On the Security of Proofs of Sequential Work in a Post-Quantum World

Jeremiah Blocki
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Seunghoon Lee
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Samson Zhou
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

A Proof of Sequential Work (PoSW) allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to perform some underlying computation. PoSWs have many applications including time-stamping, blockchain design, and universally verifiable CPU benchmarks. Mahmoody, Moran, and Vadhan (ITCS 2013) gave the first construction of a PoSW in the random oracle model though the construction relied on expensive depth-robust graphs. In a recent breakthrough, Cohen and Pietrzak (EUROCRYPT 2018) gave an efficient PoSW construction that does not require expensive depth-robust graphs.

In the classical parallel random oracle model, it is straightforward to argue that any successful PoSW attacker must produce a long $\mathcal{H}$-sequence and that any malicious party running in sequential time $T-1$ will fail to produce an $\mathcal{H}$-sequence of length T except with negligible probability. In this paper, we prove that any quantum attacker running in sequential time $T-1$ will fail to produce an $\mathcal{H}$-sequence except with negligible probability – even if the attacker submits a large batch of quantum queries in each round. The proof is substantially more challenging and highlights the power of Zhandry's recent compressed oracle technique (CRYPTO 2019). We further extend this result to establish post-quantum security of a non-interactive PoSW obtained by applying the Fiat-Shamir transform to Cohen and Pietrzak's efficient construction (EUROCRYPT 2018).

2012 ACM Subject Classification Security and privacy → Hash functions and message authentication codes; Security and privacy → Information-theoretic techniques

Keywords and phrases Proof of Sequential Work, Parallel Quantum Random Oracle Model, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ITC.2021.22

Related Version *Full Version*: `https://arxiv.org/abs/2006.10972`

Funding *Jeremiah Blocki*: Research supported in part by NSF CNS-1755708 and NSF CNS-1931443. *Seunghoon Lee*: Research supported in part by NSF Award CNS-1755708 and by the Center for Science of Information at Purdue University (NSF CCF-0939370).

Acknowledgements The authors wish to thank Fang Song (shepherd) and other anonymous reviewers for comments which improved the presentation of this paper.

1 Introduction

As we make progress towards the development of quantum computers, it is imperative to understand which cryptographic primitives can be securely and efficiently instantiated in a post-quantum world. In this work, we consider the security of proofs of sequential work against quantum adversaries.

A proof of sequential work (PoSW) [37, 23, 3, 26] is a protocol for proving that one spent significant sequential computation work to validate some statement χ. One motivation for a proof of sequential work is in time-stamping, e.g., if Bob can produce a valid proof π_χ that

N sequential steps were spent to validate χ, then Bob can prove that he must have known about χ at least time $\Omega(N)$ seconds in the past. A verifier should be able to validate the proof π_χ quickly, i.e., in time polylog(N).

Mahmoody et al. [37] gave the first construction of a proof of sequential work in the random oracle model. Their construction was based on labeling a depth-robust graph, i.e., given a random oracle $\mathcal{H}$ and a directed acyclic graph $G = (V = [N], E)$ with N nodes and an initial input x, we can compute labels $\ell_1, \ldots, \ell_N$, where the label of the source node is $\ell_1 = \mathcal{H}(\chi, 1, x)$ and an internal node v with parents $v_1, \ldots, v_\delta$ has label $\ell_v = \mathcal{H}(\chi, v, \ell_{v_1}, \ldots, \ell_{v_\delta})$.

The prover commits to labels $\ell'_1, \ldots, \ell'_N$ (a cheating prover might commit to the wrong labels) and then the verifier selects a random subset $S \subset [N]$ of $|S| = c$ challenge nodes. For each challenge node $v \in S$ with parents $v_1, \ldots, v_\delta$, the prover reveals ℓ'_v along with $\ell'_{v_1}, \ldots, \ell'_{v_\delta}$ and the verifier checks that v is locally consistent, i.e., $\ell'_v = \mathcal{H}(\chi, v, \ell'_{v_1}, \ldots, \ell'_{v_\delta})$. If we let R denote the subset of locally inconsistent nodes, then the verifier will accept with probability at most $(1 - |R|/N)^c$.

Mahmoody et al. [37] selected G such that G was ϵ-extremely depth-robust[1], meaning that for any set $R \subseteq [N]$ of locally inconsistent nodes, there is a directed path of length $T + 1 = (1 - \epsilon)N - R$. This path $P = v_0, \ldots, v_T$ corresponds to an $\mathcal{H}$-sequence of length T where an $\mathcal{H}$-sequence is any sequence of strings $x_0, \ldots, x_T$ with the property that $\mathcal{H}(x_i)$ is a substring of x_{i+1} for each $i < T$. Note that the labels $\ell'_{v_0}, \ldots, \ell'_{v_T}$ have this property. In the classical parallel random oracle model (pROM), it is relatively straightforward to prove that any algorithm running in $T - 1$ rounds and making at most q queries in total fails to produce an $\mathcal{H}$-sequence except with probability $\tilde{\Omega}\left(q^2 2^{-\lambda}\right)$ when $\mathcal{H} : \{0,1\}^{\delta\lambda} \to \{0,1\}^\lambda$ outputs binary strings of length λ [23].

The ϵ-extreme depth-robust graphs used in the construction of Mahmoody et al. [37] were quite expensive, having indegree $\delta = \tilde{\Omega}(\log N)$. Alwen et al. [7] showed how to construct ϵ-extreme depth-robust graphs with indegree just $\mathcal{O}(\log N)$ though the hidden constants were quite large. Cohen and Pietrzak [23] gave an efficient (practical) construction that avoids depth-robust graphs entirely by cleverly modifying the Merkle tree structure to obtain a graph G on $N = 2^{n+1} - 1$ nodes[2], for any integer $n \geq 1$.

Both proofs of sequential work can (optionally) be converted into a non-interactive proof by applying the Fiat-Shamir paradigm [28], i.e., given a commitment c' to labels $\ell'_1, \ldots, \ell'_N$ we can use public randomness $r = \mathcal{H}(\chi, N+1, c')$ to sample our set of challenge nodes S. The non-interactive version could be useful in cases where a prover wants to silently timestamp a statement χ without even signaling that s/he might have a statement important enough to timestamp, e.g., a researcher who believes they might resolved a famous open problem may wish to timestamp the discovery without signaling the community until s/he carefully double checks the proof.

In all of the above constructions, security relies on the hardness of computing $\mathcal{H}$-sequences of length T in sequential time $T - 1$. While this can be readily established in the classical parallel random oracle model, proving that this task is in fact hard for a quantum attacker is a much more daunting challenge. As Boneh et al. [16] pointed out, many of the convenient

[1] A DAG G is said to be *ϵ-extremely depth-robust* if it is (e, d)-depth robust for any $e, d > 0$ such that $e + d \leq (1 - \epsilon)N$ where N is the number of nodes in G. Recall that a DAG $G = (V, E)$ is (e, d)-depth robust if for any subset $S \subseteq V$ with $|S| \leq e$ there exists a path of length d in $G - S$.

[2] The graph G is "weighted" depth robust. In particular, there is a weighting function $w : V \to \mathbb{R}_{\geq 0}$ with the property that $\sum_v w(v) \in \mathcal{O}(N \log N)$ and for any subset $S \subseteq V$ with sufficiently small weight $\sum_{v \in S} w(v) \leq cN$ the DAG $G - S$ contains a path of length $\Omega(N)$.

properties (e.g., extractability, programmability, efficient simulation, rewinding, etc.) that are used in classical random oracle security proofs no longer apply in the quantum random oracle model (qROM). An attacker in the (parallel) quantum random oracle model is able to submit entangled queries, giving the attacker much more power. For example, given y a quantum attacker can find a preimage x' such that $\mathcal{H}(x') = y$ with just $\mathcal{O}\left(2^{\lambda/2}\right)$ quantum random oracle queries using Grover's algorithm. By contrast, a classical attacker would need at least $\Omega(2^{\lambda})$ queries to a classical random oracle. Similarly, a quantum attacker can find hash collisions with at most $\mathcal{O}\left(2^{\lambda/3}\right)$ queries, while a classical attacker requires $\Omega(2^{\lambda/2})$ queries. In this paper, we explore the post-quantum security of proofs of sequential work in the parallel quantum random oracle model. We aim to answer the following questions:

Can a quantum attacker running in $T-1$ sequential rounds produce an $\mathcal{H}$-sequence of length T?

Can a quantum attacker running in time $T = (1-\alpha)N$ produce a valid non-interactive proof of sequential work with non-negligible probability?

1.1 Our Contributions

We answer these questions in the negative, thus confirming the security of proof of sequential work schemes in a post-quantum world. We first prove that any quantum attacker making $N-1$ rounds of queries cannot produce an $\mathcal{H}$-sequence of length N, except with negligible probability.

▶ **Definition 1** ($\mathcal{H}$-Sequence). *An $\mathcal{H}$-sequence $x_0, x_1, \ldots, x_s \in \{0,1\}^*$ satisfies the property that for each $1 \leq i \leq s$, there exist $a, b \in \{0,1\}^*$ such that $x_i = a||\mathcal{H}(x_{i-1})||b$. For indexing reasons, we say such an $\mathcal{H}$-sequence has length s (even though there are $s+1$ variables x_i).*

In the classical random oracle model (ROM), it is straightforward to argue that any PoSW prover must find a long $\mathcal{H}$-sequence to pass the audit phase with non-negligible probability. Thus, this result already provides compelling evidence that proofs of sequential work are post-quantum secure in the parallel random oracle model.

Next we consider a non-interactive proof of sequential work applying the Fiat-Shamir transform to the efficient construction of Cohen and Pietrzak [23], and we prove that this construction is secure in the quantum parallel random oracle model. In particular, we show that any attacker running in sequential time $T = (1-\alpha)N$ will fail to produce a valid proof π_χ for any statement $\chi \in \{0,1\}^\lambda$.

While Cohen and Pietrzak [23] proved analogous results in the classical random oracle model, we stress that from a technical standpoint, proving security in the quantum random oracle model is significantly more challenging. In general, there is a clear need to develop new techniques to reason about the security of cryptographic protocols in the quantum random oracle model. Most of the techniques that are used in classical random oracle model do not carry over to the (parallel) quantum random oracle model [16]. For example, if we are simulating a classical attacker, then we can see (extract) all of the random oracle queries that the attacker makes, while we cannot observe a quantum query without measuring it, which would collapse the attacker's quantum state might significantly alter the final output.

Warm-Up Problem: Iterative Hashing

As a warm-up, we first prove an easier result in Theorem 2 that an attacker cannot compute $\mathcal{H}^N(x)$ in sequential time less than $N-1$ in the parallel quantum random oracle model, where a similar result was previously proved by Unruh [39] in the (non-parallel) quantum

random oracle model. Along the way we highlight some of the key challenges that make it difficult to extend the proof to arbitrary $\mathcal{H}$-sequences.

▶ **Theorem 2.** *Given a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ and a random input x, any quantum attacker that makes up to q queries in each of $N-1$ sequential steps can only compute $\mathcal{H}^N(x)$ with probability at most $\frac{N^2}{2^\lambda} + \frac{1}{2^{\lambda-N}} + \sqrt{\frac{48\lambda N^4 q^2 T}{2^{\lambda/2}}}$ in the quantum parallel random oracle model.*

The proof of Theorem 2 is straightforward and we defer it to the full version. Intuitively, iteratively computing $H^N(x)$ induces an $\mathcal{H}$-sequence $x_0, x_1, \ldots, x_n$ with $x_0 = x$, $x_N = \mathcal{H}^N(x)$ and $x_{i+1} = \mathcal{H}(x_i)$. One can easily define a sequence of indistinguishable hybrids where in the last hybrid the final output $x_N = \mathcal{H}^N(x) = \mathcal{H}(x_{N-1})$ is information theoretically hidden from the attacker. In general, in hybrid i, for each $j \leq i$, the value $x_j = \mathcal{H}^j(x) = \mathcal{H}(x_{j-1})$ remains information theoretically hidden until round j. In particular, we replace the random oracle $\mathcal{H}$ with a new stateful oracle $\mathcal{H}'_i(\cdot)$ that is almost identical to $\mathcal{H}(\cdot)$, except that for any $j \leq i$ if the query $\mathcal{H}(x_j)$ is submitted to $\mathcal{H}'(\cdot)$ before round j then the response will be a random unrelated λ-bit string instead of $\mathcal{H}(x_j)$.

We can argue indistinguishability of hybrids i using a result of [10] because if $j > i$, then x_j is information theoretically hidden up until round i and the total query magnitude of $x_j = \mathcal{H}^j(x)$ during round i is negligible. Here, the total query magnitude of a string x_j during round i is defined as the sum of squared amplitudes on states where the attacker is querying string x_j. It then follows that except with negligible probability a quantum attacker cannot compute $\mathcal{H}^N(x)$. The argument does rely on the assumption that the running time T of the attacker is bounded, e.g., $T \leq 2^{c\lambda}$ for some constant $c > 0$.

Our main results are summarized in Theorem 3 and Theorem 4. We show that quantum attackers running in at most $N-1$ sequential steps cannot find an $\mathcal{H}$-sequence of length N with high probability. We also show that for any quantum attackers making at most q quantum queries to the random oracle $\mathcal{H}$ over at most $(1-\alpha)N$ rounds will only be able to produce a valid PoSW with negligible probability.

Technical Challenges: Iterative Hashing vs $\mathcal{H}$-Sequences

Proving that an attacker cannot find an $\mathcal{H}$-sequence of length N in $N-1$ rounds of parallel queries is significantly more challenging. One key difference is that there are exponentially many distinct $\mathcal{H}$-sequences of length N that are consistent with the initial string x_0. By contrast, when we analyze a hash chain, each value on the chain $\mathcal{H}^j(x_0)$ can be viewed as fixed a priori. For $\mathcal{H}$-sequences, it is not clear how one would even define a hybrid where all candidate values of x_i are information-theoretically hidden because these values are not known a priori and there might be exponentially many such candidates. In fact, for any $2 \leq i \leq N$ and *any* string y, it is *likely* that there exists an $\mathcal{H}$-sequence $x_0, \ldots, x_N$ such that $y = x_i$.

Instead, we use a recent idea introduced by [42] that views the random oracle as a superposition of *databases* rather than queries. This view facilitates intuitive simulation of quantum random oracles in a manner similar to classical models, which provides intuitive simulation for queries and circumvents the need to "record all possible queries", which would give an exponential number of possible $\mathcal{H}$-sequences in our case. We give significantly more intuition in Section 4, after formalizing the relevant definitions.

▶ **Theorem 3.** *Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random hash function and let $\delta \geq 1$ be a parameter. Let p be the probability that a quantum adversary making at most q queries over $N-1$ rounds outputs $(x_0, y_0), \ldots, (x_{N-1}, y_{N-1})$ and x_N s.t. $|x_i| \leq \delta\lambda$, $y_i = \mathcal{H}(x_i)$ and $\mathsf{Substring}(y_{i-1}, x_i) = 1$ for each i, i.e., $x_0, \ldots, x_N$ is an $\mathcal{H}$-sequence. Then*

$$p \leq \frac{64q^3\delta\lambda}{2^\lambda} + \frac{2N}{2^\lambda}.$$

Here, $\mathsf{Substring}(y_{i-1}, x_i) = 1$ means that y_{i-1} is a substring of x_i, i.e., there exist $a, b \in \{0,1\}^*$ such that $x_i = a\|y_{i-1}\|b$.

From $\mathcal{H}$-Sequences to Proof of Sequential Work

Theorem 4 focuses on a non-interactive proof of sequential work obtained by applying the Fiat-Shamir transform to the efficient construction of Cohen and Pietrzak [23]. This construction is based on a DAG G with $N = 2^{n+1} - 1$ nodes and maximum indegree n. Given a random oracle $\mathcal{H} : \{0,1\}^{\lambda(n+2)} \to \{0,1\}^\lambda$, an honest prover can generate a proof for any statement $\chi \in \{0,1\}^\lambda$ in sequential time $\mathcal{O}(N)$. We prove that for any constant $\alpha > 0$, an attacker making q queries over $s = N(1-\alpha)$ rounds will fail to produce a valid proof of sequential work for any statement except with negligible probability.

▶ **Theorem 4.** *Suppose $\mathcal{A}$ makes at most q quantum queries to our random oracle $\mathcal{H}$ over at most $s = N(1-\alpha)$ rounds and let p denote the probability that $\mathcal{A}$ outputs a valid (non-interactive) proof of sequential work. Then*

$$p \leq 32q^2(1-\alpha)^{\lfloor\lambda/n\rfloor} + \frac{2q^3}{2^\lambda} + \frac{64q^3(n+2)\lambda}{2^\lambda} + \frac{2\lfloor\lambda/n\rfloor(n+2)}{2^\lambda}.$$

The main intuition for the proof Theorem 4 works as follows. Given a quantum database $\mathcal{D} = \{(x_i, y_i) : i \geq 1\}$ where y_i encodes the output on input x_i with λ bits, we define a set LUCKY_s of databases $\mathcal{D}$ based on the graph coloring (see the full version), in which $\mathcal{D}$ does not contain any collision or $\mathcal{H}$-sequence of length s, yet still contains a "lucky" Merkle tree that has a green path from the challenged node to the root that can be used to extract a proof of sequential work. We show that any attacker making (possibly parallel) q queries can only succeed in measuring a lucky database $\mathcal{D}$ with negligible probability. Finally, we show that any attacker who produces a valid PoSW must measure a database $\mathcal{D}$ that either (1) contains an $\mathcal{H}$-sequence of length s, (2) contains a collision, or (3) is a lucky database. Since each of these events has negligible probability, then it follows that with high probability, the attacker cannot produce a valid PoSW.

1.2 Related Work

Functions that are inherently sequential to compute are a cryptographic primitive used in many applications, such as proof of sequential work [37], verifiable delay functions [15], and time-lock puzzles [36]. The original construction [37] used depth-robust graphs, which have found applications in many areas of cryptography including memory-hard functions (e.g., [8, 5, 6, 14, 13, 7, 12]), proofs of replication [29, 20], and proofs of space [27, 38]. Recently, Cohen and Pietrzak [23] show that $\mathcal{H}$-sequences are difficult for a classical adversary to compute in the classical parallel random oracle model.

The Quantum Random Oracle Model (qROM) was introduced by Boneh et al. [16], who pointed out that for any real world instantiation for the hash function $\mathcal{H}$ (e.g., SHA3), one can build a quantum circuit implementing $\mathcal{H}$. Boneh et al. [16] also provided an example

of a protocol that is secure in the classical ROM, but not in the qROM. Quantum attacks and constructions under the quantum random oracle model have been studied in a number of previous settings, such as unclonable public-key quantum money [1, 2], quantum Merkle puzzles [19, 18], signature schemes [17] and construction of random functions [41].

Security reductions in the classical ROM often exploit properties such as programability and extractability of queries – properties that are lost in the qROM. Zhandry introduced compressed oracles [42] as a way to record quantum queries so that they can be viewed after computation has completed. The new technique has proven to be a useful tool to extend many classical security proofs to the quantum random oracle model, e.g., [11, 21, 34, 4, 31]. Don et. al. [25] recently showed how queries can be extracted on-the-fly in certain settings, e.g., once the algorithm outputs a classical commitment t (e.g., $t = \mathcal{H}(x)$ or $t = \mathsf{Enc}_{pk}(\mathcal{H}(x))$) that is tightly related to the input x.

The non-interactive PoSW we consider in this work is obtained by applying the Fiat-Shamir transform to the interactive PoSW construction of Cohen and Pieterzak [23]. While there is a recent line of work analyzing the security of the Fiat-Shamir transform [28] in the quantum random oracle model [33, 24, 35], applying these results would require us to first establish the security of the interactive PoSW in the (parallel) qROM. We find it easier to directly show that the non-interactive PoSW construction is secure in the (parallel) qROM.

There have been a number of work on parallelizing quantum algorithms or considering parallel queries in the quantum random oracle model. Zalka [40] showed that the parallel version of Grover's algorithm is optimal, e.g., in the ideal cipher model, any parallel key-recovery attacker making at most $q = \mathcal{O}(\sqrt{k2^\lambda})$ quantum queries to the ideal cipher must run in sequential time $\Omega(\sqrt{2^\lambda/k})$. Grover and Radhakrishnan [30] generalized Zalka's result in the setting of multiple items to search. Jeffery et al. [32] studied the parallel quantum query complexity for the element distinctness and the k-sum problem. Ambainis et al. [9] provided an improved one-way to hiding (O2H) theorem in the parallel quantum random oracle model.

In independent work, Chung et al. [22] also studied the problem of finding an $\mathcal{H}$-sequence and non-interactive proofs of sequential work in the parallel quantum random oracle model. They gave comparable bounds also using Zhandry's compressed oracle technique [42], while leveraging an abstract view of Fourier transforms for arbitrary finite Abelian groups. By comparison, our proofs avoid the need for an understanding of abstract algebra, instead using quantum information theory to bound the quantum query complexity through a reduction to classical query complexity. Thus we believe our techniques to be of independent interest, perhaps appealing to a more general audience while also providing the necessary framework to analyze the security of other classical protocols in a post-quantum world.

2 Preliminaries

Let $\mathbb{N}$ denote the set $\{0, 1, \ldots\}$, $[n]$ denote the set $\{1, 2, \ldots, n\}$, and $[a, b] = \{a, a+1, \ldots, b\}$ where $a, b \in \mathbb{N}$ with $a \leq b$. For a function $f(x)$, we recursively define $f^N(x) = f \circ f^{N-1}(x)$ where $\circ$ is a function/operator composition. We say that a non-negative function $\mu(x)$ is *negligible*, if for all polynomials $p(x)$, it holds that $0 \leq \mu(x) < \frac{1}{p(x)}$ for all sufficiently large x.

Let $\{0,1\}^n$ be the set of all bitstrings of length n. Then we define $\{0,1\}^{\leq n} = \cup_{i=0}^{n}\{0,1\}^i$ to be the set of all bitstrings of length at most n including an empty string ε. We denote $||$ as the concatenation of bitstrings. For a bitstring $x \in \{0,1\}^*$, $x[i]$ denotes its i^{th} bit, and $x[i...j] = x[i]||\ldots||x[j]$.

Given quantum states $|\phi\rangle = \sum \alpha_x |x\rangle$ and $|\psi\rangle = \sum \beta_x |x\rangle$, we define the Euclidean distance between the two states to be the quantity $\sqrt{\sum |\alpha_x - \beta_x|^2}$. The *magnitude* of $|x\rangle$ in $|\phi\rangle = \sum \alpha_x |x\rangle$ is α_x and the query probability is $|\alpha_x|^2$ – when we measure the state $|\phi\rangle_x$ we will observe $|x\rangle$ with probability $|\alpha_x|^2$.

2.1 Quantum Random Oracle Model

In the (sequential) quantum random oracle model (qROM), an adversary is given oracle access to a random hash function $\mathcal{H} : \{0,1\}^m \to \{0,1\}^\lambda$. The adversary can submit quantum states as queries to the oracle, so that $\mathcal{H}$ takes as input superposition $|\phi_1\rangle, |\phi_2\rangle, \ldots$. Each $|\phi_i\rangle$ can be expanded as $|\phi_i\rangle = \sum \alpha_{i,x,y} |x, y\rangle$ so that the output is $\sum \alpha_{i,x,y} |x, y \oplus \mathcal{H}(x)\rangle$. Note that when the initial state is of the form $|\phi\rangle = \sum \alpha_x |x, 0^w\rangle$, then the output state will be of the form $\sum \alpha_x |x, \mathcal{H}(x)\rangle$.

Compressed Oracle Technique in the Sequential qROM

Here we introduce the compressed oracle representation introduced by Zhandry [42], which is equivalent to the standard oracle in function. However, the difference between the compressed oracle and the regular oracle is in the encodings of the oracle and query registers as queries are made to the oracles. We will extend the ideas of this technique to the parallel qROM later on.

First, we formally define a database $\mathcal{D}$. A database $\mathcal{D}$ is defined by $\mathcal{D} = \{(x_i, y_i) : i \geq 1\}$ where we write $\mathcal{D}(x_i) = y_i$ to denote that y_i encodes the output on input x_i with λ bits. When $\mathcal{D} = \{\}$ is empty, it is equivalent to viewing the random oracle as being in superposition of all possible random oracles. After q queries, the state can be viewed as $\sum_{x,y,z,\mathcal{D}} \alpha_{x,y,z} |x, y, z\rangle \otimes |\mathcal{D}\rangle$, where $\mathcal{D}$ is a compressed dataset of at most q input/output pairs, x, y are the query registers, and z is the adversary's private storage.

Formally, the compressed oracle technique for the sequential qROM works as follows. Let $\mathcal{H} : \{0,1\}^m \to \{0,1\}^\lambda$ be a random hash function and suppose an adversary is given an oracle access to $\mathcal{H}$. Then we have the following observations:

- It is equivalent to view the usual random oracle mapping $|x, y\rangle \mapsto |x, y \oplus \mathcal{H}(x)\rangle$ (denote as StO) as the *phase* oracle PhsO that maps $|x, y\rangle$ to $(-1)^{y \cdot \mathcal{H}(x)} |x, y\rangle$ by applying Hadamard transforms before and after the oracle query.[3]
- It is also equivalent to view the oracle $\mathcal{H}$ as being in (initially uniform) superposition $\sum_{\mathcal{H}} |\mathcal{H}\rangle$ where we can encode $\mathcal{H}$ as a binary vector of length $2^m \times \lambda$ encoding the λ-bit output for each m-bit input string. Under this view the oracle maps the state $|\phi\rangle = \sum_{x,y} \alpha_{x,y} |x, y\rangle \otimes \sum_{\mathcal{H}} |\mathcal{H}\rangle$ to $\sum_{x,y} \alpha_{x,y} |x, y\rangle \otimes \sum_{\mathcal{H}} |\mathcal{H}\rangle (-1)^{y \cdot \mathcal{H}(x)}$.

If the attacker makes at most q queries, then we can compress the oracle $\mathcal{H}$ and write $|\phi\rangle = \sum_{x,y} \alpha_{x,y} |x, y\rangle \otimes \sum_{\mathcal{D}} |\mathcal{D}\rangle$, where each dataset $\mathcal{D} \in \{0,1\}^{\lambda \times 2^m}$ is sparse, i.e., $\mathcal{D}(x) \neq \perp$ for at most q entries. Intuitively, when $\mathcal{D}(x) = \perp$, we view the random oracle as being in a uniform superposition over potential outputs. Moreover, we can think of the basis state $|\mathcal{D}\rangle$ as corresponding to the superposition $\sum_{\mathcal{H} \in \mathcal{H}_\mathcal{D}} |\mathcal{H}\rangle$ where $\mathcal{H}_\mathcal{D} \subseteq \{0,1\}^{2^m \lambda}$ denote the set of all random oracles that are consistent with $\mathcal{D}$, i.e., if $\mathcal{H} \in \mathcal{H}_\mathcal{D}$ then for all inputs x we either have $\mathcal{D}(x) = \perp$ or $\mathcal{D}(x) = \mathcal{H}(x)$. When viewed in this way, the basis state $|\mathcal{D}\rangle$ encodes prior queries to the random oracle along with the corresponding responses. We can use a compressed phase oracle CPhsO (described below) to model a phase oracle.

[3] Notice that both StO and PhsO are unitary matrices and $\mathsf{StO} = (I^m \otimes \mathcal{H}^{\otimes\lambda})\mathsf{PhsO}(I^m \otimes \mathcal{H}^{\otimes\lambda})$ where I^m is the identity matrix on the first m qubits and $\mathcal{H}^{\otimes\lambda}$ is the Hadamard transform on the λ output qubits.

Compressed Phase Oracle

To properly define a compressed phase oracle CPhsO in the sequential qROM, a unitary local decompression procedure $\mathsf{StdDecomp}_x$ that acts on databases was first defined in [42]. Intuitively, $\mathsf{StdDecomp}_x$ decompresses the value of the database at position x when the database $\mathcal{D}$ is not specified on x and there is a room to expand $\mathcal{D}$, and $\mathsf{StdDecomp}_x$ does nothing when there is no room for decompression. If $\mathcal{D}$ is already specified on x, then we have two cases: if the corresponding y registers are in a state orthogonal to a uniform superposition, then $\mathsf{StdDecomp}_x$ is the identity (no need to decompress). If the y registers are in the state of a uniform superposition, then $\mathsf{StdDecomp}_x$ removes x from $\mathcal{D}$. We refer to the full version for a full description of $\mathsf{StdDecomp}_x$. Now we define $\mathsf{StdDecomp}, \mathsf{Increase}, \mathsf{CPhsO}'$ on the computational basis states as

$$\begin{aligned}\mathsf{StdDecomp}\left(|x,y\rangle \otimes |\mathcal{D}\rangle\right) &= |x,y\rangle \otimes \mathsf{StdDecomp}_x|\mathcal{D}\rangle,\\ \mathsf{Increase}\left(|x,y\rangle \otimes |\mathcal{D}\rangle\right) &= |x,y\rangle \otimes |\mathcal{D}\rangle|(\perp,0^\lambda)\rangle, \text{ and}\\ \mathsf{CPhsO}'\left(|x,y\rangle \otimes |\mathcal{D}\rangle\right) &= (-1)^{y\cdot\mathcal{D}(x)}|x,y\rangle \otimes |\mathcal{D}\rangle,\end{aligned}$$

where the procedure $\mathsf{Increase}$ appends a new register $|(\perp,0^\lambda)\rangle$ at the end of the database. Note that $|\mathcal{D}\rangle|(\perp,0^\lambda)\rangle$ is a database that computes the same partial function as $\mathcal{D}$, but the upper bound on the number of points is increased by 1. Here, we remark that we define $\perp \cdot y = 0$ when defining CPhsO', which implies that CPhsO' does nothing if (x,y) has not yet been added to the database $\mathcal{D}$. Finally, the compressed phase oracle CPhsO can be defined as follows:

$$\mathsf{CPhsO} = \mathsf{StdDecomp} \circ \mathsf{CPhsO}' \circ \mathsf{StdDecomp} \circ \mathsf{Increase},$$

which means that when we receive a query, we first make enough space by increasing the bound and then decompress at x, apply the query, and then re-compress the database. We remark that CPhsO successfully keeps track of positions that are orthogonal to the uniform superposition only because if (x,y) was already specified in $\mathcal{D}$ and the y registers are in the state of a uniform superposition, then $\mathsf{StdDecomp}$ removes x from $\mathcal{D}$ so that CPhsO' does nothing as explained before and the second $\mathsf{StdDecomp}$ in CPhsO will revert (x,y) back to the database.

2.2 Useful Lemmas for Compressed Oracles

Next we introduce some useful lemmas given by Zhandry [42] that are helpful for proving our main result. We first introduce the following variant of Lemma 5 from [42], which is still true for CPhsO because StO and PhsO are perfectly indistinguishable by applying a Hadamard transform before and after each query.

▶ **Lemma 5** ([42]). *Consider a quantum algorithm $\mathcal{A}$ making queries to a random oracle H and outputting tuples $(x_1,\ldots,x_k,y_1,\ldots,y_k,z)$. Let R be a collection of such tuples. Suppose with probability p, $\mathcal{A}$ outputs a tuple such that (1) the tuple is in R, and (2) $\mathcal{H}(x_i) = y_i$ for all i. Now consider running $\mathcal{A}$ with the oracle CPhsO, and suppose the database $\mathcal{D}$ is measured after $\mathcal{A}$ produces its output. Let p' be the probability that (1) the tuple is in R, and (2) $\mathcal{D}(x_i) = y_i$ for all i (and in particular $\mathcal{D}(x_i) \neq \perp$). Then $\sqrt{p} \leq \sqrt{p'} + \sqrt{k/2^n}$.*

We say that a database $\mathcal{D}$ contains a collision if we have $(x,y),(x',y) \in \mathcal{D}$ for $x \neq x'$. We use the notation $\mathsf{COLLIDE}$ to denote the set of all databases that contain a collision. Zhandry upper bounded the probability of finding collisions in a database after making queries to a compressed oracle in the following lemma.

▶ **Lemma 6** ([42]). *For any adversary making q queries to* CPhsO *and an arbitrary number of database read queries, if the database $\mathcal{D}$ is measured after the q queries, the resulting database will contain a collision with probability at most $q^3/2^\lambda$.*

3 Parallel Quantum Random Oracle Model

Quantum Query Bounds with Compressed Dataset

As a warm-up, we review how Zhandry [42] used his compressed oracle technique to provide a greatly simplified proof that Grover's algorithm is asymptotically optimal. Theorem 7 proves that the final measured database $\mathcal{D}$ will not contain a pre-image of 0^λ except with probability $\mathcal{O}\left(q^2/2^\lambda\right)$. We sketch some of the key ideas below as a warm-up and to highlight some of the additional challenges faced in our setting.

▶ **Theorem 7** ([42]). *For any adversary making q queries to* CPhsO *and an arbitrary number of database read queries, if the database $\mathcal{D}$ is measured after the q queries, the probability it contains a pair of the form $(x, 0^\lambda)$ is at most $\mathcal{O}\left(q^2/2^\lambda\right)$.*

We can view Theorem 7 as providing a bound for amplitudes of basis states with a database $\mathcal{D}$ in a set BAD that is defined as

$$\mathsf{BAD} = \{\mathcal{D} : \mathcal{D} \text{ contains a pair of the form } (x, 0^\lambda)\}.$$

Given a basis state $|x, y, z\rangle \otimes |\mathcal{D}\rangle$ with $\mathcal{D} \notin \mathsf{BAD}$ and $x \notin \mathcal{D}$, then the random oracle CPhsO maps this basis state to

$$|\psi\rangle = |x, y, z\rangle \otimes \frac{1}{2^{\lambda/2}} \sum_w (-1)^{y \cdot w} |\mathcal{D} \cup (x, w)\rangle,$$

where the amplitude on states with the corresponding database $\mathcal{D}$ in BAD is just $2^{-\lambda/2}$. We use the following notation to generalize this approach for other purposes:

▶ **Definition 8.** *For a collection of basis states $\widetilde{\mathcal{S}}$ and $|\psi\rangle = \sum_X \alpha_X |X\rangle$, we define*

$$L_2(|\psi\rangle, \widetilde{\mathcal{S}}) = \sqrt{\sum_{X \in \widetilde{\mathcal{S}}} |\alpha_X|^2}$$

to denote the root of the sum of the squared magnitudes of the projection of ψ onto the set of basis states $\widetilde{\mathcal{S}}$. *If $\mathcal{S}$ is a set of databases and $|\psi\rangle = \sum_{x,y,z,\mathcal{D}} \alpha_{x,y,z,\mathcal{D}} |x, y, z\rangle \otimes |\mathcal{D}\rangle$ is a state we define $L_2(|\psi\rangle, \mathcal{S}) = \sqrt{\sum_{x,y,z} \sum_{\mathcal{D} \in \mathcal{S}} |\alpha_{x,y,z,\mathcal{D}}|^2}$.*

An equivalent way to define $L_2(|\psi\rangle, \widetilde{\mathcal{S}})$ is $L_2(|\psi\rangle, \widetilde{\mathcal{S}}) = \left\| P_{\widetilde{\mathcal{S}}}(|\psi\rangle) \right\|_2$ where $P_{\widetilde{\mathcal{S}}}$ projects $|\psi\rangle$ onto the space spanned by $\widetilde{\mathcal{S}}$ e.g., if $|\psi\rangle = \sum_{x,y,z,\mathcal{D}} \alpha_{x,y,z,\mathcal{D}} |x, y, z\rangle \otimes |\mathcal{D}\rangle$ then

$$P_{\widetilde{\mathcal{S}}}(|\psi\rangle) = \sum_{|x,y,z\rangle \otimes |\mathcal{D}\rangle \in \widetilde{\mathcal{S}}} \alpha_{x,y,z,\mathcal{D}} |x, y, z\rangle \otimes |\mathcal{D}\rangle.$$

Using this notation, we can view the proof of Theorem 7 as bounding $L_2(\mathsf{CPhsO}|\psi\rangle, \mathsf{BAD}) - L_2(|\psi\rangle, \mathsf{BAD})$, i.e., the increase in root of squared amplitudes on bad states after each random oracle query.

There are a number of challenges to overcome when directly applying this idea to analyze $\mathcal{H}$-sequences. First, we note that Theorem 7 works in the sequential qROM, which means that the attacker can make only one query in each round. In our setting, the quantum

attacker is allowed to make more than T queries provided that the queries are submitted in parallel batches over $T-1$ rounds. Without the latter restriction, an attacker that makes T total queries will trivially be able to find an $\mathcal{H}$-sequence, even if the attacker is not quantum, by computing $\mathcal{H}^T(x)$ over T rounds. We formalize the *parallel quantum random oracle model* pqROM in Section 3.1 to model an attacker who submits batches $(x_1, y_1), \ldots$ of random oracle queries in each round.

The second primary challenge is that we can not find a static (a priori fixed) set BAD. A naïve approach would fix BAD to be the set of databases that contain an $\mathcal{H}$-sequence of length T, but this would not allow us to bound $L_2(\mathsf{CPhsO}|\psi\rangle, \mathsf{BAD}) - L_2(|\psi\rangle, \mathsf{BAD})$. In particular, if our state $|\psi\rangle$ after round r has non-negligible squared amplitudes on datasets $\mathcal{D}$ that contain an $\mathcal{H}$-sequence of length $r+1$, then it is likely that the attacker will be able to produce an $\mathcal{H}$-sequence of length T after round $T-1$ – in this sense a bad event has already occurred. In our setting, the bad sets must be defined carefully in a round-dependent fashion r. Intuitively, we want to show that $L_2(\mathsf{CPhsO}^k|\psi\rangle, \mathsf{BAD}_{r+1}) - L_2(|\psi\rangle, \mathsf{BAD}_r)$ is small, where BAD_r contains datasets $\mathcal{D}$ that contain an $\mathcal{H}$-sequence of length $r+1$ and CPhsO^k denotes a parallel phase oracle that processes $k \geq 1$ queries in each round. However, reasoning about the behavior of CPhsO^k introduces its own set of challenges when $k > 1$. We address these challenges by carefully defining sets $\mathsf{BAD}_{r,j}$, i.e., the bad set of states after the first j (out of k) queries in round j have been processed. See Section 4 for more details.

3.1 Parallel Quantum Random Oracle Model pqROM

Recall that in the sequential quantum random oracle model (qROM), an adversary can submit quantum states as queries to the oracle, so that a random hash function H takes as input superposition $|\phi_1\rangle, |\phi_2\rangle, \ldots$, and so on. Each $|\phi_i\rangle$ can be expanded as $|\phi_i\rangle = \sum \alpha_{i,x,y}|x, y\rangle$ so that the output is $\sum \alpha_{i,x,y}|x, y \oplus \mathcal{H}(x)\rangle$. Note that when the initial state is of the form $|\phi\rangle = \sum \alpha_x |x, 0^w\rangle$, then the output state will be of the form $\sum \alpha_x |x, \mathcal{H}(x)\rangle$. In the *parallel* quantum random oracle model (pqROM), the adversary can make a batch of queries $q_1, q_2, \ldots$ each round and receives the corresponding output for each of the queries. More precisely, if r_i is the number of queries made in round i, then the input takes the form $|(x_1, y_1), \ldots, (x_{r_i}, y_{r_i})\rangle$ and the corresponding output is $|(x_1, y_1 \oplus \mathcal{H}(x_1)), \ldots, (x_{r_i}, y_{r_i} \oplus \mathcal{H}(x_{r_i}))\rangle$.

We also remark that the equivalence of the standard and phase oracles that we discussed in Section 2.1 remains true with parallelism (by applying Hadamard transforms before and after the query); therefore, we will only consider extending the compressed oracles only on the CPhsO by convenience.

Extending Compressed Oracle Technique to pqROM

As mentioned before, we need to extend CPhsO to be able to handle parallel queries. To make the analysis simpler, we have an approach that essentially sequentially simulates a batch of parallel queries, which as a result, is equivalent to process parallel queries at once. Consider the following example; given a state $|B_i\rangle = |(x_1, y_1), \ldots, (x_k, y_k), z\rangle \otimes |\mathcal{D}\rangle$, let $|\psi_1\rangle$ be the state after processing the first query (x_1, y_1), and let $|\psi_2\rangle$ be the state after processing the second query (x_2, y_2) so that $|\psi_2\rangle = \mathsf{CPhsO}|\psi_1\rangle$. However, recall that CPhsO only acts on the first coordinate. Thus to handle the parallel query sequentially, we would need to switch the order of the coordinates to process the second query properly. Hence, we make a slight modification and redefine $|\psi_2\rangle = \mathsf{Swap}_{1,2} \circ \mathsf{CPhsO} \circ \mathsf{Swap}_{1,2}|\psi_1\rangle$, where

$$\mathsf{Swap}_{1,2}|(x_1, y_1), (x_2, y_2), \ldots\rangle = |(x_2, y_2), (x_1, y_1), \ldots\rangle,$$

and similarly $\mathsf{Swap}_{i,j}(\cdot) = \mathsf{Swap}_{j,i}(\cdot)$ swaps the positions of queries x_i and x_j. Thus, we now define our *parallel* version of a CPhsO oracle, called as CPhsO^i, which handle i parallel queries, recursively as

$$\mathsf{CPhsO}^i = \mathsf{Swap}_{1,i} \circ \mathsf{CPhsO} \circ \mathsf{Swap}_{1,i} \circ \mathsf{CPhsO}^{i-1},$$

where $\mathsf{CPhsO}^1 = \mathsf{CPhsO}$. For a compact notation, we define

$$\mathsf{SCPhsO}_i := \mathsf{Swap}_{1,i} \circ \mathsf{CPhsO} \circ \mathsf{Swap}_{1,i}.$$

That is, we can interpret CPhsO^i as applying SCPhsO_j (essentially) sequentially for $j = 1, \ldots, i$, i.e., $\mathsf{CPhsO}^i = \prod_{j=1}^{i} \mathsf{SCPhsO}^j = \mathsf{SCPhsO}^i \circ \cdots \circ \mathsf{SCPhsO}^1$.

We remark that there are other possible approaches in extending CPhsO to the pqROM. For example, see the full version for further details regarding an additional approach to extending CPhsO to the pqROM.

4 Finding $\mathcal{H}$-Sequences in the pqROM

In this section, we show that quantum adversaries cannot find $\mathcal{H}$-sequences of length N using fewer than $N-1$ steps, thereby showing the security of a construction for proof of sequential work in the *parallel* quantum random oracle model in the next section. Recall that an $\mathcal{H}$-sequence $x_0, x_1, \ldots, x_s \in \{0,1\}^*$ satisfies the property that for each $0 \leq i \leq s-1$, there exist $a, b \in \{0,1\}^*$ such that $x_{i+1} = a||\mathcal{H}(x_i)||b$. Note that the sequence $\mathcal{H}(x), \mathcal{H}^2(x), \ldots, \mathcal{H}^N(x)$ is an $\mathcal{H}$-sequence, so in fact, this proves that quantum adversaries are even more limited than being unable to compute $\mathcal{H}^N(x)$ for a given input x in fewer than $N-1$ steps. In our analysis, we require that $\mathcal{H}$ outputs a λ-bit string but permit each term x_i in the $\mathcal{H}$-sequence to have length $\delta\lambda$, for some variable parameter $\delta \geq 1$.

We begin by introducing some helpful notation. Given a database $\mathcal{D} = \{(x_1, y_1), \ldots, (x_q, y_q)\}$ in the compressed standard oracle view, we can define a directed graph $G_{\mathcal{D}}$ on q nodes $(v_{x_1}, \ldots, v_{x_q})$ so that there is an edge from node v_{x_i} to node v_{x_j} if and only if there exist strings a, b such that $x_j = a||y_i||b$. Thus, the graph $G_{\mathcal{D}}$ essentially encodes possible $\mathcal{H}$-sequences by forming edges between nodes v_{x_i} and v_{x_j} if and only if y_i is a substring of x_j. More precisely, given a path $P = (v_{x_{i_0}}, v_{x_{i_1}}, \ldots, v_{x_{i_k}})$ in $G_{\mathcal{D}}$, we define

- $\mathsf{HSeq}(P) := (x_{i_0}, x_{i_1}, \ldots, x_{i_k})$ denotes a corresponding $\mathcal{H}$-sequence of length k, and
- $\mathsf{LAST}(P) := v_{x_{i_k}}$ denotes the endpoint of the path P in $G_{\mathcal{D}}$ that corresponds to the output of the last label in the corresponding $\mathcal{H}$-sequence.

We also define a predicate $\mathsf{Substring}(x, y)$ where $\mathsf{Substring}(x, y) = 1$ if and only if x is a substring of y, i.e., there exist $a, b \in \{0,1\}^*$ such that $y = a||x||b$, and $\mathsf{Substring}(x, y) = 0$ otherwise.

▶ **Example 9.** Suppose that $\mathcal{D} = \{(x_1, y_1), \ldots, (x_8, y_8)\}$ where $(x_1, y_1) = (00000, 000)$, $(x_2, y_2) = (00010, 001)$, $(x_3, y_3) = (00101, 010)$, $(x_4, y_4) = (00110, 011)$, $(x_5, y_5) = (01001, 100)$, $(x_6, y_6) = (01100, 101)$, $(x_7, y_7) = (10010, 110)$, and $(x_8, y_8) = (11001, 111)$. We observe that the graph $G_{\mathcal{D}}$ induced from the database $\mathcal{D}$ should include the edge (v_1, v_2) since $x_2 = 00010 = y_1||10$, and so forth. Then we have the following graph $G_{\mathcal{D}}$ (see the full version), which includes an $\mathcal{H}$-sequence of length $s = 5$. In this example, we can say that for a path $P = (v_1, v_2, v_3, v_5, v_7, v_8)$ of length 5, we have a corresponding $\mathcal{H}$-sequence $\mathsf{HSeq}(P) = (x_1, x_2, x_3, x_5, x_7, x_8)$ of length 5 since we have $x_2 = y_1||10$, $x_3 = y_2||01$, and so on. Note that in this case we have $\mathsf{LAST}(P) = v_8$.

▶ **Definition 10.** *We define* PATH_s *to be the set of the databases (compressed random oracles)* $\mathcal{D}$ *such that* $G_{\mathcal{D}}$ *contains a path of length* s*.*

$$\mathsf{PATH}_s := \{\mathcal{D} : G_{\mathcal{D}} \text{ contains a path of length } s\}.$$

Note that PATH_s *intuitively corresponds to an* $\mathcal{H}$*-sequence of length* s*. We also define* $\widetilde{\mathsf{PATH}}_s$ *to be the set of basis states with* $\mathcal{D}$ *in* PATH_s *as follows:*

$$\widetilde{\mathsf{PATH}}_s := \{|(x_1, y_1), \ldots, (x_k, y_k), z\rangle \otimes |\mathcal{D}\rangle : \mathcal{D} \in \mathsf{PATH}_s\}.$$

Challenges of Quantum Query Bounds on Finding an $\mathcal{H}$-Sequence

To bound the probability that a single round of queries finds an $\mathcal{H}$-sequence of length $s+1$ conditioned on the previous queries not finding an $\mathcal{H}$-sequence of length s, we consider the set of basis states $\{|B_i\rangle\}_i$ of the form $|(x_1, y_1), \ldots, (x_k, y_k), z\rangle \otimes |\mathcal{D}\rangle$, where $\mathcal{D}$ contains at most $q-k$ entries and $\mathcal{D} \notin \mathsf{PATH}_s$. Let $|\psi\rangle = \sum \alpha_i |B_i\rangle$ be an arbitrary state that is a linear combination of $\{|B_i\rangle\}_i$ and let $|\psi'\rangle = \mathsf{CPhsO}^k|\psi\rangle$. Then we would like to bound $L_2(|\psi'\rangle, \mathsf{PATH}_{s+1})$, but there are substantial challenges in computing $L_2(|\psi'\rangle, \mathsf{PATH}_{s+1})$ directly.

For example, given a decomposition of $|\psi\rangle = \sum_B \alpha_B |B\rangle$ into basis states, we might like to compute $\eta_B = \mathsf{CPhsO}^k(|B\rangle)$ for each basis state $|B\rangle$ and then decompose $|\psi'\rangle = \sum_B \alpha_B \eta_B$. However, each term η_B is no longer a basis state making it difficult to describe the state $|\psi'\rangle$ in a helpful way so that we can bound $L_2(|\psi'\rangle, \widetilde{\mathsf{PATH}}_{s+1})$. The challenges are amplified as $|\psi'\rangle$ is the result of k parallel queries.

Overcoming the Challenges

Our approach is to consider an intermediate sequence of states $|\psi_0\rangle = |\psi\rangle, \ldots, |\psi_k\rangle = |\psi'\rangle$, where $|\psi_i\rangle$ intuitively encodes the state after the i^{th} query (in the block of parallel queries) is processed. Then from the definition of CPhsO^i, we have $|\psi_{i+1}\rangle = \mathsf{Swap}_{1,i+1} \circ \mathsf{CPhsO} \circ \mathsf{Swap}_{1,i+1}|\psi_i\rangle = \mathsf{SCPhsO}_{i+1}|\psi_i\rangle$ for all $i \in [k]$. This approach presents a new subtle challenge. Consider a basis state $|B\rangle = |(x_1, y_1), \ldots, (x_k, y_k)\rangle \otimes |\mathcal{D}\rangle$, where the longest path in $G_{\mathcal{D}}$ (the $\mathcal{H}$-sequence) has length $s-1$. We can easily argue that $L_2(\mathsf{SCPhsO}_1|B\rangle, \widetilde{\mathsf{PATH}}_{s+1})$ is negligible since the initial basis state $|B\rangle \notin \widetilde{\mathsf{PATH}}_s$. Now we would like to argue that $L_2(\mathsf{SCPhsO}_2 \circ \mathsf{SCPhsO}_1|B\rangle, \widetilde{\mathsf{PATH}}_{s+1})$ is negligibly small, but it is unclear how to prove this since we might have $L_2(\mathsf{SCPhsO}_1|B\rangle, \widetilde{\mathsf{PATH}}_s) = 1$, e.g., all of the datasets $\mathcal{D}$ found in the support of $\mathsf{SCPhsO}_1|B\rangle$ have paths of length s.

Overcoming this barrier requires a much more careful definition of our "bad" states. We introduce some new notions to make the explanations clearer. Suppose that a database $\mathcal{D} \notin \mathsf{PATH}_s$. If $\mathcal{D}$ has no $\mathcal{H}$-sequence of length s, it may not be the case that $|\psi_i\rangle$ has no $\mathcal{H}$-sequence of length s. However, the intuition is that since the queries $x_1, \ldots, x_k$ are made in the same round, then it is acceptable to have an $\mathcal{H}$-sequence of length s, provided that the last entry in the $\mathcal{H}$-sequence involves some (x_i, y_i). Thus we define $\mathsf{PATH}_{s,i}(x_1, \ldots, x_k)$ to be a set of the databases with the induced graph $G_{\mathcal{D}}$ having a path of length s that does not end in a term that contains $\mathcal{H}(x_i)$:

$$\mathsf{PATH}_{s,i}(x_1, \ldots, x_k) := \{\mathcal{D} : G_{\mathcal{D}} \text{ contains a path } P \text{ of length } s \text{ and } \mathsf{LAST}(P) \notin \{v_{x_1}, \ldots, v_{x_i}\}\},$$

where we recall that $\mathsf{LAST}(P)$ denotes the endpoint of the path P in $G_{\mathcal{D}}$, which corresponds to the output of the last label in the corresponding $\mathcal{H}$-sequence as defined before. We then define

$$\widetilde{\mathsf{PATH}}_{s,i} := \{|(x_1, y_1), \ldots, (x_k, y_k), z\rangle \otimes |\mathcal{D}\rangle : \mathcal{D} \in \mathsf{PATH}_{s,i}(x_1, \ldots, x_k)\},$$

which denotes the set of the states where the corresponding database $\mathcal{D}$ is in the set $\mathsf{PATH}_{s,i}(x_1, \ldots, x_k)$.

Now we define a set $\mathsf{Contain}_{s,i}$, which intuitively represents the set of databases so that the queries correspond to the guesses for preimages:

$$\mathsf{Contain}_{s,i}(x_1, \ldots, x_k) := \{\mathcal{D} \ : \exists j \leq k \text{ s.t. } \mathsf{Substring}(\mathcal{D}(x_i), x_j) = 1 \text{ and } G_{\mathcal{D}} \text{ contains a path of length } s \text{ ending at } x_i\}.$$

We then define

$$\widetilde{\mathsf{Contain}}_{s,i} := \{|(x_1, y_1), \ldots, (x_k, y_k), z\rangle \otimes |\mathcal{D}\rangle : \mathcal{D} \in \mathsf{Contain}_{s,i}(x_1, \ldots, x_k)\},$$

which denotes the set of the states where the corresponding database $\mathcal{D}$ is in the set $\mathsf{Contain}_{s,i}(x_1, \ldots, x_k)$. Therefore, we define $\mathsf{BAD}_{s,i}(x_1, \ldots, x_k)$ to be the set of databases that are not in PATH_s but upon the insertion of $(x_1, y_1), \ldots, (x_i, y_i)$, is a member of PATH_{s+1}:

$$\mathsf{BAD}_{s,i}(x_1, \ldots, x_k) := \mathsf{PATH}_{s,i}(x_1, \ldots, x_k) \cup \bigcup_{j=1}^{i} \mathsf{Contain}_{s,j}(x_1, \ldots, x_k)$$

Finally, we define

$$\widetilde{\mathsf{BAD}}_{s,i} := \{|(x_1, y_1), \ldots, (x_k, y_k), z\rangle \otimes |\mathcal{D}\rangle : \mathcal{D} \in \mathsf{BAD}_{s,i}(x_1, \ldots, x_k)\}$$

to represent the set of the states where the corresponding database $\mathcal{D}$ is in the set $\mathsf{BAD}_{s,i}(x_1, \ldots, x_k)$.

We now process each query $(x_1, y_1), \ldots, (x_k, y_k)$ sequentially and argue that the mass projected onto PATH_{s+1} by each step CPhsO^i is negligible. To prove this, we argue that $L_2(|\psi_i\rangle, \widetilde{\mathsf{BAD}}_{s,i})$ is negligible for all $i \leq k$. We use the convention that $|\psi_0\rangle$ is the initial state and $|\psi_i\rangle = \mathsf{SCPhsO}_i|\psi_{i-1}\rangle$ for all $i \in [k]$ so that $|\psi_k\rangle$ is the last state, after all the queries have been processed. Similarly, we use the convention that $\widetilde{\mathsf{BAD}}_{s,0} = \widetilde{\mathsf{PATH}}_s$.

We first give the following lemma.

▶ **Lemma 11.** *Suppose that $\mathcal{D}'$ is a database such that $\mathcal{D}'(x_{i+1}) = \perp$. If $\mathcal{D} = \mathcal{D}' \cup (x_{i+1}, w) \notin \mathsf{BAD}_{s,i}(x_1, \ldots, x_k)$, then $\mathcal{D} \notin \mathsf{BAD}_{s,i+1}(x_1, \ldots, x_k)$.*

Proof. Since $\mathcal{D} \notin \mathsf{PATH}_{s,i}(x_1, \ldots, x_k)$, any path of length s must end at one of the vertices $v_{x_1}, \ldots, v_{x_i}$ in the graph $G_{\mathcal{D}}$. Hence, no path of length s ends at $v_{x_{i+1}}$ unless we have a duplicate query $x_j = x_{i+1}$ for some $j < i + 1$. Now we have two cases:

(1) If x_{i+1} is distinct from x_j for all $j < i+1$, then by the previous observation we immediately have that $\mathcal{D} \notin \mathsf{PATH}_{s,i+1}(x_1, \ldots, x_k)$. Furthermore, $G_{\mathcal{D}}$ contains no path of length s ending at node $v_{x_{i+1}}$ since any path of length s must end at one of $v_{x_1}, \ldots, v_{x_i}$. Hence, $\mathcal{D} \notin \mathsf{Contain}_{s,i+1}(x_1, \ldots, x_k)$. Taken together, we have that $\mathcal{D} \notin \mathsf{BAD}_{s,i+1}(x_1, \ldots, x_k)$.

(2) If $x_{i+1} = x_j$ $(j < i + 1)$ is a duplicate query, then there might be a path of length s ending at $v_{x_{i+1}} = v_{x_j}$ in $\mathcal{D}$. However, due to the duplicate we have $\{v_{x_1}, \ldots, v_{x_i}\} = \{v_{x_1}, \ldots, v_{x_{i+1}}\}$. Therefore, $\mathcal{D} \notin \mathsf{PATH}_{s,i+1}(x_1, \ldots, x_k)$. Now we want to argue that

$\mathcal{D} \notin \mathsf{Contain}_{s,i+1}(x_1,\ldots,x_k)$. Note that $\mathcal{D}(x_{i+1}) = \mathcal{D}(x_j)$ for some $j < i+1$, which implies that $\mathsf{Substring}(\mathcal{D}(x_j), x_l) = 1 \Leftrightarrow \mathsf{Substring}(\mathcal{D}(x_{i+1}), x_l) = 1$ for all $l \in [k]$. If $\mathcal{D} \in \mathsf{Contain}_{s,i+1}(x_1,\ldots,x_k)$, then there exists a path of length s ending at x_j and $\mathsf{Substring}(\mathcal{D}(x_j), x_l) = 1$ for some $l \leq k$. This implies that $\mathcal{D} \in \mathsf{Contain}_{s,j}(x_1,\ldots,x_k)$ and therefore $\mathcal{D} \in \mathsf{BAD}_{s,i}(x_1,\ldots,x_k)$, which is a contradiction. Hence, we have that $\mathcal{D} \notin \mathsf{Contain}_{s,i+1}(x_1,\ldots,x_k)$ and therefore $\mathcal{D} \notin \mathsf{BAD}_{s,i+1}(x_1,\ldots,x_k)$ in this case as well. Taken together, we can conclude that if $\mathcal{D} \notin \mathsf{BAD}_{s,i}(x_1,\ldots,x_k)$, then it is also the case that $\mathcal{D} \notin \mathsf{BAD}_{s,i+1}(x_1,\ldots,x_k)$. ◀

▶ **Lemma 12.** *For each $i \in \{0,1,\ldots,k-1\}$,*

$$L_2(|\psi_{i+1}\rangle, \widetilde{\mathsf{BAD}}_{s,i+1}) - L_2(|\psi_i\rangle, \widetilde{\mathsf{BAD}}_{s,i}) \leq \frac{4\sqrt{q\delta\lambda + k\delta\lambda}}{2^{\lambda/2}}.$$

Proof. To argue that the projection onto $\widetilde{\mathsf{BAD}}_{s,i}$ increases by a negligible amount for each query, we use a similar argument to [42]. Recall that $\mathsf{SCPhsO}_{i+1} = \mathsf{Swap}_{1,i+1} \circ \mathsf{CPhsO} \circ \mathsf{Swap}_{1,i+1}$. Namely, we consider the projection of $|\psi_{i+1}\rangle = \mathsf{SCPhsO}_{i+1}|\psi_i\rangle$ onto orthogonal spaces as follows:

- We first define P_i (resp. P) to be the projection onto the span of basis states $|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes |\mathcal{D}\rangle \in \widetilde{\mathsf{BAD}}_{s,i}$ (resp. basis states in $\widetilde{\mathsf{BAD}}_{s,i+1}$).
- Next we define Q_i to be the projection onto states $|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes |\mathcal{D}\rangle \notin \widetilde{\mathsf{BAD}}_{s,i}$ such that $y_{i+1} \neq 0$ and $\mathcal{D}(x_{i+1}) = \perp$. Intuitively, Q_i represents the projection onto states that are not bad where $\mathsf{SCPhsO}_{i+1}|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes |\mathcal{D}\rangle = |(x_1,y_1),\ldots,(x_k,y_k),z\rangle \sum_w \otimes |\mathcal{D} \cup (x_{i+1},w)\rangle$ will add a new tuple (x_{i+1},w) to the dataset.
- We then define R_i to be the projection onto states $|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes |\mathcal{D}\rangle$ such that $|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes |\mathcal{D}\rangle \notin \widetilde{\mathsf{BAD}}_{s,i}$, $y_{i+1} \neq 0$ and $\mathcal{D}(x_{i+1}) \neq \perp$, so that the value of x_{i+1} has been specified in databases corresponding to these states.
- Finally, we define S_i to be the projection onto states $|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes |\mathcal{D}\rangle$ such that $|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes |\mathcal{D}\rangle \notin \widetilde{\mathsf{BAD}}_{s,i}$ and $y_{i+1} = 0$.

Since P_i, Q_i, R_i, S_i project onto disjoint states that span the entirety of $|\psi_{i+1}\rangle$ then we have $P_i + Q_i + R_i + S_i = \mathbb{I}$, where $\mathbb{I}$ denotes the identity operator. We analyze how P acts on these components separately. For the component $P_i|\psi_i\rangle$, it is easy to verify that $\|P \circ \mathsf{SCPhsO}_{i+1}(P_i|\psi_i\rangle)\|_2 \leq \|\mathsf{SCPhsO}_{i+1}(P_i|\psi_i\rangle)\|_2 \leq \|P_i|\psi_i\rangle\|_2$. See the full version for the formal proof of Lemma 13.

▶ **Lemma 13.** $\|P \circ \mathsf{SCPhsO}_{i+1}(P_i|\psi_i\rangle)\|_2 \leq \|P_i|\psi_i\rangle\|_2$.

Next, to analyze how the projection P acts on $\mathsf{SCPhsO}_{i+1}(Q_i|\psi_i\rangle)$, we note that $\mathsf{SCPhsO}_{i+1}(|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes |\mathcal{D}\rangle) = |(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes \sum_w 2^{-\lambda/2}(-1)^{w\cdot y_{i+1}}|\mathcal{D} \cup (x_{i+1},w)\rangle$ for any basis state in the support of $Q_i|\psi_i\rangle$. We then use a classical counting argument to upper bound the number of strings w such that $|(x_1,y_1),\ldots,(x_k,y_k),z\rangle \otimes \sum_w |\mathcal{D} \cup (x_{i+1},w)\rangle \in \widetilde{\mathsf{BAD}}_{s,i+1}$ by decomposing the databases in $\mathsf{BAD}_{s,i+1}(x_1,\ldots,x_k)$ into the databases in $\mathsf{PATH}_{s,i+1}(x_1,\ldots,x_k)$ and the databases in $\bigcup_{j=1}^{i+1} \mathsf{Contain}_{s,j}(x_1,\ldots,x_k)$. Intuitively, since $\mathcal{D} \notin \mathsf{BAD}_{s,i}(x_1,\ldots,x_k)$ the only way to have $\mathcal{D} \cup (x_{i+1},w) \in \mathsf{PATH}_{s,i+1}(x_1,\ldots,x_k)$ is if w is a substring of some x_j with $j \leq k$ or w is the substring of some other input x in the database $\mathcal{D}$. We bound the number of databases in $\mathsf{PATH}_{s,i+1}(x_1,\ldots,x_k)$ by noting that *any* string $x \in \{0,1\}^{\delta\lambda}$ contains *at most* $\delta\lambda$ unique contiguous substrings of length λ, so there are at most $\delta\lambda$ values of w such that $\mathsf{Substring}(w,x) = 1$. Since $|\mathcal{D} \cup (x_{i+1},w)\rangle$ contains at most q entries, then by a union bound, there are at most $q\delta\lambda$ strings w such

that exists $x \in \{0,1\}^*$ such that $\mathsf{Substring}(w,x) = 1$ and $\mathcal{D}(x) \neq \perp$ or $x = x_{i+1}$. Thus, $|\{w : D \cup (x_{i+1}, w) \in \mathsf{BAD}_{s,i+1}(x_1, \ldots, x_k)\}| \leq q\delta\lambda$. We similarly bound the number of databases in $\bigcup_{j=1}^{i+1} \mathsf{Contain}_{s,j}(x_1, \ldots, x_k)$ by noting that if $\mathcal{D} \notin \mathsf{BAD}_{s,i}(x_1, \ldots, x_k)$ then the only way for $\mathcal{D} \cup (x_{i+1}, w)$ to be in $\bigcup_{j=1}^{i+1} \mathsf{Contain}_{s,j}(x_1, \ldots, x_k)$ is if for some $j \leq k$ the string w is a substring of x_j i.e., $\mathsf{Substring}(w, x_j) = 1$. A similar argument shows that the number of strings w such that $\mathcal{D} \cup (x_{i+1}, w) \in \bigcup_{j=1}^{i+1} \mathsf{Contain}_{s,j}(x_1, \ldots, x_k)$ is at most $k\delta\lambda$.

We thus show the following (see the full version for the full proof):

▶ **Lemma 14.** $\|P \circ \mathsf{SCPhsO}_{i+1}(Q_i|\psi_i\rangle)\|_2^2 \leq \frac{q\delta\lambda + k\delta\lambda}{2^\lambda}$.

We next consider how P acts upon the basis states of $\mathsf{SCPhsO}_{i+1}(R_i|\psi_i\rangle)$. We first relate this quantity to $\mathsf{SCPhsO}_{i+1}(|x,y,z\rangle \otimes |\mathcal{D}' \cup (x_{i+1}, w)\rangle)$, where $\mathcal{D}'$ is the database $\mathcal{D}$ with x_{i+1} removed. Since $\mathcal{D} = |\mathcal{D}' \cup (x_{i+1}, w)\rangle \notin \mathsf{BAD}_{s,i}(x_1, \ldots, x_k)$, then we again use a similar classical counting argument to upper bound the number of strings w' such that $\mathcal{D}' \cup (x_{i+1}, w') \in \mathsf{BAD}_{s,i+1}(x_1, \ldots, x_k)$. Lemma 15 then follows from algebraic manipulation similar to [42]. See the full version for the formal proof.

▶ **Lemma 15.** $\|P \circ \mathsf{SCPhsO}_{i+1}(R_i|\psi_i\rangle)\|_2^2 \leq \frac{9(q\delta\lambda + k\delta\lambda)}{2^\lambda}$.

Finally, we bound the projection of P onto the states of $\mathsf{SCPhsO}_{i+1}(S_i|\psi_i\rangle)$:

▶ **Lemma 16.** $\|P \circ \mathsf{SCPhsO}_{i+1}(S_i|\psi_i\rangle)\|_2 = 0$.

Proof. We observe that $P \circ \mathsf{SCPhsO}_{i+1}(S_i|\psi_i\rangle) = 0$, since for any basis state $|(x_1,y_1), \ldots, (x_k,y_k), z\rangle \otimes |\mathcal{D}\rangle$ state in the support of $S_i|\psi_i\rangle$, we have

$$\mathsf{SCPhsO}_{i+1}|(x_1,y_1), \ldots, (x_k,y_k), z\rangle \otimes |\mathcal{D}\rangle = |(x_1,y_1), \ldots, (x_k,y_k), z\rangle \otimes |\mathcal{D}\rangle,$$

i.e., $\mathsf{SCPhsO}_{i+1}(S_i|\psi_i\rangle) = S_i|\psi_i\rangle$. We also note that since $\mathcal{D} \notin \mathsf{BAD}_{s,i}(x_1, \ldots, x_k)$ and x_{i+1} is not being inserted into the dataset that $\mathcal{D} \notin \mathsf{BAD}_{s,i+1}(x_1, \ldots, x_k)$. Hence, we have that $\|P \circ \mathsf{SCPhsO}_{i+1}(S_i|\psi_i\rangle)\|_2 = 0$. ◀

Thus from Lemma 13, Lemma 14, Lemma 15, Lemma 16, and by triangle inequality, we have

$$\begin{aligned}
\|P \circ \mathsf{SCPhsO}_{i+1}|\psi_i\rangle\|_2 &\leq \|P \circ \mathsf{SCPhsO}_{i+1}(P_i|\psi_i\rangle)\|_2 + \|P \circ \mathsf{SCPhsO}_{i+1}(Q_i|\psi_i\rangle)\|_2 \\
&\quad + \|P \circ \mathsf{SCPhsO}_{i+1}(R_i|\psi_i\rangle)\|_2 + \|P \circ \mathsf{SCPhsO}_{i+1}(S_i|\psi_i\rangle)\|_2 \\
&\leq \|P_i|\psi_i\rangle\|_2 + \frac{3\sqrt{q\delta\lambda + k\delta\lambda}}{2^{\lambda/2}} + \frac{\sqrt{q\delta\lambda + k\delta\lambda}}{2^{\lambda/2}} \\
&\leq \|P_i|\psi_i\rangle\|_2 + \frac{4\sqrt{q\delta\lambda + k\delta\lambda}}{2^{\lambda/2}}.
\end{aligned}$$

Since we have $\|P \circ \mathsf{SCPhsO}_i|\psi_i\rangle\|_2 = L_2(|\psi_{i+1}\rangle, \widetilde{\mathsf{BAD}}_{s,i+1})$ and $\|P_i|\psi_i\rangle\|_2 = L_2(|\psi_i\rangle, \widetilde{\mathsf{BAD}}_{s,i})$, we can conclude that $L_2(|\psi_{i+1}\rangle, \widetilde{\mathsf{BAD}}_{s,i+1}) - L_2(|\psi_i\rangle, \widetilde{\mathsf{BAD}}_{s,i}) \leq \frac{4\sqrt{q\delta\lambda + k\delta\lambda}}{2^{\lambda/2}}$. ◀

We now bound the probability that a single round of queries finds an $\mathcal{H}$-sequence of length $s+1$ conditioned on the previous queries not finding an $\mathcal{H}$-sequence of length s.

▶ **Lemma 17.** *Let $|\psi\rangle$ be an initial state and let $|\psi'\rangle = \mathsf{CPhsO}^k|\psi\rangle$. Then we have $L_2(|\psi'\rangle, \widetilde{\mathsf{PATH}}_{s+1}) - L_2(|\psi\rangle, \widetilde{\mathsf{PATH}}_s) \leq \frac{4k\sqrt{q\delta\lambda + k\delta\lambda}}{2^{\lambda/2}}$.*

Proof. We consider the sequence of states $|\psi\rangle = |\psi_0\rangle, \ldots, |\psi_k\rangle = |\psi'\rangle$ with $|\psi_i\rangle = \mathsf{CPhsO}^i|\psi\rangle$. By Claim 18 it suffices to bound $L_2(|\psi_k\rangle, \widetilde{\mathsf{BAD}}_{s,k})$ as $L_2(|\psi_k\rangle, \widetilde{\mathsf{PATH}}_{s+1}) \leq L_2(|\psi_k\rangle, \widetilde{\mathsf{BAD}}_{s,k})$. See the full version for the proof of Claim 18.

▷ Claim 18. $\widetilde{\mathsf{PATH}}_{s+1} \subseteq \widetilde{\mathsf{BAD}}_{s,k}$.

Recall that we use the convention $\widetilde{\mathsf{BAD}}_{s,0} = \widetilde{\mathsf{PATH}}_s$ and $|\psi_0\rangle$ is the initial state so that by Lemma 12,

$$\begin{aligned} L_2(|\psi_k\rangle, \widetilde{\mathsf{BAD}}_{s,k}) &= L_2(|\psi_0\rangle, \widetilde{\mathsf{BAD}}_{s,0}) + \sum_{i=0}^{k-1} \left[L_2(|\psi_{i+1}\rangle, \widetilde{\mathsf{BAD}}_{s,i+1}) - L_2(|\psi_i\rangle, \widetilde{\mathsf{BAD}}_{s,i}) \right] \\ &\leq L_2(|\psi_0\rangle, \widetilde{\mathsf{BAD}}_{s,0}) + \sum_{i=0}^{k-1} \frac{4\sqrt{q\delta\lambda + k\delta\lambda}}{2^{\lambda/2}} \\ &= L_2(|\psi_0\rangle, \widetilde{\mathsf{PATH}}_s) + \frac{4k\sqrt{q\delta\lambda + k\delta\lambda}}{2^{\lambda/2}}. \end{aligned}$$

Hence, $L_2(|\psi_k\rangle, \widetilde{\mathsf{PATH}}_{s+1}) \leq L_2(|\psi_k\rangle, \widetilde{\mathsf{BAD}}_{s,k}) \leq L_2(|\psi_0\rangle, \widetilde{\mathsf{PATH}}_s) + \frac{4k\sqrt{q\delta\lambda+k\delta\lambda}}{2^{\lambda/2}}$ which implies that $L_2(|\psi'\rangle, \widetilde{\mathsf{PATH}}_{s+1}) - L_2(|\psi\rangle, \widetilde{\mathsf{PATH}}_s) \leq \frac{4k\sqrt{q\delta\lambda+k\delta\lambda}}{2^{\lambda/2}}$. ◀

We now show that a quantum adversary that makes up to q rounds over $N-1$ rounds can only find an $\mathcal{H}$-sequence of length N with negligible probability.

▶ **Lemma 19.** *Suppose that in each round $i \in [N-1]$, the adversary $\mathcal{A}$ makes a query to the parallel oracle CPhsO^{k_i} and that the total number of queries is bounded by q, i.e., $\sum_{i=1}^{N-1} k_i \leq q$. Then $\mathcal{A}$ measures a database in PATH_N with probability at most $\frac{32q^3\delta\lambda}{2^\lambda}$.*

Proof. Let $|\psi_0\rangle$ be the initial state and let U_r represent a unitary transform applied by $\mathcal{A}$ in between batches of queries to the quantum oracle. Then we define $|\psi_r\rangle = U_r \circ \mathsf{CPhsO}^{k_r}|\psi_{r-1}\rangle$ for each round $r \in [N-1]$. Thus, the attacker $\mathcal{A}$ yields a sequence of states $|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle$. We remark that U_r may only operate on the state $|x, y, z\rangle$ and cannot impact the compressed oracle $\mathcal{D}$, e.g., $U_r\left(|x', y', z'\rangle \otimes |\mathcal{D}\rangle\right) = \sum_{x,y,z} \alpha_{x,y,z}|x, y, z\rangle \otimes |\mathcal{D}\rangle$. Thus,

$$L_2(U_r \circ \mathsf{CPhsO}^{k_r}|\psi_{r-1}\rangle, \widetilde{\mathsf{PATH}}_{r+1}) = L_2(\mathsf{CPhsO}^{k_r}|\psi_{r-1}\rangle, \widetilde{\mathsf{PATH}}_{r+1}),$$

so we can effectively ignore the intermediate unitary transform U_r in our analysis below. Now we can apply the previous lemma to conclude that

$$L_2(|\psi_i\rangle, \widetilde{\mathsf{PATH}}_{i+1}) \leq \frac{4k_i\sqrt{2q\delta\lambda}}{2^{\lambda/2}} + L_2(|\psi_{i-1}\rangle, \widetilde{\mathsf{PATH}}_i) \ .$$

By the triangle inequality we have

$$L_2(|\psi_{N-1}\rangle, \widetilde{\mathsf{PATH}}_N) \leq \sum_{i=0}^{N-1} \frac{4k_i\sqrt{2q\delta\lambda}}{2^{\lambda/2}} \leq \frac{\sqrt{32q^3\delta\lambda}}{2^{\lambda/2}} \ .$$

Hence, $\mathcal{A}$ measures a database in PATH_N with probability at most $\left(\frac{\sqrt{32q^3\delta\lambda}}{2^{\lambda/2}}\right)^2 = \frac{32q^3\delta\lambda}{2^\lambda}$. ◀

Thus we have shown that a quantum adversary that makes $N-1$ rounds of parallel queries should generally not find an $\mathcal{H}$-sequence of length N within their queries. Then we bound the probability that the quantum adversary outputs an $\mathcal{H}$-sequence of length N by a standard approach, e.g. [23, 42] of additionally the probability that the quantum adversary makes a "lucky guess".

▶ **Theorem 3.** *Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random hash function and let $\delta \geq 1$ be a parameter. Let p be the probability that a quantum adversary making at most q queries over $N-1$ rounds outputs $(x_0, y_0), \ldots, (x_{N-1}, y_{N-1})$ and x_N s.t. $|x_i| \leq \delta\lambda$, $y_i = \mathcal{H}(x_i)$ and* $\mathsf{Substring}(y_{i-1}, x_i) = 1$ *for each i, i.e., $x_0, \ldots, x_N$ is an $\mathcal{H}$-sequence. Then*

$$p \leq \frac{64q^3\delta\lambda}{2^\lambda} + \frac{2N}{2^\lambda}.$$

Proof. By Lemma 5 the probability p is upper bounded by $2p' + 2N2^{-\lambda}$ where p' denotes the probability that an attacker measures $\mathcal{D} \in \mathsf{PATH}_N$. By Lemma 19 we have $p' \leq \frac{32q^3\delta\lambda}{2^\lambda}$ and the result follows immediately. ◀

5 Security of PoSW in the pqROM

In this section, we show the security of a construction for proofs of sequential work in the *parallel* quantum random oracle model (pqROM). Here, we focus on the non-interactive version of PoSW, which can be obtained by applying a Fiat-Shamir transform to the PoSW defined in [23]. Note that the PoSW defined in both [37] and [23] are interactive, in which a statement χ is randomly sampled from the verifier and the prover constructs a proof based on the input statement χ.

5.1 The Definition of Non-Interactive PoSW

We first formally define the non-interactive PoSW in the random oracle model.

▶ **Definition 20** (Non-Interactive PoSW). *A* Non-Interactive Proof of Sequential Work *(*niPoSW*) consists of polynomial-time oracle algorithms $\Pi_{\mathsf{niPoSW}} = (\mathsf{Solve}, \mathsf{Verify})$ that use public parameters, as defined below.*

- ***Public Parameters.*** *Security parameter $\lambda \in \mathbb{N}$ and a random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$.*
- ***Solve.*** *Given a time parameter $T \in \mathbb{N}$, the statement χ, $\mathcal{P}$ computes a solution $\pi \leftarrow \mathsf{Solve}^{\mathcal{H}(\cdot)}(1^\lambda, T, \chi)$. The final proof is (χ, π).*
- ***Verify.*** *$\mathcal{P}$ can verify that the proof is genuine by running $\{0,1\} \leftarrow \mathsf{Verify}^{\mathcal{H}(\cdot)}(1^\lambda, T, \chi, \pi)$.*

We require the following properties:

(1) Correctness. For any $\chi \in \{0,1\}^\lambda$, $\lambda, T \in \mathbb{N}$ we have

$$\mathsf{Verify}^{\mathcal{H}(\cdot)}(1^\lambda, T, \chi, \mathsf{Solve}^{\mathcal{H}(\cdot)}(1^\lambda, T, \chi)) = 1.$$

That is, an honest prover should always produce a valid proof with probability 1, regardless of the choice of the statement $\chi \in \{0,1\}^*$, running in time parameter T and security parameter λ.

(2) Efficiency. Solve should run in time $T \cdot \mathsf{poly}(\lambda)$ and Verify should run in time $\mathsf{poly}(\lambda, \log T)$. Similarly, the solution π must have size $\mathsf{poly}(\lambda, \log T)$.

(3) Security. We say that a construction Π_{niPoSW} is $(T(\cdot), q(\cdot), \epsilon(\cdot))$-*secure (resp.* Π_{niPoSW} *is* $(T(\cdot), q(\cdot), \epsilon(\cdot))$-*quantum secure)* if any algorithm $\mathcal{A}$ running in sequential time at most $T = T(\lambda)$ in the pROM (resp. pqROM) and making at most $q = q(\lambda)$ total queries to the random oracle should fail to produce a valid proof for any statement $\chi \in \{0,1\}^\lambda$ (selected by the adversary) except with a negligible probability $\epsilon(\lambda)$, i.e., if $(\pi', \chi) \leftarrow \mathcal{A}^{\mathcal{H}(\cdot)}(1^\lambda, T)$ denotes the solution generated from $\mathcal{A}$, then

$$\Pr_{\mathcal{H}(\cdot)}\left[\mathsf{Verify}^{\mathcal{H}(\cdot)}(1^\lambda, T, \chi, \pi') = 1\right] \leq \epsilon(\lambda),$$

where the probability is taken over the randomness of the random oracle $\mathcal{H}$.

5.2 The Underlying DAG G_n^{PoSW} ([23])

We will use the same DAG in our construction of the non-interactive PoSW as the DAG defined in [23]. Here, we briefly recall their construction of the DAG G_n^{PoSW} (the figure is given in the full version). For $n \in \mathbb{N}$, let $N = 2^{n+1} - 1$ and first construct a complete binary tree $B_n = (V, E')$ of depth n, where $|V| = N$ and all the directed edges go from the leaves towards the root. We identify the N nodes $V = \{0,1\}^{\leq n}$ with the binary strings of length at most n except for the root, and we let the root be the empty string ε. Below the root we add directed edges from nodes 0 and 1 to node ε. Similarly, for each node v we add directed edges from nodes $(v\|0)$ and $(v\|1)$ to v. Here, $\|$ denotes the concatenation of the strings. Now we define the DAG $G_n^{\mathsf{PoSW}} = (V, E)$ by starting with $B_n = (V, E')$ and appending some edges as follows:

- For all leaf nodes $u \in \{0,1\}^n$, add an edge (v, u) for any v that is a left sibling of a node on the path from u to the root ε.

For example, for $u = 0110$, the path from u to the root is $0110 \to 011 \to 01 \to 0 \to \varepsilon$ and the left siblings of the nodes on this path are 010 and 00. Hence, we add the edges $(010, 0110)$ and $(00, 0110)$ to E'. We refer to [23] for the full description of G_n^{PoSW}.

5.3 The Non-Interactive Version of [23] Construction

Applying the Fiat-Shamir transform to the interactive PoSW construction [23], we have the following algorithms Solve and Verify in the non-interactive PoSW construction. For the notational simplicity, let $G_n = G_n^{\mathsf{PoSW}}$ be the underlying DAG from [23].

$\mathsf{Solve}^{\mathcal{H}(\cdot)}(1^\lambda, T, \chi, C)$:

- Compute the labels of the graph G_n with $n = 1 + \lceil \log T \rceil$, i.e., compute $\ell_i = \mathcal{H}_\chi(i, \ell_{p_1^i}, \ldots, \ell_{p_{d_i}^i}), 1 \leq i \leq N$, where $p_1^i, \ldots, p_{d_i}^i$ denotes the parents[4] of node i and $\mathcal{H}_\chi(\cdot) := \mathcal{H}(\chi, \cdot)$.
- Compute $R = \mathcal{H}_\chi(N+1, \ell_\varepsilon)$ and parse R to get $k = \lfloor \lambda/n \rfloor$ strings $c_1, \ldots, c_k \in \{0,1\}^n$. Compute the challenges $C = \{c_1, \ldots, c_k\}$ where each n-bit string c_i corresponds to a leaf node in G_n.
- Prove that everything on the path from node c_i to the root is locally consistent. This can be done by a Merkle tree reveal $\mathsf{MT.Reveal}$, which reveals the labels of all the siblings on path from node c_i to the root. More precisely, for a node $y \in \{0,1\}^{\leq n}$, we define

  $$\mathsf{MT.Reveal}(y) = \{\ell_{y[0\ldots j-1]\|(y[j]\oplus 1)}\}_{j=1}^{k},$$

  where we recall that $y[0\ldots j]$ denotes the substring of y to the j^{th} bit and $y[0\ldots 0]$ denotes the empty string. In this way, we can reveal the labels of all the siblings on path from x to the root ε. In particular, a solution π consists of the following:

  $$\pi = \{\ell_\varepsilon, c_i, \mathsf{MT.Reveal}(c_i) \text{ for } 1 \leq i \leq k\}.$$

- Output (χ, π).

$\mathsf{Verify}^{\mathcal{H}(\cdot)}(1^\lambda, T, \chi, \pi)$:

- Parse π to extract ℓ'_ε and $c'_1, \ldots, c'_k$. Set $R' = \mathcal{H}_\chi(N+1, \ell'_\varepsilon)$ and split R' to obtain $k = \lfloor \lambda/n \rfloor$ challenges $c''_1, \ldots, c''_k$ each of length n. Output 0 if $c''_i \neq c'_i$ for any $i \leq k$. Otherwise, let $(p_1^{i\prime}, \ldots, p_{d_i}^{i\prime}) = \mathsf{parents}(c'_i)$ for each i.

[4] Given a DAG G and a directed edge (u, v) we say that node u is a parent of node v. While this is the standard notion of parent in a DAG it can be counter-intuitive since the "tree" edges in our DAG are directed towards the root i.e., nodes 011 and 010 are both parents of node 01.

- Extract $\ell'_{c'_i}$ and $\ell'_{p^i_j{}'}$ from π for each $i \le k$ and $j \le d'_i$.
- Check that each leaf node c'_i is locally consistent. That is, one checks that $\ell'_{c'_i}$ is correctly computed from its parent labels:

$$\ell'_{c'_i} \stackrel{?}{=} \mathcal{H}_\chi(c'_i, \ell'_{p^i_1{}'}, \ldots, \ell_{p^i_{d_i}{}'}) \text{ where } (p^i_1{}', \ldots, p^i_{d_i}{}') = \mathsf{parents}(c'_i).$$

  (Note that in G_n all of c'_i's parents are siblings of nodes on the path from c'_i to the root ϵ).
- Check the Merkle tree openings for consistency. That is, for $i = n-1, n-2, \ldots, 0$, check that

$$\ell_{c'_i[0\ldots i]} := \mathcal{H}_\chi(c'_i[0\ldots i], \ell_{c'_i[0\ldots i]\|0}, \ell_{c'_i[0\ldots i]\|1}),$$

  and verify that the computed root $\ell_{c'_i[0\ldots 0]}$ is equal to $\ell_\varepsilon \in \pi$ that we received in the proof.

5.4 Security

We argue that for any fixed constant $\alpha > 0$ the non-interactive proof of sequential work outline above is $(T = (1-\alpha)N, q, \epsilon)$-secure for $\epsilon(\lambda) = 32q^2(1-\alpha)^{\lfloor \lambda/n \rfloor} + 2q^3 2^{-\lambda} + 64q^3(n+2)\lambda 2^{-\lambda} + 2\lfloor \lambda/n \rfloor (n+2)2^{-\lambda}$. We first define a set LUCKY_s of databases $\mathcal{D}$ in which $\mathcal{D}$ contains no collision or $\mathcal{H}$-sequence of length s, yet the dataset $\mathcal{D}$ contains a lucky Merkle tree that can be used to extract a proof of sequential work for some statement χ. We then argue that *any* attacker making q queries fails to measure such a lucky dataset $\mathcal{D}$ except with negligible probability. This is true even if the attacker is not restricted to run in sequential time s. Finally, to complete the argument we argue that any cheating attacker who produces a valid proof of sequential work can be converted into an algorithm that measures a dataset $\mathcal{D}$ that either (1) contains an $\mathcal{H}$-sequence of length s, (2) contains a collision or (3) is in LUCKY_s. Assuming that our attacker is sequentially bounded and makes at most q queries, the probability of any of these three outcomes must be negligible. Thus, the PoSW construction must be secure against any sequentially bounded attacker.

Coloring the Graph G^n_{PoSW}

Given a database $\mathcal{D}$, a statement χ, and a candidate PoSW solution $y = \ell_\epsilon$ we define an algorithm $\mathtt{ColoredMT}_{\mathcal{D}}(\chi, y)$ which returns a copy of the DAG G^{PoSW}_n in which each node is colored red or green. Intuitively, green nodes indicate that the corresponding labels are locally consistent with the database $\mathcal{D}$ while red nodes are locally inconsistent. If the PoSW solution ℓ_ϵ is entirely consistent with $\mathcal{D}$ then every node in G^n_{PoSW} will be colored green. On the other hand, if there is no entry of the form $(x, y) \in \mathcal{D}$ then the root node in G^n_{PoSW} would be colored red along with every other node below it. To define $\mathtt{ColoredMT}_{\mathcal{D}}(\chi, y)$ we use a recursive helper function $\mathtt{ColorSubTree}_{\mathcal{D}}$ which outputs a colored subgraph rooted at an intermediate node v. We briefly introduce how these algorithms work below and refer to the full version for the complete descriptions.

(1) The algorithm $\mathtt{ColorSubTree}_{\mathcal{D}}(\chi, v, x_v, y_v)$ generates a subset of nodes that consists of a Merkle subtree along with the coloring of each node in the set.

[5] Note that if we have another entry $(x'_1, y_1) \in \mathcal{D}$ satisfying $x'_1 = \chi\|1\|\ell_{10}\|\ell_{11}$ then we have a collision in the database and we do not have a parsing fail here. Similar argument holds for another parsing fail case in (3) as well. We will deal with the case that the database has collisions separately and assume that we do not have any collisions so that a unique Merkle subtree is generated as output.

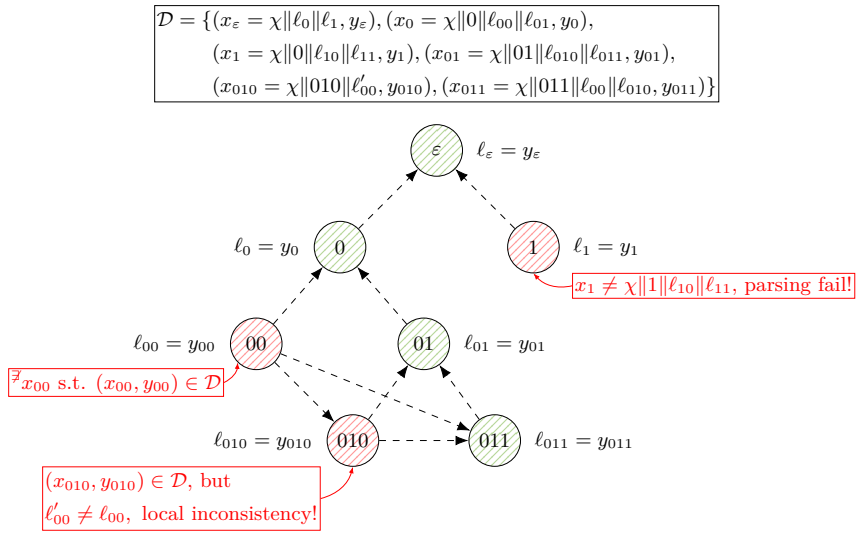

Figure 1 A succinct illustration of $\texttt{ColorSubTree}_{\mathcal{D}}(\chi, \varepsilon, x_\varepsilon, y_\varepsilon)$ where ε is an empty string and the database $\mathcal{D}$ is defined as above. Note that since $n = 3$, the nodes on the lowest layer are leaf nodes and the dashed edges are shown to help understand how the coloring works (we do not actually draw the edges in the algorithm). As described above, we have the following cases to color the node to red: (1) in node 1, there exists x_1 such that $(x_1, y_1) \in \mathcal{D}$, however, x_1 cannot be parsed properly, i.e., $x_1 \neq \chi\|1\|\ell_{10}\|\ell_{11}$ (parsing fail[5]), (2) in node 00, there is no x_{00} such that $(x_{00}, y_{00}) \in \mathcal{D}$ (undefined entry in $\mathcal{D}$), and (3) in node 010 – which is a leaf node – we have an entry $(x_{010}, y_{010}) \in \mathcal{D}$, but when parsing x_{010} into $\chi\|010\|\ell'_{00}$, we observe that the predefined label ℓ_{00} of node 00 and the value ℓ'_{00} does not match (local inconsistency).

- The algorithm takes as input (χ, v, x_v, y_v) where $\chi \in \{0,1\}^*$ is a statement, $v \in \{0,1\}^{\leq n}$ denotes a node in $G_{\mathcal{D}}$[6], and $x_v \in \{0,1\}^*, y_v \in \{0,1\}^\lambda$ are the bitstrings. Here, y_v is a potential candidate to be the label of node v. It outputs a subset of nodes $V' \subseteq V(G_{\mathcal{D}})$, which consists of a Merkle subtree with root node v and the corresponding coloring set $\texttt{Color}(V') := \{\texttt{Color}(v) : v \in V'\}$.
- Recall that a node v is *green* if it is locally consistent; for example, let $(x, y) \in \mathcal{D}$ and for node v with label $\ell_v = y$, if v_0, v_1 with labels y_0, y_1 are the parents of v then v is locally consistent if and only if it satisfies $\mathcal{H}_\chi(v, y_0, y_1) = y$. Since we satisfies $\mathcal{H}(x) = y$ as $(x, y) \in \mathcal{D}$, one would need to satisfy $x = \chi\|v\|y_0\|y_1$ for v to be locally consistent.
- Hence, we start parsing x_v into $\chi\|v'\|y_{v\|0}\|y_{v\|1}$ and see if it succeeds. That is, check if $v' = v$. If it fails, then we say it as "parsing fail", which is illustrated in Figure 1 (node 1). In this case, we color the node red and stop generating the subtree.
- If we succeed to parse x_v, then we color v to green and can proceed to its parents and see if there is an entry in the database $\mathcal{D}$ with having its y-coordinate as the label of its parent node. If it fails, then it becomes our second fail and we color the node red and stop generating the subtree. It is illustrated in Figure 1 (node 00).
- When we color the leaf nodes, we follow the same procedure except that we could have more than two parents based on the edge structure in [23], and we have another

[6] We remark that v corresponds to the identification of a node that is a binary string of length at most n, different from the label of the node ℓ_v.

possibility of "parsing fail" because the labels of its parents should be predefined by construction. If this kind of parsing fail occurs (node 010 in Figure 1), then we color the node to red.

(2) $\texttt{ColoredMT}_{\mathcal{D}}(\chi, y)$ generates a complete Merkle tree rooted at a node ε with label $\ell_\varepsilon = y$ and appends the edges as shown in [23]. The algorithm works simple; find an entry $(x, y) \in \mathcal{D}$ and call $\texttt{ColorSubTree}_{\mathcal{D}}(\chi, \varepsilon, x, y)$. Fill the missing nodes with label $\perp$ and color them all red. Then we add the edges as described in Section 5.2. If there is no such (x, y) in $\mathcal{D}$ then we abort the entire algorithm. We refer to Figure 2 for an example of running the algorithm.

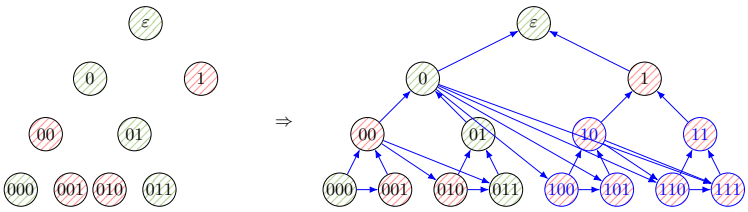

Figure 2 One example of $\texttt{ColoredMT}_{\mathcal{D}}(\chi, y)$ where $n = 3$ and $(x, y) \in \mathcal{D}$. On the left side is the output of $\texttt{ColorSubTree}_{\mathcal{D}}(\chi, \varepsilon, x, y)$ (different from Figure 1) and we fill the undefined nodes colored red and add the edges on the right side. Note that newly added nodes and edges are shown in blue.

Notations

Recall that in Definition 10, we defined PATH_s to be the set of the databases (compressed random oracles) $\mathcal{D}$ such that $G_{\mathcal{D}}$ contains a path of length s, which corresponds to the $\mathcal{H}$-sequence of length s. Now we define the set $\mathsf{COLLIDE}$ to be the set of the databases that contains a collision:

$$\mathsf{COLLIDE} := \{\mathcal{D} : \mathcal{D} \text{ contains pairs } (x, y), (x', y) \text{ such that } x \neq x'\}.$$

Given a node (string) $v = (v_1 \| \dots \| v_n) \in \{0,1\}^n$, we use $v_{\leq i} \in \{0,1\}^i$ to denote the substring $v_1 \| \dots \| v_i$, $v_{\leq 0} := \varepsilon$, and we use $\mathsf{PTR}(v, \chi) = \{v_{\leq i} \ : 0 \leq i \leq n\}$ to denote the set of all nodes on the direct path from v to the root of a Merkle tree constructed from $\texttt{ColoredMT}_{\mathcal{D}}(\chi, \cdot)$. Given a coloring of the Merkle tree, we define the predicate $\mathsf{gPTR}(v, \chi)$, which verifies that every node on the path from v to the root is green.[7] That is, $\mathsf{gPTR}(v, \chi) = 1$ if and only if $\texttt{Color}(v') = \text{green}$ for all $v' \in \mathsf{PTR}(v, \chi)$ and 0 otherwise. For example, in Figure 2, we have $\mathsf{gPTR}(011, \chi) = 1$ because the color of nodes in $\mathsf{PTR}(011, \chi) = \{011, 01, 0, \varepsilon\}$ are all green. On the other hand, we observe that $\mathsf{gPTR}(000, \chi) = 0$ despite the node 000 is green as we have an intermediate red node 00 in $\mathsf{PTR}(000, \chi)$.

Now we define $\mathsf{LUCKY}(\mathcal{D}, \chi, y)$ to be the set of λ-bit strings that produce lucky challenges for the Merkle tree rooted at y:

$$\mathsf{LUCKY}(\mathcal{D}, \chi, y) := \{w \in \{0,1\}^\lambda : w = w_1 \| \dots \| w_k \| z \text{ where } k = \lfloor \lambda / n \rfloor, \\ |w_i| = n, \text{ and } \mathsf{gPTR}(w_i, \chi) = 1 \ \forall 0 \leq i \leq k\}.$$

[7] Here, PTR stands for "Path To the Root" and gPTR stands for "green Path To the Root".

Then the set LUCKY_s is defined to be the set of databases that contains lucky challenges not in $\mathsf{COLLIDE}$ and PATH_s, i.e.,

$$\mathsf{LUCKY}_s := \{\mathcal{D} : \exists (x,y) \in \mathcal{D} \text{ s.t. } x = \chi \| N+1 \| \ell_\epsilon \ \wedge \ y \in \mathsf{LUCKY}(\mathcal{D}, \chi, \ell_\epsilon)\} \setminus (\mathsf{COLLIDE} \cup \mathsf{PATH}_s)\,.$$

We also define $\widetilde{\mathsf{LUCKY}}_s$ to be the set of basis states with $\mathcal{D}$ in LUCKY_s as follows:

$$\widetilde{\mathsf{LUCKY}}_s := \{|x,y,z\rangle \otimes |\mathcal{D}\rangle : \mathcal{D} \in \mathsf{LUCKY}_s\}.$$

We remark that when defining the set LUCKY_s, we need to assume that $\mathcal{D} \notin \mathsf{COLLIDE}$ to ensure that we get a unique Merkle tree rooted at ℓ_ε and we additionally need to assume that $\mathcal{D} \notin \mathsf{PATH}_s$ otherwise the set $\{v \in \{0,1\}^n : \mathsf{gPTR}(v,\chi) = 0\}$ may not be large, i.e., if all nodes are green.

We also define $\mathsf{PRE}(\mathcal{D})$ to be the set of λ-bit strings w that become a preimage of a hash value. It happens when w is a substring of x where $(x,y) \in \mathcal{D}$.

$$\mathsf{PRE}(\mathcal{D}) := \{w \in \{0,1\}^\lambda : \exists (x,y) \in \mathcal{D} \text{ s.t. } \mathsf{Substring}(w,x) = 1\}.$$

Security of PoSW against Quantum Attackers

Let $G_n = (V,E), \mathtt{Color}(V) \leftarrow \mathtt{ColoredMT}_{\mathcal{D}}(\chi,\cdot)$. We first introduce a helper lemma that is a classical argument (not quantum) and comes immediately from rephrasing the intermediate claim in the proof of [23, Theorem 1].

▶ **Lemma 21** ([23]). *If $\mathcal{D} \notin \mathsf{COLLIDE}$ and $\mathcal{D} \notin \mathsf{PATH}_T$ with $T = (1-\alpha)N$ for some constant $0 < \alpha < 1$, then*

$$|\{v \in \{0,1\}^n : \mathsf{gPTR}(v,\chi) = 0\}| \geq \alpha 2^n\ ,$$

i.e., at least $\alpha 2^n$ out of 2^n challenges (leaf nodes) in G_n must fail to respond correctly.

We immediately have the following corollary which bounds the number of tuples $(v_1,\ldots,v_k,y)$ such that all challenges v_i are lucky i.e., $\mathsf{gPTR}(v_i,\chi) = 1\ \forall i \leq k$. Here, $y \in \{0,1\}^{k'}$ is an auxiliary string.

▶ **Corollary 22.** *If $v_1,\ldots,v_k$ are the leaf nodes in G_n (i.e., $v_i \in V, |v_i| = n\ \forall i \leq k$), then we have that*

$$\left|\{(v_1,\ldots,v_k,y) : \mathsf{gPTR}(v_i,\chi) = 1\ \forall i \leq k,\, y \in \{0,1\}^{k'}\}\right| \leq 2^{nk+k'}(1-\alpha)^k.$$

▶ **Lemma 23.** *Let α be any constant satisfying $q \leq \frac{2^\lambda (1-\alpha)^{\lfloor \lambda/n \rfloor}}{(n+1)\lambda}$ then for any state*

$$|\phi\rangle = \sum_{x,y,z,\mathcal{D}:|\mathcal{D}|\leq q} \alpha_{x,y,z,\mathcal{D}} |x,y,z\rangle \otimes |\mathcal{D}\rangle$$

whose database register is a superposition of databases with at most q entries, we have

$$L_2(\mathsf{CPhsO}|\phi\rangle, \widetilde{\mathsf{LUCKY}}_s) - L_2(|\phi\rangle, \widetilde{\mathsf{LUCKY}}_s) \leq 4(1-\alpha)^{\frac{\lfloor \lambda/n \rfloor}{2}}.$$

Proof. (Sketch) The proof of Lemma 23 is similar to Lemma 12. We provide a complete proof in the full version and sketch the high level details here. We consider the projection of $|\phi'\rangle = \mathsf{CPhsO}|\phi\rangle$ onto orthogonal spaces P, Q, R, S where (1) P projects onto the span of basis states $|x, y, z\rangle \otimes |\mathcal{D}\rangle \in \widetilde{\mathsf{LUCKY}}_s$, (2) Q projects onto states $|x, y, z\rangle \otimes |\mathcal{D}\rangle$ such that $|x, y, z\rangle \otimes |\mathcal{D}\rangle \notin \widetilde{\mathsf{LUCKY}}_s$, $y \neq 0$, and $\mathcal{D}(x) = \perp$, (3) R projects onto states $|x, y, z\rangle \otimes |\mathcal{D}\rangle$ such that $|x, y, z\rangle \otimes |\mathcal{D}\rangle \notin \widetilde{\mathsf{LUCKY}}_s$, $y \neq 0$, and $\mathcal{D}(x) \neq \perp$, and (4) S projects onto states $|x, y, z\rangle \otimes |\mathcal{D}\rangle$ such that $|x, y, z\rangle \otimes |\mathcal{D}\rangle \notin \widetilde{\mathsf{LUCKY}}_s$ and $y = 0$. Then since P, Q, R, S project onto disjoint states that span the entirety of $|\phi'\rangle$ then we have $P + Q + R + S = \mathbb{I}$, where $\mathbb{I}$ denotes the identity operator. We analyze how P acts on these components separately and have that $\|P \circ \mathsf{CPhsO}(P|\phi\rangle)\|_2 \leq \||P|\phi\rangle\|_2$, $\|P \circ \mathsf{CPhsO}(Q|\phi\rangle)\|_2^2 \leq (1-\alpha)^{\lfloor \lambda/n \rfloor}$, $\|P \circ \mathsf{CPhsO}(R|\phi\rangle)\|_2^2 \leq 9(1-\alpha)^{\lfloor \lambda/n \rfloor}$., and $\|P \circ \mathsf{CPhsO}(S|\phi\rangle)\|_2 = 0$. Then by triangle inequality, we have that

$$\begin{aligned}
\|P \circ \mathsf{CPhsO}|\phi\rangle\|_2 &\leq \|P \circ \mathsf{CPhsO}(P|\phi\rangle)\|_2 + \|P \circ \mathsf{CPhsO}(Q|\phi\rangle)\|_2 \\
&\quad + \|P \circ \mathsf{CPhsO}(R|\phi\rangle)\|_2 + \|P \circ \mathsf{CPhsO}(S|\phi\rangle)\|_2 \\
&\leq \||P|\phi\rangle\|_2 + (1-\alpha)^{\frac{\lfloor \lambda/n \rfloor}{2}} + 3(1-\alpha)^{\frac{\lfloor \lambda/n \rfloor}{2}} \\
&\leq L_2(|\phi\rangle, \widetilde{\mathsf{LUCKY}}_s) + 4(1-\alpha)^{\frac{\lfloor \lambda/n \rfloor}{2}}.
\end{aligned}$$

Since we have that $\|P \circ \mathsf{CPhsO}|\phi\rangle\|_2 = L_2(\mathsf{CPhsO}|\phi\rangle, \widetilde{\mathsf{LUCKY}}_s)$, we complete the proof. ◀

From Lemma 23, we have the following Lemma. The proof of Lemma 24 can be found in the full version.

▶ **Lemma 24.** *Suppose that our quantum attacker $\mathcal{A}$ makes at most q queries to CPhsO then the probability p' of measuring a database $\mathcal{D} \in \mathsf{LUCKY}_s$ for $s = N(1-\alpha)$ is at most $16q^2(1-\alpha)^{\lfloor \lambda/n \rfloor}$.*

▶ **Theorem 4.** *Suppose $\mathcal{A}$ makes at most q quantum queries to our random oracle $\mathcal{H}$ over at most $s = N(1-\alpha)$ rounds and let p denote the probability that $\mathcal{A}$ outputs a valid (non-interactive) proof of sequential work. Then*

$$p \leq 32q^2(1-\alpha)^{\lfloor \lambda/n \rfloor} + \frac{2q^3}{2^\lambda} + \frac{64q^3(n+2)\lambda}{2^\lambda} + \frac{2\lfloor \lambda/n \rfloor (n+2)}{2^\lambda}.$$

Proof. Suppose that $\mathcal{A}$ make queries to a random oracle $\mathcal{H}$ and outputs tuples $((x_1, y_1), \ldots, (x_k, y_k), z)$ and let R be a collection of such tuples that contain a valid PoSW for some statement $\chi \in \{0,1\}^\lambda$. Recall that with probability p, the algorithm $\mathcal{A}$ outputs a tuple such that (1) the tuple is in R (contains a valid PoSW), and (2) $\mathcal{H}(x_i) = y_i$ for all i. Now consider running $\mathcal{A}$ with the oracle CPhsO (applying the Hadamard Transform before/after each query) and measuring the database $\mathcal{D}$ after $\mathcal{A}$ outputs. We first observe that if the final tuple is in R and $\mathcal{D}(x_i) = y_i$ for all i, then we must have $\mathcal{D} \in \mathsf{LUCKY}_{s+1} \cup \mathsf{PATH}_{s+1} \cup \mathsf{COLLIDE}$. In particular, if $\mathcal{D}$ does not contain an $\mathcal{H}$-sequence of length $s+1$ or a collision, then we must have $\mathcal{D} \in \mathsf{LUCKY}_{s+1}$ since the proof of sequential work is valid.

However, the probability of measuring a dataset $\mathcal{D} \in \mathsf{LUCKY}_{s+1} \cup \mathsf{PATH}_{s+1} \cup \mathsf{COLLIDE}$ can be upper bounded by $16q^2(1-\alpha)^{\lfloor \lambda/n \rfloor} + 32q^3(n+2)\lambda 2^{-\lambda} + q^3 2^{-\lambda}$ by applying Lemma 24 (Lucky Merkle Tree), Lemma 19 (Long $\mathcal{H}$-sequence), and Lemma 6 to upper bound the probability that $\mathcal{D} \in \mathsf{LUCKY}_{s+1}$, $\mathcal{D} \in \mathsf{PATH}_{s+1}$ and $\mathcal{D} \in \mathsf{COLLIDE}$, respectively.

We also observe that the number of input/output pairs in our PoSW is $k = \lfloor \lambda/n \rfloor (n+2)$ since we have $\lfloor \lambda/n \rfloor$ challenges where each challenge consists of a statement χ, a node itself, and at most n parents. Hence, by applying Lemma 5, we have that

$$\sqrt{p} \leq \sqrt{16q^2(1-\alpha)^{\lfloor \lambda/n \rfloor} + \frac{32q^3(n+2)\lambda}{2^\lambda} + \frac{q^3}{2^\lambda}} + \sqrt{\frac{\lfloor \lambda/n \rfloor (n+2)}{2^\lambda}},$$

which implies that

$$p \leq 32q^2(1-\alpha)^{\lfloor \lambda/n \rfloor} + \frac{2q^3}{2^\lambda} + \frac{64q^3(n+2)\lambda}{2^\lambda} + \frac{2\lfloor \lambda/n \rfloor (n+2)}{2^\lambda},$$

since $\sqrt{a} \leq \sqrt{b} + \sqrt{c}$ implies $a \leq b + c + 2\sqrt{bc} \leq 2(b+c)$ for any $a, b, c > 0$. ◀

6 Conclusion

We have shown that any attacker in the parallel quantum random oracle model making $q \ll 2^{\lambda/3}$ total queries cannot find an $\mathcal{H}$-sequence of length N in $N-1$ sequential rounds except with negligible probability. Using this result as a building block, we then prove that the non-interactive proof of sequential work of Cohen and Pietrzak [23] is secure against any attacker making $q \ll 2^{\lambda/n}$ queries and running in sequential time $T = (1-\alpha)N$. We leave it as an open question whether or not the λ/n term from this lower bound is inherent or whether the construction could be tweaked to establish security when $q \ll 2^{\lambda/n}$. The main technical hurdle is extracting more than λ/n challenges from a single random oracle output or adapting the proof to handle a modified construction where we extract challenges from multiple random oracle outputs. An alternative approach would be to introduce a second random oracle with longer outputs, which could be used to extract $\Omega(\lambda)$ challenges.

Our results also highlight the power of the recent compressed random oracle technique of Zhandry [42] and raises a natural question about whether or not these techniques could be extended to establish the security of important cryptographic primitives such as memory-hard functions or proofs of space in the quantum random oracle model. Alwen and Serbinenko [8] previously gave a classical pebbling reduction in the classical parallel random oracle model showing that the cumulative memory complexity of a data-independent memory hard function is tightly characterized by the pebbling complexity of the underlying graph. Would it be possible to establish the post-quantum security of memory hard functions through a similar reduction in the parallel quantum random oracle model?

References

1 Scott Aaronson. Quantum copy-protection and quantum money. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC*, pages 229–242, 2009.

2 Scott Aaronson and Paul Christiano. Quantum money from hidden subspaces. *Theory of Computing*, 9:349–401, 2013.

3 Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, pages 277–291, 2019.

4 Gorjan Alagic, Christian Majenz, Alexander Russell, and Fang Song. Quantum-access-secure message authentication via blind-unforgeability. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 788–817. Springer, Heidelberg, May 2020. PATHdoi:10.1007/978-3-030-45727-3_27.

5 Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 241–271. Springer, Heidelberg, August 2016. PATHdoi:10.1007/978-3-662-53008-5_9.

6 Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 3–32. Springer, Heidelberg, April / May 2017. PATHdoi:10.1007/978-3-319-56617-7_1.

7 Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 99–130. Springer, Heidelberg, April / May 2018. PATHdoi:10.1007/978-3-319-78375-8_4.

8 Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 595–603. ACM Press, June 2015. PATHdoi:10.1145/2746539.2746622.

9 Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 269–295. Springer, Heidelberg, August 2019. PATHdoi:10.1007/978-3-030-26951-7_10.

10 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh V. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

11 Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 61–90. Springer, Heidelberg, December 2019. PATHdoi:10.1007/978-3-030-36033-7_3.

12 Jeremiah Blocki, Benjamin Harsha, Siteng Kang, Seunghoon Lee, Lu Xing, and Samson Zhou. Data-independent memory hard functions: New attacks and stronger constructions. In *Advances in Cryptology - CRYPTO - 39th Annual International Cryptology Conference, Proceedings, Part II*, pages 573–607, 2019.

13 Jeremiah Blocki, Ling Ren, and Samson Zhou. Bandwidth-hard functions: Reductions and lower bounds. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 1820–1836, 2018.

14 Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of argon2i. In *Theory of Cryptography - 15th International Conference, TCC Proceedings, Part I*, pages 445–465, 2017.

15 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Proceedings, Part I*, pages 757–788, 2018.

16 Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011. PATHdoi:10.1007/978-3-642-25385-0_3.

17 Dan Boneh and Mark Zhandry. Secure signatures and chosen ciphertext security in a quantum computing world. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference. Proceedings, Part II*, pages 361–379, 2013.

18 Gilles Brassard, Peter Hoyer, Kassem Kalach, Marc Kaplan, Sophie Laplante, and Louis Salvail. Merkle puzzles in a quantum world. In *Advances in Cryptology - CRYPTO 2011. Proceedings*, pages 391–410, 2011.

19 Gilles Brassard and Louis Salvail. Quantum merkle puzzles. In *Second International Conference on Quantum, Nano, and Micro Technologies, ICQNM*, pages 76–79, 2008.

20 Ethan Cecchetti, Ben Fisch, Ian Miers, and Ari Juels. PIEs: Public incompressible encodings for decentralized storage. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1351–1367. ACM Press, November 2019. PATHdoi:10.1145/3319535.3354231.

21 Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 1–29. Springer, Heidelberg, December 2019. PATHdoi:10.1007/978-3-030-36033-7_1.

22 Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao. On the compressed-oracle technique, and post-quantum security of proofs of sequential work, 2020. *To appear at EUROCRYPT 2021*. PATHarXiv:2010.11658.

23 Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 451–467. Springer, Heidelberg, April / May 2018. PATHdoi:10.1007/978-3-319-78375-8_15.

24 Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 356–383. Springer, Heidelberg, August 2019. PATHdoi:10.1007/978-3-030-26951-7_13.

25 Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model, 2021. PATHarXiv:2103.03085.

26 Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, pages 292–323, 2019.

27 Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, August 2015. PATHdoi:10.1007/978-3-662-48000-7_29.

28 Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. PATHdoi:10.1007/3-540-47721-7_12.

29 Ben Fisch. Tight proofs of space and replication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 324–348. Springer, Heidelberg, May 2019. PATHdoi:10.1007/978-3-030-17656-3_12.

30 Lov K. Grover and J. Radhakrishnan. Quantum search for multiple items using parallel queries. *arXiv: Quantum Physics*, 2004.

31 Yassine Hamoudi and Frédéric Magniez. Quantum time-space tradeoff for finding multiple collision pairs, 2020. PATHarXiv:2002.08944.

32 Stacey Jeffery, Frederic Magniez, and Ronald de Wolf. Optimal parallel quantum query algorithms. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014*, pages 592–604, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

33 Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Heidelberg, April / May 2018. PATHdoi:10.1007/978-3-319-78372-7_18.

34 Qipeng Liu and Mark Zhandry. On finding quantum multi-collisions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 189–218. Springer, Heidelberg, May 2019. PATHdoi:10.1007/978-3-030-17659-4_7.

35 Qipeng Liu and Mark Zhandry. Revisiting post-quantum Fiat-Shamir. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 326–355. Springer, Heidelberg, August 2019. PATHdoi:10.1007/978-3-030-26951-7_12.

36 Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In *Advances in Cryptology - CRYPTO. Proceedings*, pages 39–50, 2011.

37 Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 373–388. ACM, January 2013. PATHdoi:10.1145/2422436.2422479.

38 Krzysztof Pietrzak. Proofs of catalytic space. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 59:1–59:25. LIPIcs, January 2019. PATHdoi:10.4230/LIPIcs.ITCS.2019.59.

39 Dominique Unruh. Revocable quantum timed-release encryption. *J. ACM*, 62(6), December 2015. PATHdoi:10.1145/2817206.

40 Christof Zalka. Grover's quantum searching algorithm is optimal. *Phys. Rev. A*, 60:2746–2751, October 1999. PATHdoi:10.1103/PhysRevA.60.2746.

41 Mark Zhandry. How to construct quantum random functions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 679–687, 2012.

42 Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 239–268. Springer, Heidelberg, August 2019. PATHdoi:10.1007/978-3-030-26951-7_9.