Two Is a Crowd? A Black-Box Separation of One-Wayness and Security under Correlated Inputs

Yevgeniy Vahlis*

University of Toronto
Canada
evahlis@cs.toronto.edu

Abstract. A family of trapdoor functions is one-way under correlated inputs if no efficient adversary can invert it even when given the value of the function on multiple correlated inputs. This powerful primitive was introduced at TCC 2009 by Rosen and Segev, who use it in an elegant black box construction of a chosen ciphertext secure public key encryption. In this work we continue the study of security under correlated inputs, and prove that there is no black box construction of correlation secure injective trapdoor functions from classic trapdoor permutations, even if the latter is assumed to be one-way for inputs from high entropy, rather than uniform distributions. Our negative result holds for all input distributions where each x_i is determined by the remaining $n-1$ coordinates. The techniques we employ for proving lower bounds about trapdoor permutations are new and quite general, and we believe that they will find other applications in the area of black-box separations.

1 Introduction

In this paper we study the following question: can classic trapdoor permutations be used to construct trapdoor functions that remain one way even when the adversary is given $F_{pub_1}(x_1), \ldots, F_{pub_n}(x_n)$ for independently chosen keys pub_i, but where the inputs x_i are correlated. In [17] Rosen and Segev introduce this problem, and highlight its importance by using such "correlation secure" injective trapdoor functions in a black box construction of chosen ciphertext secure public key encryption. Although this important type of public key encryption can be constructed from classic trapdoor permutations (see e.g., the seminal work of Dolev *et al* [8,9]), the constructions that achieve this goal make use of non-black-box techniques, which tend to be quite inefficient. In recent years there has been renewed effort to obtain constructions that use simpler primitives in a black box manner, yet so far no such constructions have been based on either semantically secure public key encryption, or even trapdoor permutations.

More generally, trapdoor permutations are a powerful public key primitive that is sufficient for many difficult applications in cryptography. Yet certain

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-11799-2_36

* Supported by the Natural Sciences and Engineering Research Council of Canada.

D. Micciancio (Ed.): TCC 2010, LNCS 5978, pp. 165–182, 2010.

tasks, such as Identity Based Encryption [4,5], have so far resisted attempts to be solved using this tool. Indeed, the limits of what can be constructed from trapdoor permutations are not well understood. In this paper we show that trapdoor permutations do not permit a black box construction of correlation secure injective trapdoor functions, even if the inputs are chosen from a distribution with very little correlation.

On a parallel line of research, our work is a step in the study of resettable security, a notion introduced by Canetti *et al* [6] in the context of Zero-Knowledge Proofs, and recently extended to general secure computation by Goyal and Sahai [14]. Informally, in resettable security the adversary is allowed to restart his security experiment while forcing the target primitive to reuse some of its previous randomness. The study of correlation security can be seen as another form of resettability: the adversary is allowed to selectively restart the experiment by preserving the random input to the functions, but forcing the regeneration of the function keys. In light of the positive results on resettability it is quite interesting that trapdoor permutations cannot be easily made resettably secure. Hence, our result can be seen as a step towards identifying which functionalities can be made resettably secure, and what is the required amount of interaction for achieving that level of security.

We now describe our problem and results in more detail, followed by an overview of related work, and an exposition of our technical approach.

Black-Box Cryptography. A common approach for constructing cryptographic primitives is to base them on some other, simpler, primitives that are believed to be secure. A construction of primitive A from primitive B is black box if the algorithms of A use the algorithms of B as oracles. A security reduction from A to B is black-box if there exists an adversary S such that for every adversary T that breaks B, S breaks A. Furthermore, S uses T as an oracle. In recent years, several breakthrough papers provide non-black-box solutions to some cryptographic tasks. Nevertheless, black-box constructions remain the most common approach. In this paper, all our results concern black-box constructions with a black-box security reduction. Such constructions are called "Fully Black-Box" in the taxonomy of Reingold *et al* [16].

Our Contributions. One-way trapdoor functions were introduced by Diffie and Hellman in [7]. Informally, a family of functions is one-way if given a description of randomly chosen function f_{pub} of that family, and its image $f_{pub}(x)$ on a randomly chosen input x, no efficient adversary can output x. A family of functions is "trapdoor" if there is a key generation algorithm that outputs a pair of strings (pub, pri) such that it is easy to invert $F_{pub}(\cdot)$ given pri. The notion of correlation security, introduced by Rosen and Segev in [17], extends the above experiment by giving the adversary $\left(F_{pub_1}(x_1), \ldots, F_{pub_n}(x_n)\right)$ where the pub_i are independently generated public keys, and $(x_i)_{i\in[n]}$ are sampled from some distribution $\mathcal{C}$. The family of functions is considered $\mathcal{C}$-correlation secure if no efficient adversary can invert the function on one of the coordinates. Of particular interest are distributions where the entire tuple (x_i) is reconstructible given some subset

of the coordinates. Correlation security under such distributions was shown in [17] to be sufficient to obtain chosen ciphertext security. In this paper we prove the following black-box separations:

- Let $\mathcal{C}_1$ be the uniform 2-repetition distribution: pairs of the form (x, x) where x is chosen uniformly at random. We show that there does not exist a black box construction of a $\mathcal{C}_1$-correlation secure family of injective trapdoor functions from classic trapdoor permutations.
- We then extend the above result to all input distributions that are $(n-1)$-reconstructible. That is, distributions of the form $(x_1, \ldots, x_n)$ where each x_i is determined by $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. This includes distributions with very weak correlation among the coordinates, such as $(n-1)$-wise independent distributions that are reconstructible in the above sense.

The base primitive in our separation actually has much stronger security properties than mere one-wayness. Indeed, our proofs show that even if one assumes a trapdoor permutation that is one-way for non-uniform (but high entropy) inputs then correlation security still cannot be achieved. Trapdoor permutations that are one-way for high entropy inputs have been shown by Bellare *et al* [1] to be sufficient to obtain deterministic public key encryption – a type of injective trapdoor functions, introduced by Bellare *et al* in [2], that hide almost all information about their input.

Related Work. In [15] Impagliazzo and Rudich introduced an approach for proving black-box separation results. In that seminal paper they prove a separation between one-way permutations and secure key-agreement. Since then a large body of research has followed their basic methodology. We provide a survey of the most relevant results, and recommend reading [16] for a more complete overview.

In this paper we study limits of public key primitives. In [11] Gertner *et al* show that public key encryption and Oblivious Transfer are incomparable under black-box reductions. They also show that trapdoor permutations cannot be constructed in a black-box way from public key encryption, and from trapdoor functions (functions which are not necessarily permutations, but allow sampling from the pre-image given trapdoor information). In [12] Gertner *et al* show that there is no black-box reduction from poly-to-one trapdoor functions to semantically secure public key encryption. Intuitively, [12] show that public key encryption is weaker than trapdoor functions because the latter allows the recovery of the complete input of the encryption algorithm, including the randomness. In contrast, a public key decryption algorithm recovers only the encrypted message, but not the randomness that was used by the encryptor. In [10] Gennaro *et al* show limits on the efficiency of cryptographic primitives constructed in a black-box way from basic tools such as one-way permutations, and trapdoor permutations. In particular they show bounds on the number of times that a trapdoor permutation needs to be invoked in order to construct a semantically secure public key encryption. Their lower bound is a function of the number of bits that the public key encryption scheme encrypts. Towards obtaining their

result, Gennaro *et al* define an oracle model which provides all algorithms access to a random trapdoor permutation family. We adopt this model partially in our work.

In [13] Gertner *et al* prove that chosen ciphertext secure public key encryption cannot be constructed in a black-box way from semantically secure public key encryption, provided that the decryption algorithm does not query the encryption oracle of the underlying primitive. In light of previous results that separate trapdoor permutations from semantically secure public key encryption the [13] result leaves open the possibility of achieving chosen ciphertext security using trapdoor permutations. Interestingly, the decryption algorithm in the construction of [17] *does* query the "encryption" algorithm of the underlying trapdoor function. In [5] Boneh *et al* show that Identity Based Encryption cannot be constructed in a black-box way from trapdoor permutations. In the context of the transformation by Boneh *et al* [3] of any Identity Based Encryption to a chosen ciphertext secure public key encryption, the work of [5] rules out one possible method of getting CCA public key encryption from trapdoor permutations. Our work rules out another such approach.

Overview of Techniques. The basic approach of most black-box separation results can be described as follows. Given a target primitive A and a base primitive B first define an "idealized" version of B. The idealized B is usually a distribution on functions that satisfy B's correctness requirements. Then, show that an adversary that is given oracle access to the ideal B cannot break its security, even if that adversary is computationally unbounded[1]. Then, describe an adversary that, by making a small number of queries to the ideal B, breaks *any* construction of A. Note that the fact that the adversaries are computationally unbounded requires any non-trivial A to make essential use of the ideal B oracle (otherwise that A is trivially broken). A common final step is to "project" the result into the realm of polynomial time computation by adding a **PSPACE** complete oracle. This oracle makes a polytime adversary effectively unbounded, but it does not help break the ideal B. For more details about this general approach we direct the reader to [15,16,18].

We use the work of [10] and [5] as a basis for defining our ideal trapdoor permutation oracle. In their work, Gennaro *et al* define a distribution on triples of functions (g, e, d) where $g(\cdot)$ is a random function that maps trapdoors to public keys, $e(pub, \cdot)$ is an independent random permutation for every public key pub, and $d(pri, \cdot)$ inverts the permutation $e(pub, \cdot)$ if pri is a trapdoor for pub. Although this model captures nicely the concept of an ideal trapdoor permutation, we cannot adopt it directly. The problem is that the triple (g, e, d) is in fact correlation secure: for each public key pub the permutation $e(pub, \cdot)$ is chosen independently at random. So, seeing $e(pub_1, x)$ and $e(pub_2, x)$ does not provide any additional information about x over just seeing $e(pub_1, x)$. Our solution is to introduce an additional oracle which we call Break so that given access to $(g, e, d, \mathsf{Break})$ it is no longer possible to obtain correlation security, but

[1] Note that this necessarily implies that there is no polynomial time implementation of the idealized B.

one-wayness is preserved with respect to independently random inputs. It is the main technical contribution of this paper to find the delicate balance that leaves the entire oracle just strong enough to preserve one-wayness, but weak enough to obtain the negative result.

On a high level, the oracle Break can be described as follows: Break takes as input a triple of circuits G, E, D which are a candidate correlation secure trapdoor function. The other inputs are, two public keys PUB_1 and PUB_2, and the values $E_{PUB_1}(x)$ and $E_{PUB_2}(x)$. The naive solution would be to return x. However, this would easily allow an adversary to invert any function simply by setting $pub_1 = pub_2$. Ideally we would like to restrict Break to return x only when pub_1 and pub_2 are independently generated public keys of the provided trapdoor permutation. This, however, seems hard to check. Indeed, how can we verify that the public keys were properly generated? Moreover, even performing a simpler test: that the public keys are valid (that is, they are outputs of the key generation algorithm), may give too much power to the adversary. In particular, an adversary trying to invert $f(x)$, where f is any function, may design a trapdoor permutation scheme where pub_1 is a valid public key if and only if the first bit of x is 0. To prevent the adversary from performing the above attack, we require her to provide a triple of functions $\mathcal{O}' = (g', e', d')$ that is defined on a small part of the domain of (g, e, d) but such that relative to $\mathcal{O}'$, pub_1 and pub_2 are valid public keys. The partial oracle $\mathcal{O}'$ is then superimposed on $\mathcal{O}$ to obtain a new complete oracle $\mathcal{O}''$ relative to which pub_1 and pub_2 are valid public keys. The oracle Break then performs its computation relative to the new oracle $\mathcal{O}''$.

This modification successfully deals with the issue of the validity of public keys, but we are still left with no way of knowing that the public keys were generated independently. This causes a problem because an adversary trying to break the random trapdoor permutation (g, e, d) may simply set pub_1 to be the public key that she is trying to break, and set $pub_2 = pub_1$. Thus, some kind of additional check seems necessary, yet testing for independence of pub_1 and pub_2 seems too much to hope for. As it turns out, we do not need the two public keys to be completely independent. As illustrated by the above example, we run into a problem when our Break oracle allows the adversary to use the same public key of (g, e, d) in both public keys of (G, E, D). But, if we require that the sets of public keys of (g, e, d) that are used to generate PUB_1 and PUB_2 are *disjoint*, then it becomes difficult to invert $y = e(pub, x)$. To do so the adversary would have to find $e(pub', x)$ for some pub' different from pub. However, this is as hard as finding x since the permutations $e(pub, \cdot)$ and $e(pub', \cdot)$ are random and independent. We formalize the above ideas in Section 3.

2 Preliminaries

2.1 Notation

Denote by $\mathbb{N}$ the set of natural numbers. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. For a set S we denote by $x \leftarrow_{\mathrm{R}} S$ the procedure of uniformly sampling an element from S and assigning the value to x. We write $x \in_{\mathrm{R}} S$ to denote the fact that x

is a uniformly sampled element of S. Although the distinction between families of functions and functions is very important, we occasionally write "trapdoor permutation" and "trapdoor function" instead of "family of trapdoor permutations" and "family of trapdoor functions". We do so to improve exposition, and only when the meaning is clear from context.

2.2 Probabilistic Lemmas

We will make use of the following simple fact:

Lemma 1. *Let $X_1, \ldots, X_{n+1}$ be independent Bernoulli random variables, where $\Pr[X_i = 1] = p$ 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, but X_{n+1} is sampled at 0. Then, $\Pr[\mathcal{E}] \leq \frac{1}{e \cdot n}$. Note that the bound is independent of p.*

2.3 Non-uniform Trapdoor Permutations in the Presence of Oracles

We prove our theorems in a non-uniform computational model. Thus, a collection of Trapdoor Permutations is specified, for each value of the security parameter m, by a triple of deterministic PT algorithms (G, E, D), and the following additional parameters:

- $\lambda(m)$ is the length parameter of the trapdoor permutation oracle (g, e, d) that is used by (G, E, D).
- $q = q(m)$ is the maximum number of oracle queries that any of the algorithms make in a single execution. For convenience we assume that the algorithms always make exactly q queries.

The functionality of each of the algorithms is as follows: $G(\cdot)$ takes as input a trapdoor $PRI \in \{0,1\}^m$, and outputs a function public key $PUB \in \{0,1\}^m$. $E_{PUB}(\cdot)$ is a permutation on strings of length m. Finally, $D_{PRI}(\cdot)$ computes the inverse function of $E_{G(PRI)}(\cdot)$.

We now define two security conditions: one-wayness, and correlation-security (or one-wayness in the presence of correlated products). Let A be an algorithm with access to the same oracle as (G, E, D). We define the advantage of A in the one-wayness experiment with respect to an input distribution D over $\{0,1\}^m$ as follows:

$$\delta_{OW}(A, D) \stackrel{def}{=} \Pr[A(PUB, E_{PUB}(x)) = x;\ PUB \leftarrow G(PRI)]$$

where PRI is chosen uniformly at random from $\{0,1\}^m$ and x is sampled according to D. For convenience, we denote $\delta_{OW}(A) \stackrel{def}{=} \delta_{OW}(A, U_m)$, where U_m is the uniform distribution over $\{0,1\}^m$.

For a distribution $\mathcal{C}$ on $(\{0,1\}^m)^n$ for some $n \in \mathbb{N}$, we define the advantage of A in the *$\mathcal{C}$-correlation security* experiment as follows:

$$\delta_{CS}(A) \stackrel{def}{=} \Pr[A((PUB_i, E_{PUB_i}(x_i))_{i \in [n]}) \in \{x_i\}_{i \in [n]};\ PUB_i \leftarrow G(PRI_i)]$$

where $(x_i)_{i\in[n]} \in_R \mathcal{C}$, and PRI_i for $i \in [n]$ are chosen uniformly at random from $\{0,1\}^m$. As a convention, we will frequently omit the lengths of strings when discussing trapdoor permutations, when the length is clear from context.

We measure the complexity of algorithms in the oracle model only by the number of oracle queries that they make. Using a standard technique of adding a **PSPACE** oracle we obtain the fully black-box separation for probabilistic polynomial time Turing Machines (see [16,18] for a detailed exposition of the approach).

3 Our Oracles

We prove our theorem in a relativized model where all algorithms have access to three random oracles (g, e, d) that roughly correspond to the algorithms G, E, and D of a Trapdoor Permutation. For every $\lambda \in \mathbb{N}$, the oracles (g, e, d) are sampled uniformly at random from the set of all functions satisfying the following conditions:

- $g : \{0,1\}^\lambda \to \{0,1\}^\lambda$. We view g as taking a secret key pri as input and outputting a public key.
- $e : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a function that on input $pub \in \{0,1\}^\lambda$ and $x \in \{0,1\}^\lambda$ outputs $e(pub, x) \in \{0,1\}^\lambda$. We require that for every $pub \in \{0,1\}^\lambda$ the function $e(pub, \cdot)$ be a permutation of $\{0,1\}^\lambda$.
- $d : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a function that on input $pri \in \{0,1\}^\lambda$ and $y \in \{0,1\}^\lambda$ outputs an $x \in \{0,1\}^\lambda$ that is the (unique) pre-image of y under the permutation defined by the function $e(g(pri), \cdot)$.

Redundancy of d. The function d is completely determined by g, e. Thus, when discussing a description of a trapdoor permutation oracle $\mathcal{O} = (g, e, d)$ we will assume that d is deduced from g, e rather than being part of the description.

Partial Oracles. In our proofs we will occasionally need to refer to trapdoor permutation oracles that are defined on a subset of the domain. We call a function $\mathcal{O}' = (g', e')$ which is defined on a subset of the domain of $\mathcal{O}$, a valid *partial oracle* if for every pub, $e'(pub, \cdot)$ is 1-1. Again, d is not part of the description of $\mathcal{O}'$. Instead, it is determined from (g', e') as follows: for strings pri, and y, $d(pri, y)$ is defined and equal to x if and only if $g'(pri) = pub$ and $e'(pub, x) = y$.

Conventions. Without loss of generality we assume that whenever an algorithm makes an oracle query of the form $d(pri, y)$, it first queries $g(pri)$. This assumption is useful for a cleaner presentation of our proofs.

The Oracle Break. In addition to the oracles $\mathcal{O} = (g, e, d)$, our adversary will have access to an oracle Break that weakens the above random trapdoor permutation. Before we can describe Break we must introduce some additional notation.

3.1 Oracle Notation

Before proceeding with the description of the oracle Break, let us introduce additional notation that we use when discussing various aspects of the trapdoor permutation oracle.

Oracle algorithms. For a function $\mathcal{O}$ and algorithm A we denote by $A^{\mathcal{O}}$ the fact that A may make queries to $\mathcal{O}$.

Functions as sets. It will occasionally be convenient to view the trapdoor permutation oracle $\mathcal{O} = (g, e, d)$ as sets of input-output pairs. We will use square brackets to denote the symbolic form of a mapping. For instance: to denote that $e(pub, x) = y$ we may write $[e(pub, x) = y] \in \mathcal{O}$. We will occasionally use a wild card form of this notation. For instance: we write $[e(pub, \cdot) = y] \in \mathcal{O}$ to denote that there exists x such that $[e(pub, x) = y] \in \mathcal{O}$.
When discussing queries we write $[e(pub, x)]$ to denote a query to $e(\cdot, \cdot)$ with inputs (pub, x). This is to differentiate the query from the actual value $e(pub, x)$ which is the image of (pub, x) under the function e. Given a query q in symbolic form we write $\mathcal{O}(q)$ to denote the mapping under $\mathcal{O}$ of that query. For example: if $q = [e(pub, x)]$ and $e(pub, x) = y$ then $O(q) = [e(pub, x) = y]$.

Adding answers. Given $\mathcal{O}$ and a set of queries Q we define $\mathcal{O}(Q)$ to be the set of queries in Q with their answers according to $\mathcal{O}$. Namely, $\mathcal{O}(Q) = \{[\alpha = \beta] | \alpha \in Q,\ \mathcal{O}(\alpha) = \beta\}$.

Public keys that are used in a query. Given a trapdoor permutation oracle $\mathcal{O} = (g, e, d)$, and a set Q of (g, e, d) queries we define $PK_e(Q)$ to be the set of all pub such that $[e(pub, \cdot)] \in Q$. Similarly, we define $PK_g(Q)$ to be the set of all pub such that $[g(\cdot) = pub] \in \mathcal{O}(Q)$.

Combining trapdoor permutation oracles. Let $\mathcal{O}_1 = (g_1, e_1)$ and $\mathcal{O}_2 = (g_2, e_2)$ be two (possibly partial) trapdoor permutation oracles. We write $\mathcal{O}_1 \diamond \mathcal{O}_2$ to denote the oracle which answers according to $\mathcal{O}_2$ if possible, and according to $\mathcal{O}_1$ otherwise. More precisely, $(\mathcal{O}_1 \diamond \mathcal{O}_2)(\alpha)$ returns $\mathcal{O}_2(\alpha)$ if it is defined, and $\mathcal{O}_1(\alpha)$ otherwise.
We also define a "corrected" version of the $\diamond$ operator. We write

$$\mathcal{O}_1 \diamond_c \mathcal{O}_2 \stackrel{def}{=} (g_1 \diamond g_2, e_1 \diamond_c e_2)$$

The oracle $e = e_1 \diamond_c e_2$ is defined as follows: let pub, x, y such that $e_2(pub, x) = y$. We set $e(pub, x) = y$. Furthermore, if there exists $x' \neq x$ such that $e_1(pub, x') = y$ then let $y' = e_1(pub, x)$ (note that y' may be equal to $\perp$ if $e_1(pub, x)$ is not defined). We then also set $e(pub, x') = y'$.
Note that the d part of the oracle $\mathcal{O}_1 \diamond_c \mathcal{O}_2$ is deduced from g and e. A useful property of the $\diamond_c$ operator is that for every two possibly partial oracles $\mathcal{O}_1$ and $\mathcal{O}_2$, the oracle $\mathcal{O}_1 \diamond_c \mathcal{O}_2$ has no collisions (i.e. it is a valid partial oracle), and for every α such that $\mathcal{O}_2(\alpha)$ is defined, $(\mathcal{O}_1 \diamond_c \mathcal{O}_2)(\alpha) = \mathcal{O}_2(\alpha)$.

3.2 The Oracle Break

As mentioned in the introduction, the random trapdoor permutations that the oracles (g, e, d) represent are, in fact, correlation secure. This is so because each

permutation is random and independent from the other permutations. Thus, we add a weakening oracle Break that allows our adversary to break correlation security of any trapdoor function that makes use of (g, e, d), yet preserves the one-wayness of (g, e, d). For a better exposition we present the oracle and the proof for the case of Trapdoor *Permutations*. However, both easily extend to handle injective trapdoor functions. The details are given in Section 6.

The functionality of Break is defined as follows:

Input. Break takes the following inputs:

1. A triple of oracle circuits $(G^{\mathcal{O}}, E^{\mathcal{O}}, D^{\mathcal{O}})$ that may contain (g, e, d) oracle gates. The functions computed by G, E, D must constitute a valid family of trapdoor permutations.
2. Two strings PUB_1, PUB_2. We think of these strings as public keys that were produced by G.
3. Two strings y_1, y_2. We think of these strings as the outputs of $E_{PUB_1}(x)$ and $E_{PUB_2}(x)$ respectively on some input x.
4. Two partial oracles $\mathcal{O}'_1$ and $\mathcal{O}'_2$, and two strings PRI_1 and PRI_2.

Computation. The following computation is performed by the oracle:

1. Verify that for every $pub \in PK_g(\mathcal{O}'_1) \cap PK_g(\mathcal{O}'_2)$, there exists a pri such that $g(pri) = pub$ and $[g(pri) = pub] \in \mathcal{O}'_1 \cup \mathcal{O}'_2$. Note that the requirement here is that $g(pri) = pub$ under the real oracle $\mathcal{O}$. Therefore, for every pub as above, we require that $\mathcal{O}'_1$ or $\mathcal{O}'_2$ provide us with a real trapdoor for it.
2. Verify that (G, E, D) is a valid family of trapdoor permutations, and for $i \in \{1, 2\}$, for every complete trapdoor permutation oracle $\mathcal{O}''$, if $\mathcal{O}'_i \subseteq \mathcal{O}''$, then for every $x \in \{0,1\}^m$, it holds that $D^{\mathcal{O}''}_{PRI_i}(E^{\mathcal{O}''}_{PUB_i}(x))$ returns x.
3. Let $\mathcal{O}''_i = \mathcal{O} \diamond_c \mathcal{O}'_i$.
4. Run $D^{\mathcal{O}''_1}_{PRI_1}(y_1)$ to obtain an output x. If $x = \perp$, return $\perp$.
5. For $i \in \{1, 2\}$, run $E^{\mathcal{O}''_i}_{PUB_i}(x)$, and return $\perp$ if one of the following events occurs: (i) the output of $E^{\mathcal{O}''_i}_{PUB_i}(x)$ is not equal to y_i, or (ii) $E^{\mathcal{O}''_i}_{PUB_i}(x)$ asks a query $\alpha = [g(pri)]$ such that $\mathcal{O}(\alpha) \neq \mathcal{O}''_i(\alpha)$. Finally, return x.

Complexity. Each query to Break is counted as $3q + |\mathcal{O}'_1| + |\mathcal{O}'_2|$ oracle queries. This is to prevent an adversary from making a very large Break query that gives away too much information about $\mathcal{O}$. The breakdown of the above cost is as follows: $3q$ comes from steps 4 and 5 where Break evaluates D once and E twice. The count $|\mathcal{O}'_1| + |\mathcal{O}'_2|$ is due to steps 1, and 3. In step 1 Break has to compare elements of the form $[g(pri) = pub] \in \mathcal{O}'_1 \cup \mathcal{O}'_2$ to $\mathcal{O}$. In step 3 Break has to know at most $|\mathcal{O}'_1| + |\mathcal{O}'_2|$ entries in $\mathcal{O}$ in order to perform the $\diamond_c$ operation.

This concludes the description of our oracles. In the following two sections we prove the two main lemmas that are required for our theorem: that the oracle Break preserves the one-wayness of $\mathcal{O} = (g, e, d)$, and that there exists an adversary which uses Break that breaks the correlation security of every family of injective trapdoor functions.

4 Breaking Security under Correlated Inputs

In this section we describe an adversary that breaks the correlation-security of any trapdoor permutation, while making only a polynomial number of queries to the oracles $(g, e, d, \mathsf{Break})$.

Let (G, E, D) be a collection of injective trapdoor functions with length parameter m, and maximal number of queries q. For simplicity, and due to lack of space we describe an adversary that breaks only constructions (G, E, D) that do not query Break, but only use (g, e, d). The extension of the adversary and the oracle Break to handle injective trapdoor functions (as opposed to permutations) that make use of Break is quite easy, and is described in Section 6.

4.1 Overview

We start with an informal description of our adversary. The adversary is initially given two independently generated public keys PUB_1 and PUB_2. Recall that in order to make use of the oracle Break the adversary has to come up with two partial oracles $\mathcal{O}'_1$ and $\mathcal{O}'_2$ such that PUB_1 and PUB_2 are valid outputs of $G^{\mathcal{O}'_1}$ and $G^{\mathcal{O}'_2}$ respectively. Since our adversary is computationally unbounded, that is, we count only the number of oracle queries that she makes, it is easy for her to find two such partial oracles. However, there are two issues that arise: (i) in order to pass check 1 of Break the adversary has to know the *correct* trapdoor for each *pub* that is appears in the generation of both PUB_1 and PUB_2; and (ii) if the actual oracle (g, e, d) is not used, it is quite possible that under these partial oracles, y_1 and y_2 will not invert to x. Both issues are dealt with simultaneously by performing a sampling procedure that discovers all the queries that are frequently asked by G, and by $E_{PUB_1}(x)$ and $E_{PUB_1}(x)$ where x is chosen randomly. The adversary then chooses $\mathcal{O}'_1$ and $\mathcal{O}'_2$ offline, without making any further oracle queries, but in a manner that is consistent with the information about (g, e, d) that was learned during the sampling procedure.

To see why the above procedure solves problem (i) recall that PUB_1 and PUB_2 are generated independently. Therefore, with high probability any public keys *pub* that are needed to generate *both* PUB_1 and PUB_2 are also needed to generate many other PUB's that G may output. This means that, with high probability, the adversary will generate at least one such PUB, and discover the correct trapdoor for *pub* in the process. Problem (ii) is solved due to the following simple fact: $\mathcal{O}'_1$ and $\mathcal{O}'_2$ are defined on a polynomial number of points. For each such point, if it is frequently accessed by one of $E_{PUB_1}(x)$ or $E_{PUB_2}(x)$ then the adversary would have discovered the correct value for it during the sampling. If the point is infrequently accessed, then with high probability it was not accessed when y_1 and y_2 were computed. For a similar reason, the adversary's query to Break passes the second check of step 5.

If the adversary manages to make a query to Break that is not answered with $\perp$ then with overwhelming probability the answer to that query is x, which is the inverse of y_1 and y_2. We are now ready to describe the adversary completely and analyze its performance.

4.2 The Adversary

For convenience, and without loss of generality, we assume $q \geq 2$. For any $\varepsilon > 0$ we provide a PPT adversary A, and a constant c such that $\delta_{CS}(A) \geq 1-\varepsilon$ and A makes at most q^c oracle queries. More precisely, given $\varepsilon > 0$ choose two integers c_1, c_2 such that (i) $\left(1 - \frac{1}{q^{c_1}}\right)^q \geq 1 - \frac{1}{q^{c_1-1}}$, and (ii) $\varepsilon \geq \left(\frac{q^{c_1+1}}{e^{q^{c_1}}} + \frac{1}{q^{c_1-1}} + \frac{4}{eq^{c_2}}\right)$. Our adversary proceeds in several steps:

1. The adversary is given PUB_1, PUB_2, y_1, y_2. It starts by initializing tables L, L_1, L_2, which will be used to store points of $\mathcal{O}$ that the adversary discovers. More precisely, these tables are partial oracles that are updated by the adversary in the following steps, and always satisfy $L, L_1, L_2 \subseteq \mathcal{O}$.
2. For $1 \leq i \leq q^{2c_1}$ the adversary chooses $PRI_i \in_{\mathrm{R}} \{0,1\}^n$, and simulates $G(PRI_i)$. For every query α asked by G during the simulation, the adversary adds the entry $(\alpha, \mathcal{O}(\alpha))$ to L.
3. For $1 \leq i \leq q^{c_2}$ the adversary chooses $x_i \in_{\mathrm{R}} \{0,1\}^m$, and simulates $E^{\mathcal{O}}_{PUB_1}(x_i)$ and $E^{\mathcal{O}}_{PUB_2}(x_i)$, recording all queries and answers in L_1 and L_2 respectively.
4. The adversary now selects partial oracles $\mathcal{O}'_i$, and strings PRI_1, PRI_2 for $i \in \{1,2\}$ such that:
   (a) $|\mathcal{O}'_i| \leq |L \cup L_1 \cup L_2| + q$.
   (b) $L_1 \cup L_2 \cup L \subseteq \mathcal{O}'_i$, and $G^{\mathcal{O}'_i}(PRI_i) = PUB_i$.
   (c) For every $pub \in PK_g(\mathcal{O}'_1) \cap PK_g(\mathcal{O}'_2)$ there exists an pri such that $[g(pri) = pub] \in L \cup L_1 \cup L_2$.

   If no such partial oracles exist then the adversary gives up and terminates.
5. The adversary queries $\mathsf{Break}(G, E, D, PUB_1, PUB_2, y_1, y_2, \mathcal{O}'_1, \mathcal{O}'_2, PRI_1, PRI_2)$. If Break returns $\perp$ then the adversary fails (this can be modeled by the adversary returning a random string $x \in \{0,1\}^m$). Otherwise, Break returns x, which the adversary returns as well.

4.3 Analysis

This concludes the description of our adversary. We now turn to proving that our adversary makes a successful query to Break which returns the correct inverse x. In order to prove this we need to show that the following two statements are true with high probability:

1. The adversary's query passes all the checks of Break.
2. Under $\mathcal{O}''_i = \mathcal{O} \diamond_c \mathcal{O}'_i$ it holds that $E^{\mathcal{O}''_1}_{PUB_1}(x) = y_1$ and $E^{\mathcal{O}''_1}_{PUB_2}(x) = y_2$.

We start with the first statement. We will use the following random variables in the statement of the lemma. Consider a run of our adversary in the correlation security experiment. Let $\mathcal{O}, PRI'_1, PRI'_2$ be the TDP oracle, and the private keys respectively, that are selected by the challenger. Let Q_{PUB_i} and $Q_{x,i}$ to be the sets of queries asked during the computations $G^{\mathcal{O}}(PRI'_i)$, and $E^{\mathcal{O}}(PUB_i, x)$ respectively. We define $T_{PUB_i} = \mathcal{O}(Q_{PUB_i})$ and $T_{x,i} = \mathcal{O}(Q_{x,i})$. For the first statement above we are interested in the following event:

Event E**.** For every pub for which there exist pri_1, pri_2 such that $[g(pri_1) = pub] \in T_{PUB_1}$ and $[g(pri_2) = pub] \in T_{PUB_2}$ there exists an pri such that $[g(pri) = pub] \in L$.

Essentially, the event E states that our adversary has discovered a trapdoor for every public key that was generated in the computation of both $G^{\mathcal{O}}(PRI'_1)$ and $G^{\mathcal{O}}(PRI'_2)$. The following claim shows that this happens with high probability. Intuitively, this is so because PRI'_1 and PRI'_2 are chosen independently at random, and our adversary samples many such computations in step 2. Thus, if a public key is likely to be generated by $G^{\mathcal{O}}(PRI)$ for a random PRI, then our adversary has already found it. If it is unlikely to be generated then it is unlikely to appear in the computation for two independent PRIs.

Claim. At step 5 of the adversary, the event E occurs with probability $\geq 1 - \frac{q^{c_1+1}}{e^{q^{c_1}}} + \frac{1}{q^{c_1-1}}$.

Proof (Sketch). In step 2 the adversary simulates $G(PRI)$ many times, thus learning all the pubs that are frequently generated by G on random PRI. In order for a public key pub to be likely to appear in the computation of two independent executions of G, it must frequently generated by G. Therefore, with high probability, all the pubs that were generated in *both* the computation of $G(PRI_1)$ and $G(PRI_2)$ have already been observed by the adversary during the sampling of step 2. The complete proof is given in the full version of this paper [19].

We now show that it is sufficient for event E to occur in order for our adversary to successfully construct the partial oracles $\mathcal{O}'_1$ and $\mathcal{O}'_2$, and make a Break query that passes checks 1 and 2.

Claim. If event E occurs then the adversary successfully constructs the oracles $\mathcal{O}'_1$ and $\mathcal{O}'_2$ in step 4. Furthermore, the adversary's query to Break successfully passes checks 1 and 2.

Proof. If event E occurs then L contains trapdoors for all pub that appear in both the computation of $G^{\mathcal{O}}(PRI_1)$ and $G^{\mathcal{O}}(PRI_2)$. Thus, one possibility for the values of $\mathcal{O}'_1, \mathcal{O}'_2, PRI_1, PRI_2$ in this case is simply the correct PRI_1, PRI_2 and the subset of $\mathcal{O}$ that is used in the generation of PUB_1 and PUB_2 respectively.

Now consider the adversary's query to Break. Check 1 passes because of the conditions imposed on the choice of the partial oracles $\mathcal{O}'_1$ and $\mathcal{O}'_2$. To see why check 2 passes, notice that under $\mathcal{O}'_i$ the made up PRI_i is a correct private key for the public key PUB_i, and so by the correctness of the trapdoor permutation (G, E, D), for every oracle $\mathcal{O}''$ such that $\mathcal{O}'_i \subseteq \mathcal{O}''$, $D^{\mathcal{O}''}(PRI_i, y)$ inverts y correctly.

Our next step is to prove the second property of our adversary: that for for $i \in \{1, 2\}$, with high probability, for every query α that is asked by $E^{\mathcal{O}}(PUB_i, x)$ the answers under the oracle $\mathcal{O}$, and the modified oracle $\mathcal{O}''_i = \mathcal{O} \diamond_c \mathcal{O}'_i$ are identical. The proof of this statement is very similar to Claim 6.9 in [5]. We repeat it here for completeness.

Lemma 2. *Let $\mathcal{O}'_i$, for $i \in \{1,2\}$, be the partial oracles chosen by the adversary, and let $\mathcal{O}''_i = \mathcal{O} \diamond_c \mathcal{O}'_i$. Then, with probability at least $1 - \frac{2}{eq^{c_2}}$, for every query α asked by $E^{\mathcal{O}}_{PUB_i}(x)$, $\mathcal{O}''_i(\alpha) = \mathcal{O}(\alpha)$.*

Proof. From the fact that $\mathcal{O}'_i$ is defined on at most q points that are not in $L \cup L_1 \cup L_2$, and the definition of the $\diamond_c$ operator we know that $\mathcal{O} \diamond_c \mathcal{O}'_i$ differs from $\mathcal{O}$ on at most $2q$ points.

More precisely, for a query α of the form $\alpha = [g(sk)]$ where $\mathcal{O}(\alpha) \neq \mathcal{O}''_i(\alpha) = pk$ it must be that $[g(sk) = pk] \in \mathcal{O}'_i \setminus L \cup L_1 \cup L_2$. Thus, queries of this form contribute at most one point on which $\mathcal{O}''_i$ and $\mathcal{O}$ differ.

If α is of the form $e(pk, x)$, and $[e(pk, x) = y] \in \mathcal{O}$ and $[e(pk, x) = w] \in \mathcal{O}''_i$, where $w \neq y$, then one of the following holds: either $[e(pk, x) = w] \in \mathcal{O}'_i \setminus L \cup L_1 \cup L_2$ or there exists x' such that $e(pk, x') = w$, and $e'_i(pk, x') = y$. Thus, queries of this form can contribute at most two points on which $\mathcal{O}''_i$ and $\mathcal{O}$ differ.

Consider a query α such that $\mathcal{O}(\alpha) \neq \mathcal{O}''_i(\alpha)$. Then, $[\alpha, \mathcal{O}(\alpha)] \notin L_1 \cup L_2$. This means that α did not appear in any of the simulations in step 3. Since the simulations in step 3 are done with independently chosen x_i, we can apply Lemma 1, and so the probability of α appearing during the computations $E^{\mathcal{O}''_i}_{PUB_i}(x) = y_i$ for $i \in \{1,2\}$ is at most $\frac{1}{eq^{c_2}}$.

Applying the union bound over all $\leq 2q$ points on which $\mathcal{O}$ and $\mathcal{O}''_i$ differ, we obtain that the probability that a query α is asked during $E^{\mathcal{O}}_{PUB_i}(, x)$ such that $\mathcal{O}(\alpha) \neq \mathcal{O}''_i(\alpha)$ is at most $\frac{2q}{eq^{c_2}}$.

We are now ready to prove the main theorem of this section: that our adversary successfully breaks the correlation security of any trapdoor permutation with probability which is arbitrarily close to one.

Theorem 1. *Given PUB_1, PUB_2, y_1, y_2 in the correlated one-wayness experiment, our adversary wins with probability $\geq 1 - \left(\frac{q^{c_1+1}}{e^{q^{c_1}}} + \frac{1}{q^{c_1-1}} + \frac{4}{eq^{c_2}}\right)$. Furthermore, it does so by making at most $5q + 3q^{c_1+1} + 6q^{c_2+1}$ oracle queries.*

Proof. The theorem follows by simple calculation from the above claims, and Lemma 2. The complete proof appears in [19].

5 One-Way Trapdoor Permutations Exist Relative to Our Oracle

In this section we show that the trapdoor permutation (G, E, D) where $G^{\mathcal{O}}(pri) = g(pri)$, $E^{\mathcal{O}}_{pub}(x) = e(pub, x)$ and $D^{\mathcal{O}}_{pri}(y) = d(pri, y)$ is a secure one-way trapdoor permutation, even when the adversary is given access to the oracle Break. Let A be an adversary that tries to break the one-wayness of (G, E, D). We show that $\delta \leq \frac{3q}{2^\lambda - q} + \frac{3q}{2^\lambda}$ where $\delta = \delta_{OW}(A)$ is the advantage of A in the one-wayness experiment. In fact, our proof carries through even when the input x is not uniform, but is chosen from a high entropy distribution.

The adversary's input is a pair (pub^*, y^*), and she is given access to oracles $(g, e, d, \mathsf{Break})$. Our proof proceeds in two steps: first, we show that if we modify

the oracle Break slightly then the adversary can simulate the modified oracle Break′ on her own with high probability. Since no adversary can break the one-wayness of a random trapdoor permutation, we obtain a bound on the advantage of an adversary that has access to Break′ instead of Break. The second step is to show that, in fact, the oracles Break and Break′ always produce the same answer. Combining the two steps together we get a bound on the advantage of A.

The Modified Oracle Break′. The oracle Break′ is parameterized by a public key pub^*, and is defined as follows: Break′ is the same as Break except step 4, which is modified in Break′ as follows:

4. Let $i \in \{1, 2\}$ such that $pub^* \notin PK_g(\mathcal{O}'_i)$. If $pub^* \in PK_g(\mathcal{O}'_1) \cap PK_g(\mathcal{O}'_2)$ or $pub^* \notin PK_g(\mathcal{O}'_1) \cup PK_g(\mathcal{O}'_2)$ then set $i = 1$. Then, run $D_{PRI_i}^{\mathcal{O}'_i}(y_i)$ to obtain an output x. If $x = \perp$ return $\perp$.

In other words, instead of always inverting y_1, Break′ inverts y_i where pub^* is not generated during the generation of PUB_i. Intuitively, this is a useful property because A is trying to break a single public key pub^*. Relying on the fact that the permutation $e(pub^*, \cdot)$ is random and independent from the rest of the oracle, A can simulate Break′ by generating the rest of the oracle by herself. She runs into trouble only when asked to invert $e(pub^*, \cdot)$. However, this is avoided by the check that is performed in step 5, and by the above modification. The check of step 5 prevents E from making a query that requires a trapdoor for pub^*. Note that although this may seem like a severe restriction, we have shown in Section 4 that it does not prevent us from breaking correlation security. The change in Break′ allows A to invert the one y_i which does not require knowledge of a trapdoor for pub^*.

5.1 Simulating Break′

The simulator itself is very technical, and is given in full detail in the full version [19]. We describe here the main ideas that are used in the construction. As previously mentioned, our adversary can generate all of $\mathcal{O}$ by herself, except for the permutation $e(pub^*, \cdot)$. Thus, if she wanted to simulate Break′ she would run into the following two problems: firstly, she is unable to compute the oracles $\mathcal{O}''_i = \mathcal{O} \diamond_c \mathcal{O}'_i$, and secondly she is unable to answer queries of the form $[d(pri, y)]$ where $[g(pri) = pub^*] \in \mathcal{O}''_i$.

The first problem is caused by the fact that the $\diamond_c$ operator resolves collisions, which requires the knowledge of entries of $\mathcal{O}$ of the form $[e(pub^*, x) = y]$ where $[e(pub^*, x') = y] \in \mathcal{O}'_i$. Our adversary may not possess this knowledge which prevents her from resolving all collisions. Instead, she resolves only the collisions that are *known* to her from the previous queries to the actual oracle $\mathcal{O}$, and the inputs of the Break′ query. Since the rest of $e(pub^*, \cdot)$ is random, she is unlikely to stumble unto any new collisions during the simulation of the Break′ query.

The second problem is caused by the fact that $\mathcal{O}'_i$ are adversarially chosen, and as such may contain an entry $[g(pri) = pub^*]$ which is incorrect according to

$\mathcal{O}$. This allows $D_{PRI_i}^{\mathcal{O}_i''}(y_i)$ and $E_{PUB_i}^{\mathcal{O}_i''}(x)$ to query $d(pri, y)$, which the adversary is unable to answer. This is dealt with as follows: to prevent D from making such queries we introduced the modification in Break'. The only case in which $\mathcal{O}_1''$ and $\mathcal{O}_2''$ may both contain trapdoors for pub is when the adversary knows at least one such trapdoor which is correct according to $\mathcal{O}$. Consequently, since it is hard to find a correct trapdoor for pub^*, there is an i such that $\mathcal{O}_i'$ does not define any fake trapdoors for it. It is now safe to simulate $D_{PRI_i}^{\mathcal{O}_i''}(y_i)$ because D is unlikely to find a true trapdoor of pub^*. To prevent E from making such queries, Break (and Break') perform a check in step 5 that forces the oracle to return $\perp$ if E makes a query of the form $g(pri)$ where $\mathcal{O}_i'$ and $\mathcal{O}$ disagree on the answer.

5.2 Equivalence of Break and Break′

We have shown that Break' provides very little help to an adversary trying to break the one-wayness of (g, e, d). We now show that Break and Break' always answer queries identically, which proves that Break does not break the one-wayness of (g, e, d).

Claim. The adversary A has advantage δ when given access to Break' instead of Break.

Proof (Sketch). The complete proof appears in [19]. Informally, let $x_i = D_{PRI_i}^{\mathcal{O}_i''}(y_i)$. The only difference between Break and Break' is that in step 4 Break always computes x_1 and Break' sometimes computes x_2. There are two cases: if $x_1 \neq x_2$ then due to the fact that E_{PUB_i} is a permutation, step 5 will return $\perp$. If $x_1 = x_2$, then both oracles perform the same computation in step 5.

5.3 Main Theorem

We are now ready to prove the main theorem of this section. We prove a strong variant that implies the security of (g, e, d) even when the input x is not uniform, but is chosen from a high entropy distribution. As previously mentioned, this implies that even a strong type of trapdoor functions (deterministic public key encryption) is insufficient to obtain correlation security.

Theorem 2. *Let G, E, D be the trapdoor permutation that forwards its input directly to the oracles g, e, d, and let D be a distribution over $\{0,1\}^\lambda$ such that $H_\infty(D) = k$. Then, for every adversary A that makes at most q oracle queries to $(g, e, d, \mathsf{Break})$, $\delta = \delta_{OW}(A, D) \leq \frac{2q}{2^k - q} + \frac{q}{2^\lambda - q} + \frac{3q}{2^\lambda}$.*

Proof (Sketch). The proof follows by a simple calculation from the equivalence of Break and Break', and the ability of the adversary to simulate Break'. The complete argument appears in [19].

6 Extensions

In this section we present several strengthenings of our basic theorem, and address the simplifications that we made for our proof.

Injective Trapdoor Functions. The oracle and proof require few changes to handle trapdoor functions that are 1-1 but not necessarily onto. The modifications are as follows:

- Step 2 in the description of Break is modified to verify that (G, E, D) is a valid injective trapdoor function instead of checking that it is a trapdoor permutation. The rest of the oracle stays as before.
- The main issue that arises from changing from permutations to injective functions is the concern that the adversary may design a family of injective trapdoor functions that give away too much information when the inversion algorithm is applied to a string that is not in the range of the function. However, in that case check 5 of both Break and Break$'$ (which is described in the proof) will always fail since both permutations are evaluated in the forward direction on the inverse obtained in step 4.

Weaker Correlation. Our result easily generalizes to obtain the following stronger theorem: for every $n, k \in \mathbb{N}$, and every distribution $\mathcal{C}$ on elements of $(\{0,1\}^n)^k$ such that, with high probability each of the k coordinates can be found given all the remaining $k-1$ coordinates of the sample, there is no black-box construction of a trapdoor permutation that is correlation secure under $\mathcal{C}$ from a one-way trapdoor permutation. On a very high level, our Break oracle can be generalized to break such constructions due to the following simple fact. In the simulation of Break$'$ by an adversary that has access only to (g, e, d) (and not to Break$'$, the simulator is unable to invert only one of the strings y_1, y_2. The simulator can easily be extended to invert all the strings $y_1, \ldots, y_k$ except one. More details are given in [19].

Families of Trapdoor Permutations That Use Break. Our adversary in Section 4 breaks only constructions of trapdoor permutations that only make use of the (g, e, d) part of the oracle, and never query Break. Similarly, our simulator in Section 5 only simulates Break queries that do not make recursive Break queries. We chose to describe the proof in this manner to simplify the presentation. Both Theorem 1 and Theorem 2 extend to the case where G, E, D are allowed to make Break queries.

One modification is to the cost of a Break query. When Break may make recursive queries to itself, a single Break query by the adversary counts as the sum of the costs of all the Break queries in the resulting recursion tree. A second modification is to the adversary that uses Break. The modified adversary keeps track of Break queries and answers that appear during the simulations of G and E in steps 2 and 3. Then, in step 4, she chooses the partial oracles $\mathcal{O}'_1$ and $\mathcal{O}'_2$ to be consistent with the previously observed queries and answers to Break.

The main property of Break that allows us to handle such constructions is that in every call to Break, only one of the values y_i may require a trapdoor for

some pub^* which happens to be the public key that our simulator is trying to break. Hence, to extend the simulator of Section 5 to handle constructions that make use of Break, the simulator is modified to recursively simulate Break by running our simulator for Break′ for each recursive call.

Adding a PSPACE Oracle. In our proofs the only measure of complexity for algorithms is the number of (g, e, d) queries that they make. This can be interpreted intuitively as ruling out a certain type of reductions between the two primitives in question. However, we are interested in showing that there is an oracle relative to which there exists a secure trapdoor permutation, and yet there exists a polytime adversary that breaks the correlation security of every construction. This is achieved by adding a **PSPACE** oracle. Then, step 4 of our adversary can be implemented in a single step by making a query to the **PSPACE** oracle. The rest of the computation that is performed by the adversary, and by the simulator is done in polynomial time. To complete the proof it is necessary to observe that a random trapdoor permutation remains secure, even when the adversary has access to a **PSPACE** oracle. For more details about the technique of adding a **PSPACE** oracle we direct the reader to [16] and [18].

References

1. Bellare, M., Fischlin, M., O'Neill, A., Ristenpart, T.: Deterministic encryption: Definitional equivalences and constructions without random oracles. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 360–378. Springer, Heidelberg (2008)
2. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007)
3. Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM J. Comput. 36(5), 1301–1328 (2006)
4. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
5. Boneh, D., Papakonstantinou, P., Rackoff, C., Vahlis, Y., Waters, B.: On the impossibility of basing identity based encryption on trapdoor permutations. In: FOCS 2008: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 283–292. IEEE Computer Society, Los Alamitos (2008)
6. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: STOC 2000: Proceedings of the thirty-second annual ACM symposium on Theory of computing, pp. 235–244. ACM, New York (2000)
7. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)
8. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography. In: STOC 1991: Proceedings of the twenty-third annual ACM symposium on Theory of computing, pp. 542–552. ACM, New York (1991)
9. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM Review 45(4), 727–784 (2003)

10. Gennaro, R., Gertner, Y., Katz, J., Trevisan, L.: Bounds on the efficiency of generic cryptographic constructions. SIAM J. Comput. 35(1), 217–246 (2005)
11. Gertner, Y., Kannan, S., Malkin, T., Reingold, O., Viswanathan, M.: The relationship between public key encryption and oblivious transfer. In: FOCS 2000: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Washington, DC, USA, p. 325. IEEE Computer Society, Los Alamitos (2000)
12. Gertner, Y., Malkin, T., Reingold, O.: On the impossibility of basing trapdoor functions on trapdoor predicates. In: FOCS 2001: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science, Washington, DC, USA, p. 126. IEEE Computer Society, Los Alamitos (2001)
13. Gertner, Y., Malkin, T., Myers, S.: Towards a separation of semantic and CCA security for public key encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 434–455. Springer, Heidelberg (2007)
14. Gertner, Y., Malkin, T., Myers, S.: Towards a separation of semantic and CCA security for public key encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 434–455. Springer, Heidelberg (2007)
15. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: STOC 1989: Proceedings of the twenty-first annual ACM symposium on Theory of computing, pp. 44–61. ACM, New York (1989)
16. Reingold, O., Trevisan, L., Vadhan, S.P.: Notions of reducibility between cryptographic primitives. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 1–20. Springer, Heidelberg (2004)
17. Rosen, A., Segev, G.: Chosen-ciphertext security via correlated products. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 419–436. Springer, Heidelberg (2009)
18. Simon, D.R.: Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 334–345. Springer, Heidelberg (1998)
19. Vahlis, Y.: Two is a crowd? a black-box separation of one-wayness and security under correlated inputs (2009), http://www.cs.toronto.edu/~evahlis