Random Oracles With(out) Programmability*

Marc Fischlin[1], Anja Lehmann[2], Thomas Ristenpart[3], Thomas Shrimpton[4], Martijn Stam[5], and Stefano Tessaro[3]

[1] Darmstadt University of Technology, Germany
[2] IBM Research Zurich, Switzerland
[3] University of California, San Diego, USA
[4] Portland State University, Portland, Oregon, USA
[5] Laboratory for Cryptologic Algorithms (LACAL), EPFL, Lausanne, Switzerland

Abstract. This paper investigates the Random Oracle Model (ROM) feature known as *programmability*, which allows security reductions in the ROM to dynamically choose the range points of an ideal hash function. This property is interesting for at least two reasons: first, because of its seeming artificiality (no standard model hash function is known to support such adaptive programming); second, the only known security reductions for many important cryptographic schemes rely fundamentally on programming. We provide formal tools to study the role of programmability in provable security. This includes a framework describing three levels of programming in reductions (none, limited, and full). We then prove that *no* black-box reductions can be given for FDH signatures when only limited programming is allowed, giving formal support for the intuition that full programming is fundamental to the provable security of FDH. We also show that Shoup's trapdoor-permutation-based key-encapsulation is provably CCA-secure with limited programmability, but no black-box reduction succeeds when no programming at all is permitted. Our negative results use a new concrete-security variant of Hsiao and Reyzin's two-oracle separation technique.

Keywords: hash functions, random oracle model, programmability, indifferentiability framework

1 Introduction

In the random oracle model (ROM) [1] parties are provided oracle access to a publicly available random function, a random oracle (RO). A random oracle is often viewed as the idealization of a cryptographic hash function, and security

* The work described in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. Marc Fischlin and Anja Lehmann were supported by the Emmy Noether Grant Fi 940/2-1 of the German Research Foundation (DFG). Marc is also supported by CASED (`www.cased.de`). Anja's work was done at TU Darmstadt. Thomas Ristenpart is supported by NSF grants CCF-0915675, CNS-0627779. Thomas Shrimpton is supported by NSF grants CNS-0627752, CNS-0845610. Stefano Tessaro's work was done at ETH Zurich, partially supported by the Swiss National Science Foundation (SNF), project no. 200020-113700/1.

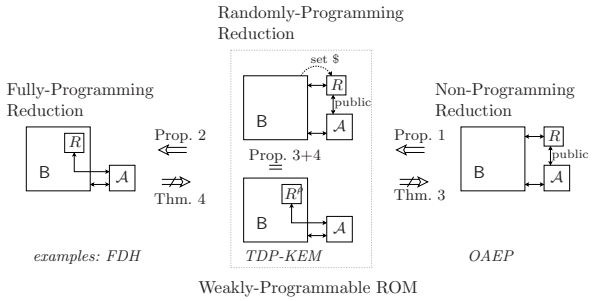

Fig. 1. Relations between the proposed models and example results. A "public" interface indicates that reduction B can see all queries of adversary $\mathcal{A}$, whereas "set $" denotes that B can re-assign random values to R, and R^ρ denotes a weakly-programmable RO. The "example: xxx" labels indicate a scheme xxx that enjoys a proof of security in the model above it, but for which we show black-box separation results implying the difficulty of proving its security in models to the right.

proofs in the ROM provide heuristic support for actual security when real hash functions are used instead. The ROM enables proofs of security for a multitude of important schemes because reductions may exploit various properties of a RO that can be realized only to a limited extent (if at all) in the standard model.

One such property is *programmability*. Loosely speaking, a random oracle can be "implemented" by dynamically selecting return values, and so long as the distribution of outputs is correct (uniform on the specified range), any method for selecting these values is permitted. The technique of programming RO output values in a security reduction seems crucial in countless positive results, e.g. [1, 3–5, 7]. However, no standard model function is known to provide the completely arbitrary and adaptive programmability provided by a RO, making it natural to wonder: which (if any) of these results could have been established *without* exploiting the full programmability of the ROM?

In this paper we formally explore models in which programmability of the random oracle in reductions is restricted. For this, we propose a form of limited programmability that is between full and no programmability. We provide two different but, surprisingly, equivalent characterizations of this limited form of programmability. We use them to show that: (1) one can prove (using a new variant of the Hsiao-Reyzin two-oracle separation technique [11]) the inability to give a programming-limited black-box reduction for the FDH signature scheme [1] and (2) that Shoup's trapdoor-permutation-based key encapsulation scheme (TDP-KEM) [15] is provably CCA secure given only limited programmability, while no black-box reduction works when programming is forbidden. For a diagrammatic summary of our main results, see Figure 1.

Modeling (Non-)Programmability in Reductions. Nielsen [13] was the first to formally investigate the role of programmability in security results. He showed that there is no way to realize a natural cryptographic functionality (non-committing non-interactive encryption) in a ROM-like model that strictly

prohibits programming of RO outputs. His result, and a more recent one by Wee [17] in the context of zero-knowledge, apply to simulation-based notions of security, and in particular restrict the ability of the simulator in these to program the RO. Unfortunately, these approaches are not sufficient to reason about the majority of ROM security proofs, which exploit programmability in *security reductions*, for example to embed an instance of a hard problem into RO outputs.

Our work considers this complementary direction, by investigating security reductions in models equipped with random oracles, but in which the ability of the reduction to program the random oracle is constrained. Along the lines of Nielsen's approach [13] in the simulation-based setting, our first contribution is to formalize *non-programming reductions* in the black-box (BB) setting (i.e. reductions only have oracle access to adversaries) by making the reduction work relative to an external RO (to which the adversary also has access).[6] We then propose a natural relaxation called *randomly-programming reductions.* Intuitively, the external random oracle is realized by a message-indexed table of randomly-chosen points, and while the reduction does not get to pick the range points, it can pick the order they appear in the table. As we shall see, this limitation on programming realizes an interesting middle point between full and no programming, and one that captures the provability of important schemes. Finally, a *fully-programming reduction* allows the reduction to arbitrarily choose output range values, as in traditional ROM proofs.

The Weakly-Programmable Random Oracle Model. A limitation of the above reduction-centric approach is the restriction to BB reductions. Indeed, as observed by Nielsen [13], providing models in which one can argue about limitations on programmability in non-BB reductions is challenging. This is because, in a non-BB setting, the reduction directly simulates all oracle queries made by an adversary and so there is no way to force the reduction to work relative to an external RO. We resolve this difficulty for the case of randomly-programming reductions by proposing a new variant of the ROM.

A Weakly-Programmable Random Oracle (WPRO) works as follows to form an idealized model of a hash function. Let ρ be an arbitrary function (whose range matches that of the hash function). For each distinct input x, the WPRO chooses its output to be $\rho(r)$ for a random coin-string r. Additionally, the WPRO allows *only* adversaries to obtain the coins r used to generate any output. Then in the WPRO model (WPROM) all parties have access to a WPRO that uses a regular, one-way function ρ. The requirements on ρ ensure that the WPROM limits programmability. For example, attempting to program an output of the oracle to a given value y requires computing $\rho^{-1}(y)$ and refuting one-wayness; regularity implies that the output of ρ is uniform, as usually required for random oracles.

The WPRO model appears to have little to do with randomly-programming reductions. Nevertheless, we prove that the two characterizations of limited

[6] The reduction does see the queries made by the adversary and the oracle's replies.

programming are strongly related. Namely, Proposition 3 states that any BB reduction in the WPROM implies a randomly-programming reduction, while Proposition 4 states that any randomly-programming reduction implies a reduction in the WPROM. Besides being convenient in proving results (the WPROM typically being easier to work with), the equivalence provides some evidence that this formulation of limited programmability is well-founded. The results discussed next point to this as well.

Implications for Practical Schemes. We put these new tools to use by reconsidering security proofs of various important schemes. These schemes can be viewed as initial and interesting case studies; we expect that one can use our techniques readily to analyze the need for programmability in many further schemes.

A first example is FDH signatures. The only known security proofs [1, 7] use reductions that embed a challenge range point for the underlying trapdoor permutation in one (or more) of the hash query responses. It may be, though, that a clever reduction exists that does not rely on programming. We give formal evidence that this is unlikely: Theorem 4 states that no BB reduction exists that shows FDH is secure in the WPROM, even for a very weak definition of unforgeability. Even if the intuition is clear that programming plays a significant role in existing reductions, we emphasize that *proving* the inability to give a reduction here is technically involved. Previous negative results use inherently *asymptotic* methods to achieve black-box separations in the uniform setting. Instead, our proof of Theorem 4 makes use of a novel approach that is non-asymptotic in nature. This result is complementary to existing negative results about FDH, e.g. [8]. (See the full version for additional discussion.)

A more involved example is Shoup's TDP-KEM [15]. Shoup's (IND-CCA) security proof does not involve embedding a challenge in the output of the RO, but rather programming is used to ensure consistency between simulation of a decapsulation oracle and simulation of the RO. We show the following surprising result: Shoup's TDP-KEM is CCA-secure in the WPROM (Theorem 2), but no non-programming BB reduction exists for showing CCA-security (Theorem 3). The negative result is even more complex than in the FDH case, involving several interesting technical hurdles (e.g. dealing with the fact that reductions can rewind adversaries, explicitly allowed in our non-programming reduction framework).

We also observe that OAEP [2] is an example of a scheme whose proof requires no programming whatsoever. This is actually evident by inspection of the proof given in [9]. We give the details in the full version, where all proofs omitted due to space constraints can also be found.

Discussion. Note that proving security with limited or no programming (still) only provides heuristic evidence for security. That said, it could be the case that proofs in the WPROM or that use a non-programming reduction provide a better heuristic than the ROM. While this would hold for any weakening of the ROM, we feel that programmability is a particularly interesting case due to its apparent

artificiality. Note, for example, that one can actually *run* a non-programming RO reduction when a concrete hash function (e.g. SHA-256) is used to realize the RO. This is not true for fully or randomly-programming reductions.

Further Related Work. Hofheinz and Kiltz [10] offer some insights on programmability from a completely different angle. Generalizing a technique due to Waters [16], they built *standard-model* hash functions that provide a limited form of programmability. Unfortunately, their hash functions are not sufficiently programmable to admit the techniques used in security arguments for ROM schemes like FDH and Fiat-Shamir. Nonetheless, their work indicates that a better understanding of programmability could lead to more broadly applicable standard-model solutions.

2 Reduction-Centric Models

In this section, we first formalize at an abstract level the general concept of a black-box reduction in the random oracle model. Furthermore, we present two variations of the black-box reduction notion where the reduction's capabilities in programming the random oracle are restricted.

2.1 Preliminaries

We begin by establishing notation and execution semantics for algorithms and oracles.

Oracle Access to Adversaries. We model all algorithms (e.g. adversaries and reductions) and ideal primitives (e.g. random oracles) as interactive Turing machines (ITM). In particular these machines can be probabilistic and can keep state. Each machine may have several communication tapes, which we usually call *interfaces*, that connect different machines to each other. We write $\mathcal{A}^{(\cdot)}$ to denote an ITM $\mathcal{A}$ with an interface that expects oracle (i.e. black-box) access to some other ITM. A reduction B with oracle access to adversary $\mathcal{A}^{(\cdot)}$ (denoted as $\mathsf{B}^{\mathcal{A}^{(\cdot)}}$) is allowed to do the following:

- At any time, B can start a copy of the algorithm $\mathcal{A}^{(\cdot)}$ on (chosen) randomness and input, where the random coins are those used to compute the first output (or oracle query).
- Once such a copy is started, B obtains each value output by $\mathcal{A}$ and must provide both the corresponding answer *and* the random coins needed for the execution of $\mathcal{A}$ to continue to its next output. (This includes queries to the given oracles.)
- At any point in time, B may halt the execution of the current copy of $\mathcal{A}$.

Note that the model is general enough so that B can, for example, "rewind" the adversary $\mathcal{A}$ to an arbitrary output by running a new copy of $\mathcal{A}$ with previously given coins.

We stress that if we write that B is given oracle access to $\mathcal{A}^{\mathcal{O}}$ for a *particular* oracle $\mathcal{O}$ (as opposed to $\mathcal{A}^{(\cdot)}$), then B does *not* get to answer $\mathcal{A}$'s queries to $\mathcal{O}$. Queries are sent directly to, and answered directly by $\mathcal{O}$ itself. We write (for example) $\mathcal{A}^{(\cdot,\mathcal{O})}$ when we wish to be explicit that queries to the first oracle are controlled by B, and the second are not. Sometimes we will simply omit some of the oracles which are controlled by B: the understanding is that any oracle which is not explicitly given to $\mathcal{A}$ in our notation can be controlled by the reduction.

Finally, we write $\mathcal{A}^{\mathcal{O}_{\text{pub}}}$ to mean the following: when $\mathcal{A}$ queries x to $\mathcal{O}$, x is forwarded to B, which can then perform some computations, call other oracles, and only after this triggers delivery of $\mathcal{O}(x)$ to $\mathcal{A}$. The answer is however given by $\mathcal{O}$ directly (but visible to B) and there is no way for B to influence it directly.[7] This construct will be useful in a number of contexts.

Security Properties. It is convenient to consider generic security properties Π for cryptographic primitives defined in terms of games involving a candidate f (called a Π-candidate) and an adversary $\mathcal{A}$ (called a Π-adversary). In particular, with each triple f, $\mathcal{A}$ and Π we associate an advantage $\mathbf{Adv}^{\Pi}_{f}(\mathcal{A})$, and f is said to be Π-secure if $\mathbf{Adv}^{\Pi}_{f}(\mathcal{A})$ is small for all efficient adversaries $\mathcal{A}$. It is convenient to assume that the advantage satisfies the following *linearity* condition: if an oracle $\mathcal{O}$ behaves as $\mathcal{O}_1$ with probability p and as $\mathcal{O}_2$ with probability $1-p$, then $\mathbf{Adv}^{\Pi}_{f^{\mathcal{O}}}(\mathcal{A}^{\mathcal{O}}) = p\cdot\mathbf{Adv}^{\Pi}_{f^{\mathcal{O}_1}}(\mathcal{A}^{\mathcal{O}_1}) + (1-p)\cdot\mathbf{Adv}^{\Pi}_{f^{\mathcal{O}_2}}(\mathcal{A}^{\mathcal{O}_2})$ for every (oracle) primitive f and all adversaries $\mathcal{A}$. Despite there being a few advantage notions that do not satisfy this property (e.g. distinguishing advantage with absolute values), an equivalent notion satisfying this property can typically be given (e.g. dispense with the absolute values).

2.2 Black-Box Reductions in the ROM

When we talk about black-box reductions, we mean *fully* black-box security reductions as defined by Reingold et al. [14]. Those reductions are paramount in cryptography, especially for random-oracle based schemes with practical efficiency as a design goal.

We present in Definition 1 below our formalization of fully-black-box reductions in the ROM, as well as our two variants with limited and no programmability of the random oracle, which we first introduce in detail.

Fully-Programming Reductions (FPRed). The first notion formalizes the standard concept of black-box reductions in the ROM. As they support the common strategy of programming the ROM without any restriction, we refer to such reductions as *fully-programming reductions*.

Non-Programming Reductions (NPRed). The first (stronger) new notion that we introduce captures the fact that the reduction has no control at all on the answers of random oracle queries. Namely, the queries are answered by a random oracle which is chosen once, independently from the reduction

[7] But the answer may be influenced through queries to related oracles.

B and its input(s), and remains the same for every execution of an adversary $\mathcal{A}$ initiated by B. While the reduction B can learn all of the RO-queries issued by $\mathcal{A}$, there is no way for B to influence their distribution. Intuitively, this models the fact that the reduction can be run with an *external* random oracle.

Randomly-Programming Reductions (RPRed). Our second variant only allows the reduction B to program the RO with random instead of arbitrary values, and is hence somewhat between fully- and non-programming reductions, To this end, we first introduce a *randomly-programmable random oracle (RPRO)* which is an idealized object that exposes three interfaces: $R_{eval}, R_{rand}, R_{prog}$ (a conventional RO can be seen as having a single interface to callers). If called via the *evaluation* interface R_{eval}, it behaves as a conventional random oracle mapping $Dom \to Rng$. A second *random* interface R_{rand} implements a random mapping $\{0,1\}^* \to Rng$. Finally, the *programming* interface R_{prog} takes $X \in Dom$ and $Y \in \{0,1\}^*$ as input, and sets $R_{eval}(X)$ to be the same as $R_{rand}(Y)$.
As $\mathcal{A}$'s queries to the evaluation interface of R_{eval} are public, the reduction B is allowed, on query X by $\mathcal{A}$, to perform a number of R_{rand} calls followed by a suitable $R_{prog}(X,Y)$ invocation in order to let the output of $\mathcal{A}$'s query satisfy a certain property before the query is actually answered to $\mathcal{A}$. This allows a minimal amount of programmability, for instance a constant number of output bits can be forced to take some input-dependent value.
We note that these interfaces allow to "reprogram" the random oracle. This supports, among other things, the ability to rewind the adversary and run it on "another", partly consistent random oracle, but where the reduction does not need to choose the actual values. Note that non-programming reductions prevent such forking techniques.

In the following, let $\mathsf{S} = \mathsf{S}^R[f]$ be a cryptographic scheme relying on a primitive f and a random oracle $R : Dom \to Rng$. Let Π and Π' be security properties which can possibly be satisfied by S and f, respectively.

Definition 1 (FPRed, NPRed, RPRed). *Let* $\mathsf{X} \in \{$*fully-programming, non-programming, randomly-programming*$\}$. *A* $(\Pi \to \Pi', \delta, t, q_{\mathcal{O}}, q_A)$*-fully-BB* X *ROM security reduction for* S *is an oracle machine* $\mathsf{B}^{(\cdot,\cdot)}$ *with the property that for* all[8] Π*-adversaries* $\mathcal{A}^{(\cdot)}$ *and all* Π'*-candidates* f*, if*

$$\mathbf{Adv}^{\Pi}_{\mathsf{S}^R[f]}(\mathcal{A}^R) > \epsilon$$

for a random oracle $R : Dom \to Rng$ *and* $\epsilon > 0$*, then*

$$\mathbf{Adv}^{\Pi'}_{f}(\mathsf{B}^{\mathcal{O}_1,\mathcal{A}^{\mathcal{O}_2}}) > \delta(\epsilon, q, \ell),$$

where q *is the total number of queries* $\mathcal{A}$ *makes and* ℓ *is the overall length of these queries. Furthermore,* B *runs in time* t*, makes* $q_{\mathcal{O}}$ *queries to the given oracle(s)*

[8] In particular, including those which are not efficiently implementable.

$\mathcal{O}_1$ and runs q_A instantiations of $\mathcal{A}^{\mathcal{O}_2}$, where all three quantities are functions of ϵ, q, and ℓ. Moreover, when:

$$\begin{aligned}
\mathsf{X} = \text{fully-programming}&\text{, then } \mathcal{O}_1 = f,\ \mathcal{O}_2 = (\cdot),\ \text{and } q_{\mathcal{O}} = q_F \\
\mathsf{X} = \text{non-programming}&\text{, then } \mathcal{O}_1 = (f, R),\ \mathcal{O}_2 = R_{\text{pub}},\ \text{and } q_{\mathcal{O}} = (q_F, q_R) \\
\mathsf{X} = \text{randomly-programming}&\text{, then } \mathcal{O}_1 = (f, R'_{eval}, R'_{prog}, R'_{rand}), \\
&\qquad\qquad \mathcal{O}_2 = R'_{eval,\text{pub}},\ \text{and } q_{\mathcal{O}} = (q_F, q_{ev}, q_{pr}, q_{ra}),
\end{aligned}$$

where $R' = (R'_{eval}, R'_{prog}, R'_{rand})$ is a RPRO.

2.3 Black-Box Separations

This paper uses a novel approach in order to obtain black-box separations in the concrete setting. The approach applies to all notions of reductions defined in this paper, but we illustrate it in the context of FPRed reductions. In order to disprove the existence of a fully-BB reduction within a certain class of reductions, for every reduction B of interest, we have to prove the existence of a Π'-candidate f and an adversary $\mathcal{A}$ such that $\mathbf{Adv}^{\Pi}_{\mathsf{S}^R[f]}(\mathcal{A}^R)$ is large, but the advantage $\mathbf{Adv}^{\Pi'}_{f}(\mathsf{B}^{f,\mathcal{A}^{(\cdot)}})$ is small. We will achieve this by first showing the existence of a *randomized* Π'-candidate F and an adversary $\mathcal{A}_P$ with *private* random coins P (i.e. not controllable by B) such that $\mathbf{Adv}^{\Pi}_{\mathsf{S}^R[F]}(\mathcal{A}_p^{f,R})$ is large for *all* values p of the random coins P and for all (fixed) primitives f obtained by fixing the coins of F, but $\mathbf{Adv}^{\Pi'}_{F}(\mathsf{B}^{F,\mathcal{A}_P^{(F,\cdot)}})$ is small for all reductions B of interest. Because of the linearity of the advantage measures, we have

$$\mathbf{Adv}^{\Pi'}_{F}(\mathsf{B}^{F,\mathcal{A}_P^{(F,\cdot)}}) = \mathbf{E}_{p,f}\left[\mathbf{Adv}^{\Pi'}_{f}(\mathsf{B}^{f,\mathcal{A}_p^{(f,\cdot)}})\right],$$

where the expected value is taken over the choice of the private coins p and the primitive f realized by F (with the corresponding probability distributions, which may in the general case even be correlated). Therefore, for all reductions B of interest, there must exist some particular f and some adversary $\mathcal{A}'^{(\cdot)} := \mathcal{A}_p^{(f,\cdot)}$ *without* private coins such that $\mathbf{Adv}^{\Pi'}_{f}(\mathsf{B}^{f,\mathcal{A}'^{(\cdot)}}) \leq \mathbf{Adv}^{\Pi'}_{F}(\mathsf{B}^{F,\mathcal{A}_P^{(F,\cdot)}})$ is small, too. Hence, such a statement (for randomized primitives) also implies the inexistence of a reduction working universally for all primitives: in particular, there is no need to apply well-known classical asymptotic (and uniform) de-randomization techniques based on the Borel-Cantelli lemma. To the best of our knowledge, this approach is novel to this paper.

3 The Weakly Programmable ROM

In the previous section programmability (or the lack thereof) is captured by considering a restricted set of reductions; from the point of view of the adversary being employed by the reduction, nothing has changed. In this section we take an

subroutine $R^{\rho}_{hon}(X)$:	**subroutine** $R^{\rho}_{adv}(X)$:
if $\mathtt{T}[X] \neq \perp$ then ret $\mathtt{T}[X]$	if $\mathtt{T}[X] \neq \perp$ then ret $\mathtt{T}[X], \mathtt{R}[X]$
$r \leftarrow\!\!\$\ \mathit{Coins}$; $z \leftarrow \rho(r)$	$r \leftarrow\!\!\$\ \mathit{Coins}$; $z \leftarrow \rho(r)$
$\mathtt{T}[X] \leftarrow z$; $\mathtt{R}[X] \leftarrow r$	$\mathtt{T}[X] \leftarrow z$; $\mathtt{R}[X] \leftarrow r$
ret $\mathtt{T}[X]$	ret $\mathtt{T}[X], \mathtt{R}[X]$

Fig. 2. The weakly-programmable random oracle ideal primitive R^{ρ} for $\rho\colon \mathit{Coins} \to \mathrm{Rng}$. Initially $\mathtt{T}[X] = \perp$ for all X.

alternative approach, modifying the random oracle itself rather than restricting the reduction.

Consider a random oracle as a mapping from *Dom* to *Rng*, where *Rng* is finite and non-empty. Since we model ideal primitives by stateful and probabilistic interactive Turing machines we can imagine the random oracle as being implemented via so-called lazy sampling: whenever a new query $X \in \mathit{Dom}$ appears, the random oracle returns a random value $z \leftarrow\!\!\$\ \mathit{Rng}$ and stores the pair (X, z) for further use. We now restrict the way the random oracle's answers z are determined. Namely, we parameterize the random oracle by a function $\rho\colon \mathit{Coins} \to \mathit{Rng}$ for a finite, non-empty set *Coins*. Each time the random oracle receives a new query X it picks $r \leftarrow\!\!\$\ \mathit{Coins}$ at random and returns $z = \rho(r)$ and stores X together with r.

Now, recall that an ideal primitive can have multiple interfaces. In what follows, we consider two: an *honest* interface for use by honest parties and protocols; and an *adversarial* interface. Loosely, the latter will give the adversary an ability to "validate" that the random oracle is behaving properly. Formally, we give the following definition of a (ρ-restricted) weakly programmable random oracle.

Definition 2 (WPRO). *For a function $\rho : \mathit{Coins} \to \mathit{Rng}$ the ideal primitive $R^{\rho} = (R^{\rho}_{hon}, R^{\rho}_{adv})$ described in Figure 2 is called a ρ-WPRO (or simply WPRO if ρ is implicitly clear).*

Notice that the honest interface of this object returns the range point z associated with the queried input. The adversarial interface returns both that range point *and* the random value r used to generate z.

At this point we have not imposed any restriction on ρ. For example, if ρ is the identity function (and $\mathit{Rng} = \mathit{Coins}$) then the resulting ideal primitive is equivalent to a normal random oracle. On the other end of the spectrum, if ρ is a constant function then it is clear that R^{ρ} would not model an ideal cryptographic hash function. Thus we establish what it means for a function ρ to be *good*.

Definition 3 (Good ρ). *A function $\rho : \mathit{Coins} \to \mathit{Rng}$ is called* good *for Rng if and only if: (1) Coins is finite, (2) $|\mathit{Rng}|$ divides $|\mathit{Coins}|$ and (3) ρ is regular, i.e. for all $y \in \mathit{Rng}$ we have*

$$|\{r \in \mathit{Coins} : \rho(r) = y\}| = \frac{|\mathit{Coins}|}{|\mathit{Rng}|}.$$

Clearly any good ρ is such that, when evaluated on a uniformly chosen domain point, one gets a uniform range point. (And conversely, if a uniform distribution on the domain of ρ induces a uniform distribution on the range, ρ is good.) Said another way, a random oracle $R : \mathit{Dom} \to \mathit{Rng}$ and WPRO R^{ρ}_{hon} (with matching domain and range) are information-theoretically indistinguishable if and only if ρ is good for Rng.

It is easy to see that various kinds of functions ρ will limit a reduction's ability to program. For the scenarios we consider, the crucial property of ρ that make the reductions—the proof of security—fail, is one-wayness of ρ (but stated in the non-asymptotic setting via an upper bound on an algorithm's inversion probability). For any function ρ and owf-adversary $\mathcal{A}$ we define the owf advantage as

$$\mathbf{Adv}^{\mathrm{owf}}_{\rho}(\mathcal{A}) = \Pr\left[\, \rho(r) = \rho(r') \;:\; r \leftarrow\!\!\$\; \mathit{Coins}\,;\; r' \leftarrow\!\!\$\; \mathcal{A}(\rho(r)) \,\right]$$

where a owf-adversary $\mathcal{A}$ is a probabilistic algorithm that takes as input a point $y \in \mathit{Rng}$ and outputs a domain point $x \in \mathit{Coins}$.

One-wayness of ρ ensures non-programmability in the following sense: Consider for example a security reduction like the traditional one for FDH. This reduction receives a random image y under a trapdoor permutation and, at some point, injects this value as the hash value in a black-box simulation for an allegedly successful adversary. But since the adversary can access the R^{ρ}_{adv} interface, the reduction would also need to provide a preimage of y under ρ, violating the one-wayness of ρ.

Reductions in the WPRO Model. One can straightforwardly define a WPRO model (WPROM) by analogy to the ROM (all honest parties have access to R_{hon}, adversarial parties have access to R_{adv}), and the notion of a black-box reduction naturally extends to this model. In particular, we consider a strong notion of reduction that allows any *good* function ρ, regardless of whether ρ is efficiently computable or not.

Definition 4 (WPROM Reduction). *A $(\Pi \to \Pi', \delta, t, q_\rho, q_F, q_A)$-fully-BB WPROM security reduction for S is an oracle machine $\mathsf{B}^{(\cdot,\cdot,\cdot)}$ with the property that for* all *Π-adversaries $\mathcal{A}^{(\cdot)}$, all good functions ρ for Rng, and all Π'-candidates f, if*

$$\mathbf{Adv}^{\Pi}_{\mathsf{S}^{R^{\rho}_{hon}}[f]}(\mathcal{A}^{R^{\rho}_{\mathrm{adv}}}) > \epsilon$$

for a ρ-WPRO $R^{\rho} = (R^{\rho}_{hon}, R^{\rho}_{\mathrm{adv}})$ with range Rng and $\epsilon > 0$, then

$$\mathbf{Adv}^{\Pi'}_{f}(\mathsf{B}^{\rho, f, \mathcal{A}^{(\cdot)}}) > \delta(\epsilon, q, \ell),$$

where q is the total number of queries $\mathcal{A}$ makes and ℓ is the overall length of these queries. Furthermore, B runs in time t, makes q_ρ and q_F queries to the given ρ and f, respectively, and runs q_A instantiations of $\mathcal{A}^{(\cdot)}$, where all three quantities are functions of $\epsilon, q,$ and ℓ.

Since the reduction notion quantifies over all good ρ, a reduction must work for one-way ρ. Indeed, it must also work for a ρ chosen randomly from the set

of all functions $Coins \to Rng$. In this way reductions must avoid making use of FDH-style programming: a reduction cannot inject a specific range point into one of WPRO's responses. As we will see in the next section, however, one can take advantage of more limited programming techniques in the WPRO model.

Although all the WPROM reductions given in this paper are fully black-box as per the definition above, we emphasize that the WPRO model is distinct from the formulations in Section 2 in that one can give non-black-box reductions in it.

4 Relationships among Types of Reductions

Having specified our reduction settings, we now establish the relationships among them. We begin by stating the intuitive implications: a non-programming BB reduction implies a randomly programming one, which in turn implies a fully programmable reduction. The straightforward proofs are omitted. Let $\mathsf{S} = \mathsf{S}^{f,R}$ be a scheme relying on a cryptographic primitive f and a random oracle $R : Dom \to Rng$. Let Π and Π' be security properties which can possibly be satisfied by S and f, respectively.

Proposition 1 (NPRed $\Rightarrow$ RPRed). *If there exists a* non-programming $(\Pi \to \Pi', \delta, t, q_F, q_R, q_A)$*-fully-BB ROM security reduction for* S*, then there exists a* randomly-programming $(\Pi \to \Pi', \delta, t, q_F, q_R, 0, 0, q_A)$*-fully-BB ROM security reduction for* S*.*

Proposition 2 (RPRed $\Rightarrow$ FPRed). *If there exists* randomly-programming $(\Pi \to \Pi', \delta, t, q_F, q_{ev}, q_{pr}, q_{ra}, q_A)$*-fully-BB ROM security reduction for* S*, then there exists a* fully-programming $(\Pi \to \Pi', \delta, t', q_F, q_A)$*-fully-BB ROM security reduction for* S*, where*[9] $t' = t + \mathcal{O}(\overline{q} \log \overline{q})$ *for* $\overline{q} = q \cdot q_A + q_{ev} + q_{pr} + q_{ra}$*.*

Next we show that schemes are secure in the WPRO model via a black-box reduction if and only if there is a randomly-programming reduction. Hence, restricting the *random oracle* in the WPRO sense, and restricting the *reduction*'s abilities to program a full-fledged random oracle, are equivalent in a black-box sense. The first result, in particular, exploits the fact that a fully-BB reduction in the WPROM must also work for a randomly chosen (regular) function ρ.

Proposition 3 (WPROM Red $\Rightarrow$ RPRed). *If a* $(\Pi \to \Pi', \delta, t, q_\rho, q_F, q_A)$*-fully-BB WPROM security reduction for* S *exists, then a randomly-programming* $(\Pi \to \Pi', \delta', t', q_F, q \cdot q_A, q \cdot q_A, q_\rho + q \cdot q_A, q_A)$*-fully-BB ROM security reduction for* S *exists, where* $\delta' = \delta - \frac{(q_A \cdot q + q_\rho)^2}{2|Dom|}$ *and* $t' = t + \mathcal{O}(q \cdot \ell)$*.*

Proposition 4 (RPRed $\Rightarrow$ WPROM Red). *If there exists a randomly-programming* $(\Pi \to \Pi', \delta, t, q_F, q_{ev}, q_{pr}, q_{ra}, q_A)$*-fully-BB ROM security reduction for* S*, then a* $(\Pi \to \Pi', \delta, t', q_F, q'_\rho, q_A)$*-fully-BB WPROM security reduction for* S *exists with* $t' = t + \mathcal{O}((q \cdot q_A \cdot \log(q \cdot q_A) \cdot \ell)$ *and* $q'_\rho = q \cdot q_A + q_{ev} + q_{pr} + q_A$*.*

[9] The extra overhead $\mathcal{O}(\overline{q} \log \overline{q})$ is due to the simulation of the RPRO in the reduction.

WPROs are not ROs, but WPROM and ROM are equivalent. Below we will confirm the expected implication that being a WPRO is actually a weaker requirement than being a full-fledged RO. Yet *existentially* WPROs and ROs are equivalent, i.e. we can efficiently construct a RO out of a WPRO.

For these comparisons we adopt the indifferentiability framework of Maurer, Renner and Holenstein [12] to reason about primitives being close to random oracles. We denote by $\mathbf{Adv}^{\text{ind-R}}_{C,H,\mathcal{S}}(\mathcal{D})$ the advantage any distinguisher $\mathcal{D}$ has in distinguishing between a construction C with component H, and an ideal primitive "R" (with an intermediary simulator $\mathcal{S}$). We denote by the superscripts "RO" and "WPRO" in the advantage the fact that the ideal primitive "R" is a random oracle or a WPRO.

First, we derive a WPRO that is not a RO, i.e. it is easily differentiable from a random oracle. This serves to show that in general WPROs and ROs are separated. Consider the composition $C^H_\rho(x) = \rho(H(x))$ of a random oracle $H : \mathit{Dom} \to \mathit{Coins}$ and a regular one-way function $\rho : \mathit{Coins} \to \mathit{Rng}$. In this case, any simulator $\mathcal{S}$ would have to invert ρ for a sound simulation.

Proposition 5 (WPRO $\not\Rightarrow$ RO). *For any function ρ that is good for Rng, for the construction $C^H_\rho(x) = \rho(H(x))$ there exists a simulator $\mathcal{S}^{WPRO}_\rho$ such that*

$$\mathbf{Adv}^{\text{ind-WPRO},\rho}_{C_\rho,H,\mathcal{S}^{WPRO}}(\mathcal{D}) = 0$$

for any distinguisher $\mathcal{D}$, but where there exists a distinguisher $\mathcal{D}^{RO}$ such that for any simulator $\mathcal{S}$,

$$\mathbf{Adv}^{\text{ind-RO}}_{C_\rho,H,\mathcal{S}}(\mathcal{D}^{RO}) \geq 1 - \mathbf{Adv}^{\text{owf}}_\rho(\mathcal{S}).$$

Despite this generic separation, it *is* possible to build a (fully programmable) random oracle out of a WPRO, essentially building RO outputs one bit at a time. Specifically, for $x \in \{0,1\}^*$ let $x|_i$ denote the ith bit of x. Given a function $H : \{0,1\}^* \to \{0,1\}^\ell$ for some $\ell > 1$ we consider the construction $\mathsf{C}^H : \{0,1\}^* \to \{0,1\}^m$ such that

$$\mathsf{C}^H(x) = H(x\|\langle 1\rangle)|_1 \,\|\, H(x\|\langle 2\rangle)|_1 \,\|\cdots\|\, H(x\|\langle m\rangle)|_1,$$

where $\|$ denotes concatenation of strings and $\langle i\rangle$ is the (suffix-free) binary encoding of an integer i. Note that the construction calls H altogether m times to achieve output length m; one can improve the efficiency by outputting more bits in each iteration at the cost of tightness in the reduction. Furthermore, due to the suffix-freeness of $\langle\cdot\rangle$ one can always decide if a given string is of the form $x\|\langle i\rangle$ for some $i \in \mathbb{N}$.

Theorem 1 (WPROM $\Leftrightarrow$ ROM). *For all good functions ρ, all integers $\tau > 0$, and a WPRO $R^\rho = (R^\rho_{\text{adv}}, R^\rho_{\text{hon}}) : \{0,1\}^* \to \{0,1\}^\ell$ there exists a simulator $\mathcal{S}_{\rho,\tau}$ such that for all distinguishers $\mathcal{D}$ issuing at most q queries to each oracle we have*

$$\mathbf{Adv}^{\text{ind-RO}}_{\mathsf{C},R^\rho,\mathcal{S}_{\rho,\tau}}(\mathcal{D}) \leq q \cdot 2^{-\tau}$$

where the simulator $\mathcal{S}_{\rho,\tau}$ invokes R at most once on each query and has running time $\mathcal{O}(\tau \cdot (\mathsf{Time}_\rho + \mathsf{Time}_{Coins}))$, where Time_ρ and Time_{Coins} are the times needed to compute ρ and to sample a random element from Coins, respectively.

5 Trapdoor-Permutation-Based Key-Encapsulation

5.1 TDP-KEM Security in the WPROM

A key-encapsulation mechanism (KEM) is a triple of algorithms denoted $\mathsf{KEM} = (\mathsf{Key}, \mathsf{Encap}, \mathsf{Decap})$ that operate as follows. The probabilistic *key-generation algorithm* returns a key-pair (pk, sk); we write $(pk, sk) \leftarrow\!\!\$\ \mathsf{Key}$. The *(key) encapsulation algorithm* Encap is a probabilistic algorithm that takes pk as input and returns a key-ciphertext pair (K, C) where $K \in \mathcal{K}$ for some non-empty set $\mathcal{K}$. The *(key) decapsulation algorithm* takes as input a pair (sk, C) and deterministically outputs a key $K \in \mathcal{K}$ or the distinguished symbol $\perp$ to denote invalidity of (sk, C). For proper operation, we require that for all pairs (pk, sk) generated by Key, if $(K, C) \leftarrow\!\!\$\ \mathsf{Encap}(pk)$ then $K \leftarrow \mathsf{Decap}(sk, C)$.

Let $\mathsf{KEM} = (\mathsf{Key}, \mathsf{Encap}, \mathsf{Decap})$ be a KEM, $\mathcal{K}$ be a non-empty set, and $\mathcal{A}$ be a KEM adversary. The security of KEM, in the WPRO model of hash function R with underlying function ρ, against an adversary $\mathcal{A}$ is defined by the following experiment:

$$\boxed{\begin{array}{l}\underline{\mathbf{Exp}^{\text{kem-cca}}_{\mathsf{KEM},R,\rho}(\mathcal{A})} \\ (pk, sk) \leftarrow\!\!\$\ \mathsf{Key};\ b \leftarrow\!\!\$\ \{0,1\}; \\ b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Decap}(sk,\cdot),R^\rho_{adv}(\cdot),\mathsf{Encap}(pk,b,\cdot)}(pk) \\ \text{if } b' = b \text{ then return 1 else 0}\end{array}}$$

The $\mathsf{Decap}(sk, \cdot)$ oracle performs the decapsulation algorithm upon its input and returns the result. The $\mathsf{Encap}(pk, b, \cdot)$ oracle takes as input a distinguished symbol RUN, picks $K_0 \leftarrow\!\!\$\ \mathcal{K}$, runs the encapsulation algorithm to produce $(K_1, C) \leftarrow\!\!\$\ \mathsf{Encap}(pk)$, and returns the challenge (K_b, C). The encapsulation oracle can be queried only once by the adversary. We then define the KEM-CCA advantage of adversary $\mathcal{A}$ in breaking the KEM scheme via a chosen-ciphertext attack as $\mathbf{Adv}^{\text{kem-cca}}_{\mathsf{KEM},R,\rho}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\text{kem-cca}}_{\mathsf{KEM},R,\rho}(\mathcal{A}) = 1] - 1/2$, where $\mathcal{A}$ is forbidden to ask the challenge ciphertext C to its decapsulation oracle.

TDP-BASED KEMS IN THE WPRO MODEL. We recall that a trapdoor permutation with domain *Dom* is a triple $\mathcal{TP} = (G, F, \overline{F})$ of efficient algorithms such that G returns a pair (pk, td), consisting of the public key and the trapdoor, with the property that $F(pk, \cdot)$ implements a permutation $f_{pk} : Dom \to Dom$, whereas $\overline{F}(td, \cdot)$ implements its inverse $f^{-1}_{pk}(\cdot)$. Consider key encapsulation mechanism $\mathsf{TDP\text{-}KEM}^R[\mathcal{TP}] = (\mathsf{Key}, \mathsf{Encap}, \mathsf{Decap})$ based on a TDP $\mathcal{TP} = (G, F, \overline{F})$ with domain *Dom* and a WPRO $R^\rho\colon Dom \to \mathcal{K}$ for some underlying good function ρ mapping *Coins* to $\mathcal{K}$, where $\mathcal{K}$ is some non-empty set. The key generation algorithm is defined by $\mathsf{Key} = G$, so it returns a pair (pk, td). The encapsulation algorithm on input pk samples $x \leftarrow\!\!\$\ Dom$, sets $K \leftarrow R^\rho_{hon}(x)$ and $C \leftarrow F(pk, x)$,

and returns (K, C). The decapsulation algorithm on input (td, C) computes $x \leftarrow \overline{F}(td, C)$, sets $K \leftarrow\!\!\$\ R^{\rho}_{hon}(x)$ and returns K.

The KEM-CCA security of this scheme is tightly bound to the OWF security of the underlying TDP. Our proof largely mirrors the one given by Shoup [15] for RSA-KEM in the ROM.

Theorem 2 (WPROM Reduction for TDP-KEM). *Let $\mathcal{TP} = (G, F, \overline{F})$ be a trapdoor permutation with domain Dom. Let $\mathsf{TDP\text{-}KEM}^R[\mathcal{TP}] = (\mathsf{Key}, \mathsf{Encap}, \mathsf{Decap})$ be the TDP-based KEM described above. Let ρ be good for the non-empty set $\mathcal{K}$ and let $\mathcal{A}$ be an adversary suitable for $\mathbf{Exp}^{\text{kem-cca}}_{\mathsf{TDP\text{-}KEM},R,\rho}(\mathcal{A})$ asking q_D queries to its decapsulation oracle, q_R queries to the WPRO and running in time t. Then there exists an adversary $\mathcal{B} = \mathsf{B}^{\rho,F,\mathcal{A}(\cdot)}$ such that*

$$\mathbf{Adv}^{\text{kem-cca}}_{\mathsf{TDP\text{-}KEM},R,\rho}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{owf}}_{\mathcal{TP}}(\mathcal{B}) + \frac{q_D}{|Dom|}$$

where $\mathcal{B}$ uses a single instantiation of $\mathcal{A}^{(\cdot)}$, runs in time at most $t + \overline{q} \log \overline{q} \cdot (\mathsf{Time}_F + \mathsf{Time}_\rho + \mathsf{Time}_{Coins} + \mathsf{Time}_{\mathcal{K}})$ for $\overline{q} = q_D + q_R + 1$ and Time_X denotes the time to execute algorithm X or to sample from set X.

Here we give a very brief overview of the proof. The reduction is required to invert the TDP on some challenge range point C^*, and it will embed this challenge along with a random key K as the response (K, C^*) to the KEM-adversary's encapsulation query. Despite not having access to the trapdoor information, the reduction *can* answer decapsulation queries by (randomly) programming the WPRO to $\rho(r)$, where r is uniform. This simulation is correct, and can easily be made consistent with WPRO queries, except in the case that Decap is queried on the TDP challenge point C^*. This case accounts for the $q_D/|Dom|$ term in the bound. Barring that case, the KEM-adversary wins its game only by querying the WPRO on the preimage of C^*, in which case the reduction succeeds to invert its challenge.

5.2 TDP-KEM is not Provable under Non-Programming Reductions

In the proof of Theorem 2, a weak form of programmability is needed to allow for consistent simulation of the decapsulation oracle. Namely, the reduction may need to return a *random* key $K \in \mathcal{K}$ for a decapsulation query C, because it does not know the associated preimage r of C under $F(pk, \cdot)$. Consequently, if the adversary queries the random oracle with input r at a later point in time, its output is programmed to K. This fact makes $\mathsf{TDP\text{-}KEM}$ amenable to an attempt to entirely avoid programmability, e.g., by means of rewinding techniques. Yet, any such approach is doomed to fail: we prove that $\mathsf{TDP\text{-}KEM}$ *cannot* be proven secure with respect to non-programming fully-BB reductions, hence showing that $\mathsf{TDP\text{-}KEM}$ is a scheme which *necessarily* requires a mild type of programmability.

This is summarized by the following theorem. Note that the result requires $q_{\mathcal{A}} \leq 2^{q-1}$: For a small number of adversarial queries q a reduction may indeed be feasible (e.g. using rewinding). Yet, for acceptable values of q the value 2^{q-1} is too large for an efficient reduction to be allowed to issue more than 2^{q-1} queries.

Theorem 3 (Non-Programming Irreducibility for TDP-KEM). *Let* $\mathsf{TDP\text{-}KEM}^R[\mathcal{TP}] = (\mathsf{Key}, \mathsf{Encap}, \mathsf{Decap})$ *be the* TDP-KEM *scheme with key space* $\mathcal{K}$*, relying on a trapdoor permutation* $\mathcal{TP} = (G, F, \overline{F})$ *with domain* Dom *and public-key/trapdoor space* $\{0,1\}^k$*, as well as on a random oracle* $R : Dom \to \mathcal{K}$*. Then, for all* $t, q > 0$*, all* $\epsilon \leq \frac{1}{2} - \frac{1}{|\mathcal{K}|}$*, and all* $(\text{kem-cca} \to \text{owp}, \delta, t, (q_G, q_F, q_{\overline{F}}), q_R, q_{\mathcal{A}})$*-fully-BB non-programming reductions* B *for* TDP-KEM*, we have*

$$\delta(\epsilon, q, q \cdot \log|Dom|) \leq \frac{(q_{\mathcal{A}} q + 1) \cdot (2 q_{\mathcal{A}} q + q_F + q_R + 1)}{|Dom|} + \frac{q_{\mathcal{A}} q}{|\mathcal{K}|} + \frac{q_G + q_{\overline{F}}}{2^k}$$

where $q_G, q_F, q_{\overline{F}}$*, and* q_R *are the number of queries of* B *to the respective oracles, and* $q_{\mathcal{A}} \leq 2^{q-1}$ *is the number of adversarial instances run by* B*.*

We provide a high-level description of the proof. We rely on an ideal trapdoor permutation $\mathcal{TP} = \mathcal{TP}^{\mathcal{F}} = (G, F, \overline{F})$ defined using the oracles $\mathcal{F} = (\mathcal{F}_\tau, \mathcal{F}_E, \mathcal{F}_{E^{-1}})$: The oracle $\mathcal{F}$ initially chooses a keyed family of *random* permutations $E : \{0,1\}^k \times Dom \to Dom$ (in other words, $E(pk, \cdot)$ is a random permutation for all k-bit pk), as well as a random permutation τ that associates to each k-bit trapdoor td a corresponding public key $pk = \tau(td)$. The oracles $\mathcal{F}_\tau$ and $\mathcal{F}_E$ allow direct evaluation of τ and E, whereas the oracle $\mathcal{F}_{E^{-1}}$, on input (td, y) computes $E^{-1}(\tau(td), y)$. The associated trapdoor permutation $\mathcal{TP}^{\mathcal{F}} = (G, F, \overline{F})$ is such that the generation algorithm $G^{\mathcal{F}}$ chooses a random uniform trapdoor $td \leftarrow\!\!\$\ \{0,1\}^k$, and sets the public key $pk \leftarrow \tau(td)$. Furthermore, the algorithms $F^{\mathcal{F}}$ and $\overline{F}^{\mathcal{F}}$ simply call $\mathcal{F}_E$ and $\mathcal{F}_{E^{-1}}$, respectively, with their inputs, and return the corresponding output. Note that, even given the public key pk, in order to be able to use $\mathcal{F}_{E^{-1}}$ for inversion of $F(pk, \cdot)$ we are required to guess $\tau^{-1}(pk)$ given only access to τ, which is of course infeasible (at least without an extra oracle).

We show that there exists a *deterministic* adversary $\mathcal{A}$ making q queries from Dom (and hence of length $\log|Dom|$ bits each) and accessing an oracle $\mathcal{O} : \{0,1\}^* \to Dom$ such that $\mathbf{Adv}^{\text{kem-cca}}_{\mathsf{TDP\text{-}KEM}^R[\mathcal{TP}],R}(\mathcal{A}^{\mathcal{O},\mathcal{TP},R}) \geq 1 - \frac{1}{|\mathcal{K}|}$ for all $\mathcal{TP}$ and $\mathcal{O}$, but whenever $\mathcal{O}$ is a random oracle and $\mathcal{TP} = \mathcal{TP}^{\mathcal{F}}$, then

$$\mathbf{Adv}^{\text{owf}}_{\mathcal{TP}}(\mathsf{B}^{\mathcal{TP},R,\mathcal{A}^{\mathcal{O},\mathcal{TP},R_{\text{pub}}}}) \leq \frac{(q_{\mathcal{A}} q + 1) \cdot (2 q_{\mathcal{A}} q + q_F + q_R + 1)}{|Dom|} + \frac{q_{\mathcal{A}} q}{|\mathcal{K}|} + \frac{q_G + q_{\overline{F}}}{2^k}$$

for *all* reductions B as in the statement of the theorem, where in particular B can run $q_{\mathcal{A}}$ instances of $\mathcal{A}$ answering both its encapsulation and the decapsulation queries. The statement of the theorem is obtained by derandomizing $\mathcal{F}$ and $\mathcal{O}$ as described in Section 2.3.

Adversary Description. Ideally, we would like the (inefficient) adversary $\mathcal{A}$ to be capable of determining whether it is being run in the actual kem-cca-game, or whether it is being used by the reduction B in order to break the one-wayness of the underlying trapdoor permutation $\mathcal{TP}$. A naive approach consists of letting $\mathcal{A}$, on input the public key pk, choose a random $r \leftarrow\!\!\$\ Dom$ and compute $C \leftarrow F(pk, r)$; the adversary subsequently asks C to the decapsulation oracle,

obtaining a value K. Finally, it issues a query r to the random oracle R, and checks whether $R(r) = K$. In the affirmative case, it assumes that it is being used in the actual kem-cca-game, and proceeds in breaking the scheme, e.g., by inverting $F(pk, \cdot)$ on the challenge ciphertext and guessing the bit b by making an additional random oracle query. Otherwise, $\mathcal{A}$ just outputs a random guess. Intuitively, since B is efficient, it cannot retrieve r given only C, and thus must give some independent answer K' back to $\mathcal{A}$'s decapsulation query, such that $\mathcal{A}$'s check will then fail.

This argument, however, has two major fallacies. First, the randomness of $\mathcal{A}$ is determined by B, and thus r is chosen (and known) by B. Second, even provided a way for $\mathcal{A}$ to issue a decapsulation query for a ciphertext C with preimage r *unknown* to B, the reduction B can still first run $\mathcal{A}$ by giving a random answer to the decapsulation query, up to the point where the random-oracle check fails, and hence finding out r (as it is queried to R_{pub}). It subsequently *rewinds* $\mathcal{A}$ so that the same query C is issued, for which now B knows the right answer $R(r)$. This allows B to invert the underlying $\mathcal{TP}$, by just giving the challenge output y as the challenge ciphertext to $\mathcal{A}$'s encapsulation query.

We overcome both these problems by using a *random* oracle $\mathcal{O} : \{0,1\}^* \to \mathit{Dom}$ and considering the following adversary $\mathcal{A}$: On input the public key pk, it asks a sequence of decapsulation queries $C_1, C_2, \ldots, C_\ell$ (for $\ell = q - 1$), where C_i is computed by applying the random oracle to pk, to $C_1, \ldots, C_{i-1}$, and to the answers of the previous queries. (We assume that such inputs can injectively be mapped into bit strings.) Then, it checks the correctness of the answers $K_\ell, K_{\ell-1}, \ldots$ in *reverse* order (as above, it checks whether $K_i = R(F^{-1}(pk, C_i))$), but stops checking as soon as the first inconsistency is found. (This is crucial for the analysis to go through.) Finally, it behaves as above depending on whether all checks have been successful or not.

The main idea is that rewinding does not help when $\mathcal{O}$ is a random oracle, since (provided some unlikely events do not occur) the best strategy for B to build a run of an instance of $\mathcal{A}$ where the correctness check is always satisfied requires exponentially many (in ℓ) executions of $\mathcal{A}$. This is proven by showing an interesting connection to a tree-based, purely combinatorial, game. This approach is similar to the schedule used by Canetti et al. [6] to prove a lower bound on the round complexity of black-box concurrent zero-knowledge proofs.

6 FDH is not Provably Secure in the WPRO Model

In this section we consider the traditional full-domain hash signature scheme and show that one cannot prove it secure under randomly-programming reductions only.[10] Hence, a stronger version of programmability is required. We carry out our proof in the WPRO model and the result follows for randomly-programming reductions by the equivalence.

[10] In fact, we prove the slightly stronger statement that not even a ρ-dependent black-box reduction in the WPRO model exists for any *one-way* good function ρ.

Full-Domain Hash. We briefly recall the FDH-signature scheme. The scheme $\mathsf{FDH}^H[\mathcal{TP}] = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Ver})$ is based on a trapdoor permutation $\mathcal{TP} = (G, F, \overline{F})$. To sign a message $M \in \mathit{Msg}$ one computes $\sigma \leftarrow \overline{F}(sk, H(M))$ for hash function $H : \mathit{Msg} \to \mathit{Sig}$, and to verify one checks that $F(pk, \sigma) = H(M)$, where (pk, sk) are the keys generated through Kg. Below we consider a very weak unforgeability notion for FDH (called *wsig*), where the adversary has to forge a signature for a random message in a key-only attack. This strengthens our result as we show that even WPROM reductions from wsig to the one-wayness of the trapdoor permutation (*owp*) perform badly.

FDH Cannot be Secure in the WPROM. We have the following result, which states that FDH cannot be proven secure (by a black-box security analysis) in the WPROM.

Theorem 4 (WPROM Irreducibility of FDH). *Let $\mathsf{FDH}^R[\mathcal{TP}] = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Ver})$ be the FDH scheme with message space Msg and signature space Sig, relying on a trapdoor permutation $\mathcal{TP} = (G, F, \overline{F})$ with domain Sig and public-key/trapdoor space $\{0,1\}^k$, as well as on a random oracle $R : \mathit{Msg} \to \mathit{Sig}$. Then, for all $t > 0$, all $\epsilon \leq 1$, and all $(\text{wsig} \to \text{owp}, \delta, t, (q_G, q_F, q_{\overline{F}}), q_\rho, q_\mathcal{A})$-fully-BB WPROM security reductions B for FDH we have*[11]

$$\delta(\epsilon) \leq \frac{q_G + q_{\overline{F}}}{2^k} + \frac{q_F + 2q_\mathcal{A} + q_\rho + 2}{|\mathit{Sig}|},$$

where $q_G, q_F, q_{\overline{F}}$, and q_ρ are the number of queries of B to $G, F, \overline{F}$, and ρ, respectively, whereas $q_\mathcal{A}$ is the number of adversarial instances run by B.

The proof adopts a variant of the *two-oracle separation technique* by Hsiao and Reyzin [11]. For $\mathcal{F}$ and the *ideal* (i.e. random) trapdoor permutation $\mathcal{TP}^\mathcal{F} = (G, F, \overline{F})$ defined as in the proof of Theorem 3, we define for all functions ρ, an oracle $\mathcal{B} = \mathcal{B}_\rho^{\mathcal{TP}^\mathcal{F}}$ such that $\mathcal{TP}^\mathcal{F}$ is one way relative to $\mathcal{B}$ as long as ρ is one way, yet there exists an adversary $\mathcal{A}_{\mathsf{FDH}}$ forging an FDH-signature given access to $\mathcal{B}$ on *any* given message, i.e. it breaks FDH in the strongest possible sense.

Roughly speaking, the oracle $\mathcal{B}$ allows inversion of $F(pk, \cdot)$ on each output y' whenever a preimage r' of y' under ρ is exhibited: This allows inversion of $F(pk, \cdot)$ for *any* output of R_{adv}^ρ, and hence arbitrary forgeries in the WPROM. Yet, in the task of inverting $F(pk, \cdot)$ on a *random* y, coming up with a valid preimage of y under ρ is as hard as inverting ρ, and thus infeasible if ρ is one way. Therefore, the oracle $\mathcal{B}$ is only used to invert $F(pk, \cdot)$ for outputs *other than* the random challenge, which does not help it to win the OWF game.

References

1. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the Annual Conference on Computer and Communications Security (CCS). ACM Press (1993)

[11] We remark that wsig adversaries are only permitted to output one forgery, and perform no queries: the function δ hence only depends on ϵ without loss of generality.

2. Bellare, M., Rogaway, P.: Optimal asymmetric encryption — how to encrypt with RSA. In: Advances in Cryptology — Eurocrypt 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1994)
3. Bellare, M., Rogaway, P.: The exact security of digital signatures — how to sign with RSA and Rabin. In: Advances in Cryptology — Eurocrypt 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
4. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. SIAM J. Comput. 32(3), 586–615 (2003)
5. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. Journal of Cryptology 17(4), 297–319 (2004)
6. Canetti, R., Kilian, J., Petrank, E., Rosen, A.: Black-box concurrent zero-knowledge requires $\tilde{\Omega}(\log n)$ rounds. In: Proceedings of the Annual Symposium on the Theory of Computing (STOC) 2001. pp. 570–579. ACM Press (2001)
7. Coron, J.S.: On the exact security of full domain hash. In: Advances in Cryptology — Crypto 2000. LNCS, vol. 1880, pp. 229–235. Springer, Heidelberg (2000)
8. Dodis, Y., Oliveira, R., Pietrzak, K.: On the generic insecurity of the full domain hash. In: Advances in Cryptology — Crypto 2005. LNCS, vol. 3621, pp. 449–466. Springer, Heidelberg (2005)
9. Fujisaki, E., Okamoto, T., Pointcheval, D., Stern, J.: RSA-OAEP is secure under the RSA assumption. In: Advances in Cryptology — Crypto 2001. LNCS, vol. 2139. Springer, Heidelberg (2001)
10. Hofheinz, D., Kiltz, E.: Programmable hash functions and their applications. In: Advances in Cryptology — Crypto 2008. LNCS, vol. 5157, pp. 21–38. Springer, Heidelberg (2008)
11. Hsiao, C.Y., Reyzin, L.: Finding collisions on a public road, or do secure hash functions need secret coins? In: Advances in Cryptology — Crypto 2004. LNCS, vol. 3152, pp. 92–105. Springer, Heidelberg (2004)
12. Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Theory of Cryptography Conference (TCC) 2004. LNCS, vol. 2951, pp. 21–39. Springer, Heidelberg (2004)
13. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Advances in Cryptology — Crypto 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (2002)
14. Reingold, O., Trevisan, L., Vadhan, S.: Notions of reducibility between cryptographic primitives. In: Theory of Cryptography Conference (TCC) 2004. LNCS, vol. 2951, pp. 1–20. Springer, Heidelberg (2002)
15. Shoup, V.: A proposal for an ISO standard for public key encryption (version 2.1). No. 2001/112 in Cryptology eprint archive, `eprint.iacr.org` (2001)
16. Waters, B.: Efficient identity-based encryption without random oracles. In: Advances in Cryptology — Eurocrypt 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
17. Wee, H.: Zero knowledge in the random oracle model, revisited. In: Advances in Cryptology — Asiacrypt 2009. LNCS, vol. 5912, pp. 417–434. Springer, Heidelberg (2009)