Uniqueness is a Different Story: Impossibility of Verifiable Random Functions from Trapdoor Permutations*

Dario Fiore[1] and Dominique Schröder[2]

[1] École Normale Supérieure, CNRS - INRIA, Paris, France. dario.fiore@ens.fr
[2] Darmstadt University of Technology, Germany. schroeder@me.com

Abstract. Verifiable random functions (VRFs), firstly proposed by Micali, Rabin, and Vadhan (FOCS 99), 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. Due to their properties, VRFs are a fascinating primitive that have found several theoretical and practical applications. However, despite their popularity, constructing VRFs seems to be a challenging task. Indeed only a few constructions based on specific number-theoretic problems are known and basing a scheme on a general assumption is still an open problem. Towards this direction, Brakerski, Goldwasser, Rothblum, and Vaikuntanathan (TCC 2009) recently showed that verifiable random functions cannot be constructed from one-way permutations in a black-box way.

In this paper we put forward the study of the relationship between VRFs and well-established cryptographic primitives. As our main result, we prove that VRFs cannot be based on the existence of trapdoor permutations (TDPs) in a black-box manner.

Our result sheds light on the nature of VRFs and can be considered interesting for at least two reasons:

- First, given the separation result of Brakerski *et al.*, one may think as though VRFs would naturally belong to the "public-key world", and thus it is interesting to figure out their relationship with other public-key primitives. In this sense, our result shows that VRFs are much stronger because we imply separations of VRFs from most of the primitives in this world: basically everything that is implied by TDPs is strictly weaker. These primitives include e.g., public-key encryption (even CCA-secure), oblivious transfer, and key-agreement.
- Second, the notion of VRFs is closely related to other two primitives: weak verifiable random functions (weak-VRFs) and verifiable pseudorandom generators (VPRGs). Weak-VRFs, defined by Brakerski *et al.*, are a relaxation of VRFs in which the pseudorandomness holds only with respect to random inputs. VPRGs, introduced by Dwork and Naor (FOCS 2000), are pseudorandom generators that allow the owner of the seed to prove the correctness of subsets of the generated bits. It is well known that "regular" PRFs can be constructed from pseudorandom generators and from weak-PRFs in a black-box way. It was thus an open problem (already recognized by Dwork and Naor in their paper) whether similar approaches could be applicable to the "verifiable" analogous of such primitives. Since weak-VRFs and VPRGs are implied by TDPs, we give a negative answer to this problem showing that the case of verifiable random functions is essentially different.

* 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. Dominique Schröder is supported by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG).

1 Introduction

The notion of verifiable random functions (VRF) was introduced by Micali, Rabin, and Vadhan in [28]. In a nutshell, VRFs are random functions with the additional property that these functions provide a proof verifying the input-output relationships. More formally, a VRF is defined by a pair of keys (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 π_x. This proof is publicly verifiable i.e., given the public key PK one can efficiently verify (using π_x) that the statement "$F(SK, x) = y$" holds. For security, VRFs must satisfy two properties: pseudorandomness and uniqueness. Informally, *pseudorandomness* states that the function looks random at any input x for which no proof has been issued. *Uniqueness* guarantees that for any x, $y = F(SK, x)$ is the only image for which a valid proof can be produced (even if the public key is maliciously-chosen).

In some sense a VRF can be seen as the public-key equivalent of a pseudorandom function. Due to their properties, VRFs are a fascinating primitive that have found application in many scenarios, both theoretical and practical: 3-rounds resettable zero-knowledge [29], non-interactive lottery systems and micropayment schemes [30], a verifiable transaction escrow scheme [23] and updatable zero-knowledge sets [25]. However, despite their popularity, constructing VRFs seems to be a challenging task. Indeed only a few constructions are known so far [28,26,8,11,1,21] (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. Moreover, 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[3]. 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. The black-box approach can be briefly 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 in [33] by Reingold *et al.* who gave different "flavors" of black-box reductions depending on the "degree" of black-box access. In this introduction we use the generic definition as sketched above and provide a more formal description in Section 2.

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 (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 as-

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

sumptions 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 in [22] 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*. In their work they ruled out any black-box construction of key-agreement protocols (KA) from one-way functions. Gertner *et al.* show in [14] 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, such as digital signatures, pseudorandom generators (PRGs), PRFs 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).

It is worth to mention that another methodology, called *meta-reductions*, for separating primitives in a black-box sense is known and put forward in e.g., [32,7,6,13]. The basic idea of this approach is to build "a reduction against the reduction". For example, [32,7,6] consider the impossibility of reductions from secure encryption or signatures to a given RSA instance. Since we do not follow this approach here, we refer the interested reader to one of the cited papers.

1.1 Our results

In this paper we investigate the relationship between verifiable random functions and well-studied cryptographic primitives. The first step towards this goal was recently given in [4] by Brakerski, Goldwasser, Rothblum and Vaikuntanathan who separated VRFs from one-way permutations. In [4] Brakerski *et al.* introduce the notion of *weak verifiable random functions (wVRFs)*, a primitive that is similar to weak-PRFs [31] as pseudorandomness only holds with respect to randomly chosen inputs. Moreover, they provide a construction of wVRFs based on the existence of trapdoor permutations and show that wVRFs are essentially equivalent to non-interactive zero knowledge proof (NIZK) systems. Since it is well known that "regular" PRFs can be constructed from weak PRFs [31,27], to study the relation between the primitives, the first attempt would be to build a VRF out of any wVRF (we intuitively show in Appendix B why this is difficult).

Another work that is closely related to this topic is that one of Dwork and Naor who propose in [12] *verifiable pseudorandom generators* (VPRGs) and show their equivalence to NIZK proofs. 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 suggest also a construction of 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 [17]. Indeed the problem of combining VPRGs to obtain a VRF was already recognized and left open by Dwork and Naor in [12].

Therefore one may naturally hope that a similar approach could also be applicable to the "verifiable" case, 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 trapdoor permutations exist. Precisely, we prove our separation theorem for stronger primitives, i.e., verifiable unpredictable functions and adaptive trapdoor permutations.

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

Verifiable unpredictable functions (VUFs) (defined in [28]) are closely related to VRFs as the output is unique. The difference is that the output is not pseudorandom but unpredictable. Therefore, VUFs can also be seen as "*unique signatures*", where, for every public key, each message can have at most one valid signature[4].

Adaptive trapdoor functions (ATDFs), recently introduced by Kiltz, Mohassel, and O'Neill in [24], are essentially the same as regular trapdoor functions but the adversary is given access to an inversion oracle. Since ATDPs trivially imply TDPs, and VRFs imply VUFs, then our black-box separation implies the separations between TDPs and VUFs and TDPs and VRFs as well.

Implications of our result. Our result sheds light on the nature of VRFs, somewhat explaining why this primitive seems so hard to realize. Furthermore it raises interesting implications (see Figure 1). First, given the separation result of Brakerski *et al.*, one can think to 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 [24], 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.* [1][5]. It is interesting to note that IBE have been already separated from TDPs [3].

Second, our result points out how much crucial is the uniqueness property into the difference between regular digital signatures and unique signatures: though the former primitive has been shown to be equivalent to OWFs, the latter one is a much stronger primitive that cannot be built neither from TDPs.

Finally, since both weak-VRFs and VPRGs are implied by TDPs, our result rules out the possibility that VRFs can be constructed from weak-VRFs and/or VPRGs, as one might have hoped following approaches similar to those adopted for building PRFs from weak-PRFs and PRGs. This shows that the verifiable analogous of these primitives are essentially different.

1.2 Overview of the Techniques

Following the works in [3,36] we model an ideal random trapdoor permutation oracle $\mathcal{O}$ 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.

We prove our result following the oracle separation methodology but in a model slightly weaker than that of Impagliazzo-Rudich [22]. Indeed Impagliazzo and Rudich prove a relativizing separation

[4] At this stage, it is interesting to observe unique and deterministic signatures are two distinct primitives. Indeed a deterministic signature scheme can be obtained from any probabilistic one by generating the randomness via a PRF. However this generic transformation is not strong enough to obtain a *unique* signature, because uniqueness must hold even with respect to maliciously-generated public keys. Moreover, the signature could be easily re-randomizable. Consider for example the signature $\sigma = \sigma' \| 0$ and let the verification algorithm ignore the last bit. Then it is obvious that uniqueness could be easily violated by flipping the last bit.

[5] 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.

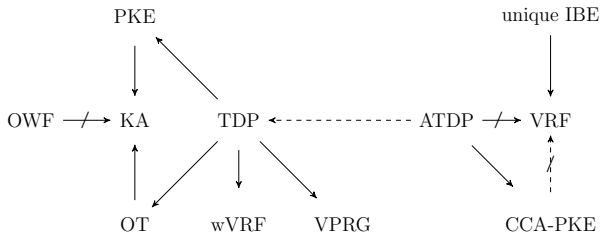

Fig. 1. This figure shows the relations among the different primitives.

between key-agreement protocols (KA) and one-way functions (OWFs) by showing an oracle relative to which OWFs exist but *any* KA fails to be secure. The same approach has been adopted in several other works (e.g. [14,15,36]) but it unfortunately fails in our case, because we cannot construct a single oracle relative to which ATDPs exist while VUFs do not. The reason is that a secure VUF in the presence of a random trapdoor permutation oracle in fact exists. The construction is the well known full-domain-hash signature scheme which is known to be secure in the random oracle model. Hence, we use a different approach for our proof. For every candidate black-box construction of VUF from $\mathcal{O}$ we show an oracle $\mathcal{B}$ such that the given VUF is insecure while a secure ATDP still exists relative to $\mathcal{O}$ and $\mathcal{B}$. We observe that this kind of impossibility results do not rule out a relativizing reduction, yet it is strong enough to rule out fully black-box reductions of VUFs to ATDPs. Our methodology is rather similar to the one that was adopted by Gertner *et al.* [16] to show that there is no black-box reduction of trapdoor functions to public-key encryption.

The core of our separation theorem is the definition of a 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 complicated. In particular the main difficulty is to prevent an attacker from exploiting $\mathcal{B}$ to break the one-wayness of an 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 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 a way such 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, π_i), where π_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 such 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, *but* that only works properly if the inputs x_i's are chosen by $\mathcal{B}_1$. Technically, we handle this by letting $\mathcal{B}_1$ and $\mathcal{B}_2$ communicate via a state, but we omit such details here.

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*) in [19] and they show that in the common random string model they are equivalent to non-interactive zero-knowledge proofs. Later, Micali, Rabin and Vadhan formally define VRFs and propose a construction (in the plain model) in [28]. 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 [18] (that extracts one random bit from polynomially-many unpredictable bits). Next, Lysyanskaya propose a VUF relying on a strong version of the Diffie-Hellman assumption in [26]. The hope following this two-steps approach is that a VUF should be easier to realize than a VRF. Unfortunately, the second step is very inefficient.

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 in [8], and Dodis and Yampolskiy give a construction based on the bilinear Diffie-Hellman inversion assumption in [11]. Abdalla, Catalano and Fiore [1] show the relationship between VRFs and a certain class of identity-based encryption schemes. 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 [21]. Another closely related work is one of Dodis and Puniya [10] who construct NIZK from verifiable random permutations (VRPs), that are the verifiable analog of pseudorandom permutations. The author also show how to convert a VRF into a VRP.

Black-Box Separations. As already mentioned, Impagliazzo and Rudich introduce the black-box separation methodology in [22]. This work inspired many researchers to follow their approach and to study the relationships between cryptographic primitives. We briefly recall the main results.

Gertner *et al.* show a black-box separation between public-key encryption (PKE) and oblivious-transfer (OT) in [14]. They show in addition, that TDPs cannot be constructed from PKE and that TDPs cannot be constructed from injective trapdoor functions (in the sense of black-box reductions). Next, Gertner, Malkin and Reingold prove that TDFs cannot be reduced (using black-box techniques) to PKE. They show that, in some sense, it is not possible to derandomize a

trapdoor predicate (i.e., the encryption algorithm). Recently, Gertner, Malkin, and Myers gave in [15] an important result on the relationship between semantically-secure public-key encryption (PKE) and PKE secure against chosen-ciphertext attacks. They rule out any black-box construction of a CCA1-secure PKE scheme from a semantically-secure one.

Boneh *et al.* prove a separation between identiy-based encryption (IBE) schemes and trapdoor permutations in [3]. Vahlis show that there is no black-box construction of correlation-secure trapdoor permutations (CP-TDPs) from classic TDPs in [36]. Given the results in [2] and [34], that show how to construct CCA secure encryption schemes from IBE and CP-TDFs respectively, the two works [3,36] are interesting as they rule out another possibility of basing CCA encryption on trapdoor permutations in a black-box way, which is still an open problem. Finally we mention the work of Dodis, Oliveira and Pietrzak [9] who used black-box separation techniques to show that no real hash function can instantiate the full-domain-hash signature scheme in the standard model.

2 Preliminaries

Before presenting our results we briefly recall some basic definitions. In what follows we denote by $\lambda \in \mathbb{N}$ the security parameter. An algorithm $\mathcal{A}$ is said to be *PPT* if it is a probabilistic Turing machine that runs in time polynomial in λ. Informally, we say that a function is *negligible* if it vanishes faster than the inverse of any polynomial. We usually refer to such a function as $\mathsf{negl}(\lambda)$. If S is a set, then $x \xleftarrow{\$} S$ indicates that x is chosen uniformly at random over S (which in particular assumes that S can be sampled efficiently).

In our proofs we will use the following fact:

Lemma 1 (Probabilistic lemma). *Let $X_1, \ldots, X_{n+1}$ be independent Bernoulli random variables such that $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$ for $i = 1, \ldots, n+1$ and some $p \in [0, 1]$. Let $\mathcal{E}$ be the event that the first n variables are sampled at 1 and X_{n+1} is sampled at 0. Then, $\Pr[\mathcal{E}] \leq \frac{1}{e \cdot n}$.*

In particular this lemma show that such probability does not depend on p.

2.1 Adaptive Trapdoor Permutations

In this section we give the formal definitions of the primitives that we use in our work. We begin with the definition of adaptive trapdoor permutations as defined in [24]. This primitive is 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 for the challenge value).

Definition 1 (Adaptive Trapdoor Permutations [24]). *A trapdoor permutation is a triple of efficient algorithms $\mathsf{ATDP} = (G, E, D)$ where:*

- *G is a probabilistic algorithm that on input 1^λ generates a pair of keys (ek, td)*
- *$E(ek, \cdot)$ implements a function $f_{ek}(\cdot)$ that is a permutation over $\{0,1\}^\lambda$*
- *$D(td, \cdot)$ uses the trapdoor key to evaluate the inverse of the function $f_{ek}(\cdot)$.*

We say that a triple $\mathsf{ATDP} = (G, E, D)$ is an adaptive trapdoor permutation (ATDP) if it satisfies the following adaptive one-wayness *property. Consider the following experiment:*

Experiment $\mathsf{Adaptive}_{\mathcal{A}}^{\mathsf{ATDP}}$
$(ek, td) \leftarrow G(1^\lambda)$;

$a \xleftarrow{\$} \{0,1\}^\lambda; b \leftarrow E(ek, a);$
$a' \leftarrow \mathcal{A}^{D(td,\cdot)}(ek, b);$
$\mathcal{A}$ wins if $a' = a$ and b was not asked to the $D(td, \cdot)$ oracle.

ATDP *is adaptive one-way if any PPT adversary $\mathcal{A}$ has at most negligible probability of succeeding in the experiment* $\mathsf{Adaptive}_{\mathcal{A}}^{\mathsf{ATDP}}$.

When the adversary is not given access to the inversion oracle then we obtain the standard notion of one-wayness of trapdoor permutations. Moreover, observe that the definitions above easily generalize to the case when the implemented function is not necessarily a permutation.

2.2 Verifiable Random Functions

Next, we review verifiable random functions (VRF). Such functions 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 Π that returns a proof π which shows that y is the output of the function on input x. Secondly, the output of the function is unique, i.e., there cannot exist two images (and proofs) that verify under the same preimage. More formally we have:

Definition 2 (Verifiable Random Functions). *A family of functions $\mathcal{F} = \{f_s : \{0,1\}^{n(\lambda)} \rightarrow \{0,1\}^{m(\lambda)}\}_{s \in \{0,1\}^{seed(\lambda)}}$ is a family of* Verifiable Random Functions *if there exists a tuple of algorithms (KG, F, Π, V) with the following functionalities:*

$KG(1^\lambda)$ is a probabilistic algorithm that on input the security parameter λ, 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 π related to x.
$V(PK, x, y, \pi)$ is a (possibly) probabilistic algorithm that outputs 1 if π is a valid proof for the statement "$f_s(x) = y$". Otherwise it outputs 0.

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

Domain Range Correctness *For all $x \in \{0,1\}^{n(\lambda)}$ we have that $F(SK, x) \in \{0,1\}^{m(k)}$ holds with all but negligible probability (over the choices of (PK, SK)).*
Completeness *For all $x \in \{0,1\}^{n(k)}$ 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 (unless with negligible probability over the coin tosses of V) $(PK, x, y_1, y_2, \pi_1, \pi_2)$ such that $y_1 \neq y_2$ and $V(PK, x, y_1, \pi_1) = V(PK, x, y_2, \pi_2) = 1$.*
Pseudorandomness *For all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we require that the probability $\mathcal{A}$ succeeds in the experiment $\mathsf{pseudo}_{\mathcal{A}}^{f}$ is at most $\frac{1}{2} + \mathsf{negl}(\lambda)$, where the experiment is defined in Figure 2.*

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

Definition 3 (Verifiable Unpredictable Functions). *A tuple (KG, F, Π, V) is a verifiable unpredictable function if the probability that any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ succeeds in the experiment $\mathsf{predict}_{\mathcal{A}}^{f}$, defined in Figure 2, is at most negligible.*

Experiment $\mathsf{pseudo}^f_{\mathcal{A}}$
$(PK, SK) \leftarrow KG(1^\lambda)$;
$(x^*, \mathsf{state}) \leftarrow \mathcal{A}_1^{Func(SK,\cdot)}(PK)$
$b \xleftarrow{\$} \{0,1\}$;
$y_0 \leftarrow F(SK, x)$; $y_1 \xleftarrow{\$} \{0,1\}^{m(k)}$
$b' \leftarrow \mathcal{A}_2^{Func(SK,\cdot)}(\mathsf{state}, y_b)$
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. 2. 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.

2.3 Black-Box Reductions

We briefly recall the formal definition of fully black-box reductions.

Definition 4 ([33]). *There exists a fully black-box (fully-BB) reduction from a primitive P to a primitive Q if there exist PPT oracle machines G and S such that:*

- ***Correctness.*** *For every implementation f of the primitive Q, G^f is a correct implementation of P.*
- ***Security.*** *For every implementation f of Q and every machine $\mathcal{A}$, if $\mathcal{A}$ breaks G^f in the sense of P, then $S^{\mathcal{A},f}$ breaks f in the sense of Q.*

3 The Black-Box Separation

In this section we introduce our impossibility result. More precisely we first describe the main ideas of our proof and then we go into the details and start to formalize our separation theorem.

In order to prove our result we follow the usual oracle separation methodology of showing an oracle relative to which ATDPs exist while VUFs do not. To be precise we show our result in a slightly weaker model where there is not a "universal" separating oracle. Instead, for every candidate reduction of VUF to ATDP we show the existence of an oracle $\mathcal{B}$ such that the given construction is insecure while a secure ATDP still exists relative to $\mathcal{B}$. Such impossibility result does not rule out a relativizing reduction, yet it is enough to show that there cannot be a black-box reduction of VUF to ATDP.

Therefore the core of our separation are two oracles, $\mathcal{O}$ and $\mathcal{B}$. The oracle $\mathcal{O} = (g, e, d)$ realizes a random trapdoor permutation (we give a formal definition together with the full proof in Section 3.2). The second oracle is a weakening oracle. Given a candidate (and correct) VUF construction $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ we show an oracle $\mathcal{B}$ such that relative to $\langle \mathcal{O}, \mathcal{B} \rangle$ a secure construction of adaptive trapdoor permutations exists while the given VUF is insecure[6]. 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 $\mathcal{B}$.

A Secure VUF Relative to $\mathcal{O}$ Exists. The first interesting question is why cannot we use the usual methodology of showing a single separating oracle, namely: Is it possible to show an oracle $\mathcal{B}$ such that *any* VUF is insecure relative to $\langle \mathcal{O}, \mathcal{B} \rangle$? The answer is no and the reason is that a secure VUF in fact exists in the presence of a random trapdoor permutation oracle (we assume that $\Pi(SK, \cdot) = F(SK, \cdot)$):

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

$\underline{KG^{\mathcal{O}}(1^\lambda)}$
$ek_0 \leftarrow g(td_0)$
$ek_1 \leftarrow g(td_1)$
$SK \leftarrow td_0,\ PK \leftarrow (ek_0, ek_1)$
return (SK, PK)

$\underline{F^{\mathcal{O}}(SK, x)}$
$y \leftarrow d(td_0, e(ek_1, x))$
return y

$\underline{V^{\mathcal{O}}(PK, x, y)}$
outputs 1 iff
$e(ek_0, y) = e(ek_1, x)$

A careful reader can easily notice that the above construction is the well-known full-domain hash signature scheme which is known to be secure in the random oracle model.

3.1 Towards the definition of $\mathcal{B}$

As one can notice, the presence of the above construction implies that proving a relativizing separation does not seem to be possible. Hence, we adopt the idea of defining a "weakening" oracle $\mathcal{B}$ for each candidate VUF construction such that $\mathcal{B}$ allows us to break the security of the VUF yet it preserves the security of ATDPs (relative to $\mathcal{O}$). The difficulty thereby 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 $\Pi^{\mathcal{O}}(SK, \cdot) = F^{\mathcal{O}}(SK, \cdot)$):

$\underline{KG^{\mathcal{O}}(1^\lambda)}$
$td \in \{0,1\}^\lambda$
$ek \leftarrow g(td)$
return $(SK, PK) = (td, ek)$

$\underline{F^{\mathcal{O}}(SK, x)}$
$y \leftarrow d(td, x)$
return y

$\underline{V^{\mathcal{O}}(PK, x, y)}$
return 1 iff $e(ek, y) = x$

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 try to extract useful information to invert the TDP. 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 her 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 (someway) 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 she queries the function oracles on a sufficiently large number of the x's, then she 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 us to require any algorithm querying $\mathcal{B}$ to provide in input sufficiently many triples (x_i, y_i, π_i) such that π_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 itself 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 the inputs for $\mathcal{B}_2$. $\mathcal{B}_2$, on the other hand, 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 $\mathcal{B}_1$ and $\mathcal{B}_2$ communicate via a state information (the details are given in the following section).

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 ones where the attacker 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.[7] 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. $PK \leftarrow KG^{\mathcal{O}''}(SK')$).

[7] 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 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 needed while running $F^{\mathcal{O}''}$. More precisely it needs to know all 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, π_i) and collect all the queries made by the algorithm. Since by Assumption 1 KG can generate 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). As discussed in the previous section, a secure VUF relative to $\mathcal{O}$ exists as well. Therefore, we follow the strategy of defining a "weakening" oracle $\mathcal{B}$ whose main task is to break the security of the 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, for every VUF construction $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ there exist an oracle $\mathcal{B}$ such that:*

(i) there exists an adversary $\mathcal{A}$ such that $\mathcal{A}^{\mathcal{O},\mathcal{B}}$ 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 $\mathcal{A}^{\mathcal{O},\mathcal{B}}$ 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 $e(g(td), a) = b$.

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 α which depends on the type of α. 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 [36]. 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, Π, 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 Π 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 Π is the corresponding π, respectively. Finally, V is given in input the public key PK, an input x, an output y and a proof π and outputs 1 if it accepts the proof, or 0 otherwise. In the above description n is a function of λ.

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 2.

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

- *each algorithm 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.*

Now, we will justify the assumptions. The first condition is reasonable because the running time of the attacker is polynomially bounded 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., [3]. The assumption 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. The basic idea is to use a PRF (applied 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 has already been addressed in [4]. Therefore, we omit it here.

A formal definition of $\mathcal{B}$. In this section 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 VUF public key PK
OUTPUT: $x_1, \ldots, x_\ell \in \{0,1\}^n$ and a binary string state.
COMPUTATION: Algorithm $\mathcal{B}_2$ selects $x_1, \ldots, x_\ell$ uniformly at random in $\{0,1\}^n$ and then uses the oracle $\mathcal{O}$ to compute state as a (deterministic) message authentication code (MAC) on $(PK, x_1, \ldots, x_\ell)$[8].

Algorithm $\mathcal{B}_2$:
INPUT: A state information state, 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 π_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:** If state is not consistent (w.r.t. MAC) with the given $(PK, x_1, \ldots, x_\ell)$, then output $\perp$.
– **Step 2:** For all $i = 1$ to ℓ run the verification 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 ℓ. 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 α such that $\mathcal{O}''(\alpha) \neq \mathcal{O}(\alpha)$, then return $\perp$. Otherwise output y^*.

[8] We assume that the two oracles share a secret information (e.g. a function from $\mathcal{O}$) that is hardcoded into them. Then it is easy to see that a perfectly random $\mathcal{O}$ can be used to compute a MAC.

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 $\mathcal{O}$. $\mathcal{B}_1$ calls $\mathcal{O}$ only to compute the MACs. Assume w.l.o.g. that computing and verifying the MAC requires q' oracle queries where $q' = poly(\lambda)$. Thus a query to $\mathcal{B}_1$ counts as q' oracle queries. Instead a query to $\mathcal{B}_2$ counts $q' + \ell q + 3q + |\mathcal{O}'|$ queries to $\mathcal{O}$ in total. This cost is obtained as follows: Step 1 needs q' queries for verifying the MAC, Step 2 makes ℓq queries as it evaluates V ℓ 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 VUF 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 PK and obtain state and $x_1, \ldots, x_\ell$.
2. Query the VUF oracles $F(SK, \cdot), \Pi(SK, \cdot)$ on x_i for all $i = 1$ to ℓ. Let $\{y_1, \pi_1, \ldots, y_\ell, \pi_\ell\}$ be the values obtained from such queries.
3. Query $\mathcal{B}_2$ on input state, PK and $\{x_1, y_1, \pi_1, \ldots, x_\ell, y_\ell, \pi_\ell\}$.
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 2. *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 $2q' + 2q^{c+1} + 4q$ oracle queries.*

As one can see our adversary $\mathcal{A}$ is very simple as it mainly relies on the oracle $\mathcal{B}$. In particular, it is easy to see that $\mathcal{A}$ always succeeds when $\mathcal{B}$ outputs a "good" pair (x^*, y^*) such that $V^{\mathcal{O}}(PK, x^*, y^*, \Pi^{\mathcal{O}}(SK, x^*))$ accepts. Therefore we need to show that the following facts happen with non-negligible probability: (i) $\mathcal{B}$ does not fail (i.e., it does not output $\perp$) and (ii) $\mathcal{B}$ correctly evaluates $F^{\mathcal{O}''}(SK', x^*) = F^{\mathcal{O}}(SK', x^*)$.

First of all we count the number of oracle queries made by $\mathcal{A}$. Observe that $\mathcal{A}$ makes one call to $\mathcal{B}_1$ and one call to $\mathcal{B}_2$. Recall that queries to $\mathcal{B}_1$ and $\mathcal{B}_2$ count q' and $q' + \ell q + 3q + |\mathcal{O}'|$ queries to $\mathcal{O}$ respectively. Since we set $\ell = q^c$, $|\mathcal{O}_Q| \leq q^{c+1}$, and because $|\mathcal{O}'| \leq |\mathcal{O}_Q| + q$ we conclude that $\mathcal{A}$ makes at most $2q' + 2q^{c+1} + 4q$ oracle queries.

Now we bound the probability that the oracle $\mathcal{B}_2$ outputs $\perp$ in Step 6, which happens when the verification equation $V^{\mathcal{O}''}(PK, x^*, F^{\mathcal{O}''}(SK, x^*))$ asks a query α such that $\mathcal{O}''(\alpha) \neq \mathcal{O}(\alpha)$. More

formally, let us call $\mathcal{E}_\alpha$ such event for a specific query α. For a fixed α we can distinguish between two cases in which α involves a public key ek that is in $Z(\mathcal{O}_Q)$ or not. Namely,

$$\begin{aligned}\Pr[\mathcal{E}_\alpha] &= \Pr[\mathcal{E}_\alpha | ek \notin Z(\mathcal{O}_Q)] \Pr[ek \notin Z(\mathcal{O}_Q)] + \Pr[\mathcal{E}_\alpha | ek \in Z(\mathcal{O}_Q)] \Pr[ek \in Z(\mathcal{O}_Q)] \\ &\leq \Pr[ek \notin Z(\mathcal{O}_Q) \wedge \mathcal{E}_\alpha] + \Pr[\mathcal{E}_\alpha | ek \in Z(\mathcal{O}_Q)]\end{aligned} \tag{1}$$

For each public key ek that appears in a run of $V^{\mathcal{O}''}(PK, x^*, F^{\mathcal{O}''}(SK, x^*))$ we can have two different cases: either it was generated by the KG algorithm or not. If we call G_{ek} the event that ek was generated by KG, then we have

$$\begin{aligned}\Pr[ek \notin Z(\mathcal{O}_Q) \wedge \mathcal{E}_\alpha] = &\Pr[ek \notin Z(\mathcal{O}_Q) | \mathcal{E}_\alpha \wedge G_{ek}] \Pr[\mathcal{E}_\alpha \wedge G_{ek}] + \\ &\Pr[ek \notin Z(\mathcal{O}_Q) | \mathcal{E}_\alpha \wedge \neg G_{ek}] \Pr[\mathcal{E}_\alpha \wedge \neg G_{ek}]\end{aligned} \tag{2}$$

By carefully looking at the definition of $\mathcal{B}_2$, it is not hard to notice that if ek was *not* generated by KG, then for any query α involving ek it must be $\mathcal{O}''(\alpha) = \mathcal{O}(\alpha)$, namely $\Pr[\mathcal{E}_\alpha \wedge \neg G_{ek}] = 0$. Hence, putting together the latter observation with equations (1) and (2), we obtain:

$$\Pr[\mathcal{E}_\alpha] \leq \Pr[ek \notin Z(\mathcal{O}_Q) | \mathcal{E}_\alpha \wedge G_{ek}] + \Pr[\mathcal{E}_\alpha | ek \in Z(\mathcal{O}_Q)].$$

Claim 1 $\Pr[ek \notin Z(\mathcal{O}_Q) | \mathcal{E}_\alpha \wedge G_{ek}] \leq \frac{q}{eq^c}$.

Proof. This event in fact occurs when the public key ek involved in the query α was not collected in the Step 2 of the $\mathcal{B}_2$ oracle algorithm. Formally α can be of the form $[g(\cdot) = ek]$ or $[e(ek, \cdot) = \cdot]$. We show that such queries only occur with probability at most $\frac{q}{eq^c}$.

Fix a public key ek among those ones that are generated during $KG^{\mathcal{O}}(SK)$ run by the real Challenger. By our Assumption 1 we know that $KG^{\mathcal{O}}$ makes at most q queries to $\mathcal{O}$, thus there are at most q of such ek's. Since in Step 2 we run the verification algorithm on $\ell = q^c$ randomly independent inputs we can apply the result of Lemma 1 to bound the probability that ek is not collected in Step 2 (and appears in α), which is at most $\frac{1}{eq^c}$. Then, applying the union bound over all the possible public keys ek generated by $KG^{\mathcal{O}}(SK)$ we obtain that $\Pr[ek \notin Z(\mathcal{O}_Q) | \mathcal{E}_\alpha \wedge G_{ek}] \leq \frac{q}{eq^c}$. □

Claim 2 $\Pr[\mathcal{E}_\alpha | ek \in Z(\mathcal{O}_Q)] \leq \frac{2q}{eq^c}$.

Proof. Observe that a query α involving a "known" public key ek may have different answers under $\mathcal{O}$ and $\mathcal{O}''$ due to the definition of the composition operator $\diamond_c$ and the fact that $\mathcal{O}'$ is defined on some points that are not in $\mathcal{O}_Q$.

In particular, if α is of the form $[g(td)]$ such that $[g(td) = ek] \in \mathcal{O}''$ and $[g(td) = ek'] \in \mathcal{O}$ with $ek \neq ek'$, then it must be $[g(td) = ek] \in \mathcal{O}' \setminus \mathcal{O}_Q$.

On the other hand, if $\alpha = [e(ek, a)]$ and, $[e(ek, a) = b] \in \mathcal{O}$ and $[e(ek, a) = b'] \in \mathcal{O}''$ with $b \neq b'$, then we distinguish between two different cases:

- $[e(ek, a) = b'] \in \mathcal{O}' \setminus \mathcal{O}_Q$. It means that this is one of the additional points chosen by $\mathcal{B}_2$ into Step 3.
- There exists a' such that $[e(ek, a') = b'] \in \mathcal{O}$ and $[e(ek, a') = b] \in \mathcal{O}'$. This is a collision created by the $\diamond_c$ operator to preserve the permutation property of $e(ek, \cdot)$.

This means that for queries α involving an $ek \in Z(\mathcal{O}_Q)$ we have that $\mathcal{O}$ and $\mathcal{O}''$ differ in at most 2 points. Thus, applying Lemma 1 and the union bound over all such α we obtain that $\Pr[\mathcal{E}_\alpha | ek \in Z(\mathcal{O}_Q)] \leq \frac{2q}{eq^c}$. □

Proof (Proof of Lemma 2). To complete the proof, let $\mathcal{F}$ be the event that $\mathcal{B}_2$ fails and outputs $\perp$. Since our adversary queries $(\mathcal{B}_1, \mathcal{B}_2)$ with the correct input, the only place where $\mathcal{B}_2$ might fail is Step 6, thus by the above claims we have $\Pr[\mathcal{F}] = \Pr[\mathcal{E}_\alpha] \leq \frac{3q}{eq^c}$.

Now we show that when $\mathcal{F}$ does not occur, then $\mathcal{B}_2$ successfully returns a pair (x^*, y^*) that is accepted by the verification algorithm (run with the real oracle) and thus allows $\mathcal{A}$ to break the unpredictability of the VUF.

First of all, observe that if the oracle $\mathcal{O}''$ is a correct trapdoor permutation oracle, then the VUF defined through the algorithms $(KG^{\mathcal{O}''}, F^{\mathcal{O}''}, \Pi^{\mathcal{O}''}, V^{\mathcal{O}''})$ is complete and thus the verification algorithm $V^{\mathcal{O}''}(PK, x^*, F^{\mathcal{O}''}(SK', x^*), \Pi^{\mathcal{O}''}(SK'', x^*))$ outputs 1.

Next, assume that the event $\mathcal{E}_\alpha$ does not occur. Then it easy to see that the verification algorithm $V^{\mathcal{O}''}(PK, x^*, F^{\mathcal{O}''}(SK, x^*), \Pi^{\mathcal{O}''}(SK, x^*))$ does not need any queries α such that $\mathcal{O}(\alpha) \neq \mathcal{O}''(\alpha)$. This means, however, that running this algorithm with access to $\mathcal{O}$ would produce the same output:

$$V^{\mathcal{O}''}(PK, x^*, F^{\mathcal{O}''}(SK', x^*), \Pi^{\mathcal{O}''}(SK', x^*)) = V^{\mathcal{O}}(PK, x^*, F^{\mathcal{O}''}(SK', x^*), \Pi^{\mathcal{O}''}(SK', x^*))$$

Thus, $V^{\mathcal{O}}(PK, x^*, F^{\mathcal{O}''}(SK', x^*), \Pi^{\mathcal{O}''}(SK', x^*))$ accepts as well. Then, recall that by our assumption $(KG^{\mathcal{O}}, F^{\mathcal{O}}, \Pi^{\mathcal{O}}, V^{\mathcal{O}})$ is a VUF implementation that is complete and unique. In particular, by the uniqueness the verification algorithm $V^{\mathcal{O}}$ does not accepts two different values for the same x^* and thus it follows that $F^{\mathcal{O}''}(SK', x^*) = F^{\mathcal{O}}(SK, x^*)$. Therefore, the probability that $\mathcal{A}$ outputs (x^*, y^*) and wins the unpredictability game is at least

$$1 - \Pr[\mathcal{F}] \geq 1 - \frac{3}{eq^{c-1}}$$

□

Finally observe that our adversary $\mathcal{A}$ runs in polynomial time, thus it does not need *any* **PSPACE** oracle to be made efficient.

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 adaptive 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 only $\mathcal{O}$. 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 an adversary $\mathcal{A}$ can simulate $\mathcal{B}$ by herself.

In the following proof we assume that $\mathcal{A}$ makes at most $\hat{q}$ oracle queries, where $\hat{q}$ includes the cost of each query to $\mathcal{B}$. Let $n = poly(\lambda)$ be the usual VUF parameter which is used in the definition of $\mathcal{B}$ (even the simulated one), and $\hat{q}, p$ and ρ are three additional parameters (that we will specify later) that are all polynomials in λ. Now we can state the following lemma:

Lemma 3. *Let $\rho = poly(\lambda)$ and 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, every adversary $\mathcal{A}$ that has access to $\langle \mathcal{O}, \mathcal{B} \rangle$ and makes at most $\hat{q}$ oracle queries has probability at most $(\frac{2\hat{q}}{2^\lambda} + \frac{3\hat{q}}{2^\lambda - \hat{q}} + \frac{p}{2^n})(1 - \frac{\rho}{2^\lambda})$ of succeeding in the adaptive one-wayness experiment against the above construction.*

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 that similar to the one used in [36]. 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 $\mathcal{S}$ has 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 occurs is very small.

Formal description of the simulator. We define the simulator $\mathcal{S}$ as follows:

Input: A public key ek^* and a value $b^* \in \{0,1\}^\lambda$. $\mathcal{S}$ has also access to the inversion oracle $I(ek^*, \cdot)$ that answers queries for $b \neq b^*$.
Output: $a^* \in \{0,1\}^\lambda$.
Algorithm: the simulator performs the following steps:

1. $\mathcal{S}$ generates a random trapdoor permutation $\mathcal{O}_{\mathcal{S}}$ which is defined on any queries except those of the form $e(ek^*, \cdot)$.
2. Run the algorithm $\mathcal{A}^{I, \langle \mathcal{O}, \mathcal{B} \rangle}(ek^*, b^*)$ and simulate $\mathcal{A}$'s oracle queries as follows:
   $\mathcal{O}$ **and** I **queries:** Letting α denote the queries $\mathcal{A}$, then we distinguish between two cases:
   - If $\alpha = [I(ek^*, \cdot)]$ or $\alpha = [e(ek^*, \cdot)]$, then forward α to the oracles I and $\mathcal{O}$ respectively and output the received answer.

- Else, answer with $\mathcal{O}_{\mathcal{S}}(\alpha)$. Collect into a partial oracle $\mathcal{O}^*$ all the mappings of the form $[e(ek^*, a) = b]$ that are discovered during the simulation through $\mathcal{A}$'s queries to $e(ek^*, \cdot)$ or $I(ek^*, \cdot)$.

$\mathcal{B}_1$ **queries:** The $\mathcal{B}_1$ oracle can be easily simulated as the real one.

$\mathcal{B}_2$ **queries:** On input state, PK and $\{(x_i, y_i, \pi_i)\}_{i=1}^{\ell}$ proceed as follows:

- Step 1 and Step 2 are simulated as the real ones.
- Step 3 is simulated offline *without making any oracle queries.* To achieve, this, the simulator finds an $\mathcal{O}'$ which is consistent with the currently known $\mathcal{O}^*$.
- The simulation of Step 4 differs slightly. Indeed $\mathcal{S}$ defines $\mathcal{O}'' = (\mathcal{O} \cup I) \diamond_c \tilde{\mathcal{O}}$ where $\tilde{\mathcal{O}} = (\mathcal{O}_{\mathcal{S}} \cup \mathcal{O}^*) \diamond_c \mathcal{O}'$. More precisely, if $\tilde{\mathcal{O}}(\alpha)$ is defined then $\mathcal{O}''(\alpha)$ outputs $\tilde{\mathcal{O}}(\alpha)$, otherwise it works as follows. If $\alpha = [d(td', b)]$ such that $[g(td') = ek^*]$ then we can safely set $\mathcal{O}''(\alpha) = I(ek^*, b)$. This is done for all $b \neq b^*$. In the special case where $\alpha = [d(td', b^*)]$, then the simulator answers $\mathcal{O}''(\alpha)$ with a random value. Answer all other queries with $\mathcal{O}$.
- Step 5: Choose a random x^* and run $y^* \leftarrow F^{\mathcal{O}''}(SK', x^*)$ and $\pi^* \leftarrow \Pi^{\mathcal{O}''}(SK', x^*)$.
- Step 6: Run $V^{\mathcal{O}''}(PK, x^*, y^*, \pi^*)$ and check that all queries made by $V^{\mathcal{O}''}$ did not use a new public key (i.e., one that was not discovered in Step 2) and that all queries are consistent with the oracle simulated to $\mathcal{A}$.

3. If during the simulation a query $[g(td)]$ occurs such that $\mathcal{S}$ answered it with ek^* and $[g(td) = ek^*] \in \mathcal{O}$, then compute $a^* \leftarrow d(td, b^*)$, output a^*, and stop.
4. If during the simulation a query $[e(ek^*, a)]$ has been asked such that $[e(ek^*, a) = b^*] \in \mathcal{O}$, then output a and stop.

5.2 Analyzing the Simulator

We now analyze the success probability of our simulator. To do so we first formalize the setting and we define the relevant events that may occur during our simulation.

Consider the input of the simulator (ek^*, b^*) and the public key PK provided by $\mathcal{A}$ in a query to $\mathcal{B}$. For every $x \in \{0,1\}^n$ let hit_x be the event that a query $[d(td^*, b^*)]$ (with $g(td^*) = ek^*$) occurs during the execution of either $F^{\mathcal{O}}(SK, x)$ or $\Pi^{\mathcal{O}}(SK, x)$. In this case, observe that a query $[e(ek^*, \cdot) = b^*]$ must appear while running $V^{\mathcal{O}}(PK, x, F^{\mathcal{O}}(SK, x), \Pi^{\mathcal{O}}(SK, x))$. Otherwise, if such a query would not appear in the verification, then it would not be "important"[9] for the verification algorithm (and thus we might change its answer without changing the output of the algorithm).

Without loss of generality, we can distinguish between two types of such public keys PK:

type-1. A public key is of **type-1** if it induces $F^{\mathcal{O}}$ to call $d(td^*, b^*)$ on a *large* fraction of the inputs, meaning that for a random $x \in \{0,1\}^n$ we have $\Pr[\mathsf{hit}_x] \geq \frac{2^n - p}{2^n}$.

type-2. A public key is of **type-2** if it causes $F^{\mathcal{O}}$ to call $d(td^*, b^*)$ only on a *small* fraction of the inputs, meaning that for a random $x \in \{0,1\}^n$ we have $\Pr[\mathsf{hit}_x] \leq \frac{p}{2^n}$.

In both cases we set $p = poly(n)$ and we define some "good" and "bad" events that may occur during our simulation:

- Good_1 is the event that a query $[g(td)]$ occurs during the simulation and $[g(td) = ek^*] \in \mathcal{O}$.
- Good_2 is the event that a query $[e(ek^*, a) = b^*]$ appears during the simulation. Note that this may happen for two reasons: either $[e(ek^*, a)]$ is asked by $\mathcal{A}$ and $\mathcal{S}$ answers with b^* or $[e(ek^*, a) = b^*]$ is found during the collection stage in Step 2.

[9] By "important" we mean that it affects the output of the algorithm.

- Bad_1 is the event that $\mathcal{A}$ can pass Step 1 by giving to $\mathcal{B}_2$ a tuple $(PK, x_1, \ldots, x_\ell)$ that is different from the one that appeared during a query to $\mathcal{B}_1$.
- Bad_2 is the event that $[g(td)]$ occurs during the simulation and $\mathcal{S}$ answers with ek^* in $\mathcal{O}_{\mathcal{S}}$.
- Bad_3 is the event that $\mathcal{O}''$ contains a mapping $[e(ek^*, a') = b]$ while $[e(ek^*, a) = b] \in \mathcal{O}$ where $a \neq a'$.
- Bad^* is the event that in Step 5 $\alpha = [d(td', b^*)]$ is answered with a random value.

Except for Bad_1 we notice that these events represent all the cases where our simulator and the real $\mathcal{B}$ oracle may differ. The next step is to bound the probability that each of these events occurs during the simulation. First observe that $\Pr[\mathsf{Bad}_1]$ is equivalent to the probability of forging a MAC for a scheme built upon a perfectly random one-way permutation. Without loss of generality we assume that such probability is negligible in λ: $\Pr[\mathsf{Bad}_1] \leq \frac{\rho}{2^\lambda}$ for some $\rho = poly(\lambda)$.

Claim 3 *The probability of events* Good_1 *and* Bad_2 *is at most* $\frac{\hat{q}}{2^\lambda}$.

Proof. Due to the randomness of the oracle $\mathcal{O}$, observe that for every query $[g(td)]$ the probability that $[g(td) = ek^*] \in \mathcal{O}$ is at most $\frac{1}{2^\lambda}$. Similarly for Bad_2, the probability that $\mathcal{S}$ chooses ek^* as the response to some query $[g(td)]$ is again $\leq \frac{1}{2^\lambda}$. Since $\mathcal{A}$ is required to ask at most $\hat{q}$ such queries, we can apply the union bound over all of them and in conclusion we obtain that $\Pr[\mathsf{Good}_1] \leq \frac{\hat{q}}{2^\lambda}$ and $\Pr[\mathsf{Bad}_2] \leq \frac{\hat{q}}{2^\lambda}$. □

Claim 4 *The probability of the event* Bad^* *is at most* $\frac{p}{2^n}$.

Proof. Recall that we distinguish between two types of keys denoted by **type-1** and **type-2**, respectively. Let Γ_i be the event that the submitted public key PK is of type i. Then we can write

$$\Pr[\mathsf{Bad}^*] = \Pr[\mathsf{Bad}^*|\Gamma_1]\Pr[\Gamma_1] + \Pr[\mathsf{Bad}^*|\Gamma_2]\Pr[\Gamma_2]$$

and consider $\Pr[\mathsf{Bad}^*|\Gamma_1]$ and $\Pr[\mathsf{Bad}^*|\Gamma_2]$ separately.

If the given public key is of **type-2**, then by our previous observation, we know that the probability that a query $[d(td', b^*)]$ has been asked in Step 5 is equal to $\Pr[\mathsf{hit}_{x^*}|\Gamma_2]$ which is $\leq \frac{p}{2^n}$. Thus, it follows that $\Pr[\mathsf{Bad}^*|\Gamma_2] \leq \frac{p}{2^n}$.

On the other hand, if the adversary submits a public key of **type-1**, then the probability that a query $d(td', b^*)$ appears in Step 5 is $\geq \frac{2^n - p}{2^n}$. However, observe that by our definition of $\mathcal{O}''$, it answered with a random value only if a query $[e(ek^*, a) = b^*]$ has never appeared in the previous steps. We can show that the probability that in this case a query $[e(ek^*, a) = b^*]$ did not appear before Step 5 is very small. For instance, if Γ_1 occurs, then we know that for any of the x_i's in Step 2 the probability that $[e(ek^*, a) = b^*]$ does not appear while evaluating $V^{\mathcal{O}}(PK, x_i, F^{\mathcal{O}}(SK, x_i), \Pi^{\mathcal{O}}(SK, x_i))$ (for $i = 1, \ldots, \ell$) is at most $1 - \Pr[\mathsf{hit}_{x_i}|\Gamma_1] \leq \frac{p}{2^n}$. Therefore the probability that such a query does not appear at all in Step 2 is $\leq (\frac{p}{2^n})^\ell \leq \frac{p}{2^n}$. Putting together the two cases, in conclusion we obtain $\Pr[\mathsf{Bad}_*] \leq \frac{p}{2^n}$. □

Claim 5 *The probability of events* Good_2 *and* Bad_3 *is at most* $\frac{\hat{q}}{2^\lambda - \hat{q}}$.

Proof. If the event Good_2 occurs, then $\mathcal{S}$ is able to compute the inverse of b^* for a random trapdoor permutation. Since $\mathcal{S}$ makes at most $\hat{q}$ queries, it follows that $\Pr[\mathsf{Good}_2] \leq \frac{\hat{q}}{2^\lambda - \hat{q}}$.

The event Bad_3 covers the case in which we discover (during the simulation) a query α of the form $[e(ek^*, \cdot) = b]$ such that $\mathcal{O}(\alpha) \neq \mathcal{O}''(\alpha)$. More precisely, we assume that $[e(ek^*, a') = b] \in \mathcal{O}''$ and $[e(ek^*, a) = b] \in \mathcal{O}$ for two distinct values a, a'.

If we consider $[e(ek^*, a') = b]$ we have two possibilities: either it is in $\mathcal{O}'$ or it is not. If $[e(ek^*, a') = b] \notin \mathcal{O}'$ then, by the definition of $\mathcal{O}''$ (and the fact that $\mathcal{O}_{\mathcal{S}}$ is not defined on $e(ek^*, \cdot)$) it must hold $[e(ek^*, a') = b] \in \mathcal{O}^*$, which means that there is no collision in this case. Otherwise, consider the case when $[e(ek^*, a') = b] \in \mathcal{O}'$ and it is also in $\mathcal{O}^*$. This case can never happen because $\mathcal{O}'$ is defined consistently with $\mathcal{O}^*$. So, assume that $[e(ek^*, a') = b] \notin \mathcal{O}^*$. If a query $\alpha = [e(ek^*, a) = b]$ appears during the simulation it means that $\mathcal{S}$ finds the inverse of b in a random permutation in the case when he already knows $|\mathcal{O}^*| \leq \hat{q}$ points and he makes at most $\hat{q}$ queries to the permutation. However we know that the probability of this event is at most $\frac{\hat{q}}{2^\lambda - \hat{q}}$. Therefore it holds $\Pr[\mathsf{Bad}_3] \leq \frac{\hat{q}}{2^\lambda - \hat{q}}$. $\square$

Once we have bound the probabilities of all our events, we can show that $\mathcal{S}$ has negligible probability of winning the one-wayness game.

First of all consider the case of an adversary that wants to fool the oracle $\mathcal{B}$ by providing a fake input. By our bound on Bad_1 we know that such probability is negligible. Formally we have:

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins}|\mathsf{Bad}_1]\Pr[\mathsf{Bad}_1] + \Pr[\mathcal{A} \text{ wins}|\neg\mathsf{Bad}_1]\Pr[\neg\mathsf{Bad}_1] \\ &\leq \Pr[\mathcal{A} \text{ wins}|\neg\mathsf{Bad}_1](1 - \Pr[\mathsf{Bad}_1])\end{aligned}$$

It is easy to see that if none of the events $\mathsf{Good}_1, \mathsf{Good}_2, \mathsf{Bad}_2, \mathsf{Bad}_3, \mathsf{Bad}^*$ occurs, then $\mathcal{S}$ simulates $\mathcal{A}$ perfectly. Moreover, all these events are disjoint as each of them induces our algorithm to halt the simulation. Therefore we have:

$$\begin{aligned}\Pr[\mathcal{S} \text{ wins}|\neg\mathsf{Bad}_1] &\geq \Pr[\mathcal{A} \text{ wins}|\neg\mathsf{Bad}_1] - \Pr[\mathsf{Good}_1] - \Pr[\mathsf{Good}_2] - \Pr[\mathsf{Bad}_2] - \Pr[\mathsf{Bad}_3] - \Pr[\mathsf{Bad}^*] \\ &\geq \Pr[\mathcal{A} \text{ wins}|\neg\mathsf{Bad}_1] - \tfrac{2\hat{q}}{2^\lambda} - \tfrac{2\hat{q}}{2^\lambda - \hat{q}} - \tfrac{p}{2^n}.\end{aligned}$$

The probability that $\mathcal{S}$ wins the one-wayness game without access to $\mathcal{B}$ is at most $\frac{\hat{q}}{2^\lambda - \hat{q}}$. Thus, in conclusion we obtain:

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] &\leq \Pr[\mathcal{A} \text{ wins}|\neg\mathsf{Bad}_1](1 - \Pr[\mathsf{Bad}_1]) \\ &\leq (\Pr[\mathcal{S} \text{ wins}|\neg\mathsf{Bad}_1] + \tfrac{2\hat{q}}{2^\lambda} + \tfrac{2\hat{q}}{2^\lambda - \hat{q}} + \tfrac{p}{2^n})(1 - \tfrac{\rho}{2^\lambda}) \\ &\leq (\tfrac{2\hat{q}}{2^\lambda} + \tfrac{3\hat{q}}{2^\lambda - \hat{q}} + \tfrac{p}{2^n})(1 - \tfrac{\rho}{2^\lambda})\end{aligned}$$

that completes the proof of Lemma 3. $\square$

Note that so far we have measured the complexity of our algorithm only in terms of oracle queries. In order to extend our black-box separation to all PPT adversaries, we can add a **PSPACE** oracle. Specifically, our simulator $\mathcal{S}$ can perform the Step 3 of $\mathcal{B}_2$ by making a query to the **PSPACE** oracle, which can be embedded using the techniques of Simon [35].

Acknowledgments

We would like to thank Yevgeniy Vahlis for helpful insights about black-box separation techniques and Michel Abdalla for helpful discussions on this work. 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. Dominique Schröder is supported by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG).

References

1. Michel Abdalla, Dario Catalano, and Dario Fiore. Verifiable random functions from identity-based key encapsulation. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 554–571, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany.
2. Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM Journal on Computing*, 36(5):915–942, 2006.
3. Dan Boneh, Periklis A. Papakonstantinou, Charles Rackoff, Yevgeniy Vahlis, and Brent Waters. On the impossibility of basing identity based encryption on trapdoor permutations. In *49th Annual Symposium on Foundations of Computer Science*, pages 283–292, Philadelphia, Pennsylvania, USA, October 25–28, 2008. IEEE Computer Society Press.
4. Zvika Brakerski, Shafi Goldwasser, Guy N. Rothblum, and Vinod Vaikuntanathan. Weak verifiable random functions. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 558–576. Springer, Berlin, Germany, March 15–17, 2009.
5. Zvika Brakerski and Yael Tauman Kalai. A framework for efficient signatures, ring signatures and identity based encryption in the standard model. Cryptology ePrint Archive, Report 2010/086, 2010. `http://eprint.iacr.org/`.
6. Emmanuel Bresson, Jean Monnerat, and Damien Vergnaud. Separation results on the "one-more" computational problems. In Tal Malkin, editor, *Topics in Cryptology – CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 71–87, San Francisco, CA, USA, April 7–11, 2008. Springer, Berlin, Germany.
7. Daniel Brown. Irreducibility to the one-more evaluation problems: More may be less, 2007. `http://eprint.iacr.org/`.
8. Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 1–17, Miami, USA, January 6–8, 2003. Springer, Berlin, Germany.
9. 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.
10. Yevgeniy Dodis and Prashant Puniya. Feistel networks made public, and applications. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 534–554, Barcelona, Spain, May 20–24, 2007. Springer, Berlin, Germany.
11. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431, Les Diablerets, Switzerland, January 23–26, 2005. Springer, Berlin, Germany.
12. Cynthia Dwork and Moni Naor. Zaps and their applications. *SIAM Journal on Computing*, 36(6):1513–1543, 2007.
13. Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In *Advances in Cryptology – EUROCRYPT 2010*, Lecture Notes in Computer Science, pages 197–215. Springer, Berlin, Germany, May 2010.
14. Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. The relationship between public key encryption and oblivious transfer. In *41st Annual Symposium on Foundations of Computer Science*, pages 325–335, Redondo Beach, California, USA, November 12–14, 2000. IEEE Computer Society Press.
15. Yael Gertner, Tal Malkin, and Steven Myers. Towards a separation of semantic and CCA security for public key encryption. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 434–455, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Berlin, Germany.
16. Yael Gertner, Tal Malkin, and Omer Reingold. On the impossibility of basing trapdoor functions on trapdoor predicates. In *42nd Annual Symposium on Foundations of Computer Science*, pages 126–135, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press.
17. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33:792–807, 1986.
18. Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, USA, May 15–17, 1989. ACM Press.
19. Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740

of *Lecture Notes in Computer Science*, pages 228–245, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Berlin, Germany.

20. Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, Lecture Notes in Computer Science, pages 654–670, Santa Barbara, CA, USA, August 2009. Springer, Berlin, Germany.
21. Susan Hohenberger and Brent Waters. Constructing verifiable random functions with large input spaces. In *Advances in Cryptology – EUROCRYPT 2010*, Lecture Notes in Computer Science, pages 656–672. Springer, Berlin, Germany, May 2010.
22. Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, Washington, USA, May 15–17, 1989. ACM Press.
23. Stanislaw Jarecki and Vitaly Shmatikov. Handcuffing big brother: an abuse-resilient transaction escrow scheme. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 590–608, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.
24. Eike Kiltz, Payman Mohassel, and Adam O'Neill. Adaptive trapdoor functions and chosen-ciphertext security. In *Advances in Cryptology – EUROCRYPT 2010*, Lecture Notes in Computer Science, pages 673–692. Springer, Berlin, Germany, May 2010.
25. Moses Liskov. Updatable zero-knowledge databases. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 174–198, Chennai, India, December 4–8, 2005. Springer, Berlin, Germany.
26. Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 597–612, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Berlin, Germany.
27. Ueli M. Maurer and Johan Sjödin. A fast and key-efficient reduction of chosen-ciphertext to known-plaintext security. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 498–516, Barcelona, Spain, May 20–24, 2007. Springer, Berlin, Germany.
28. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 120–130, New York, New York, USA, October 17–19, 1999. IEEE Computer Society Press.
29. Silvio Micali and Leonid Reyzin. Soundness in the public-key model. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 542–565, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.
30. Silvio Micali and Ronald L. Rivest. Transitive signature schemes. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 236–243, San Jose, CA, USA, February 18–22, 2002. Springer, Berlin, Germany.
31. Moni Naor and Omer Reingold. Synthesizer and their applications to the parallel construction of pseudo-random functions. *Journal of Computer and System Sciences*, 58(2), 1999.
32. Pascal Paillier and Jorge L. Villar. Trading one-wayness against chosen-ciphertext security in factoring-based encryption. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 252–266, Shanghai, China, December 3–7, 2006. Springer, Berlin, Germany.
33. Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, Cambridge, MA, USA, February 19–21, 2004. Springer, Berlin, Germany.
34. Alon Rosen and Gil Segev. Chosen-ciphertext security via correlated products. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 419–436. Springer, Berlin, Germany, March 15–17, 2009.
35. Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 334–345, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.
36. Yevgeniy Vahlis. Two is a crowd? a black-box separation of one-wayness and security under correlated inputs. In *TCC 2010: 7th Theory of Cryptography Conference*, Lecture Notes in Computer Science, pages 165–182. Springer, Berlin, Germany, 2010.

A Static Verifiable Unpredictable Functions

In this section we introduce a weaker notion that we call *static verifiable unpredictable functions* and we show that it can be constructed using ATDPs.

Definition 5 (Static Verifiable Unpredictable Functions). *A function f is* static unpredictable *if the probability that any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ succeeds the experiment $\mathsf{static}^f_{\mathcal{A}}$ is at most negligible, where*

Experiment $\mathsf{static}^f_{\mathcal{A}}$
$(x_1, \ldots, x_q, \mathsf{state}) \leftarrow \mathcal{A}_1(1^\lambda)$
$(PK, SK) \leftarrow KG(1^\lambda)$;
$y_i \leftarrow F(SK, x_i)$, $\pi_i \leftarrow \Pi(SK, x_i)$ *for* $i = 1, \ldots, q$
$(x^*, y^*) \leftarrow \mathcal{A}_2(PK, \mathsf{state}, (y_1, \pi_1), \ldots, (y_q, \pi_q))$
Output 1 iff $y^* = F(SK, x^*)$.

A construction from ATDPs satisfying such definition can be obtained in two steps.

First, consider the following construction: the key generation produces a key-pair of the ATDP $(ek, td) \leftarrow G(1^\lambda)$ and publishes ek as PK and stores td as SK. The evaluation algorithm $F(SK, x)$ returns $y \leftarrow D(SK, x)$ (the proof algorithm computes the same value). For verification, it simply checks whether $E(PK, y) = x$. It is easy to see that this construction can be proved secure according to a definition of unpredictability slightly weaker than the one given above. More precisely, the one where x^* is chosen (at random) by the challenger and given in input to $\mathcal{A}_2$.

Next, in order to obtain a static VUF we can apply the prefix technique as put forward by Hohenberger and Waters in [20] and later generalized by Brakerski and Tauman Kalai in [5].

B Intuitive argument on the difficulty of building VRFs from weak-VRFs

We give here an intuitive argument why any black-box construction of VRF from a weak-VRF *must* already be a VRF. To see this, recall that a weak-VRF is unique and the pseudorandomness only holds w.r.t. random inputs. The first observation is that any construction must describe a deterministic algorithm that takes as input a value x and outputs a unique and random value x' (which is the input of the wVRF). Moreover, such a transformation must also specify a proof algorithm that proves the relation between x and x'. We now argue that these algorithms already specify a VRF: Firstly, uniqueness follows immediately from the unique mapping between x and x'. If this mapping would not be unique, then the transformed VRF could not be unique. Secondly, the output of the construction must be random, otherwise it cannot be used as input to the wVRF. But if the ouput is already random, then this construction would immediately fulfill the pseudorandomness property. Thus, it seems that building a VRF out of any wVRF is as hard as constructing a VRF directly.