A Note on the Instantiability of the Quantum Random Oracle

Edward Eaton[1] and Fang Song[2]

[1] Cheriton School of Computer Science, University of Waterloo,
eeaton@uwaterloo.ca
[2] Department of Computer Science and Engineering, Texas A&M University,
fang.song@tamu.edu

Abstract. In a highly influential paper from fifteen years ago [10], Canetti, Goldreich, and Halevi showed a fundamental separation between the Random Oracle Model (ROM) and the standard model. They constructed a signature scheme which can be proven secure in the ROM, but is insecure when instantiated with any hash function (and thus insecure in the standard model). In 2011, Boneh et al. defined the notion of the *Quantum* Random Oracle Model (QROM), where queries to the random oracle may be made in quantum superposition. Because the QROM generalizes the ROM, a proof of security in the QROM is stronger than one in the ROM. This leaves open the possibility that security in the QROM could imply security in the standard model. In this work, we show that this is not the case, and that security in the QROM cannot imply standard-model security. We do this by showing that the original schemes that show a separation between the standard model and the ROM are also secure in the QROM. We consider two schemes that establish such a separation, one with length-restricted messages, and one without, and show both to be secure in the QROM. Our results give further understanding to the landscape of proofs in the ROM versus the QROM or standard model, and point towards the QROM and ROM being much closer to each other than either is to standard model security.

1 Introduction

In this note, we show that there exist digital signature schemes that can be proven secure against any poly-time quantum adversaries in the *quantum random-oracle model* [7], but they can be broken by a *classical* poly-time adversary when the random oracle is instantiated by any poly-time computable hash function family. This extends to the quantum setting the impossibility of instantiating a classical random oracle [3, 9, 10, 18, 26, 28].

Given the classical result (e.g., [10]) that there exists a secure signature scheme in the random oracle model but insecure under any efficient instantiation, the first doubt to clear up is probably why it does not immediately follow that a *quantum* random oracle cannot be instantiated as well. The reason is that the signature scheme in the classical result may as well get *broken* in the quantum random oracle model. In other words, all one needs to do is to prove quantum security of these classical constructions in the quantum random oracle model. This is exactly what this work does: we show that three examples in the classical setting [9, 10, 26] can be proven secure in the quantum random oracle model, and hence they demonstrate that the quantum random oracle model is *unsound* in general.

We dive into an overview of the proofs right away, so that those who are familiar with this subject can quickly digest the gist and walk away satisfied (or disappointed). If you are a more patient reader, you can come back here after enjoying the (more conventional) introduction.

Let us first review the classical examples [9, 10, 26] to be analyzed in the quantum random oracle model, and we present them under a unified framework which we hope will be easy to grasp. They all start with a secure signature scheme Σ and a function F, and Σ is "punctured" so that the signing algorithm would simply reveal the signing key when the function F is "non-random" (e.g., instantiated by a concrete hash function). To break it, an adversary just needs to convince the signing algorithm that F is indeed non-random. Therefore, it boils down to designing a proof system where a prover (adversary in the signature setting) proves "non-randomness" of a given function to a verifier (signing algorithm); whereas if the function is indeed random, no prover can fool the verifier to accept. The natural approach to such a proof system is based off the intuition that it is difficult to *predict* the output of a random oracle on an unknown input. The three classical examples nurture this intuition in two variations: predicting on a *single* input or *multiple* inputs.

1. The basic idea in [10] is to have the prover provide *an* input where the output is predictable and can be efficiently verified by the verifier. For starters, suppose we want to rule out a specific hash function H, then the prover can pick an arbitrary x and the verifier just checks if $F(x) = H(x)$. The verifier always accepts when F is instantiated by H, but accepts only with negligible probability if F is random. This immediately implies that for any function family, in particular the family of *poly-time computable* functions $\mathcal{H} = \{\mathcal{H}_\lambda = \{H_s\}_{s \in \{0,1\}^\lambda}\}$[3], we can construct a signature scheme following the idea above, where a (random) member in $\mathcal{H}$ is chosen as implementation of F, and the signing algorithm reveals the signing key whenever the "non-randomness" verification passes. Note that, nonetheless, the construction depends on the function family, which is *weaker* than the goal of establishing a signature scheme that is secure in the random oracle model, but insecure however when implement it from function family $\mathcal{H}$.

   Diagonalization comes in handy to reverse the quantifiers. The prover will provide a description s of a function H_s, which purportedly describes the function F. Then the verifier runs H_s on s and checks if it matches $F(s)$. Clearly, when F is implemented by a member $H_s \in \mathcal{H}$, the description s

[3] We assume a canonical encoding of functions into binary strings, under which s is the description of a function. Complexity is measured under security parameter λ.

is public (i.e., part of the verification key), and it is trivial for the prover to convince the verifier. Nonetheless, if F is a random oracle $\mathcal{O}$, the event $\mathcal{O}(s) = H_s(s)$ occurs only with negligible probability for any s that a prover might provide.

A technicality arises though due to the time complexity for computing $H_s(s)$ for all $\mathcal{H}$. Loosely speaking, we need a universal machine for the family $\mathcal{H}$ that on a description s computes $H_s(\cdot)$. Such a machine exists, but would require slightly super-polynomial time, which makes the verifier (i.e., signing algorithm) inefficient. This final piece of the puzzle is filled by CS-proofs introduced by Micali [27]. A CS-proof allows verifying the computation of a machine M, where the verifier spends significantly less time that the time to run M directly. This naturally applies to the problem here. Instead of running the universal machine to check $H_s(s) = F(s)$ by the verifier, the prover generates a CS-proof on the input $\langle M, s\rangle^F$ (relative to F) certifying the statement $M(s) := H_s(s) = F(s)$, which the verifier can check in polytime. When $F = \mathcal{O}$ is a random oracle, $\langle M, s\rangle^{\mathcal{O}}$ (relative to $\mathcal{O}$) is almost always a false statement, and the soundness of the CS-proof ensures that verifier will reject with high probability. Micali proved in general the soundness of CS-proofs in the random oracle model (to avoid confusion, in CS-proofs think of an random oracle independent of F).

2. Another strategy for proving "non-randomness", as employed in Maurer *et al.* [26] and Canneti *et al.* [9], is to predict on *multiple* inputs. This offers a direct *information-theoretical* analysis without relying on CS-proofs.

   In essence, a prover provides a machine π that allegedly predicts the output of F on sufficiently many inputs, and the verifier can run π and compare with the answers from F. This is easy for the prover when F is instantiated by $\mathcal{F}$ where the description s is given. On the other hand, by tuning the parameters, a counting argument would show that the randomness in a random oracle is overwhelming for any single machine (even inefficient ones!) to predict. Specifically, the "predicting" machine π needs to match with F on $q = 2|\pi| + \lambda$ inputs (i.e., the number of correct predictions has to be significantly more than the length of the description of the machine). Suppose that F is a random oracle $\mathcal{O} \leftarrow \{f : \{0,1\}^* \to \{0,1\}\}$ that outputs one bit (for the sake of simplicity), then for any π the probability that it will match $\mathcal{O}$ on q inputs is at most $2^{-(2|\pi|+\lambda)}$. A union bound on all machines of length n shows that p_n, the probability that some length-n machine is a good predictor, is at most $2^n \cdot 2^{-(2n+\lambda)} = 2^{-n-k}$. Another union bound shows that regardless of their length, no machine can be a good predictor, since $p := \sum_{n=1}^{\infty} p_n = 2^{-\lambda} \sum_n 2^{-n} \leq 2^{-\lambda-1}$ is negligible.

3. Both examples above suffer from an artifact. Namely the signature schemes need to be able to sign *long* messages or otherwise maintain *states* of prior signatures. This is rectified in [9], where a *stateless* scheme that signs only messages of polylogarithmic length is proven secure in the random oracle model but insecure under any instantiation.

   At the core of this construction is an *interactive* counterpart of the *non-interactive* proof system in part 2 above. It can be viewed as a *memory* delegation protocol, where a verifier with limited (e.g., poly-logarithmic) memory wants to check if the machine provided by the prover is a good predictor. Roughly speaking, it will execute the machine step by step and use the prover to bookkeep intermediate configurations of the machine. However, the configurations may be too long for the verifier to store and transit to the prover. Instead, the verifier employs a Merkle tree and only communicates an *authentication path* of the configuration with the prover. In particular, the verifier will memorize only a secret authentication key in between subsequent rounds. The security of the "punctured" signature scheme reduces to essentially a stronger *unforgeability* of a valid authentication path in a Merkle tree with respect to a random oracle, which is proven classically.

Proving security of separation examples in QRO. Once the constructions and classical analysis are clearly laid out, proving their security in the quantum random oracle model becomes more or less mechanic, given the techniques developed for QRO so far [1,2,16,33,34].

1^Q. (Example in [10] with CS-proofs.) Following the classical proof, we first show that the quantum security reduces to one of three cases: 1) hardness of a Grover-type search problem, which ensures that an adversary cannot feed the CS-proof a true statement in the case of a random oracle; 2) security of the original signature scheme; and 3) soundness of CS-proofs against quantum adversaries. A precise query lower bound for the search problem follows by standard techniques. And thanks to a recent work [11], CS-proofs are proven sound against quantum adversaries.

2^Q. (Example in [9,26] based on information-theoretical analysis.) It is easy to verify that the information-theoretical argument sketched above holds regardless of the kind of adversaries, and as a result the "punctured" signature scheme remains secure in the quantum random oracle model (and against quantum adversaries).

3^Q. (Example in [9] that only needs to sign short messages.) Our proof follows the classical one, where we first carefully verify and lift the reduction to the (stronger) unforgeability of Merkle tree against quantum adversaries, and then prove this property in the quantum random oracle model.

Specifically, we can model the unforgeability game as follows. Think of two correlated random oracles $\mathcal{O} : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$ and $\mathcal{O}' := \mathcal{O}(ak, \cdot)$ where ak is a random authentication key kept secret. Given *quantum* access to $\mathcal{O}$ and *classical* access to $\mathcal{O}'$, the adversary needs to come up with an authentication path $(\langle \sigma_1, \ldots, \sigma_d \rangle, \langle (v_{1,0}, v_{1,1}), \ldots, (v_{d,0}, v_{d,1}) \rangle, t)$ where $\sigma_i \in \{0,1\}$, $t = \mathcal{O}'(d, \mathcal{O}(0, v_{1,0}, v_{1,1}))$ and $v_{i,\sigma_i} = \mathcal{O}(i, v_{i+1,0}, v_{i+1,1})$ for every $i = 1, \ldots, d-1$. We prove that this is infeasible by reductions from a randomized decisional search problem and collision finding in random functions [8,21,35].

Background and motivation. The random oracle model, since its introduction [4], has proven a popular methodology for designing cryptographic schemes[4]. Basically a construction is first described and analyzed in an idealized setting where a random function is available as a black-box. To implement it in the real-world, one substitutes a cryptographic hash function for the random oracle. This methodology often leads to much more efficient schemes than alternatives. Examples include digital signatures by the Fiat-Shamir transform [17], hybrid public-key encryption following Fujisaki-Okamoto-type transforms [5], as well as succinct non-interactive zero-knowledge arguments that rise with the trending technology of blockchain and cryptocurrencies [31]. Its popularity is also attributed to the fact that one can often prove security in the random oracle model which is otherwise much more challenging or simply unknown.

It is, however, exactly the latter advantage that stirred considerable debate. *What does a security proof in the random oracle model mean?* To be pragmatic, a random-oracle proof at least serves as a sanity check that rules out inherent design flaws. Indeed, in practice most constructions that are instantiated from ones proven secure in the random oracle model have stood up extensive crypt-analysis. More formal pursuit, however, arrives at an irritating message. There are separation examples which show secure constructions in the random oracle model, but will be trivially broken whatever "nice" functions we use to instantiate the random oracle. Namely, the methodology is *unsound* in general. This does not mean all schemes following this approach are insecure. In fact, some random-oracle scheme can be instantiated under strong but reasonable assumptions and achieve desirable security in the real-world [24]. To say the least, a question mark lingers on schemes developed under this methodology.

Quantum computing adds another layer of complication to the issue here (and the overall landscape of cryptography). Because of the threats to widely deployed cryptosystems [32], a growing effort is undertaken to design and transition to so called *post-quantum cryptography* – a new set of cryptosystems that hopefully resist quantum attacks. In particular, the random oracle model has been re-examined in the presence of quantum adversaries. Since eventually a scheme (designed in RO) will be realized via a cryptographic function, whose specification is known in public, a quantum adversary can in principle construct a coherent quantum circuit that evaluates the hash function in quantum *superposition*. Consequently, when analyzing the scheme in the idealized setting, it seems necessary to grant quantum superposition queries to the random oracle by a quantum adversary. This brings about the quantum random oracle model [7]. The rationale is, very informally, good cryptographic functions are lacking structures for a quantum computer to exploit (aside from generic speedup due to quantum search), and hence realizing a scheme proven secure relative to a quantum random oracle this way is a fine practice.

Formally analyzing security in the quantum random oracle model turns out to be challenging. Many classical proof techniques, such as simulating and pro-

[4] According to Google Scholar, [4] has a citation count at 5089 (November 2019), which would be ranked top 20 at a (dated) list of most cited computer science papers [12].

gramming a random oracle on-the-fly or recording the queries, seem to fail due to unique features of quantum information. Thanks to a lot of continued effort, in recent years, researchers managed to develop various techniques for reasoning about the quantum random oracle model, and restored the security of many important constructions against quantum adversaries [2, 13, 14, 16, 20–22, 30, 33, 35, 36]. In fact, quantum random oracle model is becoming a booming research topic. To get an idea, as of writing, there are 166 citations in total of [7] and about 90 since 2018, of which about **60** came out since 2019. Also out of the 9 signature scheme submissions that made the second round of the post-quantum cryptography standardization at NIST [29], 5 of them involve the quantum random oracle. This just adds more at stake regarding "what does it mean that a scheme is proven secure in the *quantum* random oracle model?"

How to interpret this result? Our work show that in general, security in the quantum random oracle model could be vacuous in a real-world implementation. What is the real implication though? There seems a dilemma, probably more puzzling than the classical situation. On the one hand, since a quantum adversary is given more power (e.g., quantum computation and superposition access), security in the quantum random oracle provides more justification that the construction is solid. And this indeed explains the difficulty in establishing security in the quantum random oracle. Putting it in the converse way, it might occur that the classical separation examples will be broken in this model that is more "generous" to potentially more powerful quantum adversaries. Our work nonetheless shows otherwise, and it in fact reveals the other side of the dilemma. Any bit of success in restoring proof techniques in the quantum random oracle model just casts another bit of shadow on this methodology, since the seeming stronger quantum security does not promise the security of real-world implementations, not even their security against *classical* adversaries only.

On the other hand, researchers have pointed out that the security of well-designed schemes proven to be secure in the random oracle model have fared well in retrospect [25]. The use of the random oracle model to get a proof can allow for schemes that are simpler and more efficient than those in the standard model. While proofs in the quantum random oracle model are naturally more difficult, every year new techniques, more general and user-friendly, are developed to establish quantum random oracle model security [14, 19, 21, 36]. This has lead researchers to question what guarantees security in the quantum random oracle provides versus the classical random oracle model. In this line, our results can be taken as further justification that the difference between these models does not appear to be a large one, as their relation to the standard model is the same.

2 Background

2.1 The (Quantum Random) Oracle Model

The random oracle model, originally devised in [4], replaces any a cryptographic hash function with an entirely random oracle. The reduction algorithm is often allowed to manage this oracle, and can perform operations like looking up the queries that the adversary makes to it, or programming the oracle on inputs of interest. Using the random oracle model can often greatly simplify a proof or even enable a proof where otherwise not known or possible.

The intuitive idea behind the soundness of the random oracle methodology is that an adversary interacting with a scheme is unlikely to take advantage of the structure of the hash function. For most cryptographic schemes, even the adversary is likely to treat the hash function as a 'black box', and so by treating it as such, we can derive proofs for schemes that otherwise may not exist.

However, as pointed out in an influential paper by Boneh et al. [7], the random oracle model makes a fundamentally classical assumption about how an adversary interacts with the hash function (or random oracle). If we are concerned about an adversary who has access to a quantum computer, than we can assume that such an adversary is capable of instantiating the hash function on a quantum computer and making queries to it in *superposition*. Such behaviour is excluded by the random oracle model, and so when considering a quantum adversary, a more cautious approach for proofs is to consider the quantum random oracle model.

In the quantum random oracle model, the reduction algorithm still manages the oracle, but now the adversary must be allowed to make a superposition query to this oracle. For an oracle $\mathcal{O} : \mathcal{D} \to \mathcal{R}$, the reduction provides access to a unitary $U_{\mathcal{O}}$ which performs the action

$$U_{\mathcal{O}} : \sum_{x \in \mathcal{D}, y \in \mathcal{R}} \alpha_{x,y} |x\rangle|y\rangle \mapsto \sum_{x \in \mathcal{D}, y \in \mathcal{R}} \alpha_{x,y} |x\rangle|y \oplus \mathcal{O}(x)\rangle,$$

where the input must be a valid quantum state, i.e., the sum of the square amplitudes of the $\alpha_{x,y}$'s must be 1.

2.2 Notation

For clarity, we will denote a random oracle as $\mathcal{O}$, while actual instantiations of random oracles (e.g., typically hash functions) are denoted H. When describing a scheme where a function may replaced with a random oracle in the proof, or with a hash function in the real world, we will denote this function with F. The security parameter of a scheme is denoted by λ, while the output length of a hash function is denoted n. While we separate these two values for generality and expressiveness, throughout this work it is the case that $\lambda = n$.

2.3 Computationally Sound Proofs

Computationally sound proofs [27] are a construction introduced by Micali in 2000. They are a proof system, allowing a prover to prove knowledge of a witness for a language $\mathcal{L}$ in poly-logarithmic time. In our context, CS-proofs are useful for showing the validity of a computation without having to run the computation. Imagine a description of an arbitrary function f, which may take super-polynomial time to run on an input x, but will result in $f(x) = y$. A CS-proof system allows us to generate a proof π that $f(x) = y$. Even though f may take a super-polynomial amount of time to run, the CS-Proof verification system allows a verifier, on input of f, x, y, and π to verify that $f(x) = y$ in only poly-logarithmic time.

The soundness of CS-proofs was originally shown in the random oracle model. Until recently, the soundness of the construction in the quantum random oracle model was an open question. In a breakthrough result by Chiesa et al. [11], the soundness of CS-proofs in the quantum random oracle model was shown. The authors did this by showing some innovations that extend ideas from Zhandry [36].

For our purposes, a CS-proof system consists of two algorithms: $\mathsf{CSProve}$ and $\mathsf{CSVerify}$. Both algorithms implicitly take a security parameter λ. $\mathsf{CSProve}$ also takes in a function f and an input x, and returns a value y and a proof π. The $\mathsf{CSVerify}$ function takes in a function f, an input x, an output y, and a proof π. It returns either accept or reject, based on the validity of the proof. Crucially, the $\mathsf{CSVerify}$ function runs in time poly-log in the security parameter λ, and not in relation to the time it takes f to run.

The correctness property states that for an honestly generated proof π, the $\mathsf{CSVerify}$ function will always accept. The soundness property ensures that if $f(x) \neq y$, then it is computationally infeasible to find a proof π that will cause $\mathsf{CSVerify}(f, x, y, \pi)$ to return accept.

3 Instantiating Quantum Random Oracles

In this section we define three signature schemes, Σ_1, Σ_2, and Σ_3, such that:

- Σ_1 is secure in the quantum random oracle model, but insecure if that random oracle is instantiated with some specific hash function H.
- Σ_2 is secure in the QROM, but insecure when the random oracle is instantiated with any of a pre-defined set of hash functions $\{H_1, \ldots, H_m\}$.
- Σ_3 is secure in the QROM, but insecure if the random oracle is ever instantiated with *any* polynomial-time function.

These signature schemes lift the results in [10] to the quantum random oracle model. In all cases, the only assumption we require is that we have a signature scheme $\Sigma_0 = (\mathsf{KeyGen}_0, \mathsf{Sign}_0, \mathsf{Vrfy}_0)$ which is existentially unforgeable in the quantum random oracle model. Examples of schemes proven secure in the quantum random oracle model with no additional assumptions include the stateful LMS signatures [15] and the stateless SPHINCS+ framework [6] (both hash-based signatures). If one is willing to accept a computational assumption such as ring-LWE, many other signature schemes, including several of those under consideration in the NIST standardization process serve as examples [23].

3.1 Warm up — schemes Σ_1 and Σ_2

The first step in considering the instantiation of a random oracle is to consider instantiation with a single hash function, H. Then we can define the scheme Σ_1 as follows. Clearly this scheme satisfies the correctness property, as Σ_0 does.

Signature scheme Σ_1

- $\mathsf{KeyGen}_1(1^\lambda)$: Generate $(pk, sk) \leftarrow \mathsf{KeyGen}_0(1^\lambda)$, and return.
- $\mathsf{Sign}_1(sk, \mathsf{msg})$:
    - Compute $\sigma_0 = \mathsf{Sign}_0(sk, \mathsf{msg})$.
    - Check to see if $F(\mathsf{msg}) = H(\mathsf{msg})$. If so, return $\sigma_1 = \sigma_0 || sk$.
    - Otherwise, return $\sigma_1 = \sigma_0 || 0$.
- $\mathsf{Vrfy}_1(pk, \mathsf{msg}, \sigma_1)$:
    - Parse σ_1 as $\sigma_0 || x$.
    - Run $\mathsf{Vrfy}_0(pk, \mathsf{msg}, \sigma_0)$.

This scheme is eu-acma secure in the (quantum) random oracle model, where F is replaced with an oracle $\mathcal{O}$. This is intuitively because in this case, the security reduces to that of Σ_0 unless an adversary is able to find a msg such that $\mathcal{O}(\mathsf{msg}) = H(\mathsf{msg})$ (which occurs for every possible input with uniform and independent probability $1/2^n$).

Furthermore, this scheme is *insecure* if it is instantiated with H replacing the random oracle. Then the adversary is able to trivially break security, as the condition $H(\mathsf{msg}) = H(\mathsf{msg})$ is always satisfied and $\sigma_1 = \sigma_0 || sk$ will be returned for any message.

The next step is considering a finite collection of m hash functions, say $\mathcal{H} = \{H_1, H_2 \ldots, H_m\}$.

Then we can define Σ_2 similarly to Σ_1, but change the condition to first check if $\mathsf{msg} \in \{1, \ldots, m\}$ (in some encoding of the integers 1 through m) and if so, further check if $F(\mathsf{msg}) = H_{\mathsf{msg}}(\mathsf{msg})$.

The analysis in the (quantum) random oracle model is again fairly straightforward. For any random oracle $\mathcal{O}$, the probability that $\mathcal{O}(i)$ matches $H_i(i)$ for any of $i = 1$ to m is at most $m \cdot \frac{1}{2^n}$. When m is small (e.g., polynomially sized in λ), this is small enough that it is likely to not be possible that an adversary can make a query that provides them with sk. Even for a large m, each $i \in \{1, \ldots, m\}$ will have the property that $\mathcal{O}$ and H_i match with probability $1/2^n$, and so an adversary must perform an unstructured search to find such an i. Hence an adversary's ability to break Σ_2 in the (quantum) random oracle model reduces to their ability to break Σ_0.

However, as before, if F is actually replaced by any one of the H_i's, an adversary can easily break the scheme by querying i to the signing oracle.

3.2 Signature scheme Σ_3

Schemes Σ_1 and Σ_2 are only to gain an intuition for the full result, Σ_3, which is a signature scheme that is secure in the quantum random oracle model, but insecure when the oracle is instantiated with any polynomial-time function as the hash function. Following the strategy for Σ_2, we would like to fix some enumeration of all algorithms that one may use as a hash function, say $\mathcal{H} = \{H_1, H_2, \ldots\}$, with $H_i : \{0,1\}^* \to \{0,1\}^n$. Then as before, we would modify an eu-acma secure scheme Σ_0 to introduce a check in the signing algorithm to interpret msg as a non-negative integer, and check if $F(\mathsf{msg}) = H_{\mathsf{msg}}(\mathsf{msg})$. However, there are several issues that must be resolved to make this fully rigorous. Such a set of functions cannot simply be defined and used in the signature scheme, as the signature scheme requires that on input i, hash function H_i is actually run.

To fix this, we start with an enumeration of *all* algorithms, $\mathcal{A} = \{A_1, A_2, A_3, \dots\}$. We make no assumptions about this enumeration except that we can efficiently swap between the index i and some standard description of A_i. Changing between A_i and i should not be seen as a computational task to carry out, but rather a reinterpretation of the same data. Algorithms are encoded, using some standard encoding depending on the computational model, into bit strings, which can then easily be interpreted as an integer. To think of a construction that achieves this, it is helpful to think of quantum circuits. If we are working with l registers, then we can interpret the index i as a value in $\{0,1\}^*$ which specifies which gates are applied to which registers in what order. From a description of a quantum circuit, it is easy to convert this into a binary string, and then an index, and vice versa. To be reversible and match the format of a hash function, we can then consider all circuits that perform the mapping $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus A_i(x)\rangle$.

Note that not all of these algorithms necessarily run in polynomial-time in the security parameter. It is of course impossible to tell which algorithms will even terminate. We would like to assume that when a random oracle is instantiated, the function it is instantiated with will run in polynomial time in the security parameter. As well, these algorithms do not necessarily have the correct output length of n bits.

To fix this, we modify each algorithm in the following way: For each algorithm, stop after taking $n^{\log n}$ steps, and pad or truncate the output (in an arbitrary way) so that each algorithm always outputs n bits. The value $n^{\log n}$ is chosen so it bounds all polynomial-time algorithms. We enumerate our modified algorithms $\mathcal{H} = \{H_1, H_2, \dots\}$. Notice that any algorithm that is polynomial time, and outputs n bit binary strings is unmodified. So, any function that would be used as a hash function is not affected by this.

We can then make a first attempt at defining Σ_3. Given an eu-acma (in the quantum random oracle model) signature scheme Σ_0 and an enumeration of hash functions $\mathcal{H}$ as described above, we define Σ_3 as follows.

There is a very noticeable problem in this scheme. We bounded the run time of the H_i's by $n^{\log n}$, in order to make sure that we could leave *every* polynomial-time algorithm unaffected. However, any algorithm A_i that runs in $\geq n^{\log n}$ steps will be modified to run in $n^{\log n}$ steps. If a message msg is signed which corresponds to such an algorithm, the signer will have to evaluate $H_{\mathsf{msg}}(\mathsf{msg})$. This means that the signing algorithm *does not run in polynomial time in the security parameter*, and so it does not fit a valid definition of a signing algorithm.

To resolve this issue, CS-proofs are employed.

Rather than directly checking to see if $F(\mathsf{msg}) = H_{\mathsf{msg}}(\mathsf{msg})$, we can instead accept a CS-proof π that $F(\mathsf{msg}) = H_{\mathsf{msg}}(\mathsf{msg})$. This scheme is still trivial

Signature Scheme Σ_3, first attempt

- KeyGen_3: The key generation algorithm remains the same as in the original scheme Σ. Run $\mathsf{KeyGen}_0(1^\lambda)$ and return (pk, sk).
- Sign_3: On input of a message msg, and the secret key sk, do the following:
  - Compute $\sigma_0 \leftarrow \mathsf{Sign}_0(sk, \mathsf{msg})$.
  - Interpret msg as a non-negative integer. Compute $H_{\mathsf{msg}}(\mathsf{msg})$.
  - Check to see if $F(\mathsf{msg}) = H_{\mathsf{msg}}(\mathsf{msg})$.
  - If so, return signature $\sigma_3 = \sigma_0 || sk$. Otherwise, return $\sigma_3 = \sigma_0 || 0^{l_{sk}}$.
- Vrfy_3: On input of a message msg, a signature σ_3 and a public key pk, we parse σ_3 as $\sigma_0 || x$, where x is a (possibly all zero) string of length l_{sk}. Then compute and return $\mathsf{Vrfy}_0(pk, \mathsf{msg}, \sigma_0)$.

to break when F is instantiated, but we are now guaranteed that the signing algorithm always runs in polynomial time, no matter what is queried.

Signature Scheme Σ_3, correct with CS-proofs

- KeyGen_3: The key generation algorithm remains the same as in the original scheme Σ_0. Run $\mathsf{KeyGen}_0(1^\lambda)$ and return (pk, sk).
- Sign_3: On input of a message msg, and the secret key sk, do the following:
  - Compute $\sigma_0 \leftarrow \mathsf{Sign}_0(sk, \mathsf{msg})$.
  - Using some standard parsing rule, parse msg as $i || \pi$, an index i and a string π.
  - Run $\mathsf{CSVerify}$ to check if π is a CS-proof that $H_i(i) = F(i)$.
  - If so, return signature $\sigma_3 = \sigma_0 || sk$. Otherwise, return $\sigma_3 = \sigma_0 || 0^{l_{sk}}$.
- Vrfy_3: On input of a message msg, a signature σ_3 and a public key pk, we parse σ_3 as $\sigma_0 || x$, where x is a (possibly trivial) string of length l_{sk}. Then compute and return $\mathsf{Vrfy}_0(pk, \mathsf{msg}, \sigma_0)$.

This allows us to state the main theorem of our paper.

Theorem 1 (Security of Σ_3). *Let $g : \{0,1\}^* \to \{0,1\}$ be a random function such that for each x, $\Pr[g(x) = 1] = \frac{1}{2^n}$ and all outputs of the function are independent.*

Let $\mathcal{Q}$ be a quantum adversary capable of breaking the existential-unforgeability of Σ_3 with probability p in the quantum random oracle model. Then there exists a reduction algorithm $\mathcal{R}$ that, in slightly super-polynomial time, is capable of either breaking Σ_0, breaking the computational soundness of the CS-proof system, or finding an $x \in \{0,1\}^*$ such that $g(x) = 1$.

3.3 Proof of Theorem 1

To prove that Σ_3 is secure in the quantum random oracle model, we reduce its security to the adversary's ability to do one of three things:

- Break signature scheme Σ_0 in the quantum random oracle model in *slightly super-polynomial time.*
- Find a marked item with respect to a random oracle g.
- Break the computational soundness of a CS-proof in the quantum random oracle model.

The reduction algorithm has two main components: How it answers random oracle queries and how it answers signature queries.

For handling a random oracle, we will need to construct a pseudo-random function f that takes in two parameters: x and y. This function must satisfy that $f(x, y)$ is a uniform random element from the set $\{0,1\}^n \setminus \{y\}$. Such a function can be quickly constructed on a quantum accessible circuit by using $2q$-wise independent hash functions.

Then consider the following oracle:

$$\mathcal{O}(i) = \begin{cases} H_i(i) & \text{if } g(i) = 1 \\ f(i, H_i(i)) & \text{otherwise} \end{cases} \tag{1}$$

By creating the proper quantum-accessible circuits, we can create such a circuit that implements such an oracle in super-polynomial time. We will give the adversary $\mathcal{Q}$ access to this oracle.

We also need to show that the adversary cannot distinguish between this oracle and a truly random oracle. In fact, we can show something stronger than this, that this is in fact a truly random oracle. To see this, take any $y \in \{0,1\}^n$, and any $i \in \{0,1\}^*$ and consider $Pr[\mathcal{O}(i) = y]$.

$$\begin{aligned} &\Pr[\mathcal{O}(i) = y] \\ =&\Pr[g(i) = 1]\Pr[\mathcal{O}(i) = y | g(i) = 1] + \Pr[g(i) = 0]\Pr[\mathcal{O}(i) = y | g(i) = 0] \\ =&\frac{1}{2^n}\Pr[\mathcal{O}(i) = y | g(i) = 1] + \frac{2^n - 1}{2^n}\Pr[\mathcal{O}(i) = y | g(i) = 0]. \end{aligned}$$

Then note that

$$\Pr[\mathcal{O}(i) = y | g(i) = 1] = \begin{cases} 1 & \text{if } y = H_i(i) \\ 0 & \text{otherwise} \end{cases}$$

$$\Pr[\mathcal{O}(i) = y | g(i) = 0] = \begin{cases} 0 & \text{if } y = H_i(i) \\ \frac{1}{2^n - 1} & \text{otherwise} \end{cases}$$

In either case, putting these values into the equation gives that $\Pr[\mathcal{O}(i) = y] = \frac{1}{2^n}$. Furthermore, we can see that as long as g and H_i are each $2q$-wise independent, the overall hash function is $2q$-wise independent, and so we have that this gives us an oracle that is indistinguishable from a truly random one, even by a quantum adversary.

We next describe how the reduction algorithm $\mathcal{R}$ handles the signature queries. On input of a query msg, our reduction does the following:

- Parse msg as $i||\pi$, an index i and a string π.
- Run the CS-verification procedure, with π as the potential proof that $H_i(i) = \mathcal{O}(i)$. If it accepts, check if $H_i(i) = \mathcal{O}(i)$
  - If it is, then by construction, $g(i) = 1$ and we have successfully found such an i, and may stop.
  - If it isn't, then we have a CS-proof of a false fact, and may stop.
- If it did not accept, then query the challenger for a signature on msg under the scheme Sign_0 and return the signature $\sigma_0||0^{l_{sk}}$ to $\mathcal{Q}$.

If we never stop on any signature query, then eventually the adversary would submit a forgery $(\mathsf{msg}^*, \sigma_3^*)$, where msg^* was never submitted to the signing oracle. We may then parse σ_3^* as $\sigma_0^*||x$. If this forgery is accepted by the verification procedure Vrfy_3, then $\mathsf{msg}^*, \sigma_0^*$ will form a forgery with respect to Σ_0.

4 Signing short messages

In this section we describe the scheme that appears in [9] and argue that the proof of security that appears in that work translates to the quantum random oracle model. This scheme has the same restrictions as the one that appears in the previous section — we want a scheme that is secure in the quantum random oracle model, but insecure when the scheme is instantiated with any polynomial-time function. At a high level, this is accomplished in the same way as before. The signing algorithm will interpret all submitted messages as a potential description of a hash function, and check to see if this hash function matches the random oracle in such a way that proves that the random oracle is in fact, the hash function. The main distinction is that the signing algorithm will only accept messages of length poly-logarithmic in the security parameter. This means that the usage of CS-proofs is no longer a possibility. To overcome this, the authors of [9] devised a proof system for an NP-language in which the verifier need only accept *multiple, short* messages.

This proof system can then be turned into a signature scheme, and the adversary (who acts as the prover) will submit a proof that the random oracle is not random by making multiple signing queries. At first glance, it may seem that it is not hard to construct a proof system that can take multiple short messages — all we need to do is to take a proof system that requires one, large message and send that message in multiple rounds. However, such a strategy would require the verifier to be *stateful.* The verifier would need to "save" the messages that the prover sends them to be verified against future messages. When translated to the context of a signature scheme, this makes the signer stateful as well. To rule out stateless signature schemes as well, the verifier in the proof system devised in [9] needed to both accept only short messages and be *stateless.*

In this section we show that this proof system remains secure in the quantum random oracle model. To do this, we first restate the proof system as it appears in [9], and then discuss how it is used in a signature scheme similar to section 3. Finally, we show how the security of the system remains unchanged in the quantum random oracle model.

4.1 A stateless interactive proof system with short messages

As mentioned, the proof system introduced in [9] is an interactive proof system with the following goals:

- It must only require short messages, so that the signing algorithm only needs to accept short messages.
- It must be stateless so that the signing algorithm also is.
- It must be unconditionally secure in the (quantum) random oracle model, again so that the signature scheme may be as well.

At a high level, these goals are accomplished with the following strategy: the proof that the verifier needs to process is modelled as a Turing machine. The initial state to this Turing machine is "fed" to the verifier, one block at a time. Each time a block of the initial state is fed to the verifier, they authenticate the current configuration, and send an updated tag to the prover. This authentication tag is submitted to the verifier as part of each subsequent update.

Remember however, that the verifier is completely stateless. While we may describe this process as the verifier learning the configuration of the Turing machine, what is really happening is that the verifier is incrementally authenticating each part of the configuration, without ever knowing the whole state.

Once the initial state is "loaded" the prover then proceeds by having the verifier execute the Turing machine, one step at a time. The prover needs to tell the verifier the parts of the machine that they need to know, as well as the authentication tags for those parts. The verifier can then execute one step, update the authentication tags, and send these back to the prover so that they may repeat the process. Since the authentication tags are small (more on this later) and the prover only needs to communicate the parts of the Turing machine that are necessary to execute one step, the communication in each round is small. Because the authentication tags cannot be forged, the only way for the prover to get the Turing machine to be in an accepting state (authenticated by the verifier) is to have to walk the verifier through each step of the computation, having them authenticate the process along the way.

We now expand on this sketch, starting by describing the machine that the verifier will be executing to establish that the oracle is non-random.

Non-randomness machine $M^{\mathcal{O}}(1^k, \pi)$

```
– Input $\pi$ is interpreted as a description of a Turing machine. Let $n = |\pi|$.
– For $i \in \{1, \ldots, 2n + k\}$ :
    • $y_i = \mathcal{O}(i)$.
    • $z_i = \pi(i)$.
    • If the first bit of $y_i$ and $z_i$ disagree, return *reject*.
– Return *accept*.
```

The configuration is described in four tapes — the security parameter tape sp, the oracle query tape q, the oracle reply tape r, and the worktape w initially containing π. The security of M when $\mathcal{O}$ is a random oracle is shown in [9].

Lemma 1 ([9], Proposition 2). *If the oracle $\mathcal{O}$ is chosen uniformly, the probability that there exists a description of a Turing machine π such that $M^{\mathcal{O}}(1^k, \pi)$ returns* accept *is less than 2^{-k}.*

Note that this lemma refers to the existence of a Turing machine π. This property holds just as well when $\mathcal{O}$ is quantum-accessible.

To iteratively load and run the machine M, we need a mechanism for the verifier to authenticate the current state of the machine, which is described by the four work tapes (sp, q, r, w), the heads of the tapes $h_1, \ldots, h_4$, and the finite control $\mathcal{F}$. These eight values describe entirely the state of the machine M. Using some standard encoding method, they may be encoded as a binary string. It is this string, denoted c that the verifier will be authenticating.

Say the oracle $\mathcal{O}$ returns values in $\{0,1\}^n$. Then we will pad the string c to one of length $n \cdot 2^d$, where d is the smallest positive integer such that $n \cdot 2^d \geq |c|$. This allows us to construct a Merkle tree out of the string c, with the leaf nodes consisting of bit strings of length n, and the tree having height d. The Merkle tree is constructed out of the oracle $\mathcal{O}$ by setting, for level i of the tree, the value of each node to be $\mathcal{O}(i, \mathsf{left}, \mathsf{right})$, where left and right are the values (in $\{0,1\}^n$) of the two nodes in the tree directly below.

Note, in particular, that domain separation is used to separate the different levels, but not for the calculations *within* a level. This is done to speed up the process of creating a Merkle tree when the configuration c is homogeneous. For the parts of the work tapes that entirely blank (as they will be in the initial configuration), when converted into a binary string, and then a Merkle tree, their will be many repeated leaf values, which means that the entire tree can be constructed in time polynomial in the security parameter, k.

The verifier will possess an authentication key ak, which is used to authenticate the root of the Merkle tree as in a MAC scheme. The authentication tag for the tree is computed as $\mathcal{O}(d, ak, \mathsf{root})$. The loading and execution machine then proceeds as follows (Full details of this process are described in [9]).

1. The prover sends a message indicating that they wish to initialize the process. In response, the verifier loads up a blank configuration c in which the tapes are all empty, the heads are at a starting position, and the finite control is empty. They compute the root of the Merkle tree where this blank configuration forms the leaf nodes, and authenticate the root of the tree, sending the authentication tag back to the prover.
2. The prover loads the initial state of the machine M leaf-node-by-leaf-node. For any leaf node i they wish to update, they send a message to the verifier with the position they want to update, the Merkle tree verification path for that leaf node, the new value they want that position to take on, and the authentication tag for the most recent root node. The verifier uses the Merkle tree verification path to reconstruct the root node, which it verifies with the

authentication tag and its key ak. Once checked, the verifier produces an authentication tag for the tree with the desired update, by swapping out the leaf node value, computing the new resulting root node (again, by using the Merkle tree verification path) and constructing a tag for the root node.

3. When the initial state of M has been loaded, the prover can then get the verifier to begin executing M. To execute a step of M, the prover must send any leaf nodes involved in one step of the computation (e.g., the leaf node the header is pointed to, the values of the headers) and the Merkle tree verification paths for those leaves, as well as the authentication tag. The verifier computes one step of the Turing machine, recomputes the root node for the new state, and sends the authentication tag for the new state to the prover. If the machine reaches the accepting state, then the verifier accepts the state as valid.

We now proceed to prove a lifting of Proposition 4 in [9] to the quantum random oracle model.

Lemma 2. *Let ak be chosen uniformly at random in $\{0,1\}^n$, then for any prover $\mathcal{P}$ it holds that*

$$\Pr_{\mathcal{O},ak}\left[V^{\mathcal{O}}(1^k, ak) \to \mathsf{accept}\right] \leq O(q^3/2^n) \tag{2}$$

Where q is the number of (quantum) oracle queries made by $\mathcal{P}$.

Proof. As noted in Lemma 1, the probability over the randomness in $\mathcal{O}$ that there exists an accepting machine π is less than 2^{-k}. Assuming there does not exist such a machine, a dishonest prover must somehow manage to trick the verifier into reaching an accepting state. Because an accepting machine cannot be loaded into the configuration, it must be the case that some machine which should not accept was instead loaded, and the execution of this machine is then tampered with by the prover. To tamper with the execution of the machine, the adversary must, at some point, provide the verifier with a leaf node that was not in the configuration that was just authenticated.

In order to load in a falsified leaf node, the adversary must still submit a correct authentication tag. There are two cases: either the associated authentication tag was provided by the verifier, or it was not.

First we consider the case where the authentication tag was provided by the verifier. We consider the first time the adversary submits a leaf node that corresponds to an invalid machine configuration. We know that the authentication tag matches a previously issued one, but the corresponding leaf node was not part of how the previous authentication tag was generated. There are two possibilities for how this may happen. It may be the case that (i) at some point along the verification path we have values left, left', right, right' and i such that $\mathcal{O}(i, \mathsf{left}, \mathsf{right}) = \mathcal{O}(i, \mathsf{left}', \mathsf{right}')$. Of course, at most one of the left and right values can be equal. The other possibility is that (ii) the root values of the resulting Merkle trees are different, but we have a collision in the authentication tag: $\mathcal{O}(d, ak, \mathsf{root}) = \mathcal{O}(d, ak, \mathsf{root}')$.

Because an adversary who is able to break the soundness of the proof system must provide enough classical information to be able to construct a collision in the quantum random oracle $\mathcal{O}$, we can bound the success probability simply by the probability of being able to find such a collision. This can be asymptotically bounded by a $O(q^3/2^n)$ term.

The second case happens when the authentication tag for the invalid machine configuration was never previously issued. This means that the adversary was able to submit a tag t, a value d, and a Merkle tree path that leads to a value root such that $t = \mathcal{O}(d, ak, \mathsf{root})$ when the value t was never before returned by the verifier.

Intuitively, this is a structureless search problem on the part of the adversary: in order to provide a valid authentication tag, they must perform an unstructured search to find the ak value that causes authentication tags to accept. To formally show this, we provide a reduction from an adversary who can create a new authentication tag to one that can distinguish between to functions, g_0 and g_1, both mapping $\{0,1\}^n$ to $\{0,1\}$. The function g_0 simply returns 0 for all inputs x, whereas for g_1 there is precisely one random input on which g_1 will return 1.

We are given quantum access to one of the two functions, g_b, and asked to determine b. To do this, we first construct two independent quantum-accessible random oracles $\mathcal{O}_0$ and $\mathcal{O}_1$. We then construct the oracle $\mathcal{O}$ as follows:

$$\mathcal{O}(d, x, y) = \begin{cases} \mathcal{O}_0(d, x, y) & \text{if } g_b(x) = 0 \\ \mathcal{O}_1(d, y) & \text{if } g_b(x) = 1. \end{cases} \tag{3}$$

Note that since there is at most one x for which $g_b(x) = 1$, $\mathcal{O}$ is itself a random oracle, for anyone who does not have direct access to $\mathcal{O}_0$ and $\mathcal{O}_1$.

When the verifier needs to produce an authentication tag on an input (d, root), this can simply be done by computing $t = \mathcal{O}_1(d, \mathsf{root})$. Then note that these tags will be valid authentication tags with respect to some authentication key only when $b = 1$, in which case the valid authentication tag will be the value x such that $g(x) = 1$.

When $b = 1$, the authentication tags that are issued by the verifier will be correctly correlated with the oracle $\mathcal{O}$, but when $b = 0$, the tags will be entirely uncorrelated with $\mathcal{O}$. In this case, the adversary's ability to produce a forgery is bounded by a simple random guess, which corresponds to a probability of $1/2^n$. This is because in this case the oracle $\mathcal{O}$ can tell the adversary no information about correct authentication tags. To try and guess an authentication tag for a configuration (d, root^*) would mean trying to guess the value of $\mathcal{O}_1(d, \mathsf{root}^*)$ without ever having queried it (and having made no quantum queries to it). Any non-negligible difference in the success probability of the adversary $\mathcal{P}$ can be used in order to determine which function we are dealing with, and thus leads to a determination of the unknown bit b.

The probability of determining such a bit in q queries to g is bounded above by $O(q^2/2^n)$ from known result [21]. Note that each quantum query $\mathcal{P}$ makes to $\mathcal{O}$ corresponds to exactly one quantum query to g_b. Because the other case is bounded by a $O(q^3/2^n)$ term, we can drop this term entirely.

Signature Scheme Σ_4

- KeyGen_4: Because the verifier in the interactive proof system requires an authentication key, we sample one as $ak \overset{\$}{\leftarrow} \{0,1\}^n$. Obtain $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}_0(1^\lambda)$ and return $(pk_4, sk_4) = (pk_0, (sk_0, ak))$.
- Sign_4: On input of a message msg, and the secret key $sk_4 = (sk_0, ak)$, do the following:
  - Compute $\sigma_0 \leftarrow \mathsf{Sign}_0(sk, \mathsf{msg})$.
  - Using some standard parsing rule, parse msg as an input to the verifier $\mathcal{V}$.
  - Run $\mathcal{V}(ak; \mathsf{msg})$, obtaining output t, and whether the machine M reached the authenticating state.
  - If so, return signature $\sigma_4 = \sigma_0 || sk_0$. Otherwise, return $\sigma_3 = \sigma_0 || t$.
- Vrfy_4: On input of a message msg, a signature σ_4 and a public key $pk_4 = pk_0$, we parse σ_4 as $\sigma_0 || x$, where x is some string. Then compute and return $\mathsf{Vrfy}_0(pk_0, \mathsf{msg}, \sigma_0)$.

4.2 Signature scheme $\boldsymbol{\Sigma_4}$

With the interactive, stateless, short messaged proof system fleshed out, we can now discuss the signature scheme Σ_4, built out of this proof system.

Theorem 2 (Security of Σ_4). *Let $\mathcal{Q}$ be a quantum adversary capable of breaking the existential-unforgeability of Σ_4 with probability p in the quantum random oracle model, with q queries to the quantum random oracle $\mathcal{O}$. Then there exists a reduction algorithm $\mathcal{R}$ that, with probability (over $\mathcal{R}$ and $\mathcal{O}$) at least $p - O(q^3/2^n)$ is capable of either breaking Σ_0 or finding an $x, x' \in \{0,1\}^*$ such that $\mathcal{O}(x) = \mathcal{O}(x')$.*

Proof. It is easy to see that

$$\Pr[\mathcal{Q} \text{ wins eu-acma} \wedge \nexists \pi : M^{\mathcal{O}}(1^\lambda, \pi) \to \mathsf{accept}] \geq p - 2^{-\lambda},$$

where the probability is taken over the randomness in the oracle $\mathcal{O}$ and the randomness of the adversary, as well as whatever randomness is needed in the signature scheme Σ_0.

There are then two cases: either the adversary submits a signing query that causes the proof system to move into an accepting state, or they do not. If they do, then since we know there is not a π such that $M^{\mathcal{O}}(1^\lambda, \pi)$, the only way for an adversary to do this is to have found a collision in $\mathcal{O}$, which can be found by looking at the (classical) signing queries made by the adversary. We can bound the probability this happens by a $O(q/2^\lambda)$ term. Assuming that the adversary does not submit such a message, then whatever forgery is submitted by the adversary will work as a valid forgery to the signature scheme Σ_0.

References

1. Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In *Advances in Cryptology – CRYPTO 2019*, pages 269–295. Springer, 2019.

2. Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. Quantum attacks on classical proof systems: The hardness of quantum rewinding. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 474–483. IEEE, 2014.
3. Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *Advances in Cryptology – EUROCRYPT 2004*, pages 171–188. Springer, 2004.
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.
5. Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology – EUROCRYPT 1994*, pages 92–111. Springer, 1994.
6. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security – CCS 2019*, pages 2129–2146, 2019.
7. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *Advances in Cryptology – ASIACRYPT 2011*, pages 41–69. Springer, 2011.
8. Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *arXiv:quant-ph/9605034*, 1996.
9. Ran Canetti, Oded Goldreich, and Shai Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In *Theory of Cryptography Conference – TCC 2004*, pages 40–57, 2004.
10. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. A preliminary version appeared in STOC98.
11. Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. Cryptology ePrint Archive, Report 2019/834, 2019. ia.cr/2019/834.
12. CiteseerX. Most cited computer science citations, 2015. https://citeseerx.ist.psu.edu/stats/citations.
13. Jan Czajkowski, Christian Majenz, Christian Schaffner, and Sebastian Zur. Quantum lazy sampling and game-playing proofs for quantum indifferentiability. Cryptology ePrint Archive, Report 2019/428, 2019. ia.cr/2019/428.
14. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In *Advances in Cryptology - CRYPTO 2019*. Springer, 2019.
15. Edward Eaton. Leighton-micali hash-based signatures in the quantum random-oracle model. In *Selected Areas in Cryptography – SAC 2017*, pages 263–280, 2017.
16. Edward Eaton and Fang Song. Making existential-unforgeable signatures strongly unforgeable in the quantum random-oracle model. In *10th Conference on the Theory of Quantum Computation, Communication and Cryptography – TQC 2015*, volume 44 of *LIPIcs*, pages 147–162. Schloss Dagstuhl, 2015.
17. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO 1986*, pages 186–194. Springer, 1986.
18. Shafi Goldwasser and Yael Tauman Kalai. On the (in) security of the fiat-shamir paradigm. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 102–113. IEEE, 2003.

19. Sean Hallgren, Adam Smith, and Fang Song. Classical cryptographic protocols in a quantum world. In *Advances in Cryptology – CRYPTO 2011*, pages 411–428, 2011.
20. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In *Theory of Cryptography Conference – TCC 2017*, pages 341–371. Springer, 2017.
21. Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography – PKC 2016*, pages 387–416. Springer, 2016.
22. Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. Ind-cca-secure key encapsulation mechanism in the quantum random oracle model, revisited. In *Advances in Cryptology – CRYPTO 2018*, pages 96–125. Springer, 2018.
23. Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In *Advances in Cryptology – EUROCRYPT 2018*, pages 552–586, 2018.
24. Eike Kiltz, Adam ONeill, and Adam Smith. Instantiability of rsa-oaep under chosen-plaintext attack. *Journal of Cryptology*, 30(3):889–919, 2017.
25. Neal Koblitz and Alfred J. Menezes. The random oracle model: a twenty-year retrospective. *Des. Codes Cryptogr.*, 77(2-3):587–610, 2015.
26. Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Theory of cryptography conference – TCC 2004*, pages 21–39. Springer, 2004.
27. Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.
28. Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology – CRYPTO 2002*, pages 111–126. Springer, 2002.
29. NIST. Post-quantum cryptography, 2019. https://csrc.nist.gov/Projects/post-quantum-cryptography.
30. Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In *Advances in Cryptology – EUROCRYPT 2018*, pages 520–551. Springer, 2018.
31. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.
32. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.
33. Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Advances in Cryptology – EUROCRYPT 2015*, pages 755–784. Springer, 2015.
34. Mark Zhandry. How to construct quantum random functions. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 679–687. IEEE, 2012.
35. Mark Zhandry. A note on the quantum collision and set equality problems. *Quantum Information & Computation*, 15(7-8):557–567, 2015.
36. Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In *Advances in Cryptology – CRYPTO 2019*, pages 239–268. Springer, 2019.