Non-Adaptive Programmability of Random Oracle

Rishiraj Bhattacharyya * Pratyay Mukherjee†

Abstract

Random Oracles serve as an important heuristic for proving security of many popular and important cryptographic primitives. But, at the same time they are criticized due to the impossibility of practical instantiation. *Programmability* is one of the most important feature behind the power of Random Oracles. Unfortunately, in the standard hash functions, the feature of programmability is limited. In recent years, there has been endeavours to restrict programmability of random oracles. However, we observe that the existing models allow adaptive programming, that is, the reduction can program the random oracle adaptively in the online phase of the security game depending on the query received from the adversary, and thus are quite far from the standard model.

In this paper, we introduce a new feature called *non-adaptive* programmability of random oracles, where the reduction can program the random oracle only in the pre-processing phase. In particular, it might non-adaptively program the RO by setting a regular function as a post-processor on the oracle output. We call this new model Non-Adaptively-Programmable Random Oracle (NAPRO) and we show that this model is actually equivalent to so-called Non-Programmable Random Oracle (NPRO) introduced by Fischlin et al. [8], hence too restrictive.

However, we also propose a slightly stronger model, called *Weak-Non-Adaptively-Programmable Random Oracle* (WNAPRO), where in addition to non-adaptive programming, the reduction is allowed to *adaptively* extract some "auxiliary information" from the RO and this "auxiliary information " interestingly plays crucial role in the security proof allowing several important RO proofs to go through! In particular we prove the following results in WNAPRO model.

1. RSA-Full-Domain Hash signature scheme (RSA-FDH), and Boneh-Franklin ID-based encryption scheme (BF-IDE) are secure in the WNAPRO model. This is in sharp contrast to strong blackbox proofs of FDH schemes, where full programmability seems to be necessary.
2. Shoup's Trapdoor-permutation based Key-encapsulation Mechanism (TDP-KEM) *can not* be proven secure via blackbox reduction from ideal trapdoor-permutations in the WNAPRO model.

Keywords: Random Oracle; Programmability; RSA-FDH; TDP-KEM

*Rishiraj Bhattacharyya, Ecole Normale Superieure de Lyon/ INRIA, France, rishiraj.bhattacharyya@gmail.com
†Pratyay Mukherjee, Aarhus University, Denmark, pratyay85@gmail.com

1 Introduction

The Random Oracle Model, introduced by Bellare and Rogaway [2], has been a very popular paradigm for proving security of several important public key cryptosystems through the past two decades. In this model, in the security game, every party including the adversary has oracle access to a truly random function R. One then proves the security of a cryptosystem assuming the underlying hash function to be such a Random Oracle and subsequently replaces this random oracle by a suitably chosen cryptographic hash function H during the implementation. The well known Random Oracle heuristic asserts that if a scheme using R is secure, then it is secure using H.

The other extreme of this scenario is the so called standard model, in which, any hash function involved in the scheme is considered to be computable by efficient (i.e. poly-size) circuits. During the security proof, the adversary, along with the other public parameters, also receives this description. Moreover, all the computations on this hash function by the adversary are done by adversary herself. Naturally any security proof proved in the standard model is guaranteed to remain valid when the scheme is implemented using a suitably chosen public hash function H.

A drawback of the Random Oracle heuristic is that it is not clear which security property of the hash function can be preserved for secure implementation. In [5], Canetti, Goldreich and Halevi presented a counter example to this heuristic. They presented a valid signature scheme, albeit pathological, which can be proved secure in the random oracle model, yet is insecure after instantiation by any standard hash function. Goldwasser and Kalai [10] later showed that similar result holds for popular Fiat-Shamir Transformation. Furthermore, Pass [15] proved that zero knowledge proofs, proved secure in the random oracle model, may loose their deniability when instantiated with a fixed hash function.

In spite of those negative results, Random Oracle is still popular among the designers. This general purpose design methodology often helps to ensure efficiency. Moreover, some practical schemes, proven secure in the random oracle model, has withstand several cryptanalytic attempts so far. On the other hand, most of the existing schemes, proven to be secure in the standard model, are very inefficient and considered unusable in practice. From this practical importance, one natural approach is to find the limitations of Random Oracle heuristic and try to prove the security of these practical schemes in less radical setting. In this paper we propose an abstraction which arguably captures this approach.

Programmability. Programmability is a common feature of random oracle, which is exploited heavily in many security reductions. Consider a standard security reduction technique of some crypto-scheme where the reduction, having blackbox access to some adversary breaking security of the scheme, tries to break the underlying hardness assumption. In Random Oracle model, the reduction simulates the Random Oracle (by answering queries made by the adversary) during the security game. A common simulation strategy is to "program" or to adaptively "set" the output to a value of reduction's choice (often the challenge reduction received). Clearly the strategy is successful as long as the programmed output is indistinguishable from a uniformly distributed point.

In contrast, the adversary can compute the hash function herself in the standard model. The description of the hash function may depend on the underlying "hard" problem and the challenge received by the reduction. Once the hash function is deployed, often as a part of the public key, the reduction can not adaptively re-program the outputs. This seems to be a natural barrier for the random oracle model security proofs to remain meaningful in practice.

In recent years, a line of research has emerged to consider security models, restricting different 'unrealistic' features (including programmability) of random oracles. In this paper, we study whether we can restrict the programming of random oracles as a *non-adaptive* feature and still prove security of popular constructions. Precisely, we propose two weaker random oracle models, prove reductions of some interesting primitives in the proposed models and finally relate them to the existing weak models. Before describing our results in detail, we briefly review the existing works in this direction.

1.1 Related Works

In the literature, Several important work has been done prior to this work to reduce the gap between the ideal world (random oracle model) and real world (standard model). Mostly, the efforts were to propose some 'less idealistic' (equivalently 'more realistic') model than the random oracle model and observe the security of several (important) cryptographic primitives in that.

- In [14], for the first time, Nielsen pointed out the 'gap' between Programmable and Non-Programmable Random Oracle in the context of multi-party computation. He defined the Non-Programmable Random

Oracle model based on the real/ideal paradigm based framework proposed in the context of UC [4]. Precisely, he showed that the security proof of *Non-interactive Non-committing Encryption* scheme, which is secure in the (fully) Programmable Random Oracle model, fails in the *Non-Programmable* model. This result vividly showed the importance of programmability in random oracle model and at the same time posed as a big blow towards the complete practical realization of random oracles.

- In [17], Unruh proposed a variant of random oracle where *oracle-dependent* auxiliary input is allowed. In this setting, the adversary gets an auxiliary input that can contain information about the random oracle. In fact he argued with strong result that this variant should be preferred over the variant with *oracle-independent* auxiliary input. He showed that, one of the most popular encryption scheme *viz.* RSA-OAEP is IND-CCA2 secure in this variant of random oracle.

- In [11], Kiltz and Hofheinz introduced the concept of *Programmable Hash Function* (PHF). Orthogonal to our direction they mainly looked into the standard model. Intuitively, PHF is a superset of Fully-Programmable Random Oracles (the authors pointed out that random oracles can be viewed as PHFs with specific parameters). In this paper, we rather deal with some subset of the Fully-Programmable random oracle by restricting the adaptive programmability of the model. In spite of similarities in the proof techniques, we argue that our model is actually much more specific than theirs. The negative result of TDP-KEM (see Section 7 for details) is not known for generic PHFs.

- Finally, in [8], the authors showed quite a few results about different schemes in random oracle models without programmability and with *weak* programmability. They use a different approach from [14]. They classified random oracle models based on the ability to program: the Non-Programmable model (NPROM), the Weakly-Programmable model (WPROM) and the Fully-Programmable model (FPROM), among them the last one is essentially the random oracle most commonly dealt with. Based on their model they (mainly) showed the following results:

  1 In the security-proof of blackbox Full Domain Hash Signature scheme, the reduction needs 'full' programmability; which means it is secure only in the FPROM.

  2 Shoup's trapdoor permutation based Key-Encapsulation Algorithm (TDP-KEM) needs some form of "weak" programmability of the Random Oracle in its security-proof. This implies that it is secure in the WPROM (and FPROM), but not in NPROM.

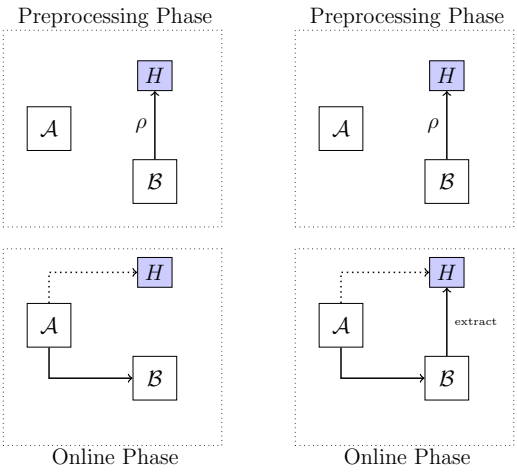
- In [1], Ananth and Bhaskar proposed a model where the reduction can not observe the interaction between the Random Oracle and the adversary. However, the reduction can fully program the oracle. In comparison, we allow the reduction to read the transcript of adversary's queries, but the programming of RO can be done only once in the pr-processing phase (and, of course, non-adaptively).

1.2 Our Contribution

In this paper we introduce the framework of non-adaptive programming of a random oracle. We present informal descriptions of the models considered in this paper below.

Non-Adaptively-Programmable Random Oracle. In this model, before publishing the public keys (we call preprocessing phase), the reduction can choose some *regular* function $\rho(\cdot) : \mathcal{D} \to \mathcal{R}$ with appropriate domains and initialize (only once in the entire game) the Random Oracle with ρ. Once the public key is published, the adversary can directly interact with the oracle via a public channel (we call online phase). The oracle works in the following way, when queried with an input x it chooses a random string $r \xleftarrow{\$} \mathcal{D}$ and compute $\rho(r)$. It returns $\rho(r)$ as the output. The distribution of r is independent to reduction's random coins and the input ρ. We stress that, the reduction *can not* modify the queries made by the adversary or the output of the oracle. However, he can query the oracle on input of his own choice. The reduction can also read adversary's queries and corresponding responses. We call this model the *Non-Adaptively-Programmable Random Oracle Model*.

Weak-Non-Adaptively-Programmable Random Oracle Model. In the second model, the reduction is additionally empowered with a *private* interface, which gives *read-only* access to the internal random coins of the oracle. Specifically, the reduction can make an "extract" query to get the sampled random string r corresponding to an hash query x. Rest of the condition is same as in the first model.



(a) Non-Adaptive Programming

(b) Weak Non-Adaptive Programming

Figure 1: Non-Adaptive-Programmability, The weak model has access to extract interface in the online phase. Here $\mathcal{A}$ and $\mathcal{B}$ are two PPT algorithms denoting the adversary and the reduction respectively.

Intuitively, even in the standard model, the hash function circuit can be thought of to be non-adaptively programmed to depend on the public key or the hard problems. Indeed, one may construct a standard model hash function as $\rho(H(.))$ where ρ is a scheme-dependent post-processing and H is a standard model hash function. However, in both our models, we consider H as Non-Programmable Random Oracle.

Equivalence with Non-Programmable Random Oracle. We argue that the 'amount of programmability' allowed by our NAPRO model is essentially minimal and substantiate this by showing an equivalence with the NPRO model of Fischlin et al. [8]. Our proof uses the fact that the programming can be done only once in the security game.

Power of Extraction. We show that interesting security reduction exists in the Weak Non-Adaptively Programmable Random Oracle (WNAPRO) Model. Specifically, we prove

1. RSA-Full Domain hash is existentially unforgeable when the underlying hash function is a WNAPRO.
2. The Boneh-Franklin Identity-Based encryption (IBE) scheme based on bilinear group (and Bilinear Diffi-Hellman assumption) is CCA secure in the WNAPRO model.

Weaker than full programmability Finally, we prove that the WNAPRO model is *strictly* weaker than the Fully-Programmable model. Specifically we show that the Trapdoor permutation based Key-encapsulation mechanism (TDP-KEM) *can not* be proven to be CCA secure in the WNAPRO model via blackbox reduction from any security property of a random permutation. Using techniques of Hsiao and Reyzin [12] we prove existence of such a reduction in the WNAPRO model would result in a reduction in the Non-Programmable Random Oracle Model, and thus would contradict the result of Fischlin et al. [8].

1.3 Comparison with Weak Programming of [8]

At a first look, our model (WNAPRO) might look similar to the Weakly-Programmable Random Oracle (WPRO) model of [8]. In that model too, the RO has an evaluation function $\rho : \mathcal{D} \to \mathcal{R}$ for some domain $\mathcal{D}$ and range $\mathcal{R}$. While queried with an input x, the RO chooses a random string $r \leftarrow \mathcal{D}$ and outputs $\rho(r)$.

However, the major difference is in *how the oracle is programmed*. In WPROM, although the reduction can not program the response to some specific value, he *can adaptively choose the random string* r, effectively based on his internal state, and program the output as $\rho(r)$ (as the function ρ is public) giving a notion of "random" albeit

adaptive programmability. Moreover, it is not hard to see that, to make this model weaker than Fully-Programmable model, ρ has to be one-way.

On the other hand, in our model, we allow the reduction to *non-adaptively* choose the function ρ but refrain to choose the random string r. In fact, the distribution of r is uniform and independent to reduction's state and transcript. The reduction can only read the sampled random string r for a message x. Hence, our model (WNAPRO) appears to be conceptually orthogonal to WPRO. This intuition is substantiated by the black-box reduction of RSA-FDH (exists in WNAPRO, but does not exist in WPRO), and TDP-KEM (exists in WPRO, does not exist in WNAPRO).

1.4 Organization

The rest of the paper is organized as follows: in Section 2, we review the notations and useful definitions. We formalize the proposed models in Section 3 followed by comparison with Non-Programmable Random Oracles in Section 4. We present the security proofs of RSA-FDH in Section 5, and Boneh-Franklin IBE in Section 6. Finally, we prove the blackbox separation of TDP-KEM in Section 7.

2 Notations and Preliminaries

Throughout this paper we follow the following notations. If $\mathcal{S}$ is a set, we denote the size of the set by $|\mathcal{S}|$. If x is chosen uniformly at random from any set $\mathcal{S}$, then it is denoted by $x \xleftarrow{\$} \mathcal{S}$. Unless otherwise mentioned we denote the security parameter by k. A function $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{R}$ is called a *negligible* function if for any polynomial function $p(\cdot)$ and for any positive integer $n \in \mathbb{N}$ we have $\varepsilon(n) < \frac{1}{p(n)}$. On the other hand a function $\alpha(\cdot) : \mathbb{N} \to \mathbb{R}$ is called a *non-negligible* function if for all negligible function $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{R}$ and for any positive integer $n \in \mathbb{N}$ we have $\alpha(n) > \varepsilon(n)$.

Now we provide the notational conventions regarding reduction, security etc. We mostly follow the notations established by [8]. 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 $\mathcal{B}$ with oracle access to adversary $A^{(\cdot)}$ (denoted $\mathcal{B}^{\mathcal{A}^{(\cdot)}}$) is allowed to do the following:

- At any time, $\mathcal{B}$ 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, $\mathcal{B}$ 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, $\mathcal{B}$ may halt the execution of the current copy of $\mathcal{A}$.

Note that the model is general enough so that $\mathcal{B}$ 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 $\mathcal{B}$ is given oracle access to $\mathcal{A}^{\mathcal{O}}$ for a particular oracle $\mathcal{O}$ (or some interface $\mathcal{O}.int$ of $\mathcal{O}$) (as opposed to $\mathcal{A}^{(\cdot)}$), then $\mathcal{B}$ 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 $\mathcal{B}$, and the second are not. Sometimes we will simply omit some of the oracles which are controlled by $\mathcal{B}$: 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 $\mathcal{A}^{\mathcal{O}_{\mathsf{pub}}}$ to mean the following: when $\mathcal{A}$ queries x to $\mathcal{O}$, x is forwarded to $\mathcal{B}$, 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 $\mathcal{B}$) and there is no way for $\mathcal{B}$ to influence it directly[1]. 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 $\mathbf{Adv}_f^{\Pi}(\mathcal{A})$ and f is said to be Π-secure if $\mathbf{Adv}_f^{\Pi}(\mathcal{A})$ is negligible function in the security parameter k which is denoted by $negl(1^k)$ for all PPT 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 $\mathbf{Adv}_{f^{\mathcal{O}}}^{\Pi}(\mathcal{A}^{\mathcal{O}}) = p \cdot \mathbf{Adv}_{f^{\mathcal{O}_1}}^{\Pi}(\mathcal{A}^{\mathcal{O}_1}) + (1-p) \cdot \mathbf{Adv}_{f^{\mathcal{O}_2}}^{\Pi}(\mathcal{A}^{\mathcal{O}_2})$ 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).

Reduction Loss Let Π and Π' be security properties. Let a primitive f be a Π-candidate. Let $\mathcal{S}[f]$ be a Π'-candidate cryptographic scheme relying on f. Let $\mathcal{A}$ be an Π'-adversary with advantage $\mathbf{Adv}_{\mathcal{S}[f]}^{\Pi'}(\mathcal{A})$. Let $\mathcal{B}^{f,\mathcal{A}}$ be a reduction from Π' to Π with advantage $\mathbf{Adv}_f^{\Pi}(\mathcal{B}^{f,\mathcal{A}})$. We define reduction loss of $\mathcal{B}$ as $\frac{\mathbf{Adv}_{\mathcal{S}[f]}^{\Pi'}(\mathcal{A})}{\mathbf{Adv}_f^{\Pi}(\mathcal{B}^{f,\mathcal{A}})}$.

Reductions in different Random Oracle models.

In general black-box reductions refer to *fully* black-box reductions as defined by Reingold et al. [16]. Here, we briefly review the reductions in different RO model established in [8].

Fully-Programmable Reductions. This is the standard setting where a reduction can *fully* program the RO without any restriction. This is the most common model of RO used in the security proofs in the literature. One can view these Random Oracles as having an interface called *program*. Whenever the adversary makes a query, the reduction can adaptively "program" the output using this interface.

Non-Programmable Reductions. In this type of reductions the reduction is not allowed to program the RO *at all*. The queries are answered (with perfectly random response as opposed to the case in the standard model) by the oracle directly independent of the reduction. While the reduction can learn the transcript of the all the queries (invoking a read transcript interface), it can not influence the response distribution.

Weakly-Programmable Reductions. This model lies somewhere between the above two. In this, the random oracle is associated with a *regular function* [2] $\rho : \mathcal{D}_\rho \to \mathcal{R}$. In general, for each query x to the RO, the oracle chooses an $r \xleftarrow{\$} \mathcal{D}$ and returns $\rho(r)$. However, the reduction is allowed some limited programming power as it can manipulate this output by using the programmability interface using a suitable random coin $r' \in \mathcal{D}$ of his own choice and forcing the oracle to output $\rho(r')$. It is easy to observe that, to make this oracle strictly weaker than Fully-Programmable RO, ρ needs to be one-way. For more detail, one may look into [8].

Some standard definitions.

In this section we recall the standard definitions of basic tools.

Trapdoor Permutations (TDPs)

Definition 2.1. A trapdoor permutation family is a triplet of PPTM $(\mathtt{G}, \mathtt{F}, \mathtt{F}^{-1})$. $\mathtt{G}$ is probabilistic and on input 1^n outputs a key-pair $(pk, td) \xleftarrow{\$} \mathtt{G}(1^k)$. $\mathtt{F}(pk, .)$ implements a permutation f_{pk} over $\{0,1\}^k$ and $\mathtt{F}^{-1}(td, .)$ implements the corresponding inverse f_{pk}^{-1}.

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

[2] By regular function we mean that, for all $y \in Range$ we have

$$|\{x \in Domain : \rho(x) = y\}| = \frac{|Domain|}{|Range|}. \tag{1}$$

INVERSION PROBLEM OF TDP FAMILY The most standard security property of TDP is *one-wayness* which says that it is hard to invert a random element without knowing the trapdoor even given the public key. Formally, for any PPTM A

$$\Pr[(pk, td) \xleftarrow{\$} \mathtt{G}(1^k), x \xleftarrow{\$} \{0,1\}^n : \mathcal{A}(f_{pk}(x), pk) = x] \leq negl(n).$$

3 NAPROM and WNAPROM

In this section we formalize our approach to program the random oracles non-adaptively. We define two variants of the Random Oracle Model where the oracle output can be programmed non-adaptively. We use the notations established in Sec. 2 (which in turn follows [8].).

NON-ADAPTIVE PROGRAMMING OF A RANDOM ORACLE. Let $\rho : \mathcal{D}_\rho \to \mathcal{R}$ be a regular function. Let $R : \mathcal{D} \to \mathcal{R}$ be a random oracle with the following interfaces

$\underline{R.init(\rho)}$

- If $L.init = 1$, return $\perp$.
- Initialize the list $L := \emptyset$.
- Set $L.\rho = \rho$, $L.init = 1$.

$\underline{R.eval(x)}$

- If $\exists (x, r) \in L$ for some r return $L.\rho(r)$. Otherwise go to the next step.
- Choose a random coin $r \xleftarrow{\$} \mathcal{D}_\rho$. Update $L := L \bigcup (x, r)$. Return $t = L.\rho(r)$.

$\underline{R.extract(x)}$

- If $x \in L$ return $(x, r, L.\rho(r))$
- Otherwise return $\perp$

R is defined by the list L and two initialization variables, $L.init$ and $L.\rho$. The variable $L.\rho$ ensures the programming of R and $L.init$ ensures that the programming can be done only once. From the implementation point of view, the variables $L.init$ and $L.\rho$ are stored in a Programmable Read Only Memory, whereas the list L is stored in the memory.

NAPROM AND WNAPROM. Let $\mathcal{S} = \mathcal{S}^R[f]$ be a cryptographic scheme relying on a primitive f and a random oracle $R : \mathcal{D} \to \mathcal{R}$. Let Π and Π' be security properties which are satisfied by $\mathcal{S}$ and f, respectively. We may use Π-$\mathcal{S}$ and Π'-f to concretely denote the respective security properties.

Now we define reductions in two models involving a non-adaptive programming of random oracles. We present a less parameter-heavy definition than [8] making it less general. We assume that the reduction and the adversary are PPT machine who asks polynomially many queries to any given oracle interface thus getting rid of the number of queries in the parameters. Moreover, we keep the definitions asymptotic instead of using concrete parameters.

Our first model is called NAPRO model where the reduction has access only to $R.init$ (to support non-adaptive programming) and $R.eval$.

Definition 3.1 (**Reduction in NAPRO model).** A $(\Pi \to \Pi')$-reduction in NAPRO model for a cryptographic scheme $\mathcal{S}$ is a PPT machine $\mathcal{B}^{(\cdot,\cdot)}$ with the property that for all Π' candidates f and for any PPT adversary $\mathcal{A}$ attacking the property Π of $\mathcal{S}^R[f]$, if $\mathbf{Adv}^{\Pi}_{\mathcal{S}^R[f]}(\mathcal{A}^R)$ is a non-negligible function of the security parameter k for a random oracle $R : \mathcal{D} \to \mathcal{R}$ then, $\mathbf{Adv}^{\Pi'}_{f}(\mathcal{B}^{(\mathcal{O}_1, \mathcal{A}^{\mathcal{O}_2})})$ must be a non-negligible function of k where $\mathcal{O}_1 \equiv (f, R.init, R.eval)$, $\mathcal{O}_2 \equiv R.eval$, and $\mathcal{B}^{(\cdot,\cdot)}$ invokes $R.init$ with a regular function ρ.

Now, we present the definition of a security reduction in the WNAPRO model, which lies somewhat between the NAPRO model and the FPRO model. In that, we give the reduction some more power compared to NAPROM.

Definition 3.2 (**Reduction in WNAPRO model).** A $(\Pi \to \Pi')$-reduction in WNAPRO model for a cryptographic scheme $\mathcal{S}$ is a PPT machine $\mathcal{B}^{(\cdot,\cdot)}$ with the property that for any PPT adversary $\mathcal{A}$ attacking Π and all Π' candidates f, if $\mathbf{Adv}^{\Pi}_{\mathcal{S}^R[f]}(\mathcal{A}^R)$ is a non-negligible function of the security parameter k for a random oracle $R : \mathcal{D} \to \mathcal{R}$ then, $\mathbf{Adv}^{\Pi'}_{f}(\mathcal{B}^{(\mathcal{O}_1, \mathcal{A}^{\mathcal{O}_2})})$ must be a non-negligible function of k where $\mathcal{O}_1 \equiv (f, R.init, R.eval, R.extract)$, $\mathcal{O}_2 \equiv R.eval$, and $\mathcal{B}^{(\cdot,\cdot)}$ invokes $R.init$ with a regular function ρ.

It is important to note that, in the security game in WNAPRO model, the reduction gets access to all three interfaces of the RO whereas the adversary can only access the interface $R.eval$. In both the cases, $R.init$ can be queried only in the initial phase and can *not* be "re-programmed". Importantly, even if the reduction rewinds the

oracle and supplies a set of new random coins, he can not re-program the oracle with a different ρ'. Although this makes the model seemingly very restrictive, we prove existence of interesting security reductions in this model.

More precisely, in WNAPRO model, only the reduction gets the sampled randomness r, which was input to the pre-defined function $\rho(\cdot)$ corresponding to some past hash-query x. Importantly, the adversary is *not* allowed to access this interface. We remark that this can be implemented by making $R.extract$ requiring a trapdoor. It remains an interesting open problem if this constraint can be removed and the adversary also gets access to the extract interface of R.

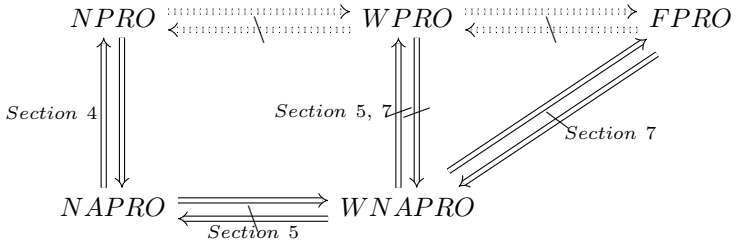


Figure 2: Relations between reductions in different RO models. Relations shown by dotted arrows were shown in [8].

4 Equivalence of security in NAPROM and NPROM

In this section we prove that security in NAPRO model is equivalent to security in NPRO model proposed in [8]. In particular we prove that for any security properties Π and Π, if there is a $(\Pi \to \Pi')$ reduction in NAPRO model, then we can construct a $(\Pi \to \Pi')$ reduction who wins the same security game in NPRO model with essentially the same asymptotic probability and vice versa.

This result might appear little counter-intuitive because in NAPROM, the reduction seems to have some power to program the random oracle by calling $R.init(\rho)$ with some regular ρ of his own choice, non-adaptively though and despite that the reduction does not get any significant advantage over NPRO model where, in contrast, any form of programmability is prohibited! Essentially this shows that even if the adversary gets to do some particular non-adaptive programming in the pre-processing phase, that turns out to be useless.[3]

First we re-define security reduction in NPRO model following our convention i.e. along the line of Def. 3.1, Def. 3.2. Before that let us define, for any Random Oracle R a new interface called $R.eval'$ as follows:

$\underline{R.eval'(x)}$

- If $\exists(x,t) \in L'$ for some t return t. Otherwise go to the next step.
- Choose a random output $t \xleftarrow{\$} \mathcal{R}$ from the codomain of R. Update $L' := L' \bigcup (x,t)$. Return t.

Note that, $R.eval'$ essentially implements the Non-Programmable Random Oracle.

Definition 4.1 (Reduction in NPRO model). A $(\Pi \to \Pi')$-reduction in NPRO model for a cryptographic scheme $\mathcal{S}$ is a PPT machine $\mathcal{B}^{(\cdot,\cdot)}$ with the property that for any PPT adversary $\mathcal{A}$ attacking Π and all Π' candidates f, if $\mathbf{Adv}^{\Pi}_{\mathcal{S}^{R}[f]}(\mathcal{A}^{R})$ is a non-negligible function of the security parameter k for a random oracle $R : \mathcal{D} \to \mathcal{R}$ then, $\mathbf{Adv}^{\Pi'}_{f}(\mathcal{B}^{(\mathcal{O}_1,\mathcal{A}^{\mathcal{O}_2})})$ must be a non-negligible function of k where $\mathcal{O}_1 \equiv (f, R.eval')$ and $\mathcal{O}_2 \equiv R.eval'$.

[3]It might be, however, possible that some other form of non-adaptive programming might make the RO strictly stronger than the NPRO. It is an interesting direction to find out such model.

Theorem 4.2. *Let ρ be a regular function and let $\mathcal{S}^f[R]$ be a cryptographic scheme based on primitive f and RO $R : \mathcal{D} \to \mathcal{R}$ such that Π is a security property of $\mathcal{S}$ and Π' is that of f. Then we have,*

1. *If there is a $(\Pi \to \Pi')$-NPRO reduction, then there exists a $(\Pi \to \Pi',)$-NAPRO reduction.*
2. *If there is a $(\Pi \to \Pi')$-NAPRO reduction, then there exists a $(\Pi \to \Pi')$-NPRO reduction.*

Proof. The first statement is intuitive and obvious. In the NAPRO model the reduction clearly has more power, in particular it can "embed" a regular function ρ into the RO which then computes the response to any query using that function. Observe that if $L.\rho$ is the identity function, the non-adaptively programming of a random oracle is essentially no programming at all. Given a reduction $\mathcal{B}_{NPRO}$ in the NPRO model, we can construct a reduction $\mathcal{B}_{NAPRO}$ in the NAPRO model, where $\mathcal{B}_{NAPRO}$ initializes the random oracle with the identity function, and invokes $\mathcal{B}_{NAPRO}$. The advantage of $\mathcal{B}_{NAPRO}$ is equal to the advantage of $\mathcal{B}_{NPRO}$.

To prove the second part we are given that there is a PPT reduction $\mathcal{B}_{NAPRO}$ in NAPRO model which implies that for a random oracle $R : \mathcal{D} \to \mathcal{R}$ and for any PPT Π-adversary $\mathcal{A}$, when $\mathbf{Adv}^{\Pi}_{\mathcal{S}^{R}[f]}(\mathcal{A})$ is non-negligible (in k) then $\mathbf{Adv}^{\Pi'}_{f}(\mathcal{B}^{((f,R.init,R.eval),\mathcal{A}^{R.eval})})$ is also non-negligible. Now we have to show the existence of a reduction $\mathcal{B}_{NPRO}$ in the NPRO model such that $\mathbf{Adv}^{\Pi'}_{f}(\mathcal{B}^{((f,R.eval'),\mathcal{A}^{R.eval'})}_{NPRO})$ becomes non-negligible. We construct the reduction $\mathcal{B}_{NPRO}$ as follows: $\mathcal{B}_{NPRO}$ runs the reduction $\mathcal{B}_{NAPRO}$ and let it interact with $\mathcal{A}$ thus simulating the NAPRO environment. In order to do that $\mathcal{B}_{NPRO}$ must simulate the interfaces $\mathcal{B}_{NAPRO}$ has access to, namely *R.init* and *R.eval* only having access to *R.eval′*. It "simulates" the interfaces as follows:

1. *R.init*(ρ): $\mathcal{B}_{NPRO}$ just receives the description of a regular function ρ, if it fails output $\perp$, otherwise *do nothing*.
2. *R.eval*(x): $\mathcal{B}_{NPRO}$ takes the input x from $\mathcal{B}_{NAPRO}$ and then just forwards it to *R.eval′*. It outputs whatever response it gets back from *R.eval′*.

Finally $\mathcal{B}_{NPRO}$ outputs what $\mathcal{B}_{NAPRO}$ outputs.

We claim that the simulated NAPRO model is statistically indistinguishable from the actual NAPRO model. In particular the view of the reduction $\mathcal{B}_{NAPRO}$ and the view of the adversary $\mathcal{A}$ is unchanged.

More formally, for a random oracle R, programmed with a fixed regular function $\rho : \mathcal{D}_\rho \to \mathcal{R}$,

$$\Pr[R.eval(x) = y] = \Pr_{r \xleftarrow{\$} \mathcal{D}_\rho}[\rho(r) = y] = \frac{1}{|Rng|} = \Pr_{t \xleftarrow{\$} \mathcal{R}}[t = y] = \Pr[R.eval'(x) = y]$$

Hence for all regular $\rho : \mathcal{D}_\rho \to \mathcal{R}$ the output distribution of *R.eval* and *R.eval′* (and hence the view of $\mathcal{B}_{NAPRO}$ and $\mathcal{A}$) is in fact identical[4]. So, the success probability of $\mathcal{B}_{NAPRO}$ remains the same in the simulated environment. Hence, the advantage of $\mathcal{B}_{NPRO}$ is exactly the advantage of $\mathcal{B}_{NAPRO}$.

□

5 RSA-Full Domain Hash Signature in WNAPRO model

In this section we prove that, the RSA Full-domain-hash signature scheme (RSA FDH in short), introduced in [2], is secure against *no-message attack* as well as *chosen message attack* in the WNAPRO model. In particular there is a black-box reduction to the hardness of RSA-problem in the WNAPRO model. First, we recall the standard definitions of a signature scheme and its security.

5.1 Signature Schemes

A signature scheme $(\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ is defined as follows:

- The *key generation algorithm* `Gen` is a probabilistic algorithm which given 1^k, outputs a pair of matching public and private keys, (pk, td).

- The *signing algorithm* `Sign` takes the message M to be signed, the public key pk and the private key td, and returns a signature $\sigma = \texttt{Sign}_{td}(M)$. The signing algorithm may be probabilistic.

[4]Notice that this is not true if the reduction could re-program ρ or could access internal variables of the oracles.

- The *verification algorithm* $\texttt{Verify}$ takes a message M, a candidate signature σ' and pk. It returns a bit $\texttt{Verify}_{pk}(M, \sigma')$, equal to 1 if the signature is accepted, and 0 otherwise. We require that if $\sigma \leftarrow \texttt{Sign}_{td}(M)$, then $\texttt{Verify}_{pk}(M, \sigma) = 1$.

5.2 Security of a Signature Scheme

A signature scheme is called *unforgeable against no-message attack* (UF-NMA) if the forger (i.e. the adversary), only getting the public key (without getting any other message signature pair), is unable to produce a valid signature of a message of her own choice. On the other hand, if the forger is unable to forge even when it can adaptively obtain signatures of messages of her choice then we call the scheme *existentially unforgeable under an adaptive chosen message attack*. In that case, a *valid forgery* is a message/signature pair (M, x) such that $\texttt{Verify}_{pk}(M, x) = 1$, and the signature of M was never requested by the forger. In Figure 3, we describe the a UF-NMA game in a random oracle model, i.e. when the hash function H involved is assumed to be a random oracle.

Experiment $Game^{nma}_{SIG,\mathcal{A}^H}$
Compute $(pk, sk) \leftarrow \texttt{Gen}(1^k)$;
Receive $(M^*, \sigma^*) \leftarrow \mathcal{A}^H(pk)$;
Compute $b \leftarrow \texttt{Verify}^H(pk, M^*, \sigma^*)$;
Return b.

Figure 3: No-message attack Game of a Signature Scheme

5.3 RSA inversion problem

We recall the RSA-inversion game for any PPT adversary $\mathcal{A}$ in Fig. 4.

Experiment $Game^{inv}_{RSA,\mathcal{A}}$
Run $(N, e, d) \leftarrow \texttt{Gen}(1^k)$;
Choose $x \xleftarrow{\$} \mathbb{Z}^*_N$;
Set $y := x^d (\mathrm{mod} N)$;
Receive $x^* \leftarrow \mathcal{A}(y, N, e)$;
If $x = x^*$,
return 1;
Else
return 0.

Figure 4: Security game of *inv*-RSA problem

The advantage of the adversary $\mathcal{A}$ is given by:

$$\mathbf{Adv}^{inv}_{\mathsf{RSA}}(\mathcal{A}) = \Pr[x^* = x]$$

5.4 RSA-Full Domain Hash.

We recall the RSA-FDH scheme of [2].

Construction 5.1 (The RSA-Full Domain Hash Signature Scheme). Let $\texttt{Gen}$ be the algorithm which, on input the security parameter 1^k, outputs a RSA-triple (N, e, d). *The RSA-Full Domain Hash* (RSA-FDH in short) signature scheme is defined as a triple of algorithms $(\texttt{Gen-FDH}, \texttt{Sign-FDH}, \texttt{Verify-FDH})$. The algorithms $\texttt{Sign-FDH}$ and $\texttt{Verify-FDH}$ has access to a hash-oracle $H : \{0,1\}^* \to \mathbb{Z}^*$. The algorithms are described as follows:

$\underline{\texttt{Gen-FDH}(1^k)}$
Compute $(N, e, d) \leftarrow \texttt{Gen}(1^k)$*;*
Set $pk := (N, e)$*;*
Set $sk := (N, d)$*;*
Return key-pair (pk, sk)*.*

$\underline{\texttt{Sign-FDH}^H(sk, m)}$
Set $y := H(m)$*;*
Set $\sigma := y^d \bmod N$*;*
Return σ*.*

$\underline{\texttt{Verify-FDH}^H(pk, m, \sigma)}$
Set $y \leftarrow \sigma^e \bmod N$*;*
Set $y' \leftarrow H(m)$*;*
If $y = y'$ *then return 1*
Else return 0

In Fig. 3, the advantage of a PPT adversary $\mathcal{A}$ is given by:

$$\mathbf{Adv}^{nma}_{\mathsf{FDH}^H[\mathsf{RSA}]}\left(\mathcal{A}^H\right) = \Pr\left[\mathsf{Ver}(M^*, \sigma^*) = 1\right]$$

where H is assumed to be a random oracle such that $H : \{0,1\}^* \to \mathbb{Z}^*$.

Now we are ready to state and prove the existence of a PPT reduction when H is a WNAPRO

Theorem 5.2. *There exists a PPT (nma-FDH $\to$ inv-RSA)-reduction in WNAPRO model with reduction loss* 1.

Proof. Let $\mathcal{A}$ be a UF-NMA adversary with advantage $\mathbf{Adv}^{nma}_{\mathsf{FDH}^H[\mathsf{RSA}]}\left(\mathcal{A}^H\right) = \varepsilon'$ against RSA-FDH for a RO H and $\mathcal{B}$ be a PPT algorithm such that the advantage of $\mathcal{B}$ is given by,

$$\mathbf{Adv}^{inv}_{\mathsf{RSA}}\left(\mathcal{B}^{(H.init, H.eval, H.extract), \mathcal{A}^{H.eval}}\right) = \varepsilon.$$

We will show that $\varepsilon' = \varepsilon$ which essentially proves the theorem.

Recall that, in WNAPRO model, the reduction has the power to initialize the Random Oracle with a function $\rho(\cdot)$, only subject to the condition that, ρ must be regular, and this initialization can be done only once in the game. However, the reduction may make an extract query to get the random strings, used as input to ρ. Recall that, in UF-NMA game, the adversary has to forge a signature without making any sign query. The objective of the reduction is to invert a RSA-challenge point y using the adversary.

Setup. Consider the RSA-inversion game $Game^{inv}_{RSA,\mathcal{B}}$ (c.f. Figure 4) where $\mathcal{B}$ receives the public parameter of the RSA problem (N, e) and the challenge y.

Programming NAPRO She describes the regular function $\rho(\cdot)$ as follows:

$$\rho(r) \stackrel{def}{=} y.r^e \bmod N.$$

$\mathcal{B}$ queries $H.init(\rho)$ to initialize the RO H. Note that, ρ, as define above is regular.

$\mathcal{B}$ runs $\mathcal{A}$ with input (N, e) as the public key of the signature scheme.

Forgery. Suppose $\mathcal{A}$ outputs a forgery (M, σ). Without loss of generality, we assume $\mathcal{A}$ has queried $H.eval$ with M. Otherwise, $\mathcal{B}$ on receiving the forgery can query $H.eval(M)$. Now, using the idea of [6],

$$\sigma^e = y \cdot r^e \bmod N \implies y = \left(\frac{\sigma}{r}\right)^e \bmod N$$

where $H.eval(M) = \rho(r)$. The reduction **queries** $H.extract(M)$ **to recover the** r and output $\left(\frac{\sigma}{r}\right) \bmod N$.

Clearly, $\mathcal{B}$ succeeds to invert y, whenever $\mathcal{A}$ outputs a valid forgery. Hence $\varepsilon' = \varepsilon$. □

Security against Chosen Message Attack. One can similarly prove the security of RSA-FDH against chosen message attacks (UF-CMA) in the WNAPRO model. Let $\mathcal{A}$ be the adversary who asks q_s signing query for any polynomial $q_s(k)$. In Fig. 5 the security game is shown.

Experiment $Game^{cma}_{SIG,\mathcal{A}^H}$
Compute $(pk, sk) \leftarrow \mathtt{Gen}(1^k)$;
Initialize $\mathsf{st} := \{pk\}$;
For $i = 1 \to q_s$;
Receive $M_i \leftarrow \mathcal{A}^R(\mathsf{st})$
Compute $\sigma_i = \mathsf{Sign}(sk, M_i)$;
Update $\mathsf{st} = \mathsf{st} \cup \{(M_i, \sigma_i)\}$
Receive $(M^*, \sigma^*) \leftarrow \mathcal{A}^R(\mathsf{st})$;
Compute $b \leftarrow \mathtt{Verify}^R(pk, M^*, \sigma^*)$;
Return b.

Figure 5: Chose-message attack Game of a Signature Scheme

the advantage of a PPT adversary $\mathcal{A}$ is given by:

$$\mathbf{Adv}^{cma}_{\mathsf{FDH}^H[\mathsf{RSA}]}\left(\mathcal{A}^H\right) = \Pr\left[\mathsf{Ver}(M^*, \sigma^*) = 1\right]$$

.

Formally we prove the following theorem.

Theorem 5.3. *There exists a PPT (cma-*FDH *→ inv-*RSA*)-reduction in WNAPRO model with a reduction loss of $\mathcal{O}(q_s)$, where q_s is the number of sign queries in the cma-*FDH *game.*

Proof. Let $\mathcal{A}$ be a UF-CMA adversary with advantage $\mathbf{Adv}^{cma}_{\mathsf{FDH}^{H}[\mathsf{RSA}]}\left(\mathcal{A}^{H}\right) = \varepsilon'$ against RSA-FDH for a RO H and $\mathcal{B}$ be a PPT algorithm such that the advantage of $\mathcal{B}$ is given by,

$$\mathbf{Adv}^{inv}_{\mathsf{RSA}}\left(\mathcal{B}^{(H.init,H.eval,H.extract),\mathcal{A}^{H.eval}}\right) = \varepsilon.$$

We will show that $\varepsilon \geq \frac{\varepsilon'}{e(1+q_s)}$ (where e is the base of natural logarithms and $q_s = O(poly(k))$ be the number of signing query) which essentially proves the theorem.

In case of UF-CMA security, the reduction also need to answer sign queries. We can apply the method of [6], with carefully defining ρ.

Setup. $\mathcal{B}$ receives the public parameter of the RSA problem (N, e) and the inversion-challenge y. Let $\ell = \log(q_s + 1)$. Define $\mathcal{D} \stackrel{def}{=} \mathbb{Z}_N^* \times \{0,1\}^\ell$.

Programming NAPRO. Let $\rho_1 : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ be defined as $\rho_1(r) = yr^e \bmod N$ and $\rho_2 : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ be defined as $\rho_2(r) = r^e \bmod N$. Finally $\rho : \mathcal{D} \to \mathbb{Z}_N^*$ is defined as

$$\rho(r\|z) \stackrel{def}{=} \begin{cases} \rho_1(r) & \text{if } z = 0^\ell \\ \rho_2(r) & \text{if Otherwise} \end{cases}.$$

$\mathcal{B}$ initializes R with the above-defined ρ, and runs $\mathcal{A}$ with input (N, e) as the public key of the signature scheme.

Simulating sign queries. For each sign query m_i, $\mathcal{B}$ queries $H.eval(m_i) = t_i$ and $H.extract(m_i) = r_i\|z_i$. Now if $t_i = r_i^e \bmod N$ (equivalently $z_i \neq 0^\ell$), $\mathcal{B}$ answers with r_i, otherwise she aborts.

Forgery to inversion. Finally, when $\mathcal{A}$ submits a forgery (m^*, σ^*), $\mathcal{B}$ queries $H.extract(m^*) = r^*\|z^*$, checks whether $r^* = \sigma^*$ (or $z^* \neq 0^\ell$). If yes, $\mathcal{B}$ aborts. Otherwise $\mathcal{B}$ outputs $\frac{\sigma^*}{r^*} \bmod N$ as output.

In WNAPRO model, the adversary does not have access to the extract interface. Both ρ_1 and ρ_2 are regular functions. The output distributions of R when $z = 0^\ell$ and $z \neq 0^\ell$ are indistinguishable in the view of $\mathcal{A}$. Hence,

$$\begin{aligned} \varepsilon &= \Pr[\mathcal{B} \text{ wins RSA-invert}] \\ &= \Pr[\mathcal{A} \text{ wins UF-CMA}] \cdot \Pr[z^* = 0] \cdot \prod_{i}^{q_s} \Pr[z_i \neq 0] \end{aligned}$$

As $H.eval$ samples the strings $r_i\|z_i$ independently and uniformly, $\Pr[z_i \neq 0] = 1 - \frac{1}{q_s+1}$ (since $\ell = \log(q_s + 1)$). Hence $\varepsilon = (1 - \frac{1}{q_s+1})^{q_s} \cdot \frac{1}{q_s+1} \geq \frac{1}{e(1+q_s)}$. □ □

6 The Boneh-Franklin IBE Scheme is secure in WNAPROM

In [3], Boneh and Franklin proposed a fully functional identity-based encryption scheme and showed the scheme has chosen ciphertext security in the random oracle model assuming a variant of the computational Diffie-Hellman problem. In this section we show that, the security of the Boneh-Franklin IBE scheme remains intact in WNAPRO model.

In [3], authors introduced new notions of security like IND-ID-CCA and IND-ID-CPA security which are applicable to IBE-schemes only. They defined a basic IBE-scheme called BasicIdent and proved that, this scheme is IND-ID-CPA secure in the random oracle model. Applying the *Fujisaki-Okamoto* transformation (detail in [9]) to BasicIdent, one gets FullIdent which was proved to be IND-ID-CPA secure in the random oracle model. The scheme BasicIdent needs two hash-oracles H_1 and H_2 and FullIdent needs another two hash-oracles, namely H_3 and H_4. We observe that, all the other random oracles except H_1 do not need any programmability at all. Hence, to prove the security, it is sufficient to prove the IND-ID-CPA security of BasicIdent when the oracle H_1 is a WNAPRO.

First we recall the standard definition of an IBE scheme and IND-ID-CPA security.

Identity-Based Encryption

An IBE-scheme $\mathcal{E}$ is specified by four algorithms: `Setup`, `Extract`, `Enc`, `Dec`. Briefly describing, `Setup` generates the system parameters (par) and the master-key (K_M); `Extract` takes those as input along with a given public-key

$ID \in \{0,1\}^*$ returning the corresponding private key d. $\texttt{Enc}$ and $\texttt{Dec}$ behaves like standard public-key scheme. The correctness requirement is:

$$\forall\ M \in \mathcal{M},\ \texttt{Dec}(par, C, d) = M \text{ where } C = \texttt{Enc}(par, M, ID),\ d = \texttt{Extract}(ID, par, K_M) \tag{2}$$

Security of Identity-Based Encryption

Since we are only dealing with the proof of BasicIdent, we briefly describe the security notion IND-ID-CPA which is similar to the standard CPA security notion with only exception that, the adversary can query the challenger with some chosen public identity ID and get the corresponding private key d from the challenger. The security game is depicted in Figure 6.

Experiment $Game^{id-cpa}_{IBE,A}$
Compute $(par, K_M) \leftarrow \mathsf{Setup}(1^k)$;
Set $\mathsf{st}_{\mathcal{A}} := \emptyset$;
Update $\mathsf{st}_{\mathcal{A}} := \mathsf{st}_{\mathcal{A}} \cup \{par\}$;
For $i = 1 \rightarrow m$ do the following:
Receive $ID_i \leftarrow \mathcal{A}^R(\mathsf{st}_{\mathcal{A}})$;
Compute $d_i \leftarrow \mathsf{Extract}(Id_i, par, K_M)$;
Update $\mathsf{st}_{\mathcal{A}} := \mathsf{st}_{\mathcal{A}} \cup \{(ID_i, d_i)\}$;
Receive $(M_0, M_1, ID) \leftarrow \mathcal{A}^R(\mathsf{st}_{\mathcal{A}})$;
If $ID \in \{ID_1, ID_2, \ldots, ID_m\}$ abort ;
Else
Choose $b \xleftarrow{\$} \{0,1\}$;
Update $\mathsf{st}_{\mathcal{A}} := \mathsf{st}_{\mathcal{A}} \cup \{\texttt{Enc}(par, M_b, Id)\}$;
For $i = m+1 \rightarrow n$;
Receive $ID_i \leftarrow \mathcal{A}^R(\mathsf{st}_{\mathcal{A}})$;
If $ID_i = ID$ Abort ;
Else
Compute $d_i \leftarrow \texttt{Extract}(Id_i, par, K_M)$;
Update $\mathsf{st}_{\mathcal{A}} := \mathsf{st}_{\mathcal{A}} \cup \{(ID_i, d_i)\}$;
Receive $b' \leftarrow \mathcal{A}^R(\mathsf{st}_{\mathcal{A}})$;
If $b = b'$ return 1;
Else return 0;

Figure 6: IND-ID-CPA security game of IBE scheme

The advantage of the adversary $\mathcal{A}$ is $\mathbf{Adv}_{\mathcal{E},\mathcal{A}}(k) = \left|\Pr[b = b'] - \frac{1}{2}\right|$. So, we consider the IBE scheme $\mathcal{E}$ is secure if this advantage is negligible in terms of the security parameter.

6.1 Bilinear Diffie-Hellman Assumption

The proof provided in [3], is based on the hardness assumption of *Bilinear Diffie-Hellman* (BDH) problem. We briefly describe the assumption here skipping most of the details. Let $\mathbb{G}_1$ be an additive group and $\mathbb{G}_2$ be a multiplicative group, both of order q (a large prime.), $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be an *admissible* bilinear map, and P be a generator of group $\mathbb{G}_1$. Let $\mathcal{G}$ be the BDH generator which is a randomized algorithm and generates the descriptions of $\mathbb{G}_1, \mathbb{G}_2, \hat{e}$. The BDH problem in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$ is as follows: Given (P, aP, bP, cP) for some $a, b, c \in \mathbb{Z}_q^*$, compute $\hat{e}(P,P)^{abc}$. The BDH-security game is described in Figure 7.

The advantage of $\mathcal{A}$ in the BDH-security game is defined as:

$$\mathbf{Adv}^{solve}_{\mathsf{BDH}}(\mathcal{A}) = \Pr\left[W = \hat{e}(P,P)^{abc}\right]$$

6.2 The scheme BasicIdent

Now we describe the scheme BasicIdent.

Experiment $Game^{solve}_{BDH,\mathcal{A}}$
Compute $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle \leftarrow \mathcal{G}(1^k)$;
Choose $P \xleftarrow{\$} \mathbb{G}_1^*$;
Choose $(a, b, c) \xleftarrow{\$} \mathbb{Z}_q^*$;
Receive $W \leftarrow \mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, aP, bP, cP)$;
If $W = \hat{e}(P, P)^{abc}$,
return 1;
Else
return 0;

Figure 7: The BDH security game

Construction 6.1 (The IBE Scheme BasicIdent)**.** The IBE scheme *BasicIdent* is defined as a 4-tuple of algorithms $\langle \texttt{Setup}, \texttt{Extract}, \texttt{Enc}, \texttt{Dec}\rangle$. Let k be the security parameter and $\mathcal{G}$ is the BDH-generator generating the BDH-parameters. The algorithms have access to two hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$ and $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ for some fixed n. The message space is $\{0,1\}^n$. The algorithms are described as follows:

$\texttt{Setup}(1^k)$
Compute $(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, P) \leftarrow \mathcal{G}$;
Choose $s \xleftarrow{\$} \mathbb{Z}_q^*$;
Set $P_{pub} := sP$;
Set $par := (q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, n, P_{pub})$;
Set $K_M \leftarrow s$;
Return (par, K_M)

$\texttt{Extract}(ID, K_M, par)$
Set $Q_{ID} := H_1(ID)$;
Set $d_{ID} := sQ_{ID}$;
Return (Q_{ID}, d_{ID})

$\texttt{Enc}(par, ID, M)$
Set $Q_{ID} := H_1(ID)$;
Choose $r \xleftarrow{\$} \mathbb{Z}_q^*$;
Compute $C \leftarrow \langle rP, M \oplus H_2(g_{ID}^r)\rangle$
(where $g_{ID} = \hat{e}(Q_{ID}, P_{pub})$*)*;
Return C

$\texttt{Dec}(par, d_{ID}, C)$
Let $\langle U, V\rangle \leftarrow C$;
Compute $M \leftarrow V \oplus H_2(\hat{e}(d_{ID}, U))$;
Return M

The advantage of adversary $\mathcal{A}$ against the above IBE in the security game shown in Fig. 6 is given by:

$$\mathbf{Adv}^{id\text{-}cpa}_{\mathsf{BASICID}^{H_1,H_2}[\mathsf{BDH}]}\left(\mathcal{A}^{H_1,H_2}\right) = \left|\Pr[b = b'] - \frac{1}{2}\right|$$

6.3 Security of BasicIdent in WNAPROM

In this section we show that BasicIdent remains secure when the hash functions are modeled as WNAPRO.

Theorem 6.2. *There exists a PPT (id-cpa-*BASICID $\to$ *solve-*BDH*)-reduction in the WNAPRO model with reduction loss of* $\mathcal{O}(q_E \cdot q_{H_2})$ *where* q_E *is the number of key extraction queries and* q_H *is the number of Random Oracle queries made by the adversary in the id-cpa-*BASICID *game.*

Proof. The proof mostly follows [3] with necessary adaptation for weak non adaptive programming. We include the full proof for completeness.

Let $\mathcal{A}$ be an IND-ID-CPA adversary to BasicIdent with advantage, $\mathbf{Adv}^{id\text{-}cpa}_{\mathsf{BASICID}^{H_1,H_2}[\mathsf{BDH}]}\left(\mathcal{A}^{H_1,H_2}\right) = \varepsilon'$ for two random oracles H_1, H_2. Suppose $\mathcal{A}$ makes at most q_E key extraction queries and q_{H_2} eval queries to H_2, (both quantities are polynomially bounded in terms of the security parameter). Let $\mathcal{B}$ be a PPT adversary solving BDH problem such that the advantage of $\mathcal{B}$ is given by,

$$\mathbf{Adv}^{solve}_{\mathsf{BDH}}\left(\mathcal{B}^{(H.init,H.eval,H.extract),\mathcal{A}^{H.eval}}\right) = \varepsilon.$$

We will show that

$$\varepsilon \geq \frac{2\varepsilon'}{e(1+q_E)\cdot q_{H_2}}$$

(where e is the base of natural logarithms) which essentially proves the theorem.

The first step is to construct a public key scheme called BasicPub and show that IND-ID-CPA attack on BasicIdent can be converted into a IND-CPA attack on BasicPub. Then we, following [3], prove that, an attacker to IND-CPA security of BasicPub reduces to an adversary winning BDH security game.

The Public-key Encryption Scheme BasicPub.

BasicPub is defined as a triple of algorithms $\langle \mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec}\rangle$. Let k be the security parameter and $\mathcal{G}$ is the BDH-generator generating the BDH-parameters. The algorithms have access to hash function $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ for some n. The algorithms are described as follows:

Public-Key Encryption Scheme BasicPub.

$\underline{\mathtt{Gen}(1^k)}$
Compute $(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, P) \leftarrow \mathcal{G}$;
Choose $s \xleftarrow{\$} \mathbb{Z}_q^*$;
Set $P_{pub} := sP$;
Choose $Q_{ID} \xleftarrow{\$} \mathbb{G}_1^*$;
Set $pk := (q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, n, P_{pub}, Q_{ID})$;
Set $d_{ID} := sQ_{ID}$;
Set $sk := d_{ID}$;
Return (pk, sk).

$\underline{\mathtt{Enc}(pk, M)}$
Choose $r \xleftarrow{\$} \mathbb{Z}_q^*$;
Compute $C \leftarrow \langle rP, M \oplus H_2(g_{ID}^r)\rangle$
(where $g_{ID} = \hat{e}(Q_{ID}, P_{pub})$);
Return C.

$\underline{\mathtt{Dec}(sk, C)}$
Let $\langle U, V\rangle \leftarrow C$;
Compute $M \leftarrow V \oplus H_2(\hat{e}(d_{ID}, U))$;
Return M.

6.3.1 From BasicIdent to BasicPub

Lemma 6.3. *Let H_1 be a WNAPRO from $\{0,1\}^*$ to $\mathbb{G}_1^*$. Let $\mathcal{A}$ be and IND-ID-CPA adversary against basicident with advantage ε'. Suppose $\mathcal{A}$ makes at most q_E extract queries. There exists an IND-CPA $\mathcal{A}'$ with advantage δ against BasicPub where*

$$\delta \geq \frac{\varepsilon'}{e(1+q_E)}$$

Proof. The proof idea is similar to that of FDH, albeit in the pairing setting. Let C_{cpa} be the challenger of IND-CPA game of BasicPub. The game starts with the random generation of the public parameter pk and the private parameter sk by running $\mathtt{Gen}(1^k)$ by C_{cpa}. More elaborately, it generates $pk = (q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, n, P_{pub}, Q_{ID})$ in addition to a WNAPRO H_2, and sets $sk = d_{ID} = sQ_{ID}$. Then C_{cpa} gives the pk to $\mathcal{A}'$. $\mathcal{A}'$ will eventually return two messages (M_0, M_1) to C_{cpa}, followed by C_{cpa} choosing a random bit b and giving back the ciphertext $C_b = \mathtt{Enc}(pk, M_b)$. At the last stage $\mathcal{A}'$ returns its guess b' and will win if, $b = b'$. The algorithm $\mathcal{A}'$ has oracle access to $\mathcal{A}$.

Setup of simulated BasicIdent. When, $\mathcal{A}'$ receives the pk from C_{cpa}, it gives the system parameter $par = (q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, n, P_{pub})$ to $\mathcal{A}$ in addition to the WNAPRO H_1 and H_2. Except H_1, all the other parameters are taken from pk. And, H_1 is a WNAPRO controlled by $\mathcal{A}'$.

Initialize H_1. Now, the main trick is how the algorithm $\mathcal{A}'$ would define H_1 to make the proof working. Let $\ell = \log(q_E + 1)$. Define $\mathcal{D} \stackrel{def}{=} \mathbb{Z}_q^* \times \{0,1\}^\ell$. $\rho_1 : \mathbb{Z}_q^* \to \mathbb{G}_1^*$ is defined as $\rho_1(r) = rQ_{ID}$. $\rho_2 : \mathbb{Z}_q^* \to \mathbb{G}_1^*$ is defined as $\rho_2(r) = rP$. Finally $\rho : \mathcal{D} \to \mathbb{Z}_N^*$ is defined as

$$\rho(r\|z) \stackrel{def}{=} \begin{cases} \rho_1(r) & \text{if } z = 0^\ell \\ \rho_2(r) & \text{if Otherwise} \end{cases}. \tag{3}$$

PHASE 1 Now, at any time $\mathcal{A}$ will call $H_1.eval(ID_i)$ with some id $ID_i \in \{0,1\}^*$ and gets the response Q_i. The triple (ID_i, Q_i, b_i) is stored in the list L_{H_1} where b_i is the random element chosen by the WNAPRO as the input to $\rho(\cdot)$. When $\mathcal{A}$ asks private-key extraction query of some ID_i, $\mathcal{A}'$ responds as follows: It runs $H_1.extract(ID_i)$ to get the triple (ID_i, Q_i, b_i). It parses b_i as $r_i \| z_i$ and if $z_i = 0^\ell$ then it reports failure. The attack on BasicPub fails. Otherwise (if $z_i \neq 0^\ell$), clearly, from Eq. (3), $Q_i = b_i P$. Define $d_i = b_i P_{pub} \in \mathbb{G}_1^*$. Observe that, $d_i = sQ_i$ and hence, d_i is the private key associated with ID_i. So, $\mathcal{A}'$ returns d_i the answer to $\mathcal{A}$.

CHALLENGE. Eventually, at some point of time when $\mathcal{A}$ decides that the above phase is over, it outputs a pair of messages $M_0, M_1 \in \mathcal{M}$ along with an identity ID_{ch}. Then $\mathcal{A}'$ responds as follows: It forwards the messages M_0, M_1 to the cpa-challenger C_{cpa}. The challenger C_{cpa} responds with the BasicPub ciphertext $C = (U, V)$ where C is the ciphertext of M_β for random bit β. Next, $\mathcal{A}'$ calls $H_1.eval(ID_{ch})$ to get the response Q_{ch}, and it calls $H_1.extract(ID_{ch})$ to get the tuple $(ID_{ch}, r_{ch} \| z_{ch}, Q_{ch})$. If $z_{ch} \neq 0^\ell$, $\mathcal{A}'$ aborts. Otherwise, we get $Q_{ch} = r_{ch} Q_{ID}$.

Following [3], when $C = (U, V)$, we have $U \in \mathbb{G}_1^*$. Set $C' = (r_{ch}^{-1} U, V)$ (r_{ch}^{-1} is the inverse of r_{ch} mod q). $\mathcal{A}'$ responds to $\mathcal{A}$ with the challenge ciphertext C'. It is easy to observe that C' is the proper BasicIdent ciphertext of the message M_β under the public identity ID_{ch} as required. Because, $Q_{ch} = H_1.eval(ID_{ch})$ and the corresponding private-key is $d_{ch} = sQ_{ch}$. And also,

$$\hat{e}(r_{ch}^{-1} U, d_{ch}) = \hat{e}(r_{ch}^{-1} U, sQ_{ch}) = \hat{e}(U, s r_{ch}^{-1} Q_{ch}) = \hat{e}(U, sQ_{ID}) = \hat{e}(U, d_{ID}) \tag{4}$$

PHASE 2. The algorithm $\mathcal{A}'$ responds more queries from $\mathcal{A}$ like before.

GUESS. Eventually, $\mathcal{A}$ comes up with a guess β' which $\mathcal{A}'$ forwards to the challenger C_{cpa}.

As argued before $\mathcal{A}$ can not distinguish output of H_1 for $z = 0^\ell$ and $z \neq 0^\ell$. Hence output distribution of $\mathcal{A}$ is independent to the distribution of z. Following [3], if the adversary $\mathcal{A}$ can win the game 10 with probability ε, the adversary $\mathcal{A}'$ wins with probability $\delta = \frac{\varepsilon'}{e1+q_E}$ where q_E is the total number of private-key extraction queries asked by $\mathcal{A}$ to $\mathcal{A}'$. □

6.3.2 IND-CPA security of BasicPub

To prove theorem 6.2, we need to show that an IND-CPA adversary to BasicPub reduces to a BDH adversary to $\mathbb{G}$ in the WNAPRO model.

Lemma 6.4. *Let $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ be a WNAPRO. Let $\mathcal{A}'$ be an IND-CPA adversary with advantage δ against* BasicPub. *Suppose $\mathcal{A}'$ makes q_{H_2} queries to H_2. There exists an adversary $\mathcal{B}$ which wins the BDH game on $(\mathbb{G}_1, \mathbb{G}_2)$ with advantage $\varepsilon = \frac{2\delta}{q_{H_2}}$.*

Proof Sketch. Here the algorithm $\mathcal{B}$ would use the IND-CPA adversary $\mathcal{A}'$ to solve the BDH problem. $\mathcal{B}$ is playing the BDH-game 7 with some BDH challenger C_{BDH} and also simulating the challenger in the IND-CPA-game 10 with $\mathcal{A}'$. The random oracle controlled by the reduction (here $\mathcal{B}$) is H_2. We observe that in the proof in [3], only required property from H_2 is the uniform output distribution and would work even in the Non-Programmable setting. As shown before, we can construct NPRO from WNAPRO by setting the evaluation function ρ as identity function and never using the extract interface.

$\mathcal{B}$ will set the $\rho : \{0,1\}^n \to \{0,1\}^n$ as the identity function by calling $H_2.init$. So, when the adversary $\mathcal{A}'$ will query the random oracle H_2 with $H_2.eval(x)$, it will get an random string from the domain $\{0,1\}^n$. The rest of the proof is exactly same as in [3] and rather straightforward. So, we skip the details.

□

Using Lemma 6.3 and Lemma 6.4, we get Theorem 6.2.

□

7 No Blackbox Reduction to TDP-Key Encapsulation Mechanism in WNAPROM

In this section we prove that, there is no blackbox reduction from the one-wayness of a trapdoor permutations to *trapdoor permutation-based key-encapsulation mechanism* (TDP-KEM) in WNAPRO model. This result proves that (i) the WNAPRO model is *strictly* weaker than the Fully-Programmable RO model and (ii) WNAPRO model

is *orthogonal* to the WPRO model of [8] as TDP-KEM is proven to be secure in that. We start with revisiting the standard construction of TDP-KEM.

7.1 Key Encapsulation Mechanism

A key-encapsulation mechanism KEM is a triple of algorithms denoted $\mathsf{KEM} = (\mathtt{Gen}, \mathtt{Encap}, \mathtt{Decap})$ that operate as follows. The probabilistic key-generation algorithm $\mathtt{Gen}$ takes a security parameter k as input and returns a key-pair (pk, sk); we write $(pk, sk) \leftarrow \mathtt{Gen}(1^k)$. The key-encapsulation algorithm $\mathtt{Encap}$ 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 $\mathtt{Decap}$ is a deterministic algorithms which takes a pair (sk, C) as input and deterministically outputs a key $K \in \mathcal{K}$ or the distinguished symbol $\perp$ to denote invalidity of (sk, C). For correctness, we require that:

$$\forall (pk, sk) \leftarrow \mathtt{Gen}(1^k) \text{ if } (K, C) \leftarrow \mathtt{Encap}(pk), \text{ then } K \leftarrow \mathtt{Decap}(sk, C) \tag{5}$$

Now consider the KEM scheme based on an ideal trapdoor permutation. Let $\mathsf{TDP} = (\mathtt{G}, \mathtt{F}, \mathtt{F}^{-1})$ be a family of trapdoor permutations and $H : \mathcal{D} \rightarrow \mathcal{K}$ be a hash function. The key generation algorithm is defined by $\mathtt{Gen} = \mathtt{G}$, on input 1^k, which returns a pair (pk, td). The encapsulation algorithm $\mathtt{Encap}$ on input pk samples $x \xleftarrow{\$} \mathcal{D}$, sets $K := H(x)$ and computes $C \leftarrow \mathtt{F}(pk, x)$, and returns (K, C). The decapsulation algorithm $\mathtt{Decap}$ on input (td, C) computes $x \leftarrow \mathtt{F}^{-1}(td, C)$, sets $K \leftarrow H(x)$ and returns K.

Security of TDP-KEM

The standard security notion for TDP-KEM is the security against chosen ciphertext attacks. In the random oracle model, the security game against the adversary is described in Figure 8.

Experiment $Game^{cca}_{KEM,\mathcal{A}}$
Compute $(pk, sk) \leftarrow \mathtt{Gen}(1^k)$;
Choose $b \xleftarrow{\$} \{0, 1\}$;
Choose $K_0 \xleftarrow{\$} \mathcal{K}$;
Compute $(K_1, C) \leftarrow \mathtt{Encap}(pk)$;
Set $\mathtt{st}_{\mathcal{A}} := \emptyset$;
Update $\mathtt{st}_{\mathcal{A}} := \mathtt{st}_{\mathcal{A}} \cup \{pk\}$;
For $i = 1 \rightarrow m$;
Receive $C_i \leftarrow \mathcal{A}^H(\mathtt{st}_{\mathcal{A}})$;
If $C_i = C$ abort;
Else
Compute $K_i \leftarrow \mathtt{Decap}(C_i, sk)$;
Update $\mathtt{st}_{\mathcal{A}} := \mathtt{st}_{\mathcal{A}} \cup \{K_i\}$;
Receive $b' \leftarrow \mathcal{A}^H(\mathtt{st}_{\mathcal{A}})$;
If $b = b'$ return 1;
Else return 0.

Figure 8: chosen ciphertext attack game of a KEM scheme

when H is a random oracle. The advantage of adversary $\mathcal{A}^H$ against the cca security of $\mathsf{KEM}^H[\mathsf{TDP}]$ is given by

$$\mathbf{Adv}^{cca}_{\mathsf{KEM}^H[\mathsf{TDP}]}(\mathcal{A}^H) = \left| Pr[b = b'] - \frac{1}{2} \right|. \tag{6}$$

Inversion problem of a trapdoor permutation

Let $TDP = (\mathtt{G}, \mathtt{F}, \mathtt{F}^{-1})$ be a family of trapdoor permutations over $\{0, 1\}^k$. We recall the inversion problem of TDP for any PPT adversary $\mathcal{A}$ in Fig. 9.

The advantage of the adversary $\mathcal{A}$ is given by:

$$\mathbf{Adv}^{inv}_{\mathsf{RSA}}(\mathcal{A}) = \Pr[x^* = x]$$

Experiment $Game^{inv}_{TDP,\mathcal{A}}$
Run $(pk, td) \xleftarrow{\$} \mathtt{G}(1^k)$;
Choose $x \xleftarrow{\$} \{0,1\}^k$;
Set $y = \mathtt{F}(pk, x)$;
Run $x^* \leftarrow \mathcal{A}(pk, y)$;
If $x = x^*$,
return 1;
Else
return 0.

Figure 9: Security game of *tdp-inv*-TDP problem

7.2 No Blackbox reduction in WNAPROM

Intuitive Idea

Let H be the hash function (modeled as a Random Oracle) and f_{pk} be the trapdoor permutation. The reduction, as usual, is playing as an adversary in the standard TDP-inversion game, and acting as a challenger in the CCA-security of TDP-KEM game. The natural technique is to set the challenge received in the ideal-tdp game as the challenge in the TDP-KEM game. The interesting part is how the reduction answers the decapsulation queries. In the Fully-Programmable RO model, the decapsulation queries are answered randomly. When the corresponding hash queries (f^{-1}_{pk} of some previous decapsulation query) are made to random oracle, the reduction sets the output of RO to the response of corresponding decapsulation queries. It is easy to check that this simulation is consistent and indeed TDP-KEM can be proven CCA secure in the random oracle model.

In the Weakly-Programmable Random Oracle model of [8], essentially the same idea with some interesting modifications is sufficient. Recall that, when H is a WPRO, the reduction can not set the output explicitly. But it can be *adaptively* programmed to be $\rho(r)$ of an r, chosen by the reduction. When the reduction gets a decapsulation query C from the adversary, it can just choose a random coin r and reply with $K = \rho(r)$ as a response. The random oracle response for query $x = f^{-1}_{pk}(C)$ (can be easily checked by $f_{pk}(x) \stackrel{?}{=} C$) is programmed using the corresponding r.

Although, our WNAPRO model looks somewhat similar (the oracle responds with $\rho(r)$ for a regular function ρ), the important difference is that, the reduction *can not* choose the random string r in the run-time to be used for any hash query response. Intuitively, while WPRO model supports adaptive "input" programming (r for the response $\rho(r)$ can be programmed), WNAPRO model supports only non-adaptive "functional" (ρ for the response $\rho(r)$ can be programmed only at the start of the game) programming. Precisely the reduction is unable to "re-program" the correspondence between x and r after pre-processing. We show that the feature of such "adaptive programming" is essential for blackbox security proof of TDP-KEM. Looking ahead, we prove that if there is a reduction from the one-wayness of a family of trapdoor permutations in the WNAPRO model, then there is a reduction from one-wayness of a family of trapdoor permutations in the NPRO model, thus contradicting Theorem 3 of [8]. For simplicity we will assume ρ to be a permutation. However, we emphasize that our technique can be extended to the setting when ρ is not a permutation.

Formally we prove the following theorem

Theorem 7.1. *If there is a (cca-*KEM $\rightarrow$ *tdp-inv-*TDP*)-reduction in WNAPRO model with advantage ε, then there exists a (cca-*KEM $\rightarrow$ *tdp-inv-*TDP*)-reduction in NPRO model with advantage at least $\varepsilon - \frac{1}{2^k}$, where k is the security parameter.*

Proof. We are given a PPT machine $\mathcal{B}$ which is a (*cca-*KEM $\rightarrow$ *tdp-inv-*TDP) reduction in WNAPRO model. In particular, for any PPT adversary $\mathcal{A}$ playing in the cca-security game of the TDP-KEM scheme KEM and any random oracle H (with appropriate domain) if $\mathbf{Adv}^{cca}_{\mathsf{KEM}^H[\mathsf{TDP}]}(\mathcal{A}^H)$ is non-negligible in the security parameter k, then $\mathbf{Adv}^{tdp\text{-}inv}_{\mathsf{TDP}}(\mathcal{B}^{((H.init,H.eval,H.extract),\mathcal{A}^{H.eval})})$ is non-negligible in k. We have to show another PPT machine, $\mathcal{B}'$ which, possibly using $\mathcal{B}$, would be a (*cca-*KEM $\rightarrow$ *tdp-inv-*TDP)-reduction in NPRO model. In particular for any PPT adversary $\mathcal{A}$ playing in the cca-security game of the TDP-KEM scheme KEM and for a random oracle H, if $\mathbf{Adv}^{cca}_{\mathsf{KEM}^H[\mathsf{TDP}]}(\mathcal{A}^{H.eval})$ is non-negligible in the security parameter k, then $\mathbf{Adv}^{tdp\text{-}inv}_{\mathsf{TDP}}(\mathcal{B}'^{((H.eval'),\mathcal{A}^{H.eval'})})$ must be non-negligible in k.

$\mathcal{B}'^{(f,H.eval')}$ is simultaneously playing

- the role of challenger in the cca-kem game with the adversary $\mathcal{A}^{H.eval'}$.
- the role of adversary in the tdp inversion game.

Recall that, the adversary $\mathcal{A}^{H.eval'}$ will make decapsulation queries. Our strategy is to use $\mathcal{B}$ to answer those decapsulation queries. When $\mathcal{B}$ will output a challenge ciphertext, we shall run $\mathcal{A}$ to get the guess, and submit the answer to $\mathcal{B}$ to successfully invert the tdp.

Recall that $\mathcal{B}$ is a reduction in the WNAPRO model. $\mathcal{B}'$, in order to use $\mathcal{B}$, needs to simulate the $H.init, H.eval$, and $H.extract$ interfaces.

We will construct such $\mathcal{B}'$ as follows:

$\underline{\mathcal{B}'}$:

- *Setup* $\mathcal{B}'^{(f,H.eval')}$ receives the public key pk and the challenge y. It simulates $\mathcal{B}$ on (pk, y). When $\mathcal{B}$ outputs the public key of the KEM to run the adversary, $\mathcal{B}'$ invokes $\mathcal{A}$ on the same public key. $\mathcal{B}'$ handles the in the following fashion.
- *Handling H.init query.* In the pre-processing phase on query $H.init(\rho)$ by $\mathcal{B}$, store the description of ρ. Also initialize an internal list $L := \emptyset$.
- *Answering query from* $\mathcal{B}^{\left((H.init,H.eval,H.extract),\mathcal{A}^{H.eval}\right)}$*:*
  1. *H.eval query.* On receiving query $H.eval(x)$ from $\mathcal{B}$ check if there exists a triple $(x, r, t) \in L$, if yes then respond with t, otherwise then query own oracle $H.eval'(x)$. Let the response to that be r. Then compute $t \leftarrow \rho(r)$ and return t to $\mathcal{B}$ as response. Finally store the triple in the list, $L \leftarrow L \cup (x, r, t)$.
  2. *H.extract query.* On receiving query $H.extract(x)$ from $\mathcal{B}$ check if there exists a triple $(x, r, t) \in L$, if not output $\perp$, otherwise output r.
  3. *Read query to the random oracle.* $\mathcal{B}$ may issue "read" query to learn the channel between $\mathcal{A}^{H.eval}$ and the $H.eval$ interface of the RO. However, since $\mathcal{B}'$ is in NPRO model, the adversary $\mathcal{A}$ has access to the interface $H.eval'$. So a simulation of the interaction between $\mathcal{A}$ and $H.eval$ is necessary. $\mathcal{B}'$ can make its own read query to simulate this channel as follows: if $\mathcal{A}$ has asked a query $H.eval'(x)$ to get a response r back from the interface $H.eval'$, then compute $t = \rho(r)$ and return the query-response pair as (x, t) to $\mathcal{B}$. Also store the triple (x, r, t) in the list L.
- *Answering decapsulation query from* $\mathcal{A}^{H.eval'}$*:*
  1. On receiving decapsulation query $(\texttt{DECAP}, C)$ forwards C to $\mathcal{B}$. Let K be the response from $\mathcal{B}$. Check whether $\exists (x, r, t) \in L$ such that $K = t$. If yes then return r to $\mathcal{A}$ as the corresponding response. If no such tuple exists, then return some random $r \xleftarrow{\$} \mathcal{K}$.

First notice that the simulated view of $\mathcal{B}$ described above is indistinguishable in poly-time from the actual view of the reduction $\mathcal{B}$ in WNAPRO model. This is straightforward from the description of $\mathcal{B}'$.

We also claim that the view of $\mathcal{A}$ here is statistically indistinguishable from the view of $\mathcal{A}$ in WNAPRO model. Essentially the view of $\mathcal{A}$ consists of the answers to the DECAP queries and the answers to the RO-evaluation queries. Let Guess be the event which takes place when there is no such triple exists in Step 1 and $\mathcal{B}'$ returns some random response to the decapsulation query. We argue that, until Guess does not take place, the view of $\mathcal{A}$ is consistent: $\mathcal{A}$ while querying $\mathcal{B}$ with some ciphertext C expects some K in return such that $K = H(x)$ and $C = \mathtt{F}(pk, x)$. In the above simulation, since $\mathcal{A}$ has access to $H.eval'$, it gets some uniform random $K \xleftarrow{\$} \mathcal{K}$ from $\mathcal{B}'$. On the other hand, in the WNAPRO model $\mathcal{A}$ expects some K such that $K = \rho(r)$ for some random $r \xleftarrow{\$} \mathcal{K}$ and some regular function ρ. But, since ρ is a permutation, the output distribution of ρ is also uniform over $\mathcal{K}$ and thus $\rho(r)$ is a uniform random value on $\mathcal{K}$. Moreover, since $\mathcal{A}$ has no access to $H.extract$ interface, it can not check (even with the knowledge of ρ) whether it is given the correctly computed $\rho(r)$ or some other random value even as long as the output is uniform random. Hence the view of $\mathcal{A}$ in both settings are identically distributed. From above argument we can conclude that when Guess does not happen the advantage of $\mathcal{B}'$ is at least equal to the advantage of $\mathcal{B}$.

However, if Guess takes place in that step then the advantage of $\mathcal{B}'$ can not depend on the advantage of $\mathcal{B}$. Nonetheless, it is easy to see that in that case $\mathcal{B}$ can win only with negligible probability. If Guess takes place, it is clear that $\mathcal{B}$ did not ask any query $H.eval(x)$ until then. This implies that $\mathcal{B}$ must have guessed the correct value of $H.eval(x)$ in order to respond to the decapsulation query forwarded by $\mathcal{B}'$. Since R is a RO, this happens only with probability $\leq \frac{1}{|\mathcal{K}|}$. Formally we can write

$$\mathbf{Adv}^{tdp\text{-}inv}_{\mathsf{TDP}}\left(\mathcal{B}'^{(H.eval',\mathcal{A}^{H.eval'})}\right) \geq \mathbf{Adv}^{tdp\text{-}inv}_{\mathsf{TDP}}\left(\mathcal{B}^{((H.init,H.eval,H.extract),\mathcal{A}^{H.eval})}\right) - \frac{1}{|\mathcal{K}|}$$

and hence if $\mathcal{B}$ has non-negligible advantage so does $\mathcal{B}'$.

□

Together with Theorem 7.2 and Theorem 3 of [8], we get

Theorem 7.2. *There is no* (cca-KEM $\rightarrow$ $tdp\text{-}inv$-TDP)*-reduction in the WNAPRO model.*

8 Conclusion

We introduced a new framework of programming random oracles non-adaptively. We presented two models to prove security reductions. We show that the first model is equivalent to the model without any power to program the oracle. Tweaking this model, we get another model where we can prove security of several important schemes like RSA Full Domain Hash Signature and Boneh-Franklin Identity Based Encryption scheme.

A natural open question is to find more existing schemes which can be proven secure in the Weak-Non-Adaptively Programmable model. Another interesting research direction is to find weaker yet interesting models, like letting adversary access the extract interface.

9 Acknowledgements

We thank Ivan Dåmgard and Vipul Goyal for useful discussions at the beginning of this work. Part of this work was done when both authors were associated with the Centre of Excellence in Cryptology of Indian Statistical Institute,Kolkata, and were supported by Ministry of Defense of Government of India.

References

[1] Prabhanjan Ananth and Raghav Bhaskar. Non observability in the random oracle model. Cryptology ePrint Archive, Report 2012/710, 2012. `http://eprint.iacr.org/2012/710`.

[2] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[3] Dan Boneh and Matthew K. Franklin. Identity based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[4] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press.

[5] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, Texas, USA, May 23–26, 1998. ACM Press.

[6] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Berlin, Germany.

[7] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany.

[8] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 303–320, Singapore, December 5–9, 2010. Springer, Berlin, Germany.

[9] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Germany.

[10] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th Annual Symposium on Foundations of Computer Science*, pages 102–115, Cambridge, Massachusetts, USA, October 11–14, 2003. IEEE Computer Society Press.

[11] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 21–38, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Berlin, Germany.

[12] Chun-Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 92–105, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Berlin, Germany.

[13] Eike Kiltz and Krzysztof Pietrzak. On the security of padding-based encryption schemes - or - why we cannot prove OAEP secure in the standard model. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 389–406, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany.

[14] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Berlin, Germany.

[15] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 316–337, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Germany.

[16] Omer Reingold and Luca Trevisan and Salil P. Vadhan Notions of Reducibility between Cryptographic Primitives, In M. Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004*, pages 1–20, Cambridge, MA, USA, February 19-21, 2004, Proceedings.

[17] Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 205–223, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Berlin, Germany.

Experiment $Game^{cpa}_{ENC,A'^R}$
Compute $(pk, sk) \leftarrow \texttt{Gen}(1^k)$;
Receive $(M_0, M_1) \leftarrow A'^R$;
Choose $b \leftarrow \{0, 1\}$
Set $C_b \leftarrow \texttt{Enc}(pk, M_b)$;
Send C_b to A'^R;
Receive $b' \leftarrow A'^R$;
If $b = b'$ return 1;
Else return 0;

Figure 10: Chosen Plaintext Attack on a Public-key Encryption Scheme

A Left-out Definitions

A.1 Public Key Encryption

A public key encryption (PKE) scheme is a triple of algorithms $\langle \texttt{Gen}, \texttt{Enc}, \texttt{Dec} \rangle$. `Gen` is a randomized algorithm which takes the security parameter 1^k as input to generate public key pair (pk, sk). `Enc` is also a randomized algorithm which takes a message $M \in \mathcal{M}$ and the public key pk to produce the ciphertext C. And the deterministic algorithm `Dec` takes the ciphertext C and the secret-key sk as input to gives back the message M. The standard IND-CPA security game is described in the standard way in figure 10.