (43), 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.

5.3 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:

$$R_{\text{preim}} := \{((x_1, y_1), \cdots, (x_t, y_t)) \in \bigcup_{s \in [q+1]} (\mathcal{X} \times \mathcal{Y})^s : \exists i : y_i = 0\}. \tag{55}$$

Lemma 17. *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.* (43) *and* R_{preim} *in Eq.* (55)*—the probability of* Find *is bounded by:*

$$\mathbb{P}[\text{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}} \setminus (R_{\text{preim}} \cup R_{\text{coll}})]] \leq 36 \frac{q^5}{|\mathcal{Y}|}, \tag{56}$$

where q is the maximal number of queries made by A*.*

Proof. Punctured oracles are defined in Definition 12. We start the proof by specifying some operations involved in that definition. 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. 3 in lines 1 and 13: counts the number of non-padding ($x \neq \perp$) entries.

The full description of the measurement involves using an auxiliary register J—note the definition of measuring a relation Def. 6—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 5 of Alg. 6 is

$$\mathsf{J}_R := \mathbb{1} \otimes |1\rangle_J\langle 1|, \tag{57}$$

$$\bar{\mathsf{J}}_R := \mathbb{1} \otimes |0\rangle_J\langle 0|, \tag{58}$$

where in addition to checking R.

In the following we focus on the punctured oracle just prior to measurement J_R. A unitary that omits the last step of Alg. 6 in $\mathsf{CPhO}_{\mathcal{Y}} \setminus R_{\text{preim}} \cup R_{\text{coll}}$ acts on registers ADJ, we define it as

$$\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R := \mathsf{Queries}^\dagger \circ \mathsf{U}_R \circ \mathsf{Queries} \circ \mathsf{CPhO}_{\mathcal{Y}}, \tag{59}$$

where the unitary U_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. (43), (55)—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. (49) to analyze the probability of Find:

$$\mathbb{P}[\text{Find} : \mathsf{A}[\mathsf{CPhO}_{\mathcal{Y}} \setminus R_{\text{preim}} \cup R_{\text{coll}}]] = 1 - \left\| \left(\prod_{i=q}^{1} \bar{\mathsf{J}}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \right) |\Psi_0\rangle|0\rangle_J \right\|^2 \tag{60}$$

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

$$+ \left\| \mathsf{J}_R \mathsf{U}_q \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \left(\prod_{i=q-1}^{1} \bar{\mathsf{J}}_R \mathsf{U}_j \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \right) |\Psi_0\rangle|0\rangle_J \right\|^2 \tag{61}$$

$$= \sum_{i=1}^{q} \left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \underbrace{\mathsf{U}_R \left(\prod_{j=i-1}^{1} \bar{\mathsf{J}}_R \mathsf{U}_j \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \right) |\Psi_0\rangle|0\rangle_J}_{:=\mathsf{U}_{i-1}|\Phi_{i-1}\rangle} \right\|^2 \tag{62}$$

$$= \sum_{i=1}^{q} \left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle \right\|^2, \tag{63}$$

where $|\Psi_0\rangle$ is the initial state of the adversary. Note that in the definition of $|\Phi_{i-1}\rangle$ we have $[\mathsf{U}_{i-1}, \bar{\mathsf{J}}_R] = 0$. 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 $\|\mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle\|^2$. Our approach to that is to propose a state $|\Psi_{i-1}^{\text{Good}}\rangle$, close to the original $|\Phi_{i-1}\rangle$, for which bounding $\left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Psi_{i-1}^{\text{Good}}\rangle|0\rangle_J \right\|^2$ 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

$$\mathcal{B}(s) := [N]^s \setminus \{(y_1, \ldots, y_s) \in [N]^s : \text{all } y_i \text{ are distinct and } \neq 0\}, \tag{64}$$

$$\mathcal{B}(1 \mid D) := \{y\}_{y \in D^Y} \cup \{0\}. \tag{65}$$

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

$$b(s) := |\mathcal{B}(1 \mid D)|, \text{ where } D \notin \mathcal{B}(s-1), \tag{66}$$

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. With the bad set defined for other relations let us present the corresponding coefficient: As examples consider R_{preim}, then $b(s) = 1$, there is just one value $y = 0$ that cause 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 $\neq 0$, matching any of them or 0 causes $D^Y \cup \{y\}$ to be in $\mathcal{B}(s)$. Through the rest of this proof we do not evaluate $b(s)$, so it is easier to reuse the proof for other relations.

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 $|\Psi_{i,R}^{\text{Good}}\rangle_{AD}$ 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 is a superposition over different database sizes. We denote the outputs given to A by $\vec{y} := (y_1, \dots, y_s)$. 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. We define the good state as:

$$|\Psi_{i,R}^{\text{Good}}\rangle_{AD} := \sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x,\eta\rangle_{A^{XY}} |\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W}$$
$$\sum_{\vec{y} \notin \mathcal{B}(s)} \frac{1}{\sqrt{(N-b(1))(N-b(2))\cdots(N-b(s))}} \omega_N^{\vec{\eta}\cdot\vec{y}} |(x_1,y_1),\dots,(x_s,y_s)\rangle_{D(\vec{x})}$$
$$\sum_{y_{s+1},\dots,y_q \in [N]} \frac{1}{\sqrt{N^{q-s}}} |(\perp, y_{s+1}),\dots,(\perp,y_q)\rangle_{D(\perp)}, \tag{67}$$

in the 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 $|\Psi_i^{\text{Good}}\rangle$ does indeed depend on R.

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 18. *For states defined as above we have*

$$\left\| |\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J - |\Phi_i\rangle_{ADJ} \right\| \le \frac{i(i+1)}{2} \left(\frac{\sqrt{2}}{\sqrt{N(N-q)}} + \frac{1}{N} \right). \tag{68}$$

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:

$$\left\| |\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J - |\Phi_i\rangle_{ADSJ} \right\| = \left\| |\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J - \bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle_{ADJ} \right\| \tag{69}$$
$$\le \left\| |\Psi_i^{\text{Good}}\rangle_{AD}|0\rangle_J - \bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Psi_{i-1}^{\text{Good}}\rangle_{AD}|0\rangle_J \right\|$$
$$+ \left\| \bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Psi_{i-1}^{\text{Good}}\rangle_{AD}|0\rangle_J - \bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle_{ADJ} \right\| \tag{70}$$
$$\le \varepsilon_{\text{step}}(i) + \left\| |\Psi_{i-1}^{\text{Good}}\rangle_{AD}|0\rangle_J - |\Phi_{i-1}\rangle_{ADJ} \right\| \le \sum_{j=1}^{i} \varepsilon_{\text{step}}(j). \tag{71}$$

We just need to calculate $\varepsilon_{\text{step}}(j) := \left\| |\Psi_j^{\text{Good}}\rangle_{AD}|0\rangle_J - \bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1} |\Psi_{j-1}^{\text{Good}}\rangle_{AD}|0\rangle_J \right\|_2$. We start calculating the claimed bound by inspecting in detail the state $\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1} |\Psi_{j-1}^{\text{Good}}\rangle_{AD}|0\rangle_J$. 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 $\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1} |\Psi_{j-1}^{\text{Good}}\rangle_{AD}|0\rangle_J$. We write

$$\mathsf{U}_{j-1} |\Psi_{j-1}^{\text{Good}}\rangle_{AD}|0\rangle_J = |\xi_{\text{ADD}}\rangle + |\xi_{\text{UPD}}\rangle + |\xi_{\text{REM}}\rangle + |\xi_{\text{NOT}}\rangle \tag{72}$$

and analyze the action of $\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R$ on the $|\xi_i\rangle$ separately.

For $|\xi_{\text{NOT}}\rangle$ there is no change to the state. For $|\xi_{\text{UPD}}\rangle$ and $|\xi_{\text{REM}}\rangle$, 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 set 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.

$$
\begin{aligned}
&\text{UPD/REM}: \\
&\mathsf{CPhO}_{\mathcal{Y}}\left(|\xi_{\text{UPD}}\rangle + |\xi_{\text{REM}}\rangle\right) = \sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x,\eta\rangle_{A^{XY}} |\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W} \\
&\sum_{\vec{y}\notin\mathcal{B}(s-1)} \frac{1}{\sqrt{(N-b(1))(N-b(2))\cdots(N-b(s-1))}} \omega_N^{\vec{\eta}\cdot\vec{y}} |(x_1,y_1),\ldots,(x_{s-1},y_{s-1})\rangle_{D(\vec{x}\setminus\{x\})} \\
&\Bigg(\sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \frac{1}{\sqrt{N-b(s)}} \omega_N^{(\eta_s+\eta)y_s} |x,y_s\rangle_{D(x)} \\
&- \frac{1}{\sqrt{N(N-b(s))}} \sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \omega_N^{(\eta_s+\eta)y_s} \sum_{y'_s\in[N]} \frac{1}{\sqrt{N}} |x,y'_s\rangle_{D(x)} \\
&+ \frac{1}{\sqrt{N(N-b(s))}} \sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \omega_N^{(\eta_s+\eta)y_s} \sum_{y'_s\in[N]} \frac{1}{\sqrt{N}} |\perp,y'_s\rangle_{D(x)} \Bigg) \\
&\sum_{y_{s+1},\ldots,y_q\in[N]} \frac{1}{\sqrt{N^{q-s}}} |(\perp,y_{s+1}),\ldots,(\perp,y_q)\rangle_{D(\perp)}. \qquad (73)
\end{aligned}
$$

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:

$$
\begin{aligned}
&\text{UPD}: \\
&\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R |\xi_{\text{UPD}}\rangle = \sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x,\eta\rangle_{A^{XY}} |\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W} \\
&\sum_{\vec{y}\notin\mathcal{B}(s-1)} \frac{1}{\sqrt{(N-b(1))(N-b(2))\cdots(N-b(s-1))}} \omega_N^{\vec{\eta}\cdot\vec{y}} |(x_1,y_1),\ldots,(x_{s-1},y_{s-1})\rangle_{D(\vec{x}\setminus\{x\})} \\
&\Bigg(\sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \frac{1}{\sqrt{N-b(s)}} \omega_N^{(\eta_s+\eta)y_s} |x,y_s\rangle_{D(x)} |0\rangle_J \\
&+ \frac{1}{N} \sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \omega_N^{(\eta_s+\eta)y_s} \sum_{y'_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \frac{1}{\sqrt{N-b(s)}} |x,y'_s\rangle_{D(x)} |0\rangle_J \\
&+ \sqrt{\frac{b(s)}{N^2(N-b(s))}} \sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \omega_N^{(\eta_s+\eta)y_s} \sum_{y'_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \frac{1}{\sqrt{b(s)}} |x,y'_s\rangle_{D(x)} |1\rangle_J \\
&- \frac{1}{\sqrt{N(N-b(s))}} \sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} \omega_N^{(\eta_s+\eta)y_s} \sum_{y'_s\in[N]} \frac{1}{\sqrt{N}} |\perp,y'_s\rangle_{D(x)} |0\rangle_J \Bigg) \\
&\sum_{y_{s+1},\ldots,y_q\in[N]} \frac{1}{\sqrt{N^{q-s}}} |(\perp,y_{s+1}),\ldots,(\perp,y_q)\rangle_{D(\perp)}. \qquad (74)
\end{aligned}
$$

In the above state we have simplified the sum $\sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))} = -\sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}$.

After removing an element from the database we have:

$$
\begin{aligned}
&\text{REM}: \\
&\mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R |\xi_{\text{REM}}\rangle = \sum_{x,\eta,\vec{x},\vec{\eta},w} \alpha_{x,\eta,\vec{x},\vec{\eta},w} |x,\eta\rangle_{A^{XY}} |\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W} \\
&\sum_{\vec{y}\notin\mathcal{B}(s-1)} \frac{1}{\sqrt{(N-b(1))(N-b(2))\cdots(N-b(s-1))}} \omega_N^{\vec{\eta}\cdot\vec{y}} |(x_1,y_1),\ldots,(x_{s-1},y_{s-1})\rangle_{D(\vec{x}\setminus\{x\})}
\end{aligned}
$$

$$\left(\sqrt{\frac{N-b(s)}{N}}\sum_{y_s\in[N]}\frac{1}{\sqrt{N}}|\bot,y_s\rangle_{D(x)}|0\rangle_J\right.$$
$$+\frac{b(s)}{N}\sum_{y_s\notin\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\frac{1}{\sqrt{N-b(s)}}\omega_N^{(\eta_s+\eta)y_s}|x,y_s\rangle_{D(x)}|0\rangle_J$$
$$\left.-\frac{\sqrt{b(s)(N-b(s))}}{N}\sum_{y_s\in\mathcal{B}(1|D(\vec{x}\setminus\{x\}))}\frac{1}{\sqrt{b(s)}}|x,y_s\rangle_{D(x)}\right)$$
$$\sum_{y_{s+1},\dots,y_q\in[N]}\frac{1}{\sqrt{N^{q-s}}}|(\bot,y_{s+1}),\dots,(\bot,y_q)\rangle_{D(\bot)}. \tag{75}$$

Now we get to adding a new entry to the database:

$$\text{ADD}:$$
$$\mathsf{CPhO}_{\mathcal{Y}}|\xi_{\text{ADD}}\rangle=\sum_{x,\eta,\vec{x},\vec{\eta},w}\alpha_{x,\eta,\vec{x},\vec{\eta},w}|x,\eta\rangle_{A^{XY}}|\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W}$$
$$\sum_{\vec{y}\notin\mathcal{B}(s)}\frac{1}{\sqrt{(N-b(1))(N-b(2))\cdots(N-b(s))}}\omega_N^{\vec{\eta}\cdot\vec{y}}|(x_1,y_1),\dots,(x_s,y_s)\rangle_{D(\vec{x})}$$
$$\sum_{y_{s+1}\in[N]}\frac{1}{\sqrt{N}}\omega_N^{\eta y_{s+1}}|x,y_{s+1}\rangle_{D(x)}\sum_{y_{s+2},\dots,y_q\in[N]}\frac{1}{\sqrt{N^{q-s-1}}}|(\bot,y_{s+2}),\dots,(\bot,y_q)\rangle_{D(\bot)}, \tag{76}$$

where for the sake of readability we again omit the proper ordering of D. Checking for R is a simple task, note that $\mathsf{Queries}$ only depends on D^X. The above state after applying $\mathsf{Queries}^\dagger\circ\mathsf{U}_R\circ\mathsf{Queries}$ is:

$$\text{ADD}:$$
$$\mathsf{CPhO}_{\mathcal{Y}}\setminus\mathsf{U}_R|\xi_{\text{ADD}}\rangle=\sum_{x,\eta,\vec{x},\vec{\eta},w}\alpha_{x,\eta,\vec{x},\vec{\eta},w}|x,\eta\rangle_{A^{XY}}|\psi(x,\eta,\vec{x},\vec{\eta},w)\rangle_{A^W}$$
$$\sum_{\vec{y}\notin\mathcal{B}(s)}\frac{1}{\sqrt{(N-b(1))\cdots(N-b(s))}}\omega_N^{\vec{\eta}\cdot\vec{y}}|(x_1,y_1),\dots,(x_s,y_s)\rangle_{D(\vec{x})}$$
$$\left(\sqrt{\frac{N-b(s+1)}{N}}\sum_{y_{s+1}\notin\mathcal{B}(1|D(\vec{x}))}\frac{1}{\sqrt{N-b(s+1)}}\omega_N^{\eta y_{s+1}}|x,y_{s+1}\rangle|0\rangle_J\right.$$
$$\left.+\sqrt{\frac{b(s+1)}{N}}\sum_{y_{s+1}\in\mathcal{B}(1|D(\vec{x}))}\frac{1}{\sqrt{b(s+1)}}\omega_N^{\eta y_{s+1}}|x,y_{s+1}\rangle|1\rangle_J\right)$$
$$\sum_{y_{s+2},\dots,y_q\in[N]}\frac{1}{\sqrt{N^{q-s-1}}}|(\bot,y_{s+2}),\dots,(\bot,y_q)\rangle_{D(\bot)}, \tag{77}$$

the appropriate position of register J is after D.

Now that we know how querying works for $|\Psi_{j-1}^{\text{Good}}\rangle$ 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 appears in states coming from applying $\mathsf{CPhO}_{\mathcal{Y}}$ (Eq. (73)).

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 $\mathsf{CPhO}_{\mathcal{Y}}\setminus\mathsf{U}_R\mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J$ by $\sqrt{\frac{N-b(s+1)}{N}}$ (Eqs. (77)). We also see this type of error is some branches where we remove the entry from D (Eqs. (73)).

Our approach to the rest of the proof is first dealing with the additive and later the multiplicative error. To this end let us define $|\psi_j^+\rangle_{ADJ}$ as the state $\bar{\mathsf{J}}_R\mathsf{CPhO}_{\mathcal{Y}}\setminus\mathsf{U}_R\mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J$

with all branches classified as the additive error excluded. The new state is defined as

$$|\psi_j^+\rangle_{ADJ} = \Bigg(\sum_s |\Psi_j^{\text{Good}}(\text{NOT}; s)\rangle + |\Psi_j^{\text{Good}}(\text{UPD}; s)\rangle \\ + \sqrt{\frac{N-b(s)}{N}}|\Psi_j^{\text{Good}}(\text{REM}; s)\rangle + \sqrt{\frac{N-b(s+1)}{N}}|\Psi_j^{\text{Good}}(\text{ADD}; s)\rangle\Bigg)|0\rangle_J, \tag{78}$$

where the states above correspond to branches of superposition where we do nothing (NOT, for $\eta = 0$), update the database, remove an entry from D, and add an entry. We add s as the argument to specify the size of the database. The multiplicative factors come from Eqs. (74), (75), and (77), the parts of the equations we look at are the first elements in the parentheses.

Bounding the difference of the states is done as follows

$$\left\| |\Psi_j^{\text{Good}}\rangle|0\rangle_J - \bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J \right\| \\ \leq \left\| |\Psi_j^{\text{Good}}\rangle|0\rangle_J - |\psi_j^+\rangle_{ADJ} \right\| + \left\| |\psi_j^+\rangle_{ADJ} - \bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J \right\|. \tag{79}$$

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

An important fact is that the joint state of the adversary and the oracle is a sum over databases of different sizes

$$|\psi_j^+\rangle = \sum_s |\psi_j^+(s)\rangle \tag{80}$$

$$\langle\psi_j^+|\psi_j^+\rangle = \sum_s \langle\psi_j^+(s)|\psi_j^+(s)\rangle \tag{81}$$

and the above is also true for $|\Psi_j^{\text{Good}}\rangle = \sum_s |\Psi_j^{\text{Good}}(s)\rangle$. To calculate the additive error we bound the norms of all orthogonal terms adding errors, the total error is the square root of the sum of squares of norms of these states. We write out only the part of the state $\bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J$ that gets modified in the branch of the superposition we focus on; We omit majority of the database and all adversary's registers. We can do this without loss of generality because the features that characterize the branch we analyze give rise to orthogonal parts of the overall states, so there is no overlap between cases we discuss; All the other amplitudes that we omit (e.g. $\alpha_{x,\eta,\vec{x},\vec{\eta},w}$) sum to 1 in absolute value squared. Whenever it is necessary to use the global features of $\bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1}|\Psi_{j-1}^{\text{Good}}\rangle|0\rangle_J$ we make it clear in the comments to the bounds.

We list all the norms important for proving Claim 18 and the bound on $\mathbb{P}[\text{Find}]$. By "bad $|0\rangle_J\langle 00|$" we denote the bad state multiplied by $|0\rangle_J$, similarly by $|01\rangle_{SJ}\langle 01|$. Note that the second error is excluded by $\bar{\mathsf{J}}_R$ but it is useful to calculate it for for the later task of calculating $\mathbb{P}[\text{Find}]$.

For updating, i.e. $\eta \neq -\eta_s$ the first state in parentheses in Eq. (73) is the good state, the rest is the error, with norm

$$\left\| \frac{1}{\sqrt{N(N-b(s))}} \sum_{y_s \notin \mathcal{B}(1|D)} \omega_N^{(\eta_s+\eta)y_s} \left(-\sum_{y'_s \notin \mathcal{B}(1|D)} \frac{1}{\sqrt{N}}|x, y'_s\rangle + \sum_{y'_s \in [N]} \frac{1}{\sqrt{N}}|\bot, y'_s\rangle \right) \right\| \\ \leq \frac{\sqrt{2}b(s)}{\sqrt{N(N-b(s))}}. \tag{82}$$

The norm of the part of the state that is in R, where in the above we change $\sum_{y'_s \notin \mathcal{B}(1|D)}$ to $\sum_{y'_s \in \mathcal{B}(1|D)}$ is

$$\sqrt{\frac{2b(s)^3}{N^2(N-b(s))}}. \tag{83}$$

For removing, i.e. $\eta = -\eta_s$ in Eq. (73) the last state in the parentheses is the good state, the rest is the error, with norm

$$\left\| \frac{1}{\sqrt{N-b(s)}} \sum_{y_s \notin \mathcal{B}(1|D)} |x, y_s\rangle - \frac{\sqrt{N-b(s)}}{\sqrt{N}} \sum_{y'_s \notin \mathcal{B}(1|D)} \frac{1}{\sqrt{N}} |x, y'_s\rangle \right\| = \frac{b(s)}{N} \tag{84}$$

In Eq. (77) there is no additive error, we just calculate the norm of the part of the state with database in R. The crucial quantity for checking for R in Eq. (77) is the branch multiplied by $|1\rangle_J$. The norm of this branch is bounded by

$$\sqrt{\frac{b(s+1)}{N}}. \tag{85}$$

To understand the above bound note that the state in Eq. (77) that is limited to the state we analyze here—so without the first element in the parentheses—is essentially equal to $|\Psi_j^{\text{Good}}\rangle$ with the exception of the range of the sum $\sum_{y_{s+1} \in \mathcal{B}(1|D(\vec{x}))}$. Now as η is given explicitly in register A^Y the inner product of registers $D(x)$ simplifies to $\sum_{y_{s+1} \in \mathcal{B}(1|D(\vec{x}))} \frac{1}{N}$, all other elements is the above norm appear in $\langle \Psi_j^{\text{Good}} | \Psi_j^{\text{Good}} \rangle$, hence are ≤ 1

The errors listed above are all weighted by the amplitudes in the state $\bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1} |\Psi_{j-1}^{\text{Good}}\rangle |0\rangle_J$; Hence, the total additive error is the maximal value that we calculated above:

$$\left\| |\psi_j^+\rangle_{ADJ} - \bar{\mathsf{J}}_R \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{j-1} |\Psi_{j-1}^{\text{Good}}\rangle |0\rangle_J \right\| \leq \frac{\sqrt{2} b(s)}{\sqrt{N(N-b(s))}} \leq \frac{\sqrt{2} b(j-1)}{\sqrt{N(N-b(q))}}, \tag{86}$$

where the bound comes from Eq. (82).

The multiplicative error is a factor that multiplies a part of the state $|\psi_j^+\rangle_{ADJ}$. We also 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, note Eq. (80). Let us write down the two parts, one affected by the error and the second not:

$$|\Psi_j^{\text{Good}}\rangle_{AD} |0\rangle_J = \sum_s \alpha(s) |\varphi_1(s)\rangle + \beta(s) |\varphi_2(s)\rangle, \tag{87}$$

$$|\psi_j^+\rangle_{ADJ} = \sum_s \alpha(s) |\varphi_1(s)\rangle + \sqrt{\frac{N-b(s)}{N}} \beta(s) |\varphi_2(s)\rangle \tag{88}$$

and we know that $\sum_s |\alpha(s)|^2 + |\beta(s)|^2 = 1$, so $\sum_s |\beta(s)|^2 \leq 1$. We continue with the bound

$$\left\| |\psi_j^+\rangle_{ADJ} - |\Psi_j^{\text{Good}}\rangle_{AD} |0\rangle_J \right\| = \left\| \sum_s \left(1 - \sqrt{\frac{N-b(s)}{N}} \right) \beta(s) |\varphi_2(s)\rangle \right\| \tag{89}$$

$$= \sqrt{\sum_s \left(1 - \sqrt{\frac{N-b(s)}{N}} \right)^2 |\beta(s)|^2} \leq \left(1 - \sqrt{\frac{N-b(j)}{N}} \right) \leq \frac{b(j)}{N} \tag{90}$$

From Eqs. (79), (86), and (90) the bound on the single step is

$$\varepsilon_{\text{step}}(j) \leq \frac{\sqrt{2} b(j-1)}{\sqrt{N(N-b(q))}} + \frac{b(j)}{N} \tag{91}$$

and the final bound is

$$\left\| |\Psi_i^{\text{Good}}\rangle_{AD} |0\rangle_J - |\Phi_i\rangle_{ADJ} \right\| \leq \sum_{j=1}^{i} j \left(\frac{\sqrt{2}}{\sqrt{N(N-b(q))}} + \frac{1}{N} \right) \tag{92}$$

$$= \frac{i(i+1)}{2} \left(\frac{\sqrt{2}}{\sqrt{N(N-b(q))}} + \frac{1}{N} \right), \tag{93}$$

where we have set $b(j) = j$. □

To calculate the probability of measuring $R = R_{\text{coll}} \cup R_{\text{preim}}$, Eq. (63) implies

$$\mathbb{P}[\text{Find}] \leq \sum_{i=1}^{q} \left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle \right\|^2 , \tag{94}$$

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

$$\left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle \right\| \leq \left\| |\Phi_{i-1}\rangle - |\Psi_{i-1}^{\text{Good}}\rangle \right\| + \left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Psi_{i-1}^{\text{Good}}\rangle \right\| , \tag{95}$$

the first norm can be bounded by Eq. (93). For the second we use the maximal bound among Eqs. (83) and (85). The maximal bound on the norm of $D \in R$ comes from Eq. (85) and the bound is

$$\left\| \mathsf{J}_R \mathsf{U}_i \mathsf{CPhO}_{\mathcal{Y}} \setminus \mathsf{U}_R \mathsf{U}_{i-1} |\Phi_{i-1}\rangle \right\| \leq \frac{i(i+1)}{2} \left(\frac{\sqrt{2}}{\sqrt{N(N-b(q))}} + \frac{1}{N} \right) + \sqrt{\frac{i}{N}} \tag{96}$$

$$\leq \frac{1}{\sqrt{N}} \left(\frac{\sqrt{2}+1}{2} i^2 + \frac{\sqrt{2}+3}{2} i \right) . \tag{97}$$

Summing the square of the above bound over $1 \leq i \leq q$ gives us the final bound:

$$\mathbb{P}[\text{Find}] \leq \frac{1}{N} \frac{3}{5} q(q+1)(q+2)(3q^2+6q+1) \leq 36 \frac{q^5}{N} \tag{98}$$

This concludes the proof of Lemma 17. □

For R_{coll} we use eq. (91) with $b(i) = i - 1$ instead of $b(i) = i$. The bound on the probability of the event Find is

$$\mathbb{P}[\text{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}} \setminus R_{\text{coll}}]] \leq 3 \frac{q^5}{|\mathcal{Y}|} . \tag{99}$$

For R_{preim} in eq. (91) we set a constant $b(j) = 1$. The bound on the probability of Find is then

$$\mathbb{P}[\text{Find} : \mathsf{A}[\mathsf{CStO}_{\mathcal{Y}} \setminus R_{\text{preim}}]] \leq 16 \frac{q^3}{|\mathcal{Y}|} . \tag{100}$$

6 Quantum Security of the Sponge Construction

We use our methods to show a detailed proof of quantum indifferentiability of the sponge construction used with a random function as the internal function.

At the end of this section we prove that quantum indifferentiability implies collapsingness.

6.1 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 7 we present the definition of the sponge construction, which we denote with Sponge [Ber+07]. The internal state $s = (\bar{s}, \hat{s}) \in \mathcal{A} \times \mathcal{C}$ of Sponge consists of two parts: the *outer* part $\bar{s} \in \mathcal{A}$ and the *inner part* $\hat{s} \in \mathcal{C}$. The number of possible outer parts $|\mathcal{A}|$ is called the *rate* of the sponge, and $|\mathcal{C}|$ is called *capacity*. Naturally the internal function is a map $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$. To denote the internal function with output limited to the part in $\mathcal{A}$ and $\mathcal{C}$ we use the same notation as for states, $\bar{\varphi}$ and $\hat{\varphi}$ respectively. Note that we use a general formulation of the construction, using any finite sets for $\mathcal{A}$ and $\mathcal{C}$. All our results also work for Sponge defined with bit-strings and addition modulo 2, as specified in [NIS14]. By PAD we denote a padding function: an efficiently computable bijection mapping an arbitrary message set to strings p of elements of $\mathcal{A}$. By $|p|$ we denote the number of characters in $\mathcal{A}$ in p. The function constructed in that way behaves as follows, $\textsc{Sponge}_\varphi : \mathcal{A}^* \times \mathbb{N} \to \mathcal{A}^*$, where $\mathcal{A}^* := \bigcup_{n=0}^{\infty} \mathcal{A}^n$. In Fig. 2 we present a scheme of the sponge construction evaluated on input m.

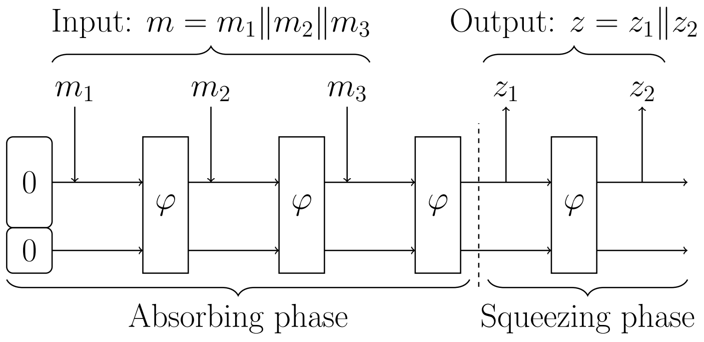


Figure 2: A schematic representation of the sponge construction: $\textsc{Sponge}_\varphi(m_1\|m_2\|m_3) = z_1\|z_2$.

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

Algorithm 7: $\textsc{Sponge}_\varphi[\textsc{pad}, \mathcal{A}, \mathcal{C}]$

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

An important feature of the sponge construction that was introduced in [Ber+07] is the fact that interaction with it can be represented on a graph $G = (\mathcal{V}, \mathcal{E})$. The set of vertices $\mathcal{V}$ corresponds to all possible states of the sponge, namely $\mathcal{V} := \mathcal{A} \times \mathcal{C}$. The outer part is controlled by the user, meaning that she can output that part and modify to any value in a future evaluation by querying an appropriate message. For that reason we group the nodes with the same inner-part value into *supernodes*, so that we have $|\mathcal{C}|$ supernodes and each such supernode consists of $|\mathcal{A}|$ nodes. A directed edge $(s,t) \in \mathcal{E}$ from a node s to a node t exists if $\varphi(s) = t$. From every node there is exactly one outgoing edge, and if φ is a permutation, then there is also exactly one edge arriving at every node. When a query algorithm interacts with the sponge construction, we can think of the sponge graph starting with no edges, and the query algorithms adding edges to $\mathcal{E}$ query by query. Then graph G then reflects the current knowledge of this algorithm about φ.

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

$$\mathsf{SpPath}(s, G) := \bar{v}_1 \| (\bar{v}_2 - \bar{w}_1) \| \cdots \| (\bar{v}_\ell - \bar{w}_{\ell-1}) \| 0. \tag{101}$$

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

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

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

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

The simulators defined in the proofs in this section are implicitly stateful. They maintain a classical or quantum state containing a database of the adversary's queries and the simulator's outputs. 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 $\varphi : \mathcal{A} \times \mathcal{C} \to \mathcal{A} \times \mathcal{C}$. Note that by such collision we also consider inputs that map to $0 \in \mathcal{C}$. We define the bound as a function of the number of queries q to φ:

$$f_{\text{coll}}(q) := \frac{q(q-1)}{2\,|\mathcal{C}|}, \tag{102}$$

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

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

$$\mathsf{H}(x, \ell) := \begin{cases} \lfloor y' \rfloor_\ell & \text{if } (x, \ell' \geq \ell) \in D \\ y' \| \left(y \xleftarrow{\$} \mathcal{A}^{\ell-\ell'} \right) & \text{if } (x, \ell' < \ell) \in D \\ y \xleftarrow{\$} \mathcal{A}^\ell & \text{otherwise} \end{cases}, \tag{103}$$

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

6.2 Classical Indifferentiability of Sponges with Random Functions

In the game-playing proofs and Algorithms 8 and 9 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 19 (SPONGE with functions, classical indifferentiability)**.** SPONGE$_\varphi$[PAD, $\mathcal{A}, \mathcal{C}$] *calling a random function* φ *is* (q, ε)*-indifferentiable from a random oracle, Eq.* (103)*, for* classical *adversaries for any* $q < |\mathcal{C}|$ *and* $\varepsilon = 8\frac{q(q+1)}{2|\mathcal{C}|}$.

Proof. The proof proceeds in six games that we show to be indistinguishable. We start with the real world: the public interface corresponding to the internal function φ is a random transformation and the private interface is SPONGE$_\varphi$. Then in a series of games we gradually change the environment of the adversary to finally reach the ideal world, where the public interface is simulated by the simulator and the private interface is a random oracle H. The simulators used in different games of the proof are defined in Alg. 8, the index of the simulator corresponds to the game in which the simulator is used. Explanations of the simulators follow.

Algorithm 8: Classical S_2, $\boxed{\mathsf{S}_3}$, $\boxed{\mathsf{S}_4}$, $\boxed{\mathsf{I}_6}$, functions

```
State  : current sponge graph $G$
Input  : $s \in \mathcal{A} \times \mathcal{C}$
Output: $\varphi(s)$

1  if $s$ has no outgoing edge then                                   // new query
2      if $\hat{s} \in \mathcal{R} \wedge \mathcal{R} \cup \mathcal{U} \neq \mathcal{C}$ then          // $\hat{s}$-rooted, no saturation
3          $\hat{t} \xleftarrow{\$} \mathcal{C}$, $\boxed{\boxed{\text{if } \hat{t} \in \mathcal{R} \cup \mathcal{U}, \text{ set Bad} = 1}}$, $\boxed{\hat{t} \xleftarrow{\$} \mathcal{C} \setminus (\mathcal{R} \cup \mathcal{U})}$
4          Construct a path to $s$: $p := \mathsf{SpPath}(s, G)$
5          if $\exists x : p = \mathrm{PAD}(x)$ then
6              $\bar{t} \xleftarrow{\$} \mathcal{A}$
7              $\boxed{\boxed{\bar{t} := \mathsf{H}(x)}}$
8          else
9              $\bar{t} \xleftarrow{\$} \mathcal{A}$
10         $t := (\bar{t}, \hat{t})$
11     else
12         $t \xleftarrow{\$} \mathcal{A} \times \mathcal{C}$
13     Add an edge $(s, t)$ to $\mathcal{E}$.
14 Set $t$ to the vertex at the end of the edge starting at $s$
15 Output $t$
```

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

$$\textbf{Game 1} := (b = 1 : b \leftarrow \mathsf{A}[\mathrm{SPONGE}_\varphi, \varphi]) . \tag{104}$$

Game 2 In the second game we introduce the simulator S_2—defined in Alg. 8—that lazy-samples the random function φ. In Alg. 8 we define all simulators of this proof at once, but note that the behavior of S_2 is not influenced by any of the conditional "if" statements (in lines 1, 2, and 5), because in the end, the output state t is picked uniformly from $\mathcal{A} \times \mathcal{C}$ anyway. The definition of the second game is

$$\textbf{Game 2} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_2}, \mathsf{S}_2]) \,. \tag{105}$$

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

$$|\mathbb{P}[\textbf{Game 2}] - \mathbb{P}[\textbf{Game 1}]| = 0. \tag{106}$$

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

$$\textbf{Game 3} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]) \,. \tag{107}$$

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 5. With such a change of the simulators we can use Lemma 5 to bound the difference of probabilities:

$$|\mathbb{P}[\textbf{Game 3}] - \mathbb{P}[\textbf{Game 2}]| \leq \mathbb{P}[\mathrm{Bad} = 1]. \tag{108}$$

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

$$\mathbb{P}[\mathrm{Bad} = 1] \leq f_{\mathrm{coll}}(q), \tag{109}$$

where f_{coll} is defined in Eq. (102). The bound is not necessarily tight as not all queries are made to rooted nodes.

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

$$\textbf{Game 4} := \left(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, \mathsf{S}_4^{\mathsf{H}}]\right) . \tag{110}$$

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

Claim 20. *Given that* $\mathrm{Bad} = 0$ *the mentioned games are the same:*

$$|\mathbb{P}[\textbf{Game 4} \mid \mathrm{Bad} = 0] - \mathbb{P}[\textbf{Game 3} \mid \mathrm{Bad} = 0]| = 0. \tag{111}$$

Proof. Note that the inner part is distributed in the same way in both games if $\mathrm{Bad} = 0$, 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 $\mathrm{Bad} = 0$ 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 $\mathbb{P}[\mathrm{Bad} = 1 : \textbf{Game 3}] = \mathbb{P}[\mathrm{Bad} = 1 : \textbf{Game 4}]$. Together with the above claim we can derive the advantage:

$$\begin{aligned}
|\mathbb{P}[\textbf{Game 4}] - \mathbb{P}[\textbf{Game 3}]| \overset{\text{Claim 20}}{=} \Big| &\mathbb{P}[\textbf{Game 4} \mid \mathrm{Bad} = 0] \\
&\cdot \underbrace{(\mathbb{P}[\mathrm{Bad} = 1 : \textbf{Game 3}] - \mathbb{P}[\mathrm{Bad} = 1 : \textbf{Game 4}])}_{=0} \\
&+ \underbrace{\mathbb{P}[\textbf{Game 3} \mid \mathrm{Bad} = 1]}_{\leq 1}\mathbb{P}[\mathrm{Bad} = 1] + \underbrace{\mathbb{P}[\textbf{Game 4} \mid \mathrm{Bad} = 1]}_{\leq 1}\mathbb{P}[\mathrm{Bad} = 1]\Big| \qquad (112) \\
&\leq 2\mathbb{P}[\mathrm{Bad} = 1]. \qquad (113)
\end{aligned}$$

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

$$\textbf{Game 5} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{S}_4^{\mathsf{H}}]\right). \tag{114}$$

Conditioned on Bad $= 0$, the outputs of the simulator in Games 4 and 5 act in the same way and are consistent with H. To calculate the adversary's advantage in distinguishing between the two games we can follow the proof of Lemma 16, with $\mathsf{H} \setminus R_1$ replaced by **Game 5**, $\mathsf{G} \setminus R_2$ replaced by **Game 4**, and event Find replaced by Bad $= 1$. As the derivation of Lemma 16 uses no quantum mechanical arguments and the assumption holds—the games are identical conditioned on Bad $= 0$—the bound holds:

$$|\mathbb{P}[\textbf{Game 5}] - \mathbb{P}[\textbf{Game 4}]| \leq 4\mathbb{P}[\text{Bad} = 1] \leq 4f_{\text{coll}}(q). \tag{115}$$

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 $\mathcal{C}$. The game is

$$\textbf{Game 6} := \left(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{I}_6^{\mathsf{H}}]\right) \tag{116}$$

and the advantage is

$$|\mathbb{P}[\textbf{Game 6}] - \mathbb{P}[\textbf{Game 5}]| \leq \mathbb{P}[\text{Bad} = 1] \leq f_{\text{coll}}(q). \tag{117}$$

following Lemma 5. 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 $q < |\mathcal{C}|$. Collecting and adding all the differences yields the claimed $\varepsilon = 8f_{\text{coll}}(q)$. □

6.3 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 an 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 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 have use of the result from Lemma (17). Let us denote the bound on inner collisions by

$$f_{\text{coll}}^{Q}(q) := 36\frac{q^5}{|\mathcal{C}|}. \tag{118}$$

The main statement of this section is:

Theorem 21 (Sponge with functions, quantum indifferentiability)**.** *Sponge$_\varphi$[pad, $\mathcal{A}$, $\mathcal{C}$] calling a random function φ is (q, ε)-indifferentiable from a random oracle, Eq.* (103)*, for* quantum *adversaries for any $q < |\mathcal{C}|$ and $\varepsilon = 288\frac{q^5}{|\mathcal{C}|} + 6\sqrt{\frac{(q+1)q^5}{|\mathcal{C}|}}$.*

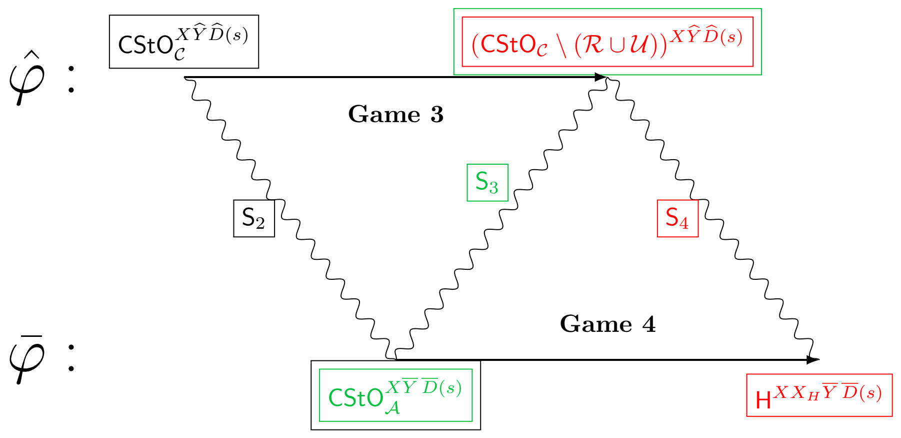
Proof. Even though we allow for quantum accessible oracles, the proof we present is very similar to the classical case. The proof follows the same structure, the biggest difference is in the simulators that use the compressed oracle to lazy-sample appropriate answers.

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. 9 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. 3 and additionally analyzed in Lemma 17. Eq. (99) provides a bound of the probability of Find (as defined in Section 5) 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 5.

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



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

Figure 3: Schematics of the simulators defined in Alg. 9, horizontal arrows signify the change introduced in the labeled game.

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

$$\textbf{Game 1} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_\varphi, \varphi]) \,. \tag{119}$$

Algorithm 9: Quantum $\boxed{\mathsf{S}_2}$, $\boxed{\mathsf{S}_3}$, $\boxed{\mathsf{S}_4}$, functions

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

Game 2 In the second game we introduce the simulator S_2, defined in Alg. 9. This algorithm is essentially a compressed random oracle, the only difference are the if statements, note that the behavior of S_2 is not influenced by any of the conditional "if" statements (in lines 3, and 6), because in the end, the output state t is picked uniformly from $\mathcal{A}\times\mathcal{C}$ anyway. The game is defined as:

$$\textbf{Game 2} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_2}, \mathsf{S}_2]) \,. \tag{120}$$

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

$$|\mathbb{P}[\textbf{Game 2}] - \mathbb{P}[\textbf{Game 1}]| = 0. \tag{121}$$

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

$$\textbf{Game 3} := (b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]) \,. \tag{122}$$

With such a change of the simulators we can use Thm. 13 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

$$|\mathbb{P}[\textbf{Game 3}] - \mathbb{P}[\textbf{Game 2}]| \leq \sqrt{(q+1)\mathbb{P}[\mathrm{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]]}. \tag{123}$$

Using Lemma 17 we have that

$$\mathbb{P}[\mathrm{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]] \leq f_{\text{coll}}^{Q}(q). \tag{124}$$

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

$$\textbf{Game 4} := \Big(b = 1 : b \leftarrow \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, \mathsf{S}_4^{\mathsf{H}}]\Big)\,. \tag{125}$$

Thanks to the classical argument we have that S_4 and S_3 are identical until bad, as in Def. 15. Then we can use Lemma 16 to bound the advantage of the adversary

$$|\mathbb{P}[\textbf{Game 4}] - \mathbb{P}[\textbf{Game 3}]| \leq 4\mathbb{P}[\text{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_3}, \mathsf{S}_3]] \leq 4f_{\text{coll}}^{Q}(q). \tag{126}$$

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:

$$\textbf{Game 5} := \Big(b = 1 : b \leftarrow \mathsf{A}[\mathsf{H}, \mathsf{S}_4^{\mathsf{H}}]\Big) \tag{127}$$

and the advantage is

$$|\mathbb{P}[\textbf{Game 5}] - \mathbb{P}[\textbf{Game 4}]| \leq 4\mathbb{P}[\text{Find} : \mathsf{A}[\textsc{Sponge}_{\mathsf{S}_4}, \mathsf{S}_4^{\mathsf{H}}]] \leq 4f_{\text{coll}}^{Q}(q). \tag{128}$$

Conditioned on $\neg$Find, the outputs of the private interface are the same, then the games are identical-until-bad and we can use Lemma 16 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 $q < |\mathcal{C}|$ as the database in every branch of the superposition increases by at most one in every query. Collecting the differences between games yields the claimed $\varepsilon = 8f_{\text{coll}}^{Q}(q) + \sqrt{(q+1)f_{\text{coll}}^{Q}(q)}$. $\square$

6.4 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:

$$\textbf{Collapse 1}: \quad (S, M, h) \leftarrow \mathsf{A}^{\mathsf{H}}(),\ m \leftarrow \mathsf{M}(M),\ b \leftarrow \mathsf{B}^{\mathsf{H}}(S, M), \tag{129}$$

$$\textbf{Collapse 2}: \quad (S, M, h) \leftarrow \mathsf{A}^{\mathsf{H}}(), \qquad b \leftarrow \mathsf{B}^{\mathsf{H}}(S, M). \tag{130}$$

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

Definition 22 (Collapsing [Unr16b])**.** *A function* H *is* collapsing *if for any valid quantum-polynomial-time adversary* (A, B)

$$|\mathbb{P}[b = 1 : \textbf{Collapse 1}] - \mathbb{P}[b = 1 : \textbf{Collapse 2}]| < \varepsilon, \tag{131}$$

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 23 (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. 21, we re-prove the result of [Cza+18] in a modular way that might come useful when indifferentiability of sponges with permutations is established.

Theorem 24 (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

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

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

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

and hence

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

□

7 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 top be devised to tackle this problem as our correctness theorem only applies to local distributions.

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, 6).

[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, 6, 7, 23, 24, 25, 26).

[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, 12).

[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, 12).

[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, 8, 9, 35, 36, 39).

[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, 10, 38).

[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, 14).

[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 pp. 4, 8).

[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).

[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, 12).

[Cza+18] Jan Czajkowski, Leon Groot Bruinderink, Andreas Hülsing, Christian Schaffner, and Dominique Unruh. "Post-quantum Security of the Sponge Construction". In: *Post-Quantum Cryptography*. Springer International Publishing, 2018, pp. 185–204. DOI: 10.1007/978-3-319-79063-3_9 (cit. on pp. 4, 44).

[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. 4).

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

[Feh18] Serge Fehr. "Classical Proofs for the Quantum Collapsing Property of Classical Hash Functions". In: *Theory of Cryptography*. Springer International Publishing, 2018, pp. 315–338. DOI: 10.1007/978-3-030-03810-6_12 (cit. on pp. 4, 44).

[HM20] Yassine Hamoudi and Frédéric Magniez. "Quantum Time-Space Tradeoffs by Recording Queries". In: *arXiv preprint arXiv:2002.08944* (2020) (cit. on p. 6).

[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. 4, 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).

[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. 18).

[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, 8, 12, 13).

[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. 14, 26).

[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. 8, 12, 14, 36).

[NIS15] NIST. *Secure Hash Standard (SHS)*. Draft FIPS 180-4. 2015. DOI: 10.6028/NIST.FIPS.180-4 (cit. on p. 12).

[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. 13).

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

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

[Unr14] Dominique Unruh. "Revocable Quantum Timed-Release Encryption". In: *Advances in Cryptology – EUROCRYPT 2014*. Springer Berlin Heidelberg, 2014, pp. 129–146. DOI: 10.1007/978-3-642-55220-5_8 (cit. on pp. 3, 4, 6, 11, 23).

[Unr16a] Dominique Unruh. "Collapse-Binding Quantum Commitments Without Random Oracles". In: *Advances in Cryptology – ASIACRYPT 2016*. Springer Berlin Heidelberg, 2016, pp. 166–195. DOI: 10.1007/978-3-662-53890-6_6 (cit. on pp. 4, 44).

[Unr16b] Dominique Unruh. "Computationally Binding Quantum Commitments". In: *Advances in Cryptology – EUROCRYPT 2016*. Springer Berlin Heidelberg, 2016, pp. 497–527. DOI: 10.1007/978-3-662-49896-5_18 (cit. on pp. 4, 43, 44).

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

[Unr19b] Dominique Unruh. *Recording quantum queries – explained*. In preparation. 2019 (cit. on pp. 15, 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 p. 27).

[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, 11, 15, 16, 23, 49).

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
$\mathfrak{A}$	Distribution of outer part of outputs of a random permutation	
A, B	An adversary, a classical or quantum algorithm	12, 25
$\mathcal{A}$	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.	12, 38
$\mathfrak{C}$	Distribution of inner part of outputs of a random permutation	
$\mathsf{CFO}_{\mathfrak{D}}$	Compressed Fourier Oracle for distribution $\mathfrak{D}$	19
$\mathcal{C}$	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	43
$\mathsf{Com}_{\mathfrak{P}}$	Compression procedure	
CH	Compressed Oracle with oracle H as Samp	
CPerO	Compressed Permutation Oracle	
$\mathsf{CPhO}_{\mathfrak{U}}$	Compressed Phase Oracle	52
$\mathsf{CStO}_{\mathfrak{D}}, \mathsf{CStO}_{\mathcal{Y}\setminus\mathcal{S}}$	Compressed Standard Oracle, for distribution $\mathfrak{D}$ and for a conditionally uniform distribution over $\mathcal{Y} \setminus \mathcal{S}$	
$D, \Delta, Д$	Prepared database in the standard basis (and the database register), prepared database in the Fourier basis, and the unprepared databse	19
$\mathsf{Dec}_{\mathfrak{D}}$	Decompression procedure	20
D	The distinguisher	13
$\mathfrak{D}$	A distribution.	16
$\mathcal{D}$	The set of outputs of queries	
$\mathcal{E}$	The set of edges of a sponge graph	36
Find	Event of measurement of the relation R returning 1	24, 26
Flip	Algorithm flipping the database in the standard basis, inputs are outputs and vice versa.	
FO	Fourier Oracle, $\mathsf{QFT}_N^{YF} \circ \mathsf{StO} \circ \mathsf{QFT}_N^{\dagger YF}$	15
H, G	Compressed Oracle	23
HT_n	The Hadamard transform	49
$\lvert\psi\rangle$	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
$O(n)$	Complexity class "big O"	
PAD	Padding function	36
$\mathsf{SpPath}(s, G)$	Function constructing an input to SPONGE leading to a given node	37
$\mathfrak{P}$	Uniform distribution over the set of permutations	
Per	Permutation Oracle	
P	The permutation unitary	54
φ	The map between states in SPONGE.	35

$\bar{\varphi}$	The general map between states with its output limited to the set $\mathcal{A}$	35
$\hat{\varphi}$	The general map between states with its output limited to the set $\mathcal{C}$	35
PhO	Phase Oracle, $\mathsf{QFT}_N^Y \circ \mathsf{StO} \circ \mathsf{QFT}_N^{\dagger Y}$	50
J_R	Projector on relation R.	24
QFT_N	The Quantum Fourier Transform	14
Rem	Removing и $= 0$ from the database	56
$\mathcal{R}$	The set of rooted supernodes	37
$\mathsf{Samp}_{\mathfrak{D}}(\mathcal{S})$	Algorithm preparing a superposition of samples of outputs of $f \leftarrow \mathfrak{D}$ on inputs from $\mathcal{S}$.	16
S	Classical and quantum simulators.	11, 38, 42
Size	Calculatte tthe numbeer of non-zero entries in D	
$\textsc{Sponge}_{\varphi}[\textsc{pad}, \mathcal{A}, \mathcal{C}]$	Sponge construction with the internal function φ, capacity set $\mathcal{C}$, and alphabet $\mathcal{A}$	36
Spread	The full set "spreading" operation.	
StO	Standard Oracle	15
$\mathfrak{U}$	The uniform distribution.	49
Upd	Updating η in the database	55
$\mathcal{U}$	The set of supernodes with outgoing edges	37
V	The individual "spreading" operation.	
$\mathcal{V}$	The set of vertices of a sponge graph	36
$\oplus$	Bitwise XOR	50
$y, \eta,$ и	Values in the Y register of a database in different bases	18

A Additional Details on Quantum-Accessible Oracles

A.1 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 [Unr19b].

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

$$\mathsf{HT}_n|x\rangle := \frac{1}{\sqrt{2^n}} \sum_{\xi \in \{0,1\}^n} (-1)^{\xi \cdot x} |\xi\rangle, \tag{132}$$

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

$$\mathsf{StO}_{\mathfrak{U}}|x,y\rangle_{XY}\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|f\rangle_F = \frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|x,y\oplus f(x)\rangle_{XY}\otimes|f\rangle_F, \tag{133}$$

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

$$\mathsf{FO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|\phi\rangle_F := |x,\eta\rangle_{XY}|\phi\oplus\chi_{x,\eta}\rangle_F, \tag{134}$$

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

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

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

$$\mathsf{PhO}_{\mathfrak{U}} := (\mathbb{1}^X_m\otimes\mathsf{HT}^Y_n)\otimes\mathbb{1}^F_{n2^m}\circ\mathsf{StO}_{\mathfrak{U}}\circ(\mathbb{1}^X_m\otimes\mathsf{HT}^Y_n)\otimes\mathbb{1}^F_{n2^m}, \tag{135}$$

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

$$\mathsf{FO}_{\mathfrak{U}} = (\mathbb{1}^{XY})\otimes\mathsf{HT}^F_{n2^m}\circ\mathsf{PhO}_{\mathfrak{U}}\circ(\mathbb{1}^{XY})\otimes\mathsf{HT}^F_{n2^m}. \tag{136}$$

The above relations show that we have a chain of oracles, similar to Eq. (19):

$$\mathsf{StO}_{\mathfrak{U}}\xleftrightarrow{\mathsf{HT}^Y_n}\mathsf{PhO}_{\mathfrak{U}}\xleftrightarrow{\mathsf{HT}^F_{n2^m}}\mathsf{FO}_{\mathfrak{U}}. \tag{137}$$

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

A.1.1 Full Oracles, Additional Details

In this section we show detailed calculations of identities claimed in Section A.1. First we analyze the Phase Oracle, introduced in Eq. (135). We can check by direct calculation that this yields the standard Phase Oracle,

$$\mathsf{PhO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|f\rangle_F = (-1)^{\eta\cdot f(x)}|x,\eta\rangle_{XY}|f\rangle_F. \tag{138}$$

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

$$\begin{aligned}
&\mathsf{PhO}_{\mathfrak{U}}|x,\eta\rangle_{XY}\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|f\rangle_F \\
&=(\mathbb{1}^X_m\otimes\mathsf{HT}^Y_n)\otimes\mathbb{1}^F_{n2^m}\mathsf{StO}_{\mathfrak{U}}|x\rangle_X\frac{1}{\sqrt{2^n}}\sum_y(-1)^{\eta\cdot y}|y\rangle_Y\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|f\rangle_F && (139)\\
&=(\mathbb{1}^X_m\otimes\mathsf{HT}^Y_n)\otimes\mathbb{1}^F_{n2^m}|x\rangle_X\frac{1}{\sqrt{2^n}}\sum_y\sum_{f\in\mathcal{F}}(-1)^{\eta\cdot y}|y\oplus f(x)\rangle_Y\frac{1}{\sqrt{|\mathcal{F}|}}|f\rangle_F && (140)\\
&=\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}|x\rangle_X\sum_\zeta\underbrace{\frac{1}{2^n}\sum_y(-1)^{\eta\cdot y}(-1)^{(y\oplus f(x))\cdot\zeta}}_{=\delta(\eta,\zeta)(-1)^{\zeta\cdot f(x)}}|\zeta\rangle_Y|f\rangle_F && (141)\\
&=\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}(-1)^{\eta\cdot f(x)}|x\rangle_X|\eta\rangle_Y|f\rangle_F. && (142)
\end{aligned}$$

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

$$\begin{aligned}
&\mathsf{HT}^F_{n2^m}\circ\mathsf{PhO}_{\mathfrak{U}}\circ\mathsf{HT}^F_{n2^m}|x,\eta\rangle_{XY}|0^{2^m n}\rangle_F=\mathsf{HT}^F_{n2^m}\frac{1}{\sqrt{|\mathcal{F}|}}\sum_{f\in\mathcal{F}}(-1)^{\eta\cdot f(x)}|x,\eta\rangle|f\rangle_F\\
&=\frac{1}{|\mathcal{F}|}\sum_{\phi,f}(-1)^{\phi\cdot f}(-1)^{\eta\cdot f(x)}|x,\eta\rangle|\phi\rangle_F\\
&=\sum_\phi\underbrace{\frac{1}{2^{n(2^m-1)}}\sum_{f(x'\neq x)}(-1)^{\phi_{x'}\cdot f(x')}}_{=\delta(\phi_{x'},0^n)}\underbrace{\frac{1}{2^n}\sum_{f(x)}(-1)^{\phi_x\cdot f(x)}(-1)^{\eta\cdot f(x)}}_{=\delta(\phi_x,\eta)}|x,\eta\rangle|\phi\rangle_F\\
&=|x,\eta\rangle|0^{2^m n}\oplus\chi_{x,\eta}\rangle && (143)
\end{aligned}$$

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

A.1.2 Compressed Oracles, Additional Details

Let us state the input-output behavior of the compressed oracle $\mathsf{CFO}_{\mathfrak{U}}$ for uniform distributions. The input-output behavior of $\mathsf{CFO}_{\mathfrak{U}}$ is given by the following equation, x_r is the smallest $x_i\in D^X$ such that $x_r\geq x$:

$$\begin{aligned}
&\mathsf{CFO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|x_1,\eta_1\rangle_{D_1}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|\bot,0^n\rangle_{D_q}=|x,\eta\rangle_{XY}|\psi_{r-1}\rangle\\
&\otimes\begin{cases}
|x_r,\eta_r\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|\bot,0^n\rangle_{D_q} & \text{if } \eta=0^n,\\
|x,\eta\rangle_{D_r}|x_r,\eta_r\rangle_{D_{r+1}}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_q} & \text{if } \eta\neq 0^n, x\neq x_r,\\
|x_r,\eta_r\oplus\eta\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-1}}|\bot,0^n\rangle_{D_q} & \text{if } \eta\neq 0^n, x=x_r,\\
 & \quad \eta\neq\eta_r,\\
|x_{r+1},\eta_{r+1}\rangle_{D_r}\cdots|x_{q-1},\eta_{q-1}\rangle_{D_{q-2}}|\bot,0^n\rangle_{D_{q-1}}|\bot,0^n\rangle_{D_q} & \text{if } \eta\neq 0^n, x=x_r,\\
 & \quad \eta=\eta_r,
\end{cases} && (144)
\end{aligned}$$

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

$$\mathsf{CPhO}_{\mathfrak{U}}:=\mathbb{1}_{n+m}\otimes\mathsf{HT}^{D^Y}_n\circ\mathsf{CFO}_{\mathfrak{U}}\circ\mathbb{1}_{n+m}\otimes\mathsf{HT}^{D^Y}_n, \qquad (145)$$

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

$$\mathsf{CPhO}_{\mathfrak{U}}|x,\eta\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}|\bot,z\rangle_D = \mathbb{1}_{n+m}\circ\mathsf{HT}_n^{D^Y}\circ\mathsf{CFO}_{\mathfrak{U}}|x,\eta\rangle_{XY}|\bot,0^n\rangle_D \tag{146}$$

$$= \mathbb{1}_{n+m}\circ\mathsf{HT}_n^{D^Y}\left((1-\delta(\eta,0^n))|x,\eta\rangle_{XY}|x,\eta\rangle_D + \delta(\eta,0^n)|x,\eta\rangle_{XY}|\bot,0^n\rangle_D\right) \tag{147}$$

$$= \frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}\left((1-\delta(\eta,0^n))(-1)^{\eta\cdot z}|x,\eta\rangle_{XY}|x,z\rangle_D + \delta(\eta,0^n)|x,0^n\rangle_{XY}|\bot,z\rangle_D\right). \tag{148}$$

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

$$\frac{1}{\sqrt{2^m}}\sum_x|x,0^n\rangle_{XY} \overset{\mathsf{CPhO}_{\mathfrak{U}}}{\mapsto} \frac{1}{\sqrt{2^m}}\sum_x|x,0^n\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_z|\bot,z\rangle_D, \tag{149}$$

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

$$\frac{1}{\sqrt{2^m}}\sum_x|x,0^n\rangle_{XY} \overset{\mathsf{CPhO}'_{\mathfrak{U}}}{\mapsto} \frac{1}{\sqrt{2^m}}\sum_x|x,0^n\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_z|x,z\rangle_D. \tag{150}$$

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

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

$$\begin{aligned}
&= |x_2,\eta_2\rangle|x_1,\eta_1\rangle\frac{1}{2^n}\sum_{z_1,z_2}\Bigg((-1)^{\eta_1\cdot z_1}\delta(\eta_2,0^n)(1-\delta(\eta_1,0^n))\underbrace{|x_1,z_1\rangle_{F_1}|\bot,z_2\rangle_{F_2}}_{=|\psi^{\text{NOT}}\rangle} \\
&+ \delta(\eta_2,0^n)\delta(\eta_1,0^n)\underbrace{|\bot,z_1\rangle_{F_1}|\bot,z_2\rangle_{F_2}}_{=|\psi^{\text{NOT}}\rangle} \\
&+ (-1)^{\eta_2\cdot z_1}(1-\delta(\eta_2,0^n))\delta(\eta_1,0^n)\underbrace{|x_2,z_1\rangle_{F_1}|\bot,z_2\rangle_{F_2}}_{=|\psi^{\text{ADD}}\rangle} \\
&+(-1)^{\eta_1\cdot z_1}(-1)^{\eta_2\cdot z_2}(1-\delta(\eta_2,0^n))(1-\delta(x_1,x_2))(1-\delta(\eta_1,0^n))\underbrace{|x_1,z_1\rangle_{F_1}|x_2,z_2\rangle_{F_2}}_{=|\psi^{\text{ADD}}\rangle} \\
&+(1-\delta(\eta_2,0^n))\delta(x_1,x_2)\delta(\eta_1,\eta_2)(1-\delta(\eta_1,0^n))\underbrace{|\bot,z_1\rangle_{F_1}|\bot,z_2\rangle_{F_2}}_{=|\psi^{\text{REM}}\rangle} \\
&+ (1-\delta(\eta_2,0^n))\delta(x_1,x_2)(1-\delta(\eta_1,\eta_2))(1-\delta(\eta_1,0^n)) \\
&\cdot(-1)^{(\eta_1\oplus\eta_2)\cdot z_1}\underbrace{|x_1,z_1\rangle_{F_1}|\bot,z_2\rangle_{F_2}}_{=|\psi^{\text{UPD}}\rangle}\Bigg),
\end{aligned} \tag{151}$$

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 $\mathsf{CPhO}_{\mathfrak{U}}$

$$\mathsf{CStO}_{\mathfrak{U}} = \mathbb{1}_m \otimes \mathsf{HT}_n^Y \circ \mathsf{CPhO}_{\mathfrak{U}} \circ \mathbb{1}_m \otimes \mathsf{HT}_n^Y. \tag{152}$$

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

$$\mathsf{CStO}_{\mathfrak{U}}|x,y\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}|\bot,z\rangle_D$$

$$= \mathbb{1}_m \otimes \mathsf{HT}_n^Y \circ \mathsf{CPhO}_{\mathfrak{U}}\frac{1}{\sqrt{2^n}}\sum_{\eta\in\{0,1\}^n}(-1)^{\eta\cdot y}|x,\eta\rangle_{XY}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}|\bot,z\rangle_D \tag{153}$$

$$= \mathbb{1}_m \otimes \mathsf{HT}_n^Y\frac{1}{\sqrt{2^n}}\sum_{\eta\in\{0,1\}^n}\frac{1}{\sqrt{2^n}}\sum_{z\in\{0,1\}^n}(-1)^{\eta\cdot y}\Bigg((1-\delta(\eta,0^n))(-1)^{\eta\cdot z}|x,\eta\rangle_{XY}|x,z\rangle_D$$
$$+\delta(\eta,0^n)|x,0^n\rangle_{XY}|\bot,z\rangle_D\Bigg) \tag{154}$$

$$= \frac{1}{2^n}\sum_{y',\eta}\frac{1}{\sqrt{2^n}}\sum_z(-1)^{\eta\cdot y}(-1)^{y'\cdot\eta}\Bigg((1-\delta(\eta,0^n))(-1)^{\eta\cdot z}|x,y'\rangle_{XY}|x,z\rangle_D$$
$$+\delta(\eta,0^n)|x,y'\rangle_{XY}|\bot,z\rangle_D\Bigg) \tag{155}$$

$$= \sum_{y'}\frac{1}{\sqrt{2^n}}\sum_z\underbrace{\frac{1}{2^n}\sum_{\eta\neq 0}(-1)^{\eta\cdot y}(-1)^{y'\cdot\eta}(-1)^{\eta\cdot z}}_{=\delta(y',y\oplus z)-\frac{1}{2^n}}|x,y'\rangle_{XY}|x,z\rangle_D$$
$$+\sum_{y'}\frac{1}{\sqrt{2^n}}\sum_z\frac{1}{2^n}|x,y'\rangle_{XY}|\bot,z\rangle_D \tag{156}$$

$$= \frac{1}{\sqrt{2^n}}\sum_z\left(|x,y\oplus z\rangle_{XY}|x,z\rangle_D - \frac{1}{2^n}\sum_{y'}|x,y'\rangle_{XY}|x,z\rangle_D + \frac{1}{2^n}\sum_{y'}|x,y'\rangle_{XY}|\bot,z\rangle_D\right). \tag{157}$$

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

A.2 Detailed Algorithm for Alg. 3: $\mathsf{CFO}_{\mathfrak{D}}$

In Algorithm 10 we present the fully-detailed version of Algorithm 3. This algorithm runs the following subroutines:

- Locate, Function 11: This subroutine locates the positions in Д where the $x-$entry coincides with the $x-$entry of the query. The result is represented as q bits, where $q_i = 1 \iff Д_i^X = x$. This result is then bitwise XOR'ed into an auxiliary register L.
- Add, Function 12: 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 13: This subroutine updates the database by subtracting η after a suitable basis transformation.
- Rem, Function 14: This subroutine removes $(x, 0)$ entries from the database and puts them to the back in the form of padding.

- Clean, Function 15: This subroutine cleans the auxiliary registers setting them back to initial values.
- Larger: This subroutine determines whether one value is larger than a second value, it works on three registers, say $D^X XA$ and flips the bit in A if the value of D^X is larger than the value in X, so

$$\mathsf{Larger}^{D^X XA}|u\rangle_{D^X}|v\rangle_X|a\rangle_A = |u\rangle_{D^X}|v\rangle_X \begin{cases} |a \oplus 1\rangle_A \text{ if } u > v \\ |a\rangle_A \text{ otherwise} \end{cases} . \tag{158}$$

  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:

$$\mathsf{P}|x_1, ..., x_q\rangle \otimes |y_1, ..., y_n\rangle := |\sigma_n \circ ... \circ \sigma_1(x_1, ..., x_q)\rangle \otimes |y_1, ..., y_n\rangle , \tag{159}$$

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 10: Detailed $\mathsf{CFO}_{\mathfrak{D}}$

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

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

Function 11: Locate

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

Function 12: Add

```
Input  : $|x\rangle_X|Д\rangle_D|l\rangle_L$
Output: $|x\rangle_X|Д'\rangle_D|l'\rangle_L$
1 Set $|a\rangle_A = |0^q \in \{0,1\}^q\rangle_A$               // initialize auxiliary register A
2 for $i = 1, ..., q$ do
3     $|a_i\rangle_{A_i} \mapsto \mathsf{Larger}(|Д_i^X\rangle_{D_i^X}|x\rangle_X|a_i\rangle_{A_i})$      // check if database entry > query
4     if $Д_i^X \neq \perp$ then                             // correct for empty entries
5         $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$
6     for $j = i+1, ..., q$ do                           // flip all higher entries
7         $|a_j\rangle_{A_j} \mapsto |a_j \oplus a_i\rangle_{A_j}$              // so we're left with one position
8 $|Д\rangle_D \mapsto \mathsf{P}^{-1}(|Д\rangle_D \otimes |a\rangle_A)$              // permute D to create empty entry
                                                            // P is defined in (159)
9 for $i = 1, ..., q$ do
10    if $a_i = 1$ then                                   // look for this empty entry
11        $|Д_i^X\rangle_{D_i^X} \mapsto |Д_i^X - x\rangle_{D_i^X}$             // add $x$−entry to the database
12        $|l_i\rangle_{L_i} \mapsto |l_i \oplus 1\rangle_{L_i}$                  // update location register
13 if $x \neq 0$ then                                    // Non zero $x$ implies non zero $a$
14    for $i = 1, ..., q$ do
15        if $l_i = 1$ then                                            // if located
16            $|a_i\rangle_{A_i} \mapsto |a_i \oplus 1\rangle_{A_i}$                 // uncompute register $A$
17 return $|x\rangle_X|Д'\rangle_D|l'\rangle_L$                   // $Д'$ is the modified database
                                                        // $l'$ is modified $l$
```

Function 13: Upd

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

Function 14: Rem

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

Function 15: Clean

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