# Uniqueness Is a Different Story: Impossibility of Verifiable Random Functions from Trapdoor Permutations

Dario Fiore[1,⋆] and Dominique Schröder[2,⋆⋆]

[1] Department of Computer Science, New York University
[2] Department of Computer Science, University of Maryland

**Abstract.** Verifiable random functions (VRFs) are pseudorandom functions with the additional property that the owner of the seed $SK$ can issue publicly-verifiable proofs for the statements "$f(SK, x) = y$", for any input $x$. Moreover, the output of VRFs is guaranteed to be unique, which means that $y = f(SK, x)$ is the only image that can be proven to map to $x$. Despite their popularity, constructing VRFs seems to be a challenging task and only a few constructions based on specific number-theoretic problems are known. Basing a scheme on general assumptions is still an open problem. Towards this direction, Brakerski *et al.* showed that verifiable random functions cannot be constructed from one-way permutations in a black-box way.

In this paper we continue the study of the relationship between VRFs and well-established cryptographic primitives. Our main result is a separation of VRFs and adaptive trapdoor permutations (ATDPs) in a black-box manner. This result sheds light on the nature of VRFs and is interesting for at least three reasons:

- First, the separation result of Brakerski *et al.* gives the impression that VRFs belong to the "public-key world", and thus their relationship with other public-key primitives is interesting. Our result, however, shows that VRFs are strictly stronger and cannot be constructed (in a black-box way) form primitives like e.g., public-key encryption (even CCA-secure), oblivious transfer, and key-agreement.
- Second, the notion of VRFs is closely related to weak verifiable random functions and verifiable pseudorandom generators which are both implied by TDPs. Dwork and Naor (FOCS 2000) asked whether there are transformation between the verifiable primitives similar to the case of "regular" PRFs and PRGs. Here, we give a negative answer to this problem showing that the case of verifiable random functions is essentially different.
- Finally, our result also shows that *unique* signatures cannot be instantiated from ATDPs. While it is well known that standard signature schemes are equivalent to OWFs, we essentially show that the uniqueness property is crucial to change the relations between primitives.

⋆ Work done while at ENS Paris.
⋆⋆ Postdoctoral fellow of the DAAD.

R. Cramer (Ed.): TCC 2012, LNCS 7194, pp. 636–653, 2012.

## 1 Introduction

Verifiable random functions (VRF) were introduced by Micali, Rabin, and Vadhan [1]. VRFs are random functions with the additional property that they provide a proof verifying the input-output relationships. Formally, a VRF is defined by a key pair $(SK, PK)$ such that: the secret seed $SK$ allows the evaluation of the function $y \leftarrow F(SK, x)$ on any input $x$ and the generation of a proof $\pi$. This proof is publicly verifiable i.e., given the public key $PK$ one can efficiently verify (using $\pi$) that the statement "$F(SK, x) = y$" holds. For security, VRFs must satisfy two properties: pseudorandomness and uniqueness. Roughly speaking, *pseudorandomness* states that the function looks random at any input $x$ for which no proof has been issued. *Uniqueness* guarantees that for any $x$, there exists only one image $y$ for which a valid proof can be produced (even for maliciously chosen public keys).

In some sense a VRF can be seen as the public-key equivalent of a pseudorandom function. This fascinating primitive has many applications, both theoretical and practical: 3-rounds resettable zero-knowledge [2], non-interactive lottery systems and micropayment schemes [3], a verifiable transaction escrow scheme [4], and updatable zero-knowledge sets [5]. However, despite their popularity, constructing VRFs seems to be challenging. In particular, only a few schemes are known so far, e.g., [1,6,7,8,9,10] (see Section 1.3 for a brief description of these works). Furthermore, all known schemes are based on specific number-theoretic problems such as RSA or different assumptions relying on bilinear maps. Constructing a VRF based on general assumptions is still an open problem.

In modern cryptography, almost all cryptographic primitives base their security on unproven computational assumptions that are considered reasonable by the community[1]. In particular, the existence of one-way functions (OWF) is one of the major open problems in cryptography. A common methodology for proving the security of a cryptographic primitive, and for better understanding its relation to other primitives, are black-box reduction techniques that can be described as follows. Let $P$ and $Q$ be two primitives. A *construction* of $P$ from $Q$ is black-box if the primitive $P$ has only oracle access to $Q$ (i.e., $P$ does not have access to the code of this primitive, but can evaluate it). A *security reduction* of $P$ to $Q$ is black-box if for any (efficient) adversary $\mathcal{A}$ that breaks $P$ there exists an (efficient) algorithm $\mathcal{S}$ that has black-box access to $\mathcal{A}$ and breaks $Q$. This approach has been extensively formalized by Reingold *et al.* who gave different "flavors" of black-box reductions depending on the "degree" of black-box access [11].

Black-box constructions and black-box proofs give clearly a limited view on the relation between the different primitives as no conclusions beyond the black-box access can be made. Nevertheless, the approach is well established as most of the cryptographic proofs are black-box and it is strong enough to show that many cryptographic primitives, such as pseudorandom functions, digital signatures, private-key encryption, are equivalent to the existence of one-way functions

[1] If one makes exception of a few cases that are proven secure in an information-theoretic sense.

(OWFs), which is considered to be one of the most basic assumptions. On the other hand, other primitives (e.g., public-key encryption) are believed to exist only under stronger assumptions (e.g., the existence of trapdoor permutations). Though such primitives and/or assumptions look different, it might be possible that many of them are related or even equivalent. Therefore, identifying the minimal assumptions on which one can base the security of a primitive is considered one of the most important goals for a better and deeper understanding of the cryptography world.

On the negative side, Impagliazzo and Rudich introduced a methodology for proving separations between primitives in the sense of black-box constructions, e.g., proving that *Q does not* imply *P in a black-box way* [12]. In their work they ruled out any black-box construction of key-agreement protocols (KA) from one-way functions. Gertner *et al.* show that the breakthrough result of Impagliazzo and Rudich can be seen as defining two separated worlds in which the cryptographic primitives can be divided: the "private cryptography" world that contains all those primitives that are equivalent to OWFs, and private-key encryption; the "public cryptography" world that contains harder primitives such as trapdoor permutations, public-key encryption (PKE), KA and oblivious transfer (OT) [13].

It is worth to mention that another methodology, called *meta-reductions*, for separating primitives in a black-box sense is known. Since we do not follow this approach, we refer the reader to e.g., [14,15,16].

### 1.1 Our Results

We investigate the relationship between verifiable random functions and well-studied cryptographic primitives. The first step towards this goal was recently given by Brakerski, Goldwasser, Rothblum, and Vaikuntanathan who separated VRFs from one-way permutations [17]. The authors introduce the notion of *weak verifiable random functions (wVRFs)* that can be seen as the public key analogue to weak-PRFs: pseudorandomness only holds with respect to randomly chosen inputs. Moreover, they construct wVRFs from (enhanced) trapdoor permutations and show that wVRFs are essentially equivalent to non-interactive zero knowledge proof (NIZK) systems in the common reference string model. In the private key setting, it is well known that "regular" PRFs can be constructed from weak PRFs in a black-box way [18,19]. Thus, a natural direction to study the relation between the primitives is to build a VRF out of any wVRF.

Another work that is closely related to this topic is the study of *verifiable pseudorandom generators* (VPRGs) due to Dwork and Naor [20]. Roughly speaking, a VPRG is a pseudorandom generator that allows the owner of the seed to prove the correctness of subsets of the generated bits while the other bits remain indistinguishable from random. Dwork and Naor constructed VPRGs from trapdoor permutations. Again, in the case of "regular" PRFs we know how to turn a PRG

into a PRF in a black-box way [21]. Dwork and Naor left open the question if a similar transformation can be found in the public key setting [20], namely:

> *Is it possible to construct a VRF from VPRGs and/or weak-VRFs in a black-box way?*

In this paper, we give a negative answer to this question and, more generally, we show that no black-box constructions of VRFs from (enhanced) trapdoor permutations exist.

**Theorem 1 (informal).** *There exists no black-box reduction of verifiable random functions to trapdoor permutations.*

Our result is actually more general than the above indicates; it separates the weaker primitive of verifiable unpredictable functions (VUFs) from the stronger primitive of *adaptive* trapdoor functions. The difference between VRFs and VUFs is that in the latter the output should be unpredictable instead of pseudorandom. Therefore, VUFs can also be seen as "*unique signatures*", where, for every public key, each message can have at most one valid signature[2].

Adaptive trapdoor functions (ATDFs), recently introduced by Kiltz, Mohassel, and O'Neill in [22], are essentially strictly stronger than trapdoor functions as the adversary is given access to an inversion oracle.

**Implications of Our Result.** Our result sheds light on the nature of VRFs and explains why this primitive seems so hard to construct. First, given the separation result of Brakerski *et al.*, one can naturally think of VRFs as though they belong to the "public cryptography" world. Then, if we consider the relationship between VRFs and the other public-key primitives, our result highlights that VRFs are much stronger as they cannot be implied by most of the primitives in this world: basically everything which is implied by TDPs, e.g. semantically-secure public-key encryption, oblivious transfer, key-agreement. Moreover, since ATDPs imply CCA-secure PKE [22], then VRFs are separated even from it. On the positive side we observe that we can obtain a construction of VRFs from identity-based encryption with *unique key derivation* following the idea of Abdalla *et al.* [9][3]. Combining this positive result with our impossibility result confirms the impossibility result of IBE from TDPs [23].

Second, our result points out the hardness of achieving the uniqueness property in the context of digital signatures: While signature schemes are equivalent to OWFs, *unique* signatures cannot be instantiated from (adaptive) TDPs in a black-box way.

Finally, since both weak-VRFs and VPRGs are implied by TDPs, our result rules out the possibility of constructing VRFs from weak-VRFs and/or VPRGs

[2] At this stage, it is interesting to observe unique and deterministic signatures are two distinct primitives. Consider for example the signature $\sigma = \sigma' \| 0$ where $\sigma'$ is deterministic and the verification algorithm ignores the last bit. Then it is obvious that uniqueness could be easily violated by flipping the last bit.

[3] Precisely, the unique key derivation algorithm immediately implies a VUF, which can then be turned into a VRF using the original idea of Micali, Rabin and Vadhan.

(in a black-box way). Thus, it seems that there is no hope that the approaches used in the private key world to build PRFs from weak-PRFs and PRGs can be adopted to the case of the public verifiable primitives. This shows that the verifiable analogous of these primitives are essentially different.

### 1.2 Overview of the Techniques

Our starting point is the so-called "two oracles" technique of Hsiao and Reyzin [24]. The main idea of this technique is to construct two oracles, say $\mathcal{O}$ and $\mathcal{B}$, such that $\mathcal{O}$ is used in the constructions, whereas both oracles $\mathcal{O}$ and $\mathcal{B}$ can be accessed by the adversaries. This approach is slightly weaker than the single oracle technique because it "only" rules out fully-black-box reductions (instead of any black-box reduction).

**Our Oracles.** In our case the oracle $\mathcal{O}$ is an ideal random trapdoor permutation oracle that is modeled as a triple of random functions $(g, e, d)$ such that: $g(\cdot)$ maps trapdoors to public keys; $e(ek, \cdot)$ is a random permutation for every public key $ek$ and $d(td, \cdot)$ is the inverse of $e(ek, \cdot)$ when $g(td) = ek$. Due to the fact that $\mathcal{O}$ is truly random, $\mathcal{O}$ is secure even in the sense of adaptive trapdoor permutations. The oracle $\mathcal{B}$ is designed to break any black-box construction of VUF based on $\mathcal{O}$.

Therefore, the core of our separation theorem is the definition of the weakening oracle $\mathcal{B}$. The proof then consists of two main parts:

(i) showing an efficient adversary that can break the unpredictability of the VUF by making a polynomial number of queries to $\mathcal{B}$;
(ii) showing an ATDP construction that is secure against any adversary that makes at most polynomially-many oracle queries.

The design of $\mathcal{B}$ is rather technical. In particular, the main difficulty is to prevent an attacker from exploiting $\mathcal{B}$ to break the one-wayness of the ATDP. A naïve construction would be an oracle that takes as input a VUF public key and returns $y^* \leftarrow F(SK, x^*)$, i.e., the evaluation of the function on a random point $x^*$. This oracle would clearly break the unpredictability of the VUF, but it would also be too strong. Consider, for instance, an adversary $\mathcal{A}$ that is given as input a public key $ek^*$ of a trapdoor permutation and that is challenged to invert it on a random point $b^*$. Now, $\mathcal{A}$ might encode $(ek^*, b^*)$ into $PK$ in such a way that the evaluation of $F(SK, x^*)$ requires to invert $b^*$. But then the attacker would learn all informations about $b^*$'s inverse. To prevent these "dangerous" queries we modify $\mathcal{B}$ such that it takes as input a certain number of triples $(x_i, y_i, \pi_i)$, where $\pi_i$ is a valid proof for "$F(SK, x_i) = y_i$". The idea follows from the intuition that the attacker can encode $b^*$ (and $ek^*$) into $PK$ in only two ways:

(i) $F(SK, \cdot)$ needs to invert $b^*$ on a large fraction of the inputs,
(ii) $F(SK, \cdot)$ needs to invert $b^*$ only on a negligible fraction of the inputs.

Now, suppose that $\mathcal{A}$ encodes $b^*$ into $PK$ as defined in the first case. In order to query the oracle, $\mathcal{A}$ has to provide valid proofs. But if $\mathcal{A}$ can compute all

proofs, then the attacker must already know $b^*$'s inverse. Otherwise, if $b^*$ is encoded into $PK$ as described in the second case, then the probability that evaluating $F(SK, x^*)$ on a random input $x^*$ requires to invert $b^*$ is negligible. Hence, returning $y^*$ does not reveal any useful informations to $\mathcal{A}$. Although this idea seems very promising, it raises another issue. In fact $\mathcal{A}$ might overcome this limitation by choosing all the $x_i$'s from the small fraction that does not require to invert $b^*$. We solve this issue by defining a two-steps oracle $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that $\mathcal{B}_1$ chooses the values $x_i$'s and $\mathcal{B}_2$ is the actual oracle as described above, such that it works properly only if the inputs $x_i$'s are chosen by $\mathcal{B}_1$.

Finally, an important detail towards the definition of $\mathcal{B}$ is that it simulates the run of $F^{\mathcal{O}}(SK, x^*)$ using a *different* oracle $\mathcal{O}'$ and a *different* secret key $SK'$ such that $SK'$ still corresponds to $PK$ under $\mathcal{O}'$. The idea is that, if $\mathcal{O}'$ is close enough to $\mathcal{O}$ (as it should be the case while trying to break the VUF), then evaluating $F^{\mathcal{O}'}(SK', x^*)$ produces the same output as $F^{\mathcal{O}}(SK, x^*)$. On the other hand, with high probability $\mathcal{O}$ and $\mathcal{O}'$ are *not* close when an ATDP adversary invokes $\mathcal{B}$.

### 1.3 Other Related Work

**Verifiable Random Functions.** Goldwasser and Ostrovsky introduce the notion of unique signatures (calling them *invariant signatures*) and they show that in the common random string model they are equivalent to non-interactive zero-knowledge proofs [25]. Later, Micali, Rabin and Vadhan formally define VRFs and propose a construction (in the plain model) [1]. The authors follow two main steps: (1) they construct a verifiable unpredictable function (VUF) based on the RSA problem and then (2) they show a generic transformation to convert a VUF into a VRF using the Goldreich-Levin theorem [26] (that extracts one random bit from polynomially-many unpredictable bits). The hope of this two-steps approach is that a VUF should be easier to realize than a VRF, but the second step is very inefficient. Finally, Lysyanskaya proposes a VUF relying on a strong version of the Diffie-Hellman assumption [6].

The subsequent works suggest direct and (more) efficient constructions of VRFs without relying on the Goldreich-Levin transformation. Dodis suggests an instantiation on the sum-free generalized DDH assumption [7], and Dodis and Yampolskiy give a construction based on the bilinear Diffie-Hellman inversion assumption [8]. Abdalla, Catalano, and Fiore show the relationship between VRFs and a certain class of identity-based encryption schemes [9]. Moreover, the authors propose a construction based on the weak bilinear Diffie-Hellman inversion assumption. All the schemes mentioned so far share the limitation of supporting only a small domain (i.e., of superpolynomial size). The only exception is the recent scheme by Hohenberger and Waters, who give the first construction having a large input space [10]. Another closely related work is one of Dodis and Puniya who construct NIZK from verifiable random permutations (VRPs), that are the verifiable analog of pseudorandom permutations [27]. The author also show how to convert a VRF into a VRP.

**Black-Box Separations.** After the seminal result of Impagliazzo and Rudich many follow up works studied the relation between different primitives, such as, e.g., [13,28,29,30,23,23,31,32]. We discuss these works in the full version [14].

## 2 Preliminaries

**Adaptive Trapdoor Permutations.** Adaptive trapdoor permutations (ATDPs) are defined similar to a trapdoor permutation, but in the security definition the adversary is provided with an oracle that inverts the function on arbitrary images (except on the challenge value). A formal definition is given in [22,14].

**Verifiable Random Functions.** Verifiable random functions (VRF) are similar to pseudorandom functions, but differ in two main aspects: Firstly, the output of the function is publicly verifiable, i.e., there exists an algorithm $\Pi$ that returns a proof $\pi$ which shows that $y$ is the output of the function on input $x$. Secondly, the output of the function is unique, i.e., no two images (and proofs) exist that verify under the same preimage.

**Definition 1 (Verifiable Random Functions).** *A family of functions* $\mathcal{F} = \{f_s : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}_{s \in \{0,1\}^{seed(\lambda)}}$ *is a family of* Verifiable Random Functions *if there exists a tuple of algorithms* $(KG, F, \Pi, V)$ *with the following functionalities:*

$KG(1^\lambda)$ *outputs a pair of keys* $(PK, SK)$.
$F(SK, x)$ *is a deterministic algorithm that evaluates* $f_s(x)$.
$\Pi(SK, x)$ *is an algorithm that outputs a proof* $\pi$ *related to* $x$.
$V(PK, x, y, \pi)$ *outputs 1 if* $\pi$ *is a valid proof for "*$f_s(x) = y$*", else it outputs 0.*

*A tuple* $(KG, F, \Pi, V)$ *is said to be a VRF if it satisfies the following properties:*

**Domain Range Correctness** *For all values* $x \in \{0,1\}^{n(\lambda)}$*, over the choices of* $(PK, SK)$*, we have that* $F(SK, x) \in \{0,1\}^{m(\lambda)}$ *holds with all but negligible probability.*

**Completeness** *For all* $x \in \{0,1\}^{n(\lambda)}$ *if* $\Pi(SK, x) = \pi$ *and* $F(SK, x) = y$ *then* $V(PK, x, y, \pi)$ *outputs 1 with overwhelming probability (over the choices of* $(PK, SK)$ *and the coin tosses of* $V$*).*

**Uniqueness** *There exist no values* $(PK, x, y_1, y_2, \pi_1, \pi_2)$*, unless with negligible probability over the coin tosses of* $V$*, such that for distinct* $y_1$ *and* $y_2$ *it holds that* $V(PK, x, y_1, \pi_1) = V(PK, x, y_2, \pi_2) = 1$.

**Pseudorandomness** *For all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *we require that the probability* $\mathcal{A}$ *succeeds in the experiment* $\mathsf{pseudo}^f_{\mathcal{A}}$ *is at most* $\frac{1}{2} + \mathsf{negl}(\lambda)$*, where the experiment is defined in Figure 1.*

Verifiable unpredictable functions (VUF) are similar to VRFs, except that unpredictability must hold instead of pseudorandomness:

**Definition 2 (Verifiable Unpredictable Functions).** *A tuple* $(KG, F, \Pi, V)$ *is a* verifiable unpredictable function *if the probability that any PPT adversary* $\mathcal{A}$ *succeeds in the experiment* $\mathsf{predict}^f_{\mathcal{A}}$*, defined in Figure 1, is at most negligible.*

**Experiment** $\mathsf{pseudo}^f_{\mathcal{A}}$
$(PK, SK) \leftarrow KG(1^\lambda)$;
$(x^*, \mathsf{state}) \leftarrow \mathcal{A}_1^{Func(SK,\cdot)}(PK)$
$b \xleftarrow{\$} \{0,1\}$;
$y_0 \leftarrow F(SK, x)$; $y_1 \xleftarrow{\$} \{0,1\}^{m(\lambda)}$
$b' \leftarrow \mathcal{A}_2^{Func(SK,\cdot)}(\mathsf{state}, y_b)$
Output 1 iff $b' = b$
and $x^*$ was not asked
to the $Func(SK, \cdot)$ oracle.

**Experiment** $\mathsf{predict}^f_{\mathcal{A}}$
$(PK, SK) \leftarrow KG(1^\lambda)$;
$(x^*, y^*) \leftarrow \mathcal{A}^{Func(SK,\cdot)}(PK)$
Output 1 iff $y^* = F(SK, x^*)$ and
$x^*$ was not asked
to the $Func(SK, \cdot)$ oracle.

**Fig. 1.** This Figure show the experiment of pseudorandomness and unpredictability. In both experiments the oracle $Func(SK, \cdot)$ computes $F(SK, \cdot)$ and $\Pi(SK, \cdot)$ and returns their output.

## 3 The Black-Box Separation

We first give a high-level overview of the main ideas of our proof before going into the details afterwards. Our starting point is the "two oracles" separation technique of Hsiao and Reyzin [24]. In the context of VRFs, we have to construct two oracles $\mathcal{O}$ and $\mathcal{B}$ relative to which ATDPs exist while VUFs do not. In particular, the constructions are restricted to have black-box access only to $\mathcal{O}$, while the adversary may access both $\mathcal{O}$ and $\mathcal{B}$.

The core of our separation are the two oracles, $\mathcal{O}$ and $\mathcal{B}$. The oracle $\mathcal{O} = (g, e, d)$ realizes a random trapdoor permutation (we give a formal definition in Section 3.2). The second oracle is a weakening oracle such that relative to $\langle \mathcal{O}, \mathcal{B} \rangle$ a secure construction of adaptive trapdoor permutations exists while any given candidate (and correct) VUF construction $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ is insecure[4]. To prove this result, we build an adversary that wins the unpredictability game with non-negligible probability. Since the description of the oracle $\mathcal{B}$ is rather technical, we first describe the high-level intuitions that guides us to the design of $\mathcal{B}$.

### 3.1 Towards the Definition of $\mathcal{B}$

Towards the definition of such $\mathcal{B}$, the main difficulty is to design an oracle that is strong enough to help predicting a value of the VUF while simultaneously being too weak to invert the ATDP.

A naïve approach for $\mathcal{B}$ would be the one that immediately breaks the VUF, by taking the VUF's public key $PK$ and a value $x$ as input; it then would return $F^{\mathcal{O}}(SK, x)$. Of course, any VUF construction breaks down in the presence of such oracle. So, it would remain to show that an ATDP is still secure in the presence of such $\langle \mathcal{O}, \mathcal{B} \rangle$, which unfortunately is not the case. To see this, consider the following VUF defined through $KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}}$ (where $\Pi^{\mathcal{O}}(SK, \cdot) = F^{\mathcal{O}}(SK, \cdot)$): The $KG^{\mathcal{O}}$ algorithm queries $ek \leftarrow g(td)$ on a random $td \in \{0,1\}^\lambda$ and sets $PK = ek$ and $SK = td$. The function evaluation algorithm on input $x$

[4] By $\langle \mathcal{O}, \mathcal{B} \rangle$ we mean that the algorithm $A^{\langle \mathcal{O}, \mathcal{B} \rangle}$ gets access to both oracles.

obtains $y \leftarrow d(td, x)$ and outputs $y$. $V(PK, x, y)$ simply checks that $e(ek, y) = x$. Observe that this construction is sound and unique (but trivially insecure). Now, we construct an adversary $\mathcal{A}$ against the ATDP that exploits the above defined $\mathcal{B}$ to invert the challenge $(ek^*, b^*)$. This attacker inverts the challenge by simply submitting $(PK = ek^*, x = y^*)$ to $\mathcal{B}$! This means that the oracle $\mathcal{B}$ that we sketched before is too strong and reveals too much information.

As one can guess, the problem are those queries to $\mathcal{B}$ that are "dangerous" in the sense that they extract too much useful information to invert the ATDP. Starting from this (toy) example we modify $\mathcal{B}$ to prevent such "dangerous queries". The first important observation is that our adversary against the unpredictability only needs to predict *some* value, rather than a specific one. This means, the attacker only needs to find $y^*$ for a fresh $x^* \in \{0,1\}^n$. Therefore, our first modification consists of changing the input that is provided to $\mathcal{B}$. Basically, we let $\mathcal{B}$ choose $x^*$ on which it evaluates $y^* \leftarrow F^{\mathcal{O}}(SK, x^*)$. This new definition of $\mathcal{B}$ still allows us to break the security of the VUF and it also avoids direct inversion queries as the attack can no longer query $x$ directly to $\mathcal{B}$.

However, this modification is not sufficient to avoid that an ATDP adversary exploits the access to $\mathcal{B}$. The problem is that an attacker $\mathcal{A}$ might encode its challenge $(ek^*, b^*)$ into the public key $PK$. For instance, $\mathcal{A}$ could create and submit a public key such that any function evaluation will require to invert $b^*$ according to the permutation $e(ek^*, \cdot)$. We show how to prevent such queries starting from the following basic intuition.

Assume that a value $b \in \{0,1\}^\lambda$ is (somehow) encoded into the public key $PK$ and recall that we denote by $x$ the input of $F^{\mathcal{O}}(SK, \cdot)$. Then we have two mutually exclusive cases:

1. $F^{\mathcal{O}}(SK, \cdot)$ inverts $b$ on a large fraction of the $x$'s;
2. $F^{\mathcal{O}}(SK, \cdot)$ inverts $b$ only on a negligible fraction of the $x$'s (even on no $x$ in the most extreme case).

Now, recall that a VUF attacker is allowed to query the function (and see the corresponding proofs) for inputs of her choice. Therefore, if $\mathcal{A}$ queries the function oracles on a sufficiently large number of the $x$'s, then $\mathcal{A}$ will learn the inverses of all the "frequent" $b$'s of type 1 with high probability. On the other hand, for any $b$ of type 2, the probability that running $F^{\mathcal{O}}(SK, x)$ on a random $x$ asks to invert $b$ is negligible.

**Ensuring $\mathcal{A}$ Has Access to the Function Oracles.** The above intuition suggests that any algorithm querying $\mathcal{B}$ must provide as additional input sufficiently many triples $(x_i, y_i, \pi_i)$ such that $\pi_i$ is a valid proof for "$F^{\mathcal{O}}(SK, x_i) = y_i$". This way, if a ATDP adversary embeds a "type 1" $b$ into $PK$, then it must know its inverse in order to provide the above triples. Or, if a "type 2" $b$ is encoded into $PK$, then with high probability the attacker $\mathcal{A}$ will not gain any further information on its inverse from seeing the evaluation of $F^{\mathcal{O}}(SK, x^*)$ for a random $x^*$.

Although such restriction seems to capture the right intuition, we observe that it is not sufficient to prevent the adversary from exploiting $\mathcal{B}$. To see this, assume that $\mathcal{A}$ encodes its challenge $(ek^*, b^*)$ into $PK$ such that $b^*$ is of type

1, namely $F^{\mathcal{O}}(SK, x)$ queries $d(td^*, b^*)$ on a large fraction of the $x$'s. Then, if the attacker $\mathcal{A}$ is allowed to choose the inputs $x_1, \ldots, x_\ell$ provided to $\mathcal{B}$, then it might take all of them from the small fraction that does not require to invert $b^*$. In this case our previous argument would fail.

Therefore, in order to prevent these dangerous queries, we deny $\mathcal{A}$ choosing the inputs $x_1, \ldots, x_\ell$. That is, we define a two-steps oracle $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ where $\mathcal{B}_1$ chooses $\ell$ random inputs, and $\mathcal{B}_2$ evaluates the VUF *only* if it gets as input values and proofs for $x$'s that were chosen by $\mathcal{B}_1$. For this we will require that $\mathcal{B}_1$ is essentially a random function that, given as input a VUF public key and a collection of oracle circuits implementing a VUF, outputs $\ell$ random strings.

Furthermore, observe that this restriction is not a problem for the attacker that we build against the VUF, because it has access to the function oracles, $F(SK, \cdot)$ and $\Pi(SK, \cdot)$, that compute these values and proofs for her. On the other hand, an ATDP adversary now has restricted power as it does not know $b^*$'s inverse.

**Avoiding Malicious Keys.** Finally, the last type of dangerous queries that we have to handle are those where the attacker $\mathcal{A}$ queries $\mathcal{B}$ on an "invalid" public key $PK$. By "invalid" we mean that $PK$ is not the output of an honest execution of the key generation algorithm $KG^{\mathcal{O}}(SK)$. The problem is again that an evaluation of $F^{\mathcal{O}}(SK, x)$ can reveal "sensitive" informations about the trapdoor permutation. Indeed, observe that an execution of $F^{\mathcal{O}}$ must use the $d(\cdot, \cdot)$ oracle in a *significant* way or the VUF cannot be secure.[5] Thus, one may think about designing $\mathcal{B}$ in such a way that it rejects any queries that involve invalid public keys. However, this solution is still dangerous as $\mathcal{B}$ might be used to test the validity of public keys. We solve the issue by defining $\mathcal{B}$ such that it computes the answer using a different key $SK'$ and a different oracle $\mathcal{O}''$ but that the new function $F^{\mathcal{O}''}(SK', \cdot)$ behaves in almost all cases as the original one $F^{\mathcal{O}}(SK, \cdot)$. More precisely, the oracle $\mathcal{B}$ evaluates $F^{\mathcal{O}''}(SK', \cdot)$ using a key $SK'$ (that is most likely different from $SK$) and an oracle $\mathcal{O}''$ which is also different from the real oracle $\mathcal{O}$. The key $SK'$ is computed such that it corresponds to the "real" key $PK$ under $\mathcal{O}''$ (i.e., $PK \leftarrow KG^{\mathcal{O}''}(SK')$). The idea is to construct $\mathcal{O}''$ such that is close to $\mathcal{O}$. Then we can show that evaluating $F^{\mathcal{O}''}(SK', x)$ is basically the same as evaluating $F^{\mathcal{O}}(SK, x)$.

The hope is that $\mathcal{O}''$ differs from $\mathcal{O}$ in the points that may represent dangerous queries. If this is the case, then we are done as computing $F^{\mathcal{O}''}(SK', x)$ will not reveal sensitive informations on the real ATDP. More precisely, our oracle $\mathcal{B}$ selects uniformly at random a secret key $SK'$ and an oracle $\mathcal{O}''$ such that $PK = KG^{\mathcal{O}''}(SK')$ and $\mathcal{O}''$ agrees with $\mathcal{O}$ on those points that are already known to the adversary.

**Discovering All ATDP Public Keys.** In order to correctly simulate a run of $F^{\mathcal{O}''}$ it is important that our oracle has discovered all the ATDP public keys $ek$ that may be needed while running $F^{\mathcal{O}''}$. More precisely it needs to know all

[5] For instance, if $F^{\mathcal{O}}$ does not use the oracles, then an exponentially-strong adversary could always evaluate the circuit associated to $F$.

the public keys that were generated during the honest execution of $KG^{\mathcal{O}}(SK)$. So, to discover these public keys we define $\mathcal{B}$ such that it runs $V^{\mathcal{O}}$ on all the received triples $(x_i, y_i, \pi_i)$ and collect all the queries made by the algorithm. Since by Assumption 1, the algorithm $KG$ generates at most $q$ of such $ek$'s, it is sufficient to repeat the above step on sufficiently many triples, say $q^c$ for some constant $c$ that we will specify later. This allows us to discover all the public keys with high probability.

### 3.2 The Formal Separation Theorem

In this section we formalize the techniques that we use to prove our result. The core of our proof is the description of two oracles $\mathcal{O}$ and $\mathcal{B}$. The first oracle $\mathcal{O} = (g, e, d)$ implements a perfectly random trapdoor permutation and it is obvious that a secure ATDP exists relative to $\mathcal{O}$ (where the security follows from the randomness of the function). Therefore, we follow the strategy of defining a "weakening" oracle $\mathcal{B}$ whose main task is to break the security of a given VUF construction. This approach is formalized in the following theorem:

**Theorem 1 (formally restated).** *Let $\mathcal{O} = (g, e, d)$ be a random trapdoor permutation oracle. Then, there exists an oracle $\mathcal{B}$ such that for every VUF construction $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ which is correct and unique we have:*

*(i) there is an adversary $\mathcal{A}$ such that $\mathcal{A}^{\langle \mathcal{O}, \mathcal{B} \rangle}$ breaks the security of the VUF with non-negligible probability;*
*(ii) there exists an ATDP construction $(G^{\mathcal{O}}, E^{\mathcal{O}}, D^{\mathcal{O}})$ relative to $\mathcal{O}$ such that no adversary $\mathcal{A}^{\langle \mathcal{O}, \mathcal{B} \rangle}$ can break its security with non-negligible probability.*

We formally prove this theorem defining the oracles $\mathcal{O}$ and $\mathcal{B}$ in the following paragraphs. Afterwards, we prove the theorem by stating two separate lemmata. The first one, given in Section 4, shows the insecurity of the VUF, whereas the second lemma (Section 5) proves the existence of a secure ATDP.

**The Oracle $\mathcal{O}$.** We prove our separation in a relativized model where each algorithm has access to a random trapdoor permutation oracle $\mathcal{O} = (g, e, d)$ where $g, e$ and $d$ are sampled uniformly at random from the set of all functions with the following conditions:

- $g : \{0,1\}^\lambda \to \{0,1\}^\lambda$ takes a trapdoor key $td$ and outputs a public key $ek$.
- $e : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a function that takes in input a public key $ek$ and a value $a$ and outputs $b$. For every $ek \in \{0,1\}^\lambda$, $e(ek, \cdot)$ is required to be a permutation over $\{0,1\}^\lambda$.
- $d : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a function that on input a pair $(td, b)$ outputs the unique $a \in \{0,1\}^\lambda$ such that $e(g(td), a) = b$.

Since the permutation is defined over $\{0,1\}^\lambda$, it is easy to see that the oracle is also an enhanced TDP.

**Notation.** We write $\mathcal{A}^{\mathcal{O}}$ to denote that an algorithm $\mathcal{A}$ is given access to an oracle $\mathcal{O}$. We will use square brackets to denote queries and mappings. For instance, we write $[e(ek, a)]$ to denote a query to $e$ with input $ek$ and $a$. Otherwise $e(ek, a)$ refers the actual value of the function $e$ on the given input. We write $[e(ek, a) = b]$ to denote that there is a mapping between $a$ and $b$ in the function $e(ek, \cdot)$. Also, for ease of presentation, we will sometimes abuse the notation and write $\mathcal{O}(\alpha)$ to denote the answer of $\mathcal{O}$ on a query $\alpha$ which depends on the type of $\alpha$. For example if $\alpha = [e(ek, a)]$, then $\mathcal{O}(\alpha) = e(ek, a)$.

Let $\mathcal{O}_k$ (with $k \in \{1, 2\}$) be a partial (aka suboracle) oracle. We define the set of all public keys that are contained into the queries of $\mathcal{O}_k$ as

$$Z(\mathcal{O}_k) = \{ek : [g(\cdot) = ek] \in \mathcal{O}_k \text{ or } [e(ek, \cdot) = \cdot] \in \mathcal{O}_k\}.$$

**Suboracles.** Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two (possibly partial) trapdoor permutation oracles. We write $\mathcal{O}_1 \diamond_c \mathcal{O}_2$ to denote the oracle that answers with $\mathcal{O}_1$ only if $\mathcal{O}_2$ is not defined. Otherwise, it answers with $\mathcal{O}_2$. If $\mathcal{O}_1 = (g_1, e_1, d_1)$ and $\mathcal{O}_2 = (g_2, e_2, d_2)$ are two trapdoor permutation oracles as defined above, then its composition is defined by composing each algorithm, namely:

$$\mathcal{O}_1 \diamond_c \mathcal{O}_2 = (g_1 \diamond_c g_2, e_1 \diamond_c e_2, d_1 \diamond_c d_2)$$

This definition needs some more explanation. We want that the oracle obtained from the composition of two oracles preserves the properties of the two individual oracles. In particular, we require that $(e_1 \diamond_c e_2)(ek, \cdot)$ is a permutation for any valid $ek$. The problem is that the permutations $e_1$ and $e_2$ may contain collisions, namely there exist $ek$ and two distinct values $a, a' \in \{0, 1\}^\lambda$ such that $e_2(ek, a) = e_1(ek, a')$. To handle such collisions we use the same technique suggested in [33]. We define $e = e_1 \diamond_c e_2$ as follows: let $ek, a, b$ be values such that $[e_2(ek, a) = b] \in \mathcal{O}_2$. We set $e(ek, a) = b$. If there exists a value $a' \neq a$ such that $[e_1(ek, a') = b] \in \mathcal{O}_1$, then let $b' = e_1(ek, a)$ and set $e(ek, a') = b'$. The composition $d = d_1 \diamond_c d_2$ is defined to be consistent with $g$ and $e$.

**VUF in the Presence of Our Oracle.** For a simpler exposition we make some general assumptions on any VUF construction with access to the oracle $\mathcal{O} = (g, e, d)$. First, we consider a slightly relaxed definition of the VUF algorithms $(KG, F, \Pi, V)$ as follows. The algorithm $KG(SK)$ takes as input a secret key $SK \in \{0, 1\}^n$ and outputs $PK \in \{0, 1\}^n$. The input of $F$ and $\Pi$ are the secret key $SK$ and a value $x \in \{0, 1\}^n$. The output of $F$ is the function value $y \in \{0, 1\}^n$, whereas the output from $\Pi$ is the corresponding $\pi$, respectively. Finally, $V$ is given in input the public key $PK$, an input $x$, an output $y$ and a proof $\pi$ and outputs 1 if it accepts the proof, or 0 otherwise. In the above description $n$ is a function of the security parameter $\lambda$.

Recall that we assume towards contradiction that there exists a black-box reduction of VUFs to ATDPs. Then we denote by $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ the corresponding VUF construction. According to our notation, each algorithm has access to the $(g, e, d)$ oracles and they have to use them in a "significant" way to implement a secure primitive. Also, by definition of black-box reduction, this

construction is a correct VUF implementation, that satisfies completeness and uniqueness according to Definition 1.

**Assumption 1.** *For a simpler exposition, in our proofs we use the following assumptions:*

- *each algorithm is unbounded, but makes at most $q = poly(\lambda)$ oracle queries during its execution;*
- *every query $d(td, \cdot)$ is followed by a query $g(td)$;*
- *the proof algorithm is deterministic;*
- *the verification algorithm is deterministic;*
- *the completeness of the VUF holds in a perfect sense.*

Before proceeding with the description of the breaking oracle, we briefly justify these assumptions. The first condition is reasonable because we consider only efficient constructions and moreover, it allows us to easily quantify the advantage of our adversaries. The second one avoids queries of the adversary to $d(\cdot, \cdot)$ using a trapdoor key without knowing the corresponding public key. This assumption is also common and has been previously used in e.g., [23]. Assuming that the proof algorithm is deterministic is not a restriction as we can turn any VRF with a probabilistic proof algorithm into one having a deterministic algorithm by applying a PRF to the input and the private seed of the VRF to derive the randomness. Completeness and uniqueness follow easily from the VRF (note that uniqueness only holds w.r.t. to the output of the function and not w.r.t. the proof). The rest follows easily applying a standard hybrid argument. The assumptions on deterministic verification and perfect completeness have already been addressed in [17], hence we omit the discussion here.

**A Formal Definition of $\mathcal{B}$.** Here, we provide a formal description of our oracle $\mathcal{B}$, which is composed by the following two algorithms $(\mathcal{B}_1, \mathcal{B}_2)$:

Algorithm $\mathcal{B}_1$:
INPUT: A collection of oracle circuits $VUF^{\mathcal{O}} = (KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ implementing a VUF, and a VUF public key $PK$
OUTPUT: $x_1, \ldots, x_\ell \in \{0,1\}^n$.
COMPUTATION: To each input $(VUF^{\mathcal{O}}, PK)$, the algorithm $\mathcal{B}_1$ associates a random function $f : \{0,1\}^n \to \{0,1\}^n$. For $i = 1$ to $\ell$, it computes $x_i = f(i)$, and finally it returns $x_1, \ldots, x_\ell$.

Algorithm $\mathcal{B}_2$:
INPUT: A collection of oracle circuits $VUF^{\mathcal{O}} = (KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ implementing a VUF, a VUF public key $PK$ and a set $\{(x_i, y_i, \pi_i)\}_{i=1}^{\ell}$ such that $x_i \in \{0,1\}^n, y_i \in \{0,1\}^m$, and $\pi_i$ is in the range of $\Pi(\cdot, \cdot)$.
OUTPUT: $x^* \in \{0,1\}^n, y^* \in \{0,1\}^m$.
COMPUTATION: The oracle performs the following computation:
- **Step 1:** Invoke $(x'_1, \ldots, x'_\ell) \leftarrow \mathcal{B}_1(VUF^{\mathcal{O}}, PK)$ and check that the values $x_1, \ldots, x_\ell$ received as input are equal to $(x'_1, \ldots, x'_\ell)$ returned by $\mathcal{B}_1$. Otherwise, output $\perp$.

- **Step 2:** For all $i = 1$ to $\ell$ run the algorithm $V^{\mathcal{O}}(PK, x_i, y_i, \pi_i)$ and collect into a partial oracle $\mathcal{O}_Q$ all the queries that are made during each run. If there is some $j$ such that the verification algorithm does not accept, stop and output $\perp$.
- **Step 3:** Find a secret key $SK'$ and a partial oracle $\mathcal{O}'$ such that:
  1. $KG^{\mathcal{O}'}(SK') = PK$, $F^{\mathcal{O}'}(SK', x_i) = y_i$ and $\Pi^{\mathcal{O}'}(SK', x_i) = \pi_i$.
  2. $\mathcal{O}' \supseteq \mathcal{O}_Q$ and $|\mathcal{O}'| \leq |\mathcal{O}_Q| + q$ where $q$ is the same value defined in Assumption 1.
- **Step 4:** Define $\mathcal{O}'' = \mathcal{O} \diamond_c \mathcal{O}'$
- **Step 5:** Choose $x^*$ uniformly at random in $\{0,1\}^n$ such that $x^* \neq x_i$ for all $i = 1$ to $\ell$. Run $y^* \leftarrow F^{\mathcal{O}''}(SK', x^*)$ and $\pi^* \leftarrow \Pi^{\mathcal{O}''}(SK', x^*)$.
- **Step 6:** Run $V^{\mathcal{O}''}(PK, x^*, y^*, \pi^*)$. If $V^{\mathcal{O}''}$ asks a query $\alpha$ such that $\mathcal{O}''(\alpha) \neq \mathcal{O}(\alpha)$, then return $\perp$. Otherwise output $y^*$.

**Complexity of $\mathcal{B}$.** Based on Assumption 1, we evaluate the cost of each query to $\mathcal{B}$ in terms of queries to the oracle $\mathcal{O}$. Since the function $f$ chosen by $\mathcal{B}_1$ is completely independent of $\mathcal{O}$, we do not count its cost. Instead a query to $\mathcal{B}_2$ counts $\ell q + 3q + |\mathcal{O}'|$ queries to $\mathcal{O}$ in total. This cost is obtained as follows: Step 2 makes $\ell q$ queries as it evaluates $V$ $\ell$ times, Step 3 is made offline, Step 4 counts $|\mathcal{O}'|$ queries that are needed to perform the $\diamond_c$ operation and finally Step 5 and Step 6 require $2q$ and $q$ queries respectively.

## 4 Insecurity of VUFs Relative to Our Oracles

In this section we formally show that for every candidate black-box construction $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ of a VUF from ATDP there is an efficient adversary $\mathcal{A}$ that breaks the unpredictability of the VUF with non-negligible probability $1 - \delta$ by making a polynomial number of oracle queries to $\langle \mathcal{O}, \mathcal{B} \rangle$.

Let $q$ be the maximum number of oracle queries that can be made by the VUF algorithms (according to Assumption 1) and $c \in \mathbb{N}$ be a sufficiently large constant specified below. Without loss of generality, in the following proof we assume $q \geq 2$ and we fix $c$ such that $\delta \leq \frac{3}{eq^{c-1}}$ and our adversary has non-negligible advantage at least $1 - \delta$. Also we set $\ell = q^c$.

Our adversary $\mathcal{A}$ works as follows:

INPUT: A public key $PK$ and access to the function oracles $F(SK, \cdot), \Pi(SK, \cdot)$.
OUTPUT: $x^*, y^* \in \{0,1\}^n$.
ALGORITHM: Our algorithm performs the following steps:

1. Query $\mathcal{B}_1$ on input $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}}), PK$ and obtain $x_1, \ldots, x_\ell$.
2. Query the VUF oracles $F(SK, \cdot), \Pi(SK, \cdot)$ on $x_i$ for all $i = 1$ to $\ell$. Let $\{y_1, \pi_1, \ldots, y_\ell, \pi_\ell\}$ be the values obtained from such queries.
3. Query $\mathcal{B}_2$ on input $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}}), PK, \{x_1, y_1, \pi_1, \ldots, x_\ell, y_\ell, \pi_\ell\}$.
4. If $\mathcal{B}_2$ returns $\perp$, then halt and fail. Otherwise, if $\mathcal{B}_2$ returns $(x^*, y^*)$, then output $(x^*, y^*)$.

Then we are able to state the following lemma:

**Lemma 1.** *The adversary $\mathcal{A}$ defined above with input PK and oracle access to $\langle \mathcal{O}, \mathcal{B} \rangle$ wins the unpredictability experiment with probability at least $1 - \frac{3}{eq^{c-1}}$ and makes at most $2q^{c+1} + 4q$ oracle queries.*

The proof is given in the full version [14].

## 5 Security of ATDPs Relative to Our Oracles

In this section we show the existence of a trapdoor permutation $(G^{\mathcal{O}}, E^{\mathcal{O}}, D^{\mathcal{O}})$ that is adaptively one-way even against adversaries that have access to $\mathcal{B}$. The construction is straightforward as each algorithm forwards its input to the corresponding oracle, namely: $G^{\mathcal{O}}(td) = g(td), E^{\mathcal{O}}(ek, a) = e(ek, a)$ and $D^{\mathcal{O}}(td, b) = d(td, b)$.

By the randomness of the oracle $\mathcal{O}$, it is easy to see that the above construction is a secure ATDP when the adversary is given access only to $\mathcal{O}$. Therefore, in order to prove its security relative to the oracle $\mathcal{B}$, we will show that $\mathcal{B}$ does not help to break the one-wayness of $(G^{\mathcal{O}}, E^{\mathcal{O}}, D^{\mathcal{O}})$, namely that $\mathcal{B}$ can be simulated to the adversary $\mathcal{A}$. Now we can state the following lemma:

**Lemma 2.** *Let $(G^{\mathcal{O}}, E^{\mathcal{O}}, D^{\mathcal{O}})$ be an adaptive trapdoor permutation where each algorithm forwards its input to $g, e$, and $d$ respectively. Then, for every adversary $\mathcal{A}$ that has access to $\langle \mathcal{O}, \mathcal{B} \rangle$ and makes at most $q$ oracle queries, there is a sufficiently large $\lambda$ such that the probability that $\mathcal{A}$ succeeds in the adaptive one-wayness experiment against the above construction is at most negligible.*

### 5.1 Defining the Simulator

Recall that the main idea is to show that $\mathcal{A}$ can simulate the oracle $\mathcal{B}$ locally. To do so, we show that for every $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that gets the same input as $\mathcal{A}$, but which does not have access to $\mathcal{B}$. We then show that the success probability of $\mathcal{S}$ is close to that of $\mathcal{A}$.

**Intuition for the Simulator.** In the first step, the simulator generates a random trapdoor permutation oracle $\mathcal{O}_{\mathcal{S}}$ locally, except for the portion concerning the permutation $e(ek^*, \cdot)$. In particular $\mathcal{O}_{\mathcal{S}}$ is defined progressively by choosing its answers uniformly at random. Moreover, we construct $\mathcal{S}$ such that it collects into a partial oracle $\mathcal{O}^*$ all the queries of the form $[e(ek^*, \cdot)]$ that $\mathcal{A}$ makes during the simulation. This way, $\mathcal{S}$ knows all the trapdoors of all the public keys (but $ek^*$) and is therefore able to evaluate all inversion queries $d(td, \cdot)$ where $g(td) \neq ek^*$.

The first three steps of the algorithm $\mathcal{B}_2$ can easily be simulated as in the real case. The first difference comes up into Step 4 where $\mathcal{S}$ has to define the oracle $\mathcal{O}''$.The difficulty here is that the simulator does not know the entire $\mathcal{O}$ and thus it cannot compute the composition $\mathcal{O} \diamond_c \mathcal{O}'$. We solve this problem using an idea similar to the one used in [33]. Namely, we define $\mathcal{O}''$ such that it is consistent

with the partial oracles that are known to $\mathcal{S}$ so far (i.e., $\mathcal{O}_{\mathcal{S}}, \mathcal{O}^*$ and $\mathcal{O}'$) and we forward all other queries to $\mathcal{O}$. This solves most of the problematic cases due to the fact that the adversary $\mathcal{A}$ only knows *queried* mappings (which are also known to $\mathcal{S}$ since it has stored all of them).

One remaining issue are those queries $[d(td', b)]$ such that $td'$ is the trapdoor that is "virtually" associated to $ek^*$ (i.e., $[g(td') = ek^*] \in \mathcal{O}'$) and there is no known mapping $[e(ek^*, \cdot) = b]$ in $\mathcal{O}^*$. Indeed, recall that the simulator does not know the real trapdoor $td^*$ such that $[g(td^*) = ek^*] \in \mathcal{O}$, and also notice that forwarding these unknown queries to $\mathcal{O}$ would inevitably lead to an inconsistent mapping. Assume for example that $\alpha = [d(td', b)]$ is answered with $\mathcal{O}(\alpha) = a$. Then we have a mapping $[e(ek^*, a) = b] \in \mathcal{O}''$, but it is very unlikely that $[e(ek^*, a) = b]$ is in $\mathcal{O}$. Such inconsistencies could potentially be discovered in Step 6 which would cause the simulation to output $\perp$ while it should not.

Fortunately, we show how to handle such queries by using the external inversion oracle $I(ek^*, \cdot)$. Finally, the last remaining problem is the query $\alpha = [d(td', b^*)]$. We cannot answer this query correctly (at least as long as the inverse of $b^*$ has not been discovered before), however we will show that this case only happens with negligible probability. The main idea is that either $\mathcal{A}$ cannot provide an accepting input to $\mathcal{B}_2$ or (in the case that we have passed all the checks and have reached Step 5) the probability that this query cannot be answered is very small.

The full description of the simulator and the proof are provided in the full version [14].

**Acknowledgments.** We thank the anonymous referees for their valuable comments. We would like to thank Yevgeniy Vahlis for helpful clarifications about black-box separation techniques and Michel Abdalla for helpful discussions on this work. We also thank Jonathan Katz and Arkady Yerukhimovich for helpful discussions about their augmented black-box model. 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 and by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG).

## References

1. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science, pp. 120–130. IEEE Computer Society Press (1999)
2. Micali, S., Reyzin, L.: Soundness in the Public-key Model. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 542–565. Springer, Heidelberg (2001)
3. Micali, S., Rivest, R.L.: Transitive Signature Schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 236–243. Springer, Heidelberg (2002)
4. Jarecki, S., Shmatikov, V.: Handcuffing Big Brother: an Abuse-Resilient Transaction Escrow Scheme. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 590–608. Springer, Heidelberg (2004)
5. Liskov, M.: Updatable Zero-Knowledge Databases. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 174–198. Springer, Heidelberg (2005)

6. Lysyanskaya, A.: Unique Signatures and Verifiable Random Functions from the DH-DDH Separation. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 597–612. Springer, Heidelberg (2002)
7. Dodis, Y.: Efficient Construction of (Distributed) Verifiable Random Functions. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 1–17. Springer, Heidelberg (2002)
8. Dodis, Y., Yampolskiy, A.: A Verifiable Random Function with Short Proofs and Keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005)
9. Abdalla, M., Catalano, D., Fiore, D.: Verifiable Random Functions from Identity-Based Key Encapsulation. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 554–571. Springer, Heidelberg (2009)
10. Hohenberger, S., Waters, B.: Constructing Verifiable Random Functions with Large Input Spaces. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 656–672. Springer, Heidelberg (2010)
11. Reingold, O., Trevisan, L., Vadhan, S.P.: Notions of Reducibility between Cryptographic Primitives. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 1–20. Springer, Heidelberg (2004)
12. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: 21st Annual ACM Symposium on Theory of Computing, pp. 44–61. ACM Press (1989)
13. Gertner, Y., Kannan, S., Malkin, T., Reingold, O., Viswanathan, M.: The relationship between public key encryption and oblivious transfer. In: 41st Annual Symposium on Foundations of Computer Science, pp. 325–335. IEEE Computer Society Press (2000)
14. Fiore, D., Schröder, D.: Uniqueness is a different story: Impossibility of verifiable random functions from trapdoor permutations. Cryptology ePrint Archive, Report 2010/648 (2010), `http://eprint.iacr.org/`
15. Bresson, E., Monnerat, J., Vergnaud, D.: Separation Results on the "One-More" Computational Problems. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 71–87. Springer, Heidelberg (2008)
16. Fischlin, M., Schröder, D.: On the Impossibility of Three-Move Blind Signature Schemes. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 197–215. Springer, Heidelberg (2010)
17. Brakerski, Z., Goldwasser, S., Rothblum, G.N., Vaikuntanathan, V.: Weak Verifiable Random Functions. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 558–576. Springer, Heidelberg (2009)
18. Naor, M., Reingold, O.: Synthesizer and their applications to the parallel construction of pseudo-random functions. Journal of Computer and System Sciences 58 (1999)
19. Maurer, U.M., Sjödin, J.: A Fast and Key-Efficient Reduction of Chosen-Ciphertext to Known-Plaintext Security. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 498–516. Springer, Heidelberg (2007)
20. Dwork, C., Naor, M.: Zaps and their applications. SIAM Journal on Computing 36, 1513–1543 (2007)
21. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM 33, 792–807 (1986)
22. Kiltz, E., Mohassel, P., O'Neill, A.: Adaptive Trapdoor Functions and Chosen-Ciphertext Security. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 673–692. Springer, Heidelberg (2010)

23. Boneh, D., Papakonstantinou, P.A., Rackoff, C., Vahlis, Y., Waters, B.: On the impossibility of basing identity based encryption on trapdoor permutations. In: 49th Annual Symposium on Foundations of Computer Science, pp. 283–292. IEEE Computer Society Press (2008)
24. Hsiao, C.-Y., Reyzin, L.: Finding Collisions on a Public Road, or Do Secure Hash Functions Need Secret Coins? In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 92–105. Springer, Heidelberg (2004)
25. Goldwasser, S., Ostrovsky, R.: Invariant Signatures and Non-interactive Zero-Knowledge Proofs Are Equivalent. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 228–245. Springer, Heidelberg (1993)
26. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: 21st Annual ACM Symposium on Theory of Computing, pp. 25–32. ACM Press (1989)
27. Dodis, Y., Puniya, P.: Feistel Networks Made Public, and Applications. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 534–554. Springer, Heidelberg (2007)
28. Dodis, Y., Oliveira, R., Pietrzak, K.: On the Generic Insecurity of the Full Domain Hash. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 449–466. Springer, Heidelberg (2005)
29. Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM Journal on Computing 36, 915–942 (2006)
30. Gertner, Y., Malkin, T., Myers, S.: Towards a Separation of Semantic and CCA Security for Public Key Encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 434–455. Springer, Heidelberg (2007)
31. Rosen, A., Segev, G.: Chosen-Ciphertext Security via Correlated Products. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 419–436. Springer, Heidelberg (2009)
32. Katz, J., Schröder, D., Yerukhimovich, A.: Impossibility of Blind Signatures from One-Way Permutations. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 615–629. Springer, Heidelberg (2011)
33. Vahlis, Y.: Two Is a Crowd? A Black-Box Separation of One-Wayness and Security under Correlated Inputs. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 165–182. Springer, Heidelberg (2010)