Post-quantum security of cryptographic transformations in the random oracle model

Huang, Y.

Chapter 5

KEM Combiners via Split-key PRFs

5.1 Introduction

In the upcoming transition to post-quantum secure cryptographic standards, *combiners* play an important role. A combiner can be used to compile several cryptographic schemes into a new, hybrid scheme, which offers the same (or a similar) functionality, and so that the new scheme is secure as long as *at least one* of the original schemes is secure. For example, combining a well-established but quantum-insecure scheme with a believed-to-be quantum-secure (but less well studied) scheme then offers the best of both worlds: it offers security against quantum attacks, should there really be a quantum computer in the future, but it also offers some protection in case the latter scheme turns out to be insecure (or less secure than expected) even against classical attacks. In other words, using a combiner in this context ensures that we are not making things less secure by trying to aim for quantum security.

In [GHP18], Giacon, Heuer, and Poettering proposed a family of KEM combiners that works as follows. To produce the combined ciphertext $C := (c_1, \ldots, c_n)$, one concatenates all component ciphertexts $c_1, \ldots, c_n$ produced by the component KEMs, and to produce the combined session key $K := F(k_1, \ldots, k_n, C)$, one feeds the component session keys $k_1, \ldots, k_n$ and C into a key-derivation function F. They then show that instantiating F with a *split-key pseudorandom function* (skPRF) would preserve the standard IND-CCA security as a KEM combiner. Roughly speaking, an skPRF is a function $F(k_1, \ldots, k_n, x)$ with multiple keys $k_1, \ldots, k_n$ that behaves like a PRF if at least one of the keys k_i is kept secret.[1] A particularly efficient skPRF proposed

[1] For technical reasons, one typically restricts the PRF attacker from querying F with the same input x twice.

by [GHP18] is

$$F(k_1, \ldots, k_n, x) := H(g(k_1, \ldots, k_n), x) \ ,$$

where H is a cryptographic hash function, and where we assume $g(k_1, \ldots, k_n)$ has high min-entropy as long as one of the keys k_i is freshly chosen. The classical security of this skPRF has been proven in the ROM (i.e. modelling H as a random oracle) considering classical attackers, which implies the classical security of the aforementioned KEM combiner.

5.1.1 Our Contributions

Given its relevance for constructing hybrid schemes from both pre-quantum and post-quantum schemes, the post-quantum security of F, and hence that of the corresponding KEM combiner, has remained an important open question prior to our work [DFH22], on which this chapter is based.

Quantum security of a skPRF. In Section 5.4, we close this gap via showing that F is a secure skPRF, considering quantum attackers as well. More specifically, we show that for every attacker $\mathcal{A}$ making at most q_H queries to H and q_F queries to F (with one of the keys k_i sampled and kept secret at the beginning), $\mathcal{A}$ cannot distinguish F from a truly random function R chosen independently of H, with the following concrete bound on the advantage

$$\mathbf{Adv}_F^{\mathsf{skPRF}}(\mathcal{A}) := \left| \Pr\left[\, 1 \leftarrow \mathcal{A}^{F,H} \,\right] - \Pr\left[\, 1 \leftarrow \mathcal{A}^{R,H} \,\right] \right| \leq 4\sqrt{2q_F^2 q_H \epsilon} + 4\sqrt{2q_H^2 q_F \epsilon} \ ,$$

where ϵ is a parameter close to zero, depending on the function g. This confirms the quantum security of F as an skPRF. Consequently, the corresponding KEM combiner as described above and in [GHP18] is also secure against quantum attackers.

By default, such an attacker $\mathcal{A}$ can then choose *adaptively*, i.e., depending on answers to previous queries, at what point to query *which oracle*. This is in contrast to a *static* $\mathcal{A}$ that has a predefined order of when it queries which oracle.[2] In certain cases, proving security for a static attacker is easier than proving security for a full fledged adaptive attacker, or taking care of adaptivity (naively) results in an unnecessary blow-up in the error term (see later).

Our security proof crucially exploits a generic compiler that we introduce in Section 5.3 to reduce every such adaptive attacker to a static one. Namely, in spirit, our security proof is a typical hybrid proof, where we replace, one by one, the queries to F by queries to R; however, the crux is that for each hybrid, corresponding to a particular function query that is to be replaced, the closeness of the current to the previous hybrid depends on the number of hash queries *between the current and the previously replaced function query*. In case

[2] In either case, we allow $\mathcal{A}$ to decide adaptively *what input* to query, when having decided (adaptively or statically) on which oracle to query.

of an adaptive $\mathcal{A}$, *each* such "window" of hash queries between two function queries could be as large as the total number of hash queries in the worst case, giving rise to a huge multiplicative blow-up when using this naive bound. Instead, for a static $\mathcal{A}$, each such window is bounded by a fixed number, with the sum of these numbers being the total number of hash queries.

By means of our compiler, we can turn the possibly adaptive $\mathcal{A}$ into a static one (almost) for free, and this way avoid an unnecessary blow-up, respectively bypassing additional complications that arise by trying to avoid this blow-up by other means.

Our adaptive-to-static compiler. More generally, we now consider the attackers given query access to n oracles $\mathcal{O}_1, \ldots, \mathcal{O}_n$. In light of the above, it is desirable to have a generic compiler that transforms any adaptive attacker $\mathcal{A}$ into a static attacker $\bar{\mathcal{A}}$ that is equally successful in the attack. And there is actually a simple, naive solution for that. Indeed, let $\mathcal{A}$ be an arbitrary oracle algorithm that makes adaptive queries to n oracles $\mathcal{O}_1, \ldots, \mathcal{O}_n$, and consider the static oracle algorithm $\bar{\mathcal{A}}$ defined as follows: $\bar{\mathcal{A}}$ simply runs $\mathcal{A}$, and at every point in time when $\mathcal{A}$ makes a query to one of $\mathcal{O}_1, \ldots, \mathcal{O}_n$ (but due to the adaptivity it will only become clear at the time of the query *which* $\mathcal{O}_i$ is to be queried then), the algorithm $\bar{\mathcal{A}}$ makes n queries, one to every $\mathcal{O}_i$, and it relays $\mathcal{A}$'s query to the right oracle, while making dummy queries to the other oracles.

At first glance, this simple solution is not too bad. It certainly transforms any adaptive $\mathcal{A}$ into a static $\bar{\mathcal{A}}$ that will be equally successful, and the blow-up in the total query complexity is a factor n only, which is mild given that the typical case is $n = 2$. However, it turns out that in many situations, considering the blow-up in the total query complexity is not good enough.

For example, consider the case of an attacker against a public-key encryption scheme in the random oracle model. In this example, it is typically assumed that $\mathcal{A}$ may make many more queries to the random oracle than to the decryption oracle, i.e., $q_H \gg q_D$, where q_H and q_D are the numbers of queries to the two different oracles respectively. But then, applying the above simple compiler, $\bar{\mathcal{A}}$ makes the same number of queries to the random oracle and to the decryption oracle; namely $\bar{q}_H = \bar{q}_D = q_H + q_D$. Furthermore, the actual figure of merit, namely the advantage of an attacker $\bar{\mathcal{A}}$, is typically not (bounded by) a function of the total query complexity, but a function of the two respective query complexities q_H and q_D *individually*. For example, if one can show that the advantage of any *static* attacker $\bar{\mathcal{A}}$ with respective query complexities $\bar{q}_H$ and $\bar{q}_D$ is bounded by, say, $\bar{q}_H \bar{q}_D^2 \epsilon$ (for some small ϵ), then the above compiler gives a bound on the advantage of any *adaptive* attacker $\mathcal{A}$ with respective query complexities q_H and q_D of $q_H^3\epsilon + 2q_H^2 q_D \epsilon + q_H q_D^2 \epsilon + q_D^3 \epsilon$. If $q_H \gg q_D$ then this is significantly worse than $\approx q_H q_D^2 \epsilon$, which one might hope for given the bound for static $\bar{\mathcal{A}}$.

Our contribution in Section 5.3, is a compiler that transforms any *adaptive* oracle algorithm $\mathcal{A}$ that makes at most q_i queries to oracle $\mathcal{O}_i$ for $i = 1, \ldots, n$ into a *static* oracle algorithm $\bar{\mathcal{A}}$ that makes at most $\bar{q}_i = nq_i$ queries to oracle $\mathcal{O}_i$ for $i = 1, \ldots, n$. Thus, rather than controlling the blow-up in the total number of queries, we can control the blow-up in the number of queries for each oracle *individually*, yet still with the same factor n. Our result applies for *any* vector $\mathbf{q} = (q_1, \ldots, q_n) \in \mathbb{Z}^n_{\geq 0}$ and contains no hidden constants. Our compiler naturally depends on $\mathbf{q}$ (or, alternatively, needs $\mathbf{q}$ as input) but otherwise only requires straight-line black-box access to $\mathcal{A}$, and it preserves efficiency: the run time of $\bar{\mathcal{A}}$ is polynomial in $Q = q_1 + \cdots + q_n$, plus the time needed to run $\mathcal{A}$. Furthermore, the compiler is applicable to any classical or quantum oracle algorithm $\mathcal{A}$, where in the latter case the queries to the oracles $\mathcal{O}_1, \ldots, \mathcal{O}_n$ may be classical or quantum as well; however, the *choice* of the oracle for each query is assumed to be classical (so that individual query complexities are well defined).

In the above made-up example of a public-key encryption scheme with advantage bounded by $\bar{q}_H \bar{q}_D^2 \epsilon$ for any static $\bar{\mathcal{A}}$ with respective query complexities $\bar{q}_H$ and $\bar{q}_D$, we now get the bound $8 q_H q_D^2 \epsilon$ for any adaptive $\mathcal{A}$ with respective query complexities q_H and q_D.

Besides applying our adaptive-to-static compiler in the main contribution (i.e. for analyzing the skPRFs), we show the usefulness of our adaptive-to-static compiler in Section 5.3.4 by discussing two additional example results from the literature. One is the security proof by Alkim *et al.* [ABB+17] of the qTESLA signature scheme [ABB+20] in the quantum random oracle model; the other is the recent work by Alagic, Bai, Katz and Majenz [ABKM22] on the quantum security of the famous Even-Mansour cipher. In both these works, the adaptivity of the attacker was a serious obstacle and caused a significant overhead and additional complications in the proof. With our results, these complications could have been avoided without sacrificing much in the security loss (as would be the case with using a naive compiler).

Interestingly, all three example applications are in the realm of quantum security (of a classical scheme). This seems to suggest that the kind of adaptivity we consider here is not so much of a hurdle in the case of classical queries. Indeed, in that case, a typical argument works by inspecting the entire query transcript and identifying an event with the property that conditioned on this event, whatever needs to be shown holds *with certainty*, and then it remains to show that this event is very likely to occur. In the case of quantum queries, this kind of reasoning does not apply since one cannot "inspect" the query transcript anymore; instead, one then typically resorts to some sort of hybrid argument where queries are replaced one-by-one, and then adaptivity of the queries may — and sometimes does, as we discuss — form a serious obstacle.

5.2 Preliminaries

We consider oracle algorithms $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}$ that make queries to (possibly unspecified) oracles $\mathcal{O}_1,\ldots,\mathcal{O}_n$, see Fig. 5.1 (left). Sometimes, and in particular when the oracles are not specified, we just write $\mathcal{A}$ and leave it implicit that $\mathcal{A}$ makes oracle calls. We allow $\mathcal{A}$ to be classical or quantum, and in the latter case we may also allow the queries (to some of the oracles) to be quantum; however, the choice of *which* oracle is queried is always classical. For the purpose of our work, we may assume $\mathcal{A}$ to have no input; any potential input could be hardwired into $\mathcal{A}$. For a vector $\mathbf{q} = (q_1,\ldots,q_n) \in \mathbb{Z}^n_{\geq 0}$, we say that $\mathcal{A}$ is a $\mathbf{q}$-*query* oracle algorithm if it makes at most q_i queries to the oracle $\mathcal{O}_i$.

In general, such an oracle algorithm $\mathcal{A}$ may decide *adaptively* which oracle to query at what step, dependent on previous oracle responses. In contrast to this, a *static* oracle algorithm has an arbitrary but pre-defined order in querying the oracles.

Our goal will be to transform any *adaptive* oracle algorithm $\mathcal{A}$ into a *static* oracle algorithm $\bar{\mathcal{A}}$ that is functionally equivalent, while keeping the blow-up in query complexity for each individual oracle, i.e., the blow-up for each individual q_i, small. By *functionally equivalent* (for certain oracle instantiations) we mean the respective executions of $\mathcal{A}^{O_1,\ldots,O_n}$ and $\bar{\mathcal{A}}^{O_1,\ldots,O_n}$ give rise to the same output distribution *for all* (the considered) instantiations $O_1,\ldots,O_n$ of the oracles $\mathcal{O}_1,\ldots,\mathcal{O}_n$. In case of *quantum* oracle algorithms, we require the output state to be the same.

For this purpose, we declare that an *interactive oracle algorithm* $\mathcal{B}$ is an interactive algorithm with two distinct interaction interfaces, one for the interaction with $\mathcal{A}$ (we call this the *simulation interface*), and one for the oracle queries (we call this the *oracle interface*), see Fig. 5.1 (middle). For any oracle algorithm $\mathcal{A}$, we then denote by $\mathcal{B}[\mathcal{A}]$ the oracle algorithm that is obtained by composing $\mathcal{A}$ and $\mathcal{B}$ in the obvious way. In other words, $\mathcal{B}[\mathcal{A}]$ runs $\mathcal{A}$ and answers all of $\mathcal{A}$'s oracle queries using its simulation interface; furthermore, $\mathcal{B}[\mathcal{A}]$ outputs whatever $\mathcal{A}$ outputs at the end of this run of $\mathcal{A}$, see Fig. 5.1 (right).[3]

In contrast to $\mathcal{A}$ (where, for our purpose, any input could be hardwired), we explicitly allow an interactive oracle algorithm $\mathcal{B}$ to obtain an input. Indeed, our transformation, which turns any adaptive oracle algorithm $\mathcal{A}$ into a static oracle algorithm $\bar{\mathcal{A}}$, needs to "know" $\mathbf{q}$, i.e., the number of queries $\mathcal{A}$ makes to the different oracles. Thus, this will be provided in the form of an input to $\mathcal{B}$; for reasons to be clear, it be provided in unary, i.e., as $1^{\mathbf{q}} := (1^{q_1},\ldots,1^{q_n})$.

We stress that we do not put any computational restriction on the oracle algorithms $\mathcal{A}$ (beyond bounding the queries to the individual oracles); however, we do want our transformation to preserve efficiency. Therefore, we say that

[3]Note, we silently assume consistency between $\mathcal{A}$ and $\mathcal{B}$, i.e. $\mathcal{A}$ should send a message when $\mathcal{B}$ expects one and the format of these messages should match the format of the messages that $\mathcal{B}$ expects (and vice versa), so that the above composition makes sense. Should $\mathcal{B}$ encounter some inconsistency, it will abort.

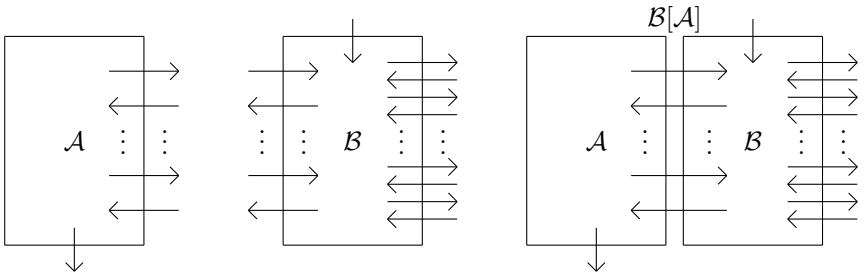

Figure 5.1: An oracle algorithm $\mathcal{A}$ (left), an interactive oracle algorithm $\mathcal{B}$ (middle), and the oracle algorithm $\mathcal{B}[\mathcal{A}]$ obtained by composing $\mathcal{A}$ and $\mathcal{B}$ (right).

an interactive oracle algorithm $\mathcal{B}$ is *polynomial-time* if the number of local computation steps it performs is bounded to be polynomial in its input size, and where we declare that copying an *incoming* message on the simulation interface to an *outgoing* message on the oracle interface, and vice versa, is unit cost (irrespectively of the size of the message). By providing $\mathbf{q}$ in unary, we thus ensure that $\mathcal{B}$ is polynomial-time in $q_1 + \cdots + q_n$.

5.3 A Generic Adaptive-to-static Compiler

5.3.1 Our Result

Let n be an arbitrary positive integer. We present here a generic adaptive-to-static compiler $\mathcal{B}$ that, on input a vector $\mathbf{q} \in \mathbb{Z}^n_{\geq 0}$, turns any *adaptive* $\mathbf{q}$-query oracle algorithm $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}$ into a *static* $n\mathbf{q}$-query algorithm.

Theorem 5.1. *There exists a polynomial-time interactive oracle algorithm $\mathcal{B}$, such that for any $\mathbf{q} \in \mathbb{Z}^n_{\geq 0}$ and any* adaptive $\mathbf{q}$*-query oracle algorithm $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}$, the oracle algorithm $\mathcal{B}[\mathcal{A}](1^{\mathbf{q}})$ is a* static $n\mathbf{q}$*-query oracle algorithm that is functionally equivalent to $\mathcal{A}$ for all* stateless *instantiations of the oracles $\mathcal{O}_1, \ldots, \mathcal{O}_n$.*

Remark 5.2. *As phrased, Theorem 5.1 applies to oracle algorithms $\mathcal{A}$ that have no input. This is merely for simplicity. In case of an oracle algorithm $\mathcal{A}$ that takes an input, we can simply apply the statement to the algorithm $\mathcal{A}(x)$ that has the input x hardwired, and so argue that Theorem 5.1 also applies in that case.*

Remark 5.3. *$\mathcal{B}[\mathcal{A}]$ is guaranteed to behave the same way as $\mathcal{A}$ for* stateless *(instantiations of the) oracles only. This is become most of the queries that $\mathcal{B}[\mathcal{A}]$ makes are actually dummy queries (i.e., queries on a default input and with the response ignored), which have no effect in case of stateless oracles,*

but may mess up things in case of stateful oracles. Theorem 5.1 extends to arbitrary stateful oracles if we allow $\mathcal{B}[\mathcal{A}]$ *to* skip *queries instead of making dummy queries (but the skipped queries would still count towards the query complexity).*

Given the vector $\mathbf{q} = (q_1, \ldots, q_n) \in \mathbb{Z}^n_{\geq 0}$, the core of the problem is to find a *fixed* sequence of $\mathcal{O}_i$'s in which each individual $\mathcal{O}_i$ occurs at most nq_i times, and so that *every* sequence of $\mathcal{O}_i$'s that contains each individual $\mathcal{O}_i$ at most q_i times can be embedded into the former. We consider and solve this abstract problem in the following section, and then we wrap up the proof of Theorem 5.1 in Section 5.3.3.

5.3.2 The Technical Core

Let Σ be an non-empty finite set of cardinality n. We refer to Σ as the *alphabet*. As is common, Σ^* denotes the set of finite strings over the alphabet Σ. In other words, the elements of Σ^* are the strings/sequences $s = (s_1, \ldots, s_\ell) \in \Sigma^\ell$ with arbitrary $\ell \in \mathbb{Z}_{\geq 0}$ (including $\ell = 0$).

Following standard terminology, for $s = (s_1, \ldots, s_\ell)$ and $s' = (s'_1, \ldots, s'_m)$ in Σ^*, the *concatenation* of s and s' is the string $s \| s' = (s_1, \ldots, s_\ell, s'_1, \ldots, s'_m)$, and s' is a *subsequence* of s, denoted $s' \sqsubseteq s$ if there exist integers $1 \leq j_1 < \ldots < j_m \leq \ell$ with $(s_{j_1}, \ldots, s_{j_m}) = (s'_1, \ldots, s'_m)$. Such an integer sequence $(j_1, \ldots, j_m)$ is then called an *embedding* of s' into s.[4]

Finally, for a function $q : \Sigma \to \mathbb{Z}_{\geq 0}, \sigma \mapsto q_\sigma$, we say that $s = (s_1, \ldots, s_\ell) \in \Sigma^*$ has *characteristic* (at most) q if $|\{i \text{ s.t. } s_i = \sigma\}| = q_\sigma$ $(\leq q_\sigma)$ for any $\sigma \in \Sigma$.

Lemma 5.4 (Embedding Lemma)**.** *Let* Σ *be an alphabet of size* n*, and let* $q : \Sigma \to \mathbb{Z}_{\geq 0}$*,* $\sigma \mapsto q_\sigma$*. Then, there exists a string* $s \in \Sigma^*$ *with characteristic* $n \cdot q : \sigma \mapsto n \cdot q_\sigma$ *such that any string* $s' \in \Sigma^*$ *with characteristic at most* q *is a subsequence of* s*, i.e.,* $s' \sqsubseteq s$*.*

The idea of the construction of the sequence s is quite simple: First, we evenly distribute $n \cdot q_\sigma$ copies of σ within the interval $(0, n]$ by "attaching" one copy of σ to every point in $(0, n]$ that is an integer multiple of $1/q_\sigma$ (see Fig. 5.2). Note that it may happen that different symbols are "attached" to the same point. Then, we walk along the interval from 0 and n and, one by one, collect the symbols we encounter in order to build up s' from left to right; in case we encounter a point with multiple symbols "attached" to it, we collect them in an arbitrary order.

It is then not too hard to convince yourself that this s indeed satisfies the claim. Namely, for any $s' = (s'_1, \ldots, s'_m)$ as considered, we can again walk along the interval from 0 and n, and we will then encounter all the symbols

[4] We use *string* and *sequence* interchangeably; however, following standard terminology, there is a difference between a *substring* and *subsequence*: namely, a substring is a subsequence that admits an embedding with $j_{i+1} = j_i + 1$.

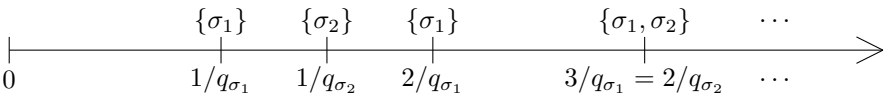

Figure 5.2: Constructing the string s by distributing the different symbols evenly within the interval $(0, n]$ (here with $3/q_{\sigma_1} = 2/q_{\sigma_2}$), and then collecting them from left to right.

of s', one by one: we will encounter the symbol s'_1 within the walk from 0 to $1/q_{s'_1}$, the symbol s'_2 then within the walk from $1/q_{s'_1}$ to $1/q_{s'_1} + 1/q_{s'_2}$, etc.[5]

Putting this idea into a formal proof is somewhat tedious, but in the end not too difficult. In order to formalize things properly, we generalize the standard notion of a sequence $s \in \Sigma^*$ in a way that allows us to talk about "attaching" a symbol to a point on $\mathbb{R}$, etc., in a rigorous way. Formally, we define a *line sequence* to be an arbitrary finite (possibly empty) subset $S \subseteq \mathbb{R} \times \Sigma$, i.e.,

$$S = \{(t_1, s_1), \ldots, (t_\ell, s_\ell)\} \in \mathcal{P}_{<\infty}(\mathbb{R} \times \Sigma)\,,$$

where w.l.o.g. we will always assume that $t_1 \leq \ldots \leq t_\ell$. We may think of the symbol s_i to "occur at the time" t_i.[6] For a subset $T \subset \mathbb{R}$, the set $\mathcal{P}_{<\infty}(T \times \Sigma)$ then obviously denotes the set of line sequences with $t_1, \ldots, t_\ell \in T$.

Assuming that the alphabet Σ is equipped with a total order $\leq$, any line sequence $S = \{(t_1, s_1), \ldots, (t_\ell, s_\ell)\}$ is naturally associated with the ordinary sequence

$$\pi(S) := (s_1, \ldots, s_\ell) \in \Sigma^*\,,$$

which is uniquely determined by the convention $t_1 \leq \ldots \leq t_\ell$ and insisting on $s_i \leq s_j$ whenever $t_i = t_j$ for $i < j$.

This *projection* $\pi : \mathcal{P}_{<\infty}(\mathbb{R} \times \Sigma) \to \Sigma^*$ preserves the characteristic of the sequence, i.e., if $s = (s_1, \ldots, s_\ell) = \pi(S)$ then

$$|\,\{t \text{ s.t. } (t, \sigma) \in S\}\,| = |\,\{i \text{ s.t. } s_i = \sigma\}\,| \tag{5.1}$$

for any $\sigma \in \Sigma$. Furthermore, for $T, T' \subset \mathbb{R}$ with $T < T'$ point-wise, and for $S \in \mathcal{P}_{<\infty}(T \times \Sigma)$ and $S' \in \mathcal{P}_{<\infty}(T' \times \Sigma)$, it is easy to see that $\pi(S \cup S') = \pi(S) \| \pi(S')$, from which it then follows that for ordinary sequences $s, s' \in \Sigma^*$

$$s \sqsubseteq \pi(S) \,\wedge\, s' \sqsubseteq \pi(S') \implies s \| s' \sqsubseteq \pi(S) \| \pi(S') = \pi(S \cup S')\,. \tag{5.2}$$

A final, simple observation, which follows directly from the definitions, is that

[5] To avoid the obvious issue of division by zero, we assume without loss of generality that each $q_\sigma > 0$.

[6] Note that we allow $t_i = t_j$ for $i \neq j$ while the definition prohibits $(t_i, s_i) = (t_j, s_j)$. If desired, one could allow the latter by letting S be a multi-set, but this is not necessary for us.

for $\sigma \in \Sigma$, i.e. a sequence of length $m = 1$, $\sigma \sqsubseteq \pi(S)$ holds if and only if there exists a time $t \in \mathbb{R}$ such that $(t, \sigma) \in S$.

Proof of Lemma 5.4. For any symbol $\sigma \in \Sigma$ let S_σ be a line sequence

$$S_\sigma := \left\{ \tfrac{1}{q_\sigma}, ..., \tfrac{nq_\sigma}{q_\sigma} \right\} \times \{\sigma\} \in \mathcal{P}_{<\infty}((0, n] \times \Sigma)\,,$$

and set $S := \bigcup_{\sigma \in \Sigma} S_\sigma$. We will show that $s := \pi(S)$ is as claimed.

The claim on the characteristic of s follows from the preservation of the characteristic under π, i.e. (5.1), and from $|\,\{t \text{ s.t. } (t, \sigma) \in S\}\,| = |\,S_\sigma\,| = n \cdot q_\sigma$, which holds by construction of S.

Let $s' = (s'_1, \ldots, s'_m) \in \Sigma^*$ be arbitrary with characteristic bounded by q. We consider the times $\tau_j := 1/q_{s'_1} + \cdots + 1/q_{s'_j}$ for $j \in \{1, \ldots, m\}$, and we let T_j be the interval

$$T_j := \big(\tau_{j-1}, \tau_j\big] = \big(\tau_{j-1}, \tau_{j-1} + \tfrac{1}{q_j'}\big] \subset \mathbb{R}\,,$$

and decompose $S = S_1 \cup \ldots \cup S_m$ with $S_j := S \cap (T_j \times \Sigma) \in \mathcal{P}_{<\infty}(T_j \times \Sigma)$. Here, we exploit that

$$\tau_m = \sum_{\sigma \in \Sigma} \frac{|\,\{i \text{ s.t. } s'_i = \sigma\}\,|}{q_\sigma} \leq \sum_{\sigma \in \Sigma} \frac{q_\sigma}{q_\sigma} = n\,,$$

and so the S_j's indeed cover all of $S \in \mathcal{P}_{<\infty}((0, n] \times \Sigma)$. Given that the interval $T_j \subset (0, n]$ has size $1/q_{s'_j}$, there exists a time $t_j \in T_j \cap \left\{ \tfrac{1}{q_\sigma}, ..., \tfrac{nq_\sigma}{q_\sigma} \right\}$. But then, $(t_j, s'_j) \in S_j$ by construction of S, and therefore $s'_j \sqsubseteq \pi(S_j)$. Finally, since $T_{j-1} < T_j$, property (5.2) implies that

$$s' = s'_1 \| \cdots \| s'_m \sqsubseteq \pi(S_1 \cup \ldots \cup S_m) = s$$

which was to be shown. □

While Lemma 5.4 above settles the existence question, the following two observations settle the corresponding efficiency questions. For concreteness, we assume $\Sigma = \{1, \ldots, n\}$ below, and thus can identify the function $q : \Sigma \to \mathbb{Z}_{\geq 0}$, $\sigma \mapsto q_\sigma$ with the vector $\mathbf{q} = (q_1, \ldots, q_n)$.

First, we observe that the line sequence S defined in the proof above, as well as its projection $s = \pi(S)$, can be computed in polynomial time in $q_1 + \cdots + q_n$; thus, we have the following.

Lemma 5.5. *There exists a polynomial-time algorithm that, on input $1^{\mathbf{q}}$, computes a string $s \in \Sigma^*$ as specified in the proof of Lemma 5.4.*

Furthermore, for any $s' \in \Sigma^*$ with characteristic at most q, for which we then know by Lemma 5.4 that s' can be embedded into s, the following ensures that this embedding can be computed efficiently and *on the fly*.

Lemma 5.6. *There exists a polynomial-time algorithm $\mathcal{E}$ such that for every string $s \in \Sigma^*$ and every subsequence $s' = (s'_1, \ldots, s'_m) \sqsubseteq s$, the following holds. Computing inductively $j_i \leftarrow \mathcal{E}(s, s'_i, j_{i-1})$ for every $i \in [m]$, where $j_0 := 0$, results in an increasing sequence $j_1 < \cdots < j_m$ with*

$$s' = (s_{j_1}, \ldots, s_{j_m}) \; .$$

The algorithm $\mathcal{E}$ simply follows the obvious greedy strategy: for each s'_i it looks for the next j_i for which $s'_i = s_{j_i}$. More formally:

Proof. The algorithm $\mathcal{E}(s, s'_i, j_{i-1})$ computes

$$j_i := \min \{k \in \mathbb{Z}_{>0} \,|\, j_{i-1} < k \leq m, s_k = s'_i\} \; . \tag{5.3}$$

It can be easily shown that the minimum is well-defined, i.e. taken over a non-empty set for each i by the assumption that s' is a subsequence of s, and thus by construction, every j_i is such that $s'_i = s_{j_i}$ while keeping $j_1 < \cdots < j_n$ increasing. This concludes the proof. □

5.3.3 Wrapping up the Proof of Theorem 5.1

The claimed interactive oracle algorithm $\mathcal{B}$ now works in the obvious way. On input $\mathbf{q}$ (provided in unary) and for any $\mathcal{A}$, $\mathcal{B}[\mathcal{A}]$ will make static oracle queries to $\mathcal{O}_{s_1}, \mathcal{O}_{s_2}, \ldots, \mathcal{O}_{s_{nQ}}$, where $s = (s_1, \ldots, s_{nQ}) \in \{1, \ldots, n\}^*$ is the string promised to exist by Lemma 5.4, with $Q = q_1 + \cdots + q_n$. In more detail, it first computes s using the algorithm from Lemma 5.5. Then, for the ith oracle query that $\mathcal{B}$ receives from $\mathcal{A}$ (starting with $i = 1$), and which consists of the identifier $s'_i \in \{1, \ldots, n\}$ of which oracle to query now and of the actual input to the oracle $\mathcal{O}_{s'_i}$, the algorithm $\mathcal{B}$ does the following: it computes $j_i \leftarrow \mathcal{E}(s, s'_i, j_{i-1})$ using the algorithm from Lemma 5.6, makes dummy queries to $\mathcal{O}_{s_{j_{i-1}+1}}, \ldots, \mathcal{O}_{s_{j_i-1}}$, and forwards $\mathcal{A}$'s query input to $\mathcal{O}_{s_{j_i}} = \mathcal{O}_{s'_i}$. The fact that $(j_1, \ldots, j_Q)$ computed this way forms an embedding of $s' = (s'_1, \ldots, s'_Q)$ into s ensures that $\mathcal{B}$ is able to forward all the queries that $\mathcal{A}$ makes to the right oracle, and so $\mathcal{A}$ will produce its output as in an ordinary run with direct adaptive access to the oracles.

5.3.4 Applications

To demonstrate the usefulness of our adaptive-to-static compiler, we briefly discuss three results from the literature. For two of them, the adaptivity of the attacker was explicitly declared as an obstacle in the security proof, and dealing with it complicated the proof substantially. These complications could be avoided/removed by means of our adaptive-to-static compiler. For the third one, we can immediately strengthen one of the results, which is restricted to

hold for static multi-oracle adversaries, by dropping this restriction via our compiler.

Quantum security of qTESLA. Our first application is in the context of qTESLA [ABB+20], which is a signature scheme that made it into the second round of the NIST post-quantum competition. Its security is based on the Ring-LWE problem, to which the authors of [ABB+17] give a reduction in the quantum random oracle model (QROM).[7] In the reduction, which starts from the security notion of *Unforgeability under Chosen Message Attack* (UF-CMA), the adversary can query a random oracle H as well as a signing oracle, where the order of oracle queries may be adaptive.

The reduction strategy of [ABB+17] applies only to a static adversary, with a fixed query pattern. Thus, the authors first compile the adaptive into a naive static attacker by letting it do q_H (the number of H-queries of the original adaptive adversary) H-queries between any two signing queries. Leaving it with this would blow up the number of H-queries to $q_S q_H$. In order to avoid that, they give the attacker a "*live-switch*", meaning that each query to H may be in superposition of making the query and not making the query, and the total "*query magnitude*" on actual H-queries is still restricted to q_H. Not so surprising, adding even more "quantumness" to the problem in this way, makes the analysis more complicated (compared to using standard "all-or-nothing" static queries and a standard classical bound on the query complexity), but it allows the authors to avoid the above blow-up in the (classical) query complexity to transpire into the security loss. The overall loss they obtain in the end is $O((q_S q_H^2 + q_S^3 + q_S^2 q_H) \cdot \epsilon)$ for small ϵ determined by the parameters of the scheme.

Since the security reduction in [ABB+17] intertwines the adaptive to static hurdle with other aspects of the proof, we cannot simply insert our Theorem 5.1 and then continue the proof as is. Still, by applying our result, we could obtain a static adversary with almost no cost in the number of H-queries, avoiding the need for the rather complicated "live-switch superposition" attacker, thus simplifying the overall proof significantly. Furthermore, looking ahead at Section 5.4, our result allows us to obtain the much better $O(\sqrt{q_O q_H^2 \epsilon} + \sqrt{q_O^2 q_H \epsilon})$ loss in a similar context — similar in the sense that it also involves two oracles where one reprograms the other at some high-entropy input. The adaptive to static reduction there allows us to apply some additional QROM tools that could potentially also be applied in the setting of qTESLA to improve the bound. However, actually doing this would require us to rewrite the entire proof of [ABB+17], which we consider outside the scope of this work.

[7] We note that some versions of qTESLA have been broken [LS19], but the attack only applies to an optimized variant that was developed for the NIST-competition, and does not apply to the scheme in [ABB+17] that we discuss here.

Quantum security of the function FX. Our second application is to [JST21], where the post-quantum security of the FX key-length extension is studied (which is a generalization of the Even-Mansour cipher). In a first part, post-quantum security of FX is shown under the restriction that the inputs to the queries are fixed in advance. In a second part, towards avoiding this restriction, the authors consider a variation of the FX construction, which they call FFX (for "function FX"), and they show in their Theorem 3 post-quantum security of FFX under the restriction that the attacker is "*order consistent*", as they call it in [JST21], which is precisely our notion of a *static* multi-oracle algorithm. Thus, by a direct application of our Theorem 5.1, this restriction can be dropped (almost) for free, i.e., with a small constant blow-up on the attackers advantage.

Quantum security of the Even-Mansour cipher. The work [ABKM22] shows full post-quantum security of the (unmodified) Even-Mansour cipher. As in the case of qTESLA, the fact that the attacker can choose adaptively whether to query the public permutation of the cipher complicates the proof. Indeed, as is explained on the 4th page in [ABKM22], this adaptivity issue forces the authors to extend the blinding lemma of Alagic *et al.* to a variant that gives a bound in terms of the *expected* number of queries. While the authors succeed in providing such an extended version of the blinding lemma (Lemma 3 in [ABKM22]), it further increases the complexity of an already involved proof.[8]

Thus, again, our Theorem 5.1 could be used to simplify the given proof by bypassing the complications that arise due to the attacker choosing adaptively which oracle to query at what point.

5.4 Quantum Security of a Split-key PRF

5.4.1 Hybrid Security and skPRFs

A *split-key pseudorandom function* (skPRF), as introduced in [GHP18], is a polynomial-time computable function $F : \mathcal{K}_1 \times \cdots \times \mathcal{K}_n \times \mathcal{X} \to \mathcal{Y}$ that is a pseudorandom function (PRF) in the standard sense *for every* $i \in [n]$ when considered as a keyed function with key space $\mathcal{K}_i$ and message space $\mathcal{K}_1 \times \cdots \times \mathcal{K}_{i-1} \times \mathcal{K}_{i+1} \times \cdots \times \mathcal{K}_n \times \mathcal{X}$, with the additional restriction that the distinguisher $\mathcal{A}$ (in the standard PRF security definition) must use a fresh $x \in \mathcal{X}$ in every query $(k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n, x)$.

[8] To be fully precise, Lemma 3 in [ABKM22] also generalizes the original blinding lemma in a different direction by allowing to reprogram to an arbitrary value instead of a uniformly random one; however, this generalization comes for free in that the original proof still applies up to obvious changes, while allowing an expected number of queries, which is needed to deal with the adaptivity issue, requires a new proof.

This restriction on the PRF distinguisher may look artificial, but is motivated by this definition of a skPRF being good enough for the intended purpose of a skPRF, namely to give rise to a *secure KEM combiner*. Indeed, [GHP18] shows that the naturally combined KEM, obtained by concatenating the individual ciphertexts to $C = (c_1, \ldots, c_n)$, and combining the individual session keys $k_1, \ldots, k_n$ using the above mentioned skPRF as

$$K = F(k_1, \ldots, k_n, C)\,,$$

is IND-CCA secure if at least one of the individual KEM's is IND-CCA secure.

The paper [GHP18] also proposes a particularly efficient hash-based construction, given by

$$F(k_1, \ldots, k_n, x) := H(g(k_1, \ldots, k_n), x) \tag{5.4}$$

where $g : \mathcal{K}_1 \times \cdots \times \mathcal{K}_n \to \mathcal{W}$ is a polynomial-time mapping with the property that, for some small ϵ,

$$\Pr_{k_i \leftarrow \mathcal{K}_i}[g(k_1, \ldots, k_n) = w] \leq \epsilon\,, \tag{5.5}$$

for every $i \in [n]$ and for every $k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n$ and every w; furthermore, $H : \mathcal{W} \to \mathcal{Y}$ is a cryptographic hash function. Simple choices for the function g are $g(k_1, \ldots, k_n) = (k_1, \ldots, k_n)$ and $g(k_1, \ldots, k_n) = k_1 + \cdots + k_n$.

It is shown in [GHP18] that this construction is a skPRF when H is modelled as a random oracle; indeed, it is shown that the distinguishing advantage is upper-bounded by $q_H\epsilon$, where q_H is the number of queries to the random oracle H.

Given the natural use of combiners in the context of the upcoming transition to post-quantum cryptography, it is natural — and well-motivated — to ask whether F can be proven to be a skPRF in the presence of a *quantum attacker*, i.e., when H is modeled as a *quantum* random oracle. Below, we answer this in the affirmative.

5.4.2 Quantum-security of the skPRF

The goal of this section is to show the security of the skPRF (5.4) in the quantum random oracle model. In essence, this requires proving that F is a PRF (in the quantum random oracle model) with respect to *any* of the k_i's being the key, subject to the restriction of asking a fresh x in each query.

To simplify the notation, we fix the index $i \in [n]$ and simply write k for k_i and x for $(k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n, x)$, and we abstract away the properties of the function g as follows. We let

$$F(k, x) := H(h(k, x))\,,$$

where $h : \mathcal{K} \times \mathcal{X} \to \mathcal{W}$ is an arbitrary function with the property that, for some parameter $\epsilon > 0$,

$$\Pr_{k \leftarrow \mathcal{K}}[h(k, x) = w] \leq \epsilon \tag{5.6}$$

for all $w \in \mathcal{W}$ and $x \in \mathcal{X}$. Furthermore, in the PRF security game, we restrict the attacker/distinguisher $\mathcal{A}$ to queries x with a fresh value of $h(k, x)$, no matter what k is.

More formally, let $\mathcal{A}^{\mathcal{H},\mathcal{O}}$ be an arbitrary quantum oracle algorithm, making quantum superposition queries to an oracle $\mathcal{H}$ and classical queries to another oracle $\mathcal{O}$, with the restriction that for every query x to $\mathcal{O}$ it holds that

$$h(\kappa, x) \neq h(\kappa, x') \ , \tag{5.7}$$

for any prior query x' to $\mathcal{O}$ and all $\kappa \in \mathcal{K}$. For any such oracle algorithm $\mathcal{A}^{\mathcal{H},\mathcal{O}}$, we consider the standard PRF security games

$$\mathsf{PR}^1 := \mathcal{A}^{H,F} \qquad \text{and} \qquad \mathsf{PR}^0 := \mathcal{A}^{H,R} \, ,$$

obtained by instantiating $\mathcal{H}$ with a random function H (the random oracle) in both games, and in one game we instantiate $\mathcal{O}$ with the pseudorandom function F, which we understand to return $F(k, x)$ on query x for a random $k \leftarrow \mathcal{K}$, chosen once and for all queries, and in the other we instantiate $\mathcal{O}$ with a truly random function R instead.

We show that the distinguishing advantage for these two games is bounded as follows.

Theorem 5.7. *Let $\mathcal{A}^{\mathcal{H},\mathcal{O}}$ be a (q_H, q_F)-query oracle algorithm satisfying (5.7). Then*

$$\left|\Pr\left[1 \leftarrow \mathsf{PR}^1\right] - \Pr\left[1 \leftarrow \mathsf{PR}^0\right]\right| \leq 4\sqrt{2q_F^2 q_H \epsilon} + 4\sqrt{2q_H^2 q_F \epsilon} \, .$$

We can now apply Theorem 5.7 to the function $h(k, x) := (g(k_1, \ldots, k_n), \tilde{x})$, where $k := k_i$ and $x := (k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n, \tilde{x})$. Indeed, the condition (5.5) on g implies the corresponding condition (5.7) on h, and the restriction on $\tilde{x}$ being fresh in the original skPRF definition implies the above restriction on $h(k, x)$ being fresh no matter what k is, i.e., 5.6). Thus, we obtain the following.

Corollary 5.8. *For any function g satisfying (5.5) for a given $\epsilon > 0$, the function $F(k_1, \ldots, k_n, x) := H(g(k_1, \ldots, k_n), x)$ is a skPRF in the quantum random oracle model with distinguishing advantage at most $4\sqrt{2q_F^2 q_H \epsilon} + 4\sqrt{2q_H^2 q_F \epsilon}$.*

5.4.3 Proof of Theorem 5.7

Proof (of Theorem 5.7). Let $\mathcal{A}^{\mathcal{H},\mathcal{O}}$ be an oracle algorithm as considered in the previous subsection. Thanks to Theorem 5.1, taking a factor-2 blow-up in the

query complexity into account, we may assume $\mathcal{A}$ to be a *static* (q_H, q_F)-query oracle algorithm. It will be convenient to write such a static algorithm as

$$\mathcal{A}^{[\mathbf{H}_0 \mathcal{O} \mathbf{H}_1 \mathcal{O} \mathbf{H}_2 \ldots \mathcal{O} \mathbf{H}_{q_F}]} ,$$

where each block $\mathbf{H}_i = \mathcal{H} \cdots \mathcal{H}$ consists of a (possibly empty) sequence of symbols $\mathcal{H}$ of length $q_i^{\mathcal{H}} = |\mathbf{H}_i|$, and with the understanding that $\mathcal{A}$ first makes $q_0^{\mathcal{H}}$ queries to $\mathcal{H}$, then a query to $\mathcal{O}$, then $q_1^{\mathcal{H}}$ queries to $\mathcal{H}$, etc., where, obviously, $q_0^{\mathcal{H}} + \cdots + q_{q_F}^{\mathcal{H}} = q_H$ then. Instantiating $\mathcal{H}$ with H, and $\mathcal{O}$ with F and R, respectively, we can then write

$$\mathsf{PR}^0 = \mathcal{A}^{[\mathbf{H}_0 R \mathbf{H}_1 \ldots R \mathbf{H}_{q_F}]} \qquad \text{and} \qquad \mathsf{PR}^1 = \mathcal{A}^{[\mathbf{H}_0 F \mathbf{H}_1 \ldots F \mathbf{H}_{q_F}]} .$$

For the proof, we introduce certain hybrid games. For this purpose, we introduce the following alternative (*stateful* and R-dependent) instantiation H' of $\mathcal{H}$. To start with, H' is set to be equal to H, but whenever R is queried on some input x, H' is *reprogrammed* at the point $h(k, x)$ to the value $H'(h(k, x)) := R(x)$. For any i, we now define the two hybrid games

$$\mathsf{PR}_i^2 := \mathcal{A}^{[\mathbf{H}_0 R \ldots R \mathbf{H}_i F \mathbf{H}'_{i+1} F \ldots F \mathbf{H}'_{q_F}]}$$
$$\widetilde{\mathsf{PR}}_i^2 := \mathcal{A}^{[\mathbf{H}_0 R \ldots R \mathbf{H}_i R \mathbf{H}'_{i+1} F \ldots F \mathbf{H}'_{q_F}]}$$

and also spell out

$$\mathsf{PR}_{i+1}^2 = \mathcal{A}^{[\mathbf{H}_0 R \ldots R \mathbf{H}_i R \mathbf{H}_{i+1} F \ldots F \mathbf{H}'_{q_F}]}$$

to emphasize its relation to $\widetilde{\mathsf{PR}}_i^2$. We note that in all of the above, the first occurrences of $\mathcal{H}$ and $\mathcal{O}$ are instantiated with R and H, respectively, but at some point we switch to R and H' instead.

The extreme cases match up the games we are interested in. Indeed,

$$\mathsf{PR}_0^2 = \mathcal{A}^{[\mathbf{H}_0 F \mathbf{H}'_1 \ldots F \mathbf{H}'_{q_F}]} = \mathcal{A}^{[\mathbf{H}_0 F \mathbf{H}_1 \ldots F \mathbf{H}_{q_F}]} = \mathsf{PR}^1 ,$$

where we exploit that there are no queries to R and thus H' remains equal to H, and, by definition,

$$\mathsf{PR}_{q_F}^2 = \mathcal{A}^{[\mathbf{H}_0 R \mathbf{H}_1 \ldots R \mathbf{H}_{q_F}]} = \mathsf{PR}^0 .$$

Our goal is to prove the closeness of the following games

$$\mathsf{PR}^1 = \mathsf{PR}_0^2 \approx \widetilde{\mathsf{PR}}_0^2 \approx \mathsf{PR}_1^2 \cdots \approx \mathsf{PR}_{q_F-1}^2 \approx \widetilde{\mathsf{PR}}_{q_F-1}^2 \approx \mathsf{PR}_{q_F}^2 = \mathsf{PR}^0 .$$

We do this by means of applying Lemma 5.9 and 5.10, which we state here and prove further down.

Lemma 5.9. *For each* $0 \le i < q_F$,

$$\left|\Pr\left[1 \leftarrow \mathsf{PR}_i^2\right] - \Pr\left[1 \leftarrow \widetilde{\mathsf{PR}}_i^2\right]\right| \le 2\sqrt{\sum_{1\le j\le i} q_j^{\mathcal{H}}\epsilon}\,.$$

Lemma 5.10. *For each* $0 \le i < q_F$,

$$\left|\Pr\left[1 \leftarrow \widetilde{\mathsf{PR}}_i^2\right] - \Pr\left[1 \leftarrow \mathsf{PR}_{i+1}^2\right]\right| \le 2q_{i+1}^{\mathcal{H}}\sqrt{q_F\epsilon}\,.$$

Indeed, by repeated applications of these lemmas, and additionally using that $q_0^{\mathcal{H}} + \cdots + q_i^{\mathcal{H}} \le q_H$ for all $0 \le i \le q_F$, we obtain

$$\begin{aligned}\left|\Pr\left[1 \leftarrow \mathsf{PR}^1\right] - \Pr\left[1 \leftarrow \mathsf{PR}^0\right]\right| &\le 2\sum_{i=0}^{q_F}\sqrt{\sum_{1\le j\le i} q_j^{\mathcal{H}}\epsilon} + 2\sum_{i=0}^{q_F} q_{i+1}^{\mathcal{H}}\sqrt{q_F\epsilon}\\ &\le 2\sqrt{q_F^2 q_H\epsilon} + 2\sqrt{q_H^2 q_F\epsilon}\end{aligned}$$

which concludes the claim of Theorem 5.7 when incorporating the factor-2 increase in q_H and q_F due to switching to a static $\mathcal{A}$. □

It remains to prove Lemma 5.9 and 5.10, which we do below. In both proofs, we use the *gentle measurement lemma* [Wil11, Lemma 9.4.1], which states that if a projective measurement has a very likely outcome then the measurement causes only little disturbance on the state. More formally, for any density operator ρ and any projector P, where $p := \mathsf{tr}(P\rho P)$ then is the probability to observe the outcome associated with P when measured using the measurement $\{P, \mathbb{I} - P\}$, the trace distance between the original state ρ and the post-measurement state $\rho' := P\rho P/p$ is bounded by $\sqrt{1-p}$. This in turn implies that ρ and ρ' can be distinguished with an advantage $\sqrt{1-p}$ only.

The proof of Lemma 5.9 additionally makes use of Zhandry's compressed oracle technique [Zha19]. It is out of scope of this work to give a self-contained description of this technique; we refer to the original work [Zha19] instead, or to [CFHL21], which offers an alternative concise description. At the core is the observation that one can *purify* the random choice of the function H and then, by switching to the Fourier basis and doing a suitable measurement, one can check whether a certain input x has been "recorded" in the database (mind though that such a measurement disturbs the state). If the outcome is negative then the oracle is still in a uniform superposition over all possible hash values for x, and as a consequence, when removing the purification by doing a full measurement of H (in the computational basis), $H(x)$ is ensured to be a "fresh" uniformly random value, with no information on $H(x)$ having been leaked in prior queries.

In the proof of Lemma 5.9, we use this technique to check whether *prior*

to the crucial query, which is to F in one and to R in the other game, there was a query to H that would reveal the difference, and we use (5.6) to argue that it is unlikely that such a query occurred. Since this measurement has a likely outcome, it is also ensured by the gentle measurement lemma that this measurement causes little disturbance.

Proof (of Lemma 5.9). For convenience, we refer to the *crucial query* as the respective query to F and R that differs between

$$\mathsf{PR}_i^2 = \mathcal{A}^{[\mathbf{H}_0 R \ldots R \mathbf{H}_i F \mathbf{H}'_{i+1} F \ldots F \mathbf{H}'_{q_F}]} \quad \text{and} \quad \widetilde{\mathsf{PR}}_i^2 = \mathcal{A}^{[\mathbf{H}_0 R \ldots R \mathbf{H}_i R \mathbf{H}'_{i+1} F \ldots F \mathbf{H}'_{q_F}]}.$$

Furthermore, we let x be the input to that query, and we set $w := h(k, x)$, with k being the key chosen and used by F. Note that up to this very query, the two games are identical. Also, by (5.7) it is ensured that for any prior query x' to R it holds that $h(k, x') \neq w$.

First, we consider the games $\mathbf{G}^1$ and $\widetilde{\mathbf{G}}^1$ that work exactly as PR_i^2 and $\widetilde{\mathsf{PR}}_i^2$, respectively, except that, at the beginning of the games we set up the compressed oracle and answer all queries made to H prior to the crucial query using the compressed oracle. Then, once x is received during the crucial query, we do a full measurement of the purified (i.e. uncompressed) oracle in order to obtain the function H, which is then to be used in the remainder of the games. We note that setting up the function H' is then necessarily also deferred to after this measurement, where H' is then set to be equal to H, except that for any prior query x' to R it is reprogrammed to $H'(h(k, x')) := R(x')$. Only once H has been measured and H' set up as above, is the crucial query then actually answered.

It follows from basic properties of the compressed oracle that the respective output distributions of $\mathbf{G}^1$ and $\widetilde{\mathbf{G}}^1$ match with those of PR_i^2 and $\widetilde{\mathsf{PR}}_i^2$.

Then, we define $\mathbf{G}^2$ and $\widetilde{\mathbf{G}}^2$ from $\mathbf{G}^1$ and $\widetilde{\mathbf{G}}^1$, respectively, by introducing one more measurement. Namely, right after x is sent by $\mathcal{A}$ and before H is measured, we measure in the compressed oracle whether the input $w = h(k, x)$ has been recorded in the database, and in case of a positive outcome, the game aborts. By the gentle measurement lemma (and basic properties of the trace distance),

$$\left|\Pr\left[1 \leftarrow \mathbf{G}^1\right] - \Pr\left[1 \leftarrow \mathbf{G}^2\right]\right| \leq \sqrt{\Pr\left[\mathbf{G}^2 \text{ aborts}\right]}$$

and similarly for $\widetilde{\mathbf{G}}^1$ and $\widetilde{\mathbf{G}}^2$, where $\widetilde{\mathbf{G}}^2$ aborts with the same probability as $\mathbf{G}^2$.

By basic properties, after $t := q_0^{\mathcal{H}} + \cdots + q_i^{\mathcal{H}}$ queries to the compressed oracle, no more than t values have been recorded. I.e., if we were to measure, for the sake of the argument, the entire compressed oracle to obtain the full database D, it would hold that $\mathrm{supp}(D) := \{u \,|\, D(u) \neq \bot\}$ has cardinality

at most t. Since k has not been used yet and so is still freshly random (i.e., independent of x and D), the high-entropy condition (5.6) then ensures that

$$\Pr\big[\widetilde{\mathbf{G}}^2 \text{ abort}\big] = \Pr\big[\mathbf{G}^2 \text{ abort}\big] = \Pr\big[w \in \mathrm{supp}(D)\big] \leq \sum_{j<i} q_j^{\mathcal{H}} \epsilon\,.$$

It remains to show that $\mathbf{G}^2$ and $\widetilde{\mathbf{G}}^2$ behave identically conditioned on not aborting. The only difference between the two games is that in $\mathbf{G}^2$ the crucial query is answered with $y := H(h(k,x)) = H(w)$ and H' is *not* reprogrammed at the point w, while in $\widetilde{\mathbf{G}}^2$ the crucial query is answered with $y := R(x)$ and H' *is* reprogrammed at the point w to $H'(w) := R(x)$. We argue that this difference is not noticeable by $\mathcal{A}$.

First, we note that y is a fresh random value in both games. In the former game it is because, conditioned on not aborting, the compressed oracle at the register $h(k,x)$ is $\perp$, and so when uncompressing and measuring to obtain H, the hash value $H(w)$ will be a fresh random value. In the latter game it is because $R(x)$ is a truly random function and, due to (5.7), x has not been queried to R before.

Second, we observe that $y = H'(w)$ in both games. Indeed, in $\widetilde{\mathbf{G}}^2$ this holds by definition; in $\mathbf{G}^2$ it holds because $H'(w) = H(w)$, which follows from the fact that H' is reprogrammed only at points $w' = h(k,x')$ with x' being a prior query to R, but then (5.7) ensures that $w' \neq w$.

Thus, in both games, from $\mathcal{A}$'s perspective, the tuple $(k, y, H', H\backslash w)$ of random variables has the same distribution, where $H\backslash w$ refers to the function (table of) H but with the value at the point w removed. The only difference is that in one game $H'(w) = H(w)$ and in the other not (necessarily). However, the future behavior of $\mathcal{A}$ in both games only depends on $(k, y, H', H\backslash w)$, and thus $\mathcal{A}$ behaves the same way in both games. Here we are exploiting that the future hash queries by $\mathcal{A}$ are to H' (and not to H anymore), and, once more, we are using the restriction (5.7), here to ensure that for any future F-query x' by $\mathcal{A}$, it holds that $h(k,x') \neq w$, and thus the response does not depend on $H(w)$. Thus, $H(w)$ does indeed not affect $\mathcal{A}$'s behavior after the crucial query.

Exploiting that $\mathsf{PR}_i^2 = \mathbf{G}^1 \approx \mathbf{G}^2 = \widetilde{\mathbf{G}}^2 \approx \widetilde{\mathbf{G}}^1 = \widetilde{\mathsf{PR}}_i^2$, with the approximations bounded as discussed further up, we obtain the claimed closeness claim. This concludes the proof. □

Proof of Lemma 5.10. In order to show the closeness between $\widetilde{\mathsf{PR}}_i^2$ and PR_{i+1}^2, we define the intermediate games

$$\mathbf{G}_{i,j} := \mathcal{A}^{[\mathbf{H}_0 R \ldots \mathbf{H}_i R \mathbf{H}'_{i,j} \mathbf{H}_{i,j} F \ldots F \mathbf{H}'_{q_F}]}$$

for $0 \leq j \leq m := q_{i+1}^{\mathcal{H}}$, where $\mathbf{H}'_{i,j}$ and $\mathbf{H}_{i,j}$ consists of j and $m-j$ copies of

H' and H respectively. Note that for the extreme cases we have

$$\mathbf{G}_{i,0} = \widetilde{\mathsf{PR}}_i^2 \qquad \text{and} \qquad \mathbf{G}_{i,m} = \mathsf{PR}_{i+1}^2 \,.$$

Thus, it suffices to show closeness between $\mathbf{G}_{i,j}$ and $\mathbf{G}_{i,j+1}$ for any $0 \leq j < m$. Note that they only differ at one query, which is either to H' or to H, which we will refer to as the *crucial query* for convenience. In the remainder, i and j are arbitrary (in the considered ranges) but fixed.

Define the games $\widetilde{\mathbf{G}}^1$ and $\mathbf{G}^1$ from $\mathbf{G}_{i,j}$ and $\mathbf{G}_{i,j+1}$ respectively as follows. Let X be the set of queries x made to R prior to the crucial query, and set $S := \{h(k,x) \,|\, x \in X\}$. We then measure the crucial query, which may be in a superposition, with the binary measurement that checks whether the crucial query is an element of S, and we abort if this is the case.

In case of a negative outcome, i.e., the crucial query is *not* in S, there is no difference between the reply provided by H and by H', and thus there is no difference between the two games — and in case of a positive outcome, they both abort. In order to argue that this measurement causes little disturbance, we again use the gentle measurement lemma to argue that

$$\left|\Pr\left[1 \leftarrow \mathbf{G}^1\right] - \Pr\left[1 \leftarrow \mathbf{G}_{i,j+1}\right]\right| \leq \sqrt{\Pr\left[\mathbf{G}^1 \text{ abort}\right]} \,,$$

and correspondingly for $\mathbf{G}_{i,j}$ and $\widetilde{\mathbf{G}}^1$. So it remains to bound the abort probability. For the purpose of the argument, let us do a full measurement of the query, and let w be the outcome. We note that k has not been used yet, and thus remains a fresh random key, independent of w and X. Thus, using (5.6),

$$\Pr\left[\mathbf{G}^1 \text{ abort}\right] = \Pr\left[\widetilde{\mathbf{G}}^1 \text{ abort}\right] = \Pr\left[w \in S\right] \leq \sum_{x \in X} \Pr\left[w = h(k,x)\right] \leq q_F \epsilon \,.$$

Adding up this error term over the sequence $\mathbf{G}_{i,0} \approx \cdots \approx \mathbf{G}_{i,m}$ of approximations, the proof is concluded. □