Innovations in Computer Science 2011

Leaky Pseudo-Entropy Functions

Mark Braverman[1,3] Avinatan Hassidim[2,3] Yael Tauman Kalai[4]
[1]University of Toronto, Toronto, Canada
[2] MIT, Cambridge USA
[3] Part of this work was done while visiting Microsoft Research, Cambridge USA
[4]Microsoft Research, Cambridge, USA
mbraverm@cs.toronto.edu avinatan@mit.edu yael@microsoft.com

Abstract: Pseudo-random functions (PRFs) introduced by Goldwasser, Goldreich, and Micali (FOCS 1984), are one of the most important building blocks in cryptography. A PRF family is a family of seeded functions $\{f_s\}$, with the property that no efficient adversary can tell the difference between getting oracle access to a random PRF function f_s, and getting oracle access to a truly random function.
In this work, we consider the problem of constructing pseudo-random functions that are resilient to leakage. Unfortunately, even if a single bit about the secret seed $s \in \{0,1\}^k$ is leaked, then there is no hope to construct a PRF, since the leakage can simply be the first bit of $f_s(0)$, and thus $f_s(0)$ is distinguishable from uniform. Therefore, when dealing with leakage, we must relax the definition.
We consider the following relaxation: Instead of requiring that for each input x, the value $f_s(x)$ looks *random*, we require that it looks like it has high *min-entropy*, even given oracle access to f_s everywhere except point x. We call such a function family a *pseudo-entropy function* (PEF) family. In particular, a leakage-resilient PEF family has the property that given leakage $L(s)$ and given oracle access to f_s, it is hard to predict f_s on any input that was not queried. We construct such a leakage-resilient PEF family under the DDH assumption (or more generally, assuming the existence of lossy functions with the property that the output size is not much larger than the input size).
We also show that leakage-resilient PEFs imply leakage-resilient *random-input* PRFs, where the requirement is that for a *random* input r, the value $f_s(r)$ looks uniform, even given the leakage $L(s)$ and given oracle access to f_s anywhere accept at point r (the leakage $L(s)$ is independent of r, but the oracle f_s is present even after the pair $(r, f_s(r))$ is given).

Keywords: pseudo random function, leakage, cryptography.

1 Introduction

Pseudo-random functions, defined by Goldwasswer, Goldreich, and Micali [18] are one of the fundamental building blocks in cryptography. Such functions are efficiently generated using a *short* secret random seed s, and yet look truly random, in the sense that oracle access to a pseudo-random function f_s is computationally indistinguishable from oracle access to a truly random function.

In this work, we investigate the following question: *What if we don't even have a short random seed, but all we have is some secret seed with min-entropy?* Alternatively: *What if an arbitrary poly-time function of the secret seed is leaked (such that some min-entropy is left)?* It is easy to see that in such cases it is impossible to generate even one bit of randomness.

Instead, the question we ask is: *Can we generate entropy?*

One of the motivations for studying this problem comes from the recent proliferation of side channel attacks, where attackers try to use physical means to get information about the secret keys. These attacks exploit the physical characteristics of the execution of a cryptographic device, such as timing, power consumption, electro-magnetic radiation, and so forth (see [21,27,28,31] and the references therein). Thus, in recent years there has been a major effort by the cryptographic community to construct cryptographic schemes that remain secure even if part of the secret key is leaked. This is in contrast to the traditional approach, which assumes that secret keys are gener-

ated using perfectly random bits, and once they are generated, they are perfectly secret.

By now, there have been many constructions of cryptographic primitives that are resilient to leakage [1-4,6,7,10,11,14-16,19,20,23-26,29,30,33,35]. In this work, we continue the study of leakage-resilient cryptography, where we focus on constructing pseudorandom functions that are resilient to leakage. However, what we find intriguing about this work, and what conceptually distinguishes this work from previous ones, is that we don't have any additional randomness beyond the seed (which is partially leaked). In all other results in the regime of leakage-resilient cryptography, fresh randomness is used either to extract randomness from the weak key (e.g., in the encryption process [1]), or to hide the secret key (e.g., in the signature process [26]). Alternatively, many results restrict the type of leakage,[1] in a way that allows the use of (deterministic) extractors [7,14,29]. Here, on the other hand, after the seed is partially leaked, we have no fresh randomness in our hands, and the only assumption we can make about our secret seed, is that it has some min-entropy. Thus, we can only hope to generate (computational) min-entropy, as opposed to (computational) randomness.

1.1 Our results

We construct a family of functions $\mathcal{F} = \{f_s\}$ such that given oracle access to a random function in the family $f_s \in \mathcal{F}$, and given an arbitrary (poly-time) leakage $L(s)$, such that the seed s still has (enough) min-entropy conditioned on $L(s)$, the value of the function at any point that was not queried has (computational) min-entropy. We call such function family a *leakage-resilient pseudo-entropy function (PEF)* family.

We consider two definitions for PEFs: A *selective* one, which requires that for any *a priori* selected input x, the value of $f_s(x)$ has (computational) min-entropy given the leakage $L(s)$ and oracle access to f_s at any point except x. The other definition is an *adaptive* one, which requires that for any input x, which may be chosen after seeing the leakage $L(s)$ and after accessing the oracle f_s, the value of $f_s(x)$ has (computational) min-entropy given the leakage $L(s)$ and oracle access to f_s at any point except x. We note that in both definitions we allow the leakage function to be *adaptive*. More specifically, the adversary may choose to leak one bit at a time in an adaptive manner, while making oracle calls to f_s in between leakages. We refer the reader to Section 3 for formal definitions.

We construct a selectively secure PEF family under the DDH assumption.[2] Then, we show that under the sub-exponential hardness of the DDH assumption,[3] our PEF family is also adaptively secure. We refer the reader to Section 4 for details about our construction.

We note that any (adaptively secure) PEF family $\mathcal{F} = \{f_s\}$ has the property that even given oracle access to a random f_s and given leakage $L(s)$, it is (computationally) hard to predict the value of the function at any point that was not queried. Thus, a leakage resilient PEF family is, in particular, a deterministic leakage-resilient message authentication code (MAC), where two players who share a secret s authenticate a message m by appending to it the value $f_s(m)$.

Deterministic MACs are interesting in the context of leakage, since they are not only resilient to leakage from the secret key, but they are also (trivially) resilient to leakage from the secret randomness used during the authentication process (as this process is deterministic, and thus no randomness is used). Most known leakage resilient MACs or leakage-resilient signature schemes cannot tolerate leakage from the secret randomness used during the authentication or signing process, and the leakage function is only a function of the secret key. An exception is the very recent (and independent) work of Boyle, Segev and Wichs [5], which constructs a signature scheme that is resilient to leakage from both the secret key and the randomness used during the signing process (assuming the *total* leakage from all the randomness used in all signatures, is bounded).[4]

We note that for some applications, generating entropy (or unpredictability) is not enough, and generating randomness is crucial. Motivated by such applications, we show how to construct a leakage-resilient *random-input PRF* family from any (selectively se-

[1] For example, they rely on the "*only computation leaks information*" assumption, introduced in the influential work of Micali and Reyzin [29].

[2] More generally, we assume the existence of a lossy function family with the property that the output size is not much larger than the input size. In Appendix B we construct such a family based on the DDH assumption.

[3] Namely, we assume that there exists a constant $\epsilon > 0$ such that no PPT adversary can distinguish between a DDH tuple and a random tuple with probability greater than $2^{-\lambda^\epsilon}$, where $\lambda \in \mathbb{N}$ is the security parameter. See Theorem 4.2.

[4] Previously, Brakerski *et. al.* [6] (among other things) proved a similar result in the *random oracle model*.

cure) leakage-resilient PEF family. The former is a family of functions $\mathcal{F}$ with the property that given an arbitrary (poly-time) leakage $L(s)$, such that s still has (enough) min-entropy conditioned on $L(s)$, for a random point r, the pair $(r, f_s(r))$ looks random even given oracle access to f_s anywhere except at point r (where the adversary may access the oracle even after the pair $(r, f_s(r))$ is given). It is the randomness of the input r that allows us to argue the (computational) randomness of $f_s(r)$, even conditioned on the leakage $L(s)$ and conditioned on the information given by the oracle.

We note that a random-input PRF is weaker than a (standard) PRF, but is stronger than a weak PRF, where the requirement is that $(r, f_s(r))$ looks random even given many pairs of the form $(r_i, f_s(r_i))$, where the r_i's are uniformly and independently distributed. Namely, for weak PRFs the adversary is not given oracle access to the function, but is given the evaluation of the function on a bunch of random inputs. It is known how to construct a leakage-resilient *weak* PRF family, but it was not known how to construct a leakage-resilient *random-input* PRF family.

1.2 Background and related work

Several leakage models were considered in the literature. Roughly speaking, one can partition these leakage models into two categories: one-time (or bounded) leakage, and continual leakage. The bounded leakage models, which our leakage model resides in, consider the case where leakage occurs only once, whereas continual leakage models consider the case where leakage occurs over and over again.

Bounded leakage models. These models consider the case that an adversary gets some leakage $L(s)$ of the secret s, and the requirement is that the scheme remains secure even given this leakage. Clearly, in order to get meaningful results, we must somehow bound the leakage function, since otherwise it can leak the entire secret, i.e., $L(s) = s$, and security is clearly breached.

Akavia, Goldwasser, and Vaikuntanathan [1] considered the class of *shrinking* leakage functions. Namely, their requirement is that $|L(s)| < |s|$. Naor and Segev [30] considered a slightly more general class of leakage functions which consists of all the leakage functions L such that $L(s)$ must leave s with some min-entropy. Our results hold in the Naor-Segev leakage model.

Other, more general, leakage models are the auxiliary input model of Dodis *et. al.* [11], and the bounded retrieval model of [8,13]. However, due to the deterministic nature of a PRF family, it is impossible to construct a leakage resilient PEF family in these models. We refer the reader to Appendix A for more details. It is known how to construct encryption schemes, identity-based encryption schemes, signature schemes, weak PRF schemes, and more, in various of these leakage models [1-3,9,11,19,26,30].

Continual leakage models. Roughly speaking, these models can again be partitioned into two categories: The first considers memory leakage, where each leakage function is a function of the entire memory (as in the bounded leakage models), and the second considers the *"only computational leaks information"* model of Micali and Reyzin [29], where each leakage function can depend only on the part of the secret that was used during the computation. Note that the former model is a generalization of the bounded leakage models. The latter model, on the other hand, is not a generalization of the bounded leakage models, and is incomparable. On the one hand, leakage can occur over and over again, but on the other hand, the leakage functions are restrictive to the part of the secret used during the computation, and thus secret randomness that was not used during the computation remains truly random and hidden. We note that until very recently, all schemes secure against continual leakage considered the "only computation leaks information" model, where it was shown how to construct signature schemes and block ciphers [14,15,33]. Using (simple) secure hardware, it was shown how to convert any circuit into a leakage resilient one [16,20,25]. Very recently, various cryptographic schemes that are secure in the continual memory leakage model were presented, including encryption schemes, identity-based encryption schemes, and signature schemes [6,10].

We note that constructing PEFs or even weak PRFs in a continual leakage model remains an interesting open problem. Recently, Dodis and Pietrzak [12] made some progress in this direction by constructing PRFs that are resilient to continual leakage in the Micali-Reyzin model of "only computation leaks information". However, their result is restricted to a *single* a priori chosen leakage function.

1.3 Our techniques

We start by giving an overview of our construction of a leakage resilient PEF family. Our construction fol-

lows the GGM paradigm [18]. However, rather than using a pseudo-random generator as the main building block (as in [18]), we use a family of lossy functions as our main building block, a notion formulated by Peikert and Waters [32][5] (yet appeared implicitly in prior work). Such a family consists of two types of functions: lossy functions and injective ones. The lossy ones (information theoretically) lose most of the information about the input; i.e., the image is significantly smaller than the domain. The injective functions, on the other hand, are injective. It is required that it is (computationally) hard to distinguish between a random lossy function in the family and a random injective function in the family. We refer the reader to Section 2.1 for the formal definition.

Let $\mathcal{F}$ be a lossy function family. Suppose for now, that we have a common reference string (CRS) which consists of $2k$ pairs of random injective functions from $\mathcal{F}$, denoted by

$$\mathsf{crs} = \begin{pmatrix} f_{1,0}, f_{2,0}, \ldots, f_{k,0} \\ f_{1,1}, f_{2,1}, \ldots, f_{k,1} \end{pmatrix}$$

(We will get rid of this assumption, by taking the crs to be part of the seed of the PEF function).

We construct a PEF function family $\mathcal{G}$, where each function in $\mathcal{G}$ is associated with a secret seed $s \in \{0,1\}^n$. For input $x = (x_1, \ldots, x_k) \in \{0,1\}^k$, define

$$g_s(x) = f_{1,x_1} \circ f_{2,x_2} \circ \ldots \circ f_{k,x_k}(s).$$

Namely, the value $g_s(x)$ is computed by first applying the function f_{k,x_k} on input s; then applying the function $f_{k-1,x_{k-1}}$ to the value $f_{k,x_k}(s)$; then applying the function $f_{k-2,x_{k-2}}$ to the value $f_{k-1,x_{k-1}}(f_{k,x_k}(s))$, and so on. Note that for g_s to be computable in polynomial time, we must require that the underlying lossy functions are not expanding by too much, as this would incur an exponential blowup. Indeed, in Appendix B, we rely on the DDH assumption to construct a lossy function family, where both the domain and the range are $\{0,1\}^n$.

Our family $\mathcal{G}$ has the property that if one of the functions $f_{1,x_1}, \ldots, f_{k,x_k}$ is lossy, then the value $g_s(x)$ (together with the crs) contains very little information about the secret seed s. On the other hand, if they are all injective then the value $g_s(x)$ (together with the crs) contains all the information about the seed s. In fact, we prove the stronger property that if all the functions $f_{1,x_1}, \ldots, f_{k,x_k}$ are injective, and all the rest are lossy, then the value $g_s(x)$ (together with the crs) contains all the information about s, and yet *all* the values $\{g_s(x')\}_{x' \neq x}$ together contain very little information about s; namely, using very little information about s, one can compute g_s everywhere except at point x.

Using this property, together with the property that it is hard to distinguish between random lossy functions and random injective functions in $\mathcal{F}$, we prove that $\mathcal{G}$ is a leakage resilient PEF family with selective security.[6] To this end, for an a priori chosen $x \in \{0,1\}^k$, we first change the functions $f_{1,1-x_1}, \ldots, f_{k,1-x_k}$ to be random lossy functions (rather than injective ones), and claim that no PPT adversary can tell the difference. Then, we claim that the amount of information needed to compute g_s everywhere except at point x, together with the crs and the leakage $L(\mathsf{crs}, s)$, is quite small. Hence, $g_s(x)$, which contains all the information about s, has high min-entropy.

We now formalize this intuition. For any given x, an oracle to g_s anywhere except point x, can be simulated given only the crs and the following values:

$$\begin{aligned} y_k &\triangleq f_{k,1-x_k}(s) \\ y_{k-1} &\triangleq f_{k-1,1-x_{k-1}}(f_{k,x_k}(s)) \\ &\vdots \\ y_1 &\triangleq f_{1,1-x_1}(f_{2,x_2}(\ldots f_{k,x_k}(s))) \end{aligned}$$

This is the case, since for any $x' \neq x$, let $i \in [k]$ be the largest index for which $x'_i \neq x_i$. Then one can efficiently compute $g_s(x')$ from y_i and crs.

Note that if the k functions $f_{1,x_1}, \ldots, f_{k,x_k}$ are injective and the other k functions $f_{1,1-x_1}, \ldots, f_{k,1-x_k}$ are lossy, in the sense that they lose all information about the input except at most n^ϵ bits, then $y_1, \ldots, y_k$ contains only $k \cdot n^\epsilon$ bits of information about s (think of $k \cdot n^\epsilon$ as significantly smaller than n).

More formally, we prove that $\mathcal{G}$ is a leakage-resilient PEF family by arguing that if there exists a PPT adversary $\mathcal{A}$ that distinguishes $g_s(x)$ from a random variable with high min-entropy, given the leakage and oracle access to g_s everywhere except x, then there exists a PPT adversary $\mathcal{B}$ who distinguishes between lossy functions and injective ones. The adversary $\mathcal{B}$ is given k functions that are either injective or lossy. He will use these functions instead of $f_{1,1-x_1}, \ldots, f_{k,1-x_k}$, and

[5] Their formulation included an additional trapdoor requirement.

[6] Later in this section, we mention how to go from selective security to adaptive security.

will try to use $\mathcal{A}$ to distinguish between $g_s(x)$ and a random variable with high min-entropy. Then if the functions were lossy, then $g_s(x)$ indeed has high min-entropy, and thus $\mathcal{A}$ will fail to distinguish, whereas if these functions are injective then he will succeed with non-negligible probability.

All this holds assuming the definition of a PEF is a selective one, in the sense that for any *a priori chosen* x, the value $g_s(x)$ is computationally indistinguishable from having high min-entropy, even given crs, the leakage $L(\mathsf{crs}, s)$ and oracle access to g_s (everywhere except at point x). In order to get *adaptive* security, where the adversary can choose x after seeing the leakage and the crs, we rely on subexponential hardness assumptions, and simply guess x in advance. Note that $x \in \{0,1\}^k$, and we can take $k = n^\delta$ for an arbitrary small constant $\delta > 0$. Thus, we guess x correctly with probability 2^{-n^δ}, and hence rely on the assumption is that it is hard to distinguish a random lossy function from a random injective function, with probability 2^{-n^δ}. We refer the reader to Section 4 for details.

Finally, we mention how we go from a (selectively secure) leakage-resilient PEF family $\mathcal{G}$ to a leakage-resilient random-input PRF family $\mathcal{G}^*$. Recall that for a random-input PRF family, the requirement is that for a *random* input r the value $g^*_s(r)$ looks random, even given oracle access to g^*_s everywhere except point r.

The idea is simply to apply a randomness extractor to the output $g_s(x)$. Unfortunately, we do not have deterministic randomness extractors, and to extract randomness from $g_s(x)$ we need fresh randomness. The question is: where do we get this randomness from? One idea is to take it from the input. Namely, define

$$g^*_s(x, r) = \mathsf{Ext}(g_s(x); r),$$

where Ext is a seeded extractor, and r is the random seed for the extractor. The problem with this solution is that the resulting g^*_s is not a random-input PRF, since given a random input (x, r) the adversary is allowed to query the oracle at points (x, r') for any $r' \neq r$, and thus can retrieve $g_s(x)$. Thus, instead, we use a collision-resistant hash function h, and define:

$$g^*_s(x) = \mathsf{Ext}\left(g_s(h(x)); x\right).$$

Note, however, that the seed x is not independent of the source $g_s(h(x))$. Therefore, instead of using a seeded extractor, we let Ext be a 2-source extractor, that takes as input two independent weak sources, where one has min-entropy grater than $1/2$. The 2-source extractor of Raz [34] has this property. The idea would be to think of $h(x)$ as fixed, and thus the sources $g_s(h(x))$ and x are independent. Moreover, x has min-entropy rate greater than $1/2$ even conditioned on $h(x)$, and $g_s(h(s))$ has (computational) min-entropy, even given $h(x)$ and given the crs and the leakage $L(\mathsf{crs}, s)$, since it is a PEF. We refer the reader to Section 5 for details.

2 Preliminaries and definitions

The min-entropy of a random variable $X \in \{0,1\}^n$ is defined by

$$H_\infty(X) \triangleq -\log\left(\max_{x\in\{0,1\}^n} \Pr[X = x]\right).$$

The conditional min-entropy of the random variable $X \in \{0,1\}^n$ conditioned on a random variable $Y \in \{0,1\}^k$ is defined by

$$\tilde{H}_\infty(X|Y) \triangleq -\log \mathbb{E}_{y\leftarrow Y}\left[2^{-H_\infty(X|Y=y)}\right]. \quad (1)$$

We say that two sequences of random variables $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable, if for any PPT algorithm $\mathcal{A}$, for any constant $c \in \mathbb{N}$, and for every sufficiently large $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{A}(X_\lambda) = 1] - \Pr[\mathcal{A}(Y_\lambda) = 1]| < 1/\lambda^c$$

This is denoted by $X \approx Y$.

In what follows, we recall the definitions of a lossy function family and a pseudo-random function family. We denote the security parameter by λ. We denote the domain of the lossy functions by $\{0,1\}^n$ and we denote the domain of the pseudo-random functions by $\{0,1\}^k$, where both n and k are functions of the security parameter λ. In our results, k will be significantly smaller than n, though both n and k will be polynomially related to λ.

2.1 Lossy functions

Lossy trapdoor functions were introduced in the seminal paper of Peikert and Waters [32]. Such function families have two types of functions: lossy functions and injective functions, where lossy functions have the property that the output contains very little information about the input (information theoretically), and where the injective ones are generated with a trapdoor for inversion. It is required that it is hard

to distinguish between a random lossy function and a random injective function in the family. In this work, we do not need the trapdoor property, and only consider families of lossy functions.

Definition 2.1. *A function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ is an* **ω-lossy family** *if the following holds:*

1. *There are two seed generation algorithms $\mathsf{Gen}_{\text{loss}}$ and $\mathsf{Gen}_{\text{inj}}$ such that*

(a) $\{\mathsf{Gen}_{\text{loss}}(1^\lambda)\}_{\lambda\in\mathbb{N}} \approx \{\mathsf{Gen}_{\text{inj}}(1^\lambda)\}_{\lambda\in\mathbb{N}}$

(b) *For every $\lambda \in \mathbb{N}$ and for every $s \in \mathsf{Gen}_{\text{inj}}(1^\lambda)$, the function $f_s : \{0,1\}^n \to \{0,1\}^*$ is injective.*

(c) *For every $\lambda \in \mathbb{N}$ and for every $s \in \mathsf{Gen}_{\text{loss}}(1^\lambda)$, the function $f_s : \{0,1\}^n \to \{0,1\}^*$ is lossy, in the sense that its image size is at most $2^{n-\omega}$.*[7]

2. *There is a poly-time evaluation algorithm Eval such that for every $\lambda \in \mathbb{N}$, every $s \in \mathsf{Gen}_{\text{loss}}(1^\lambda) \cup \mathsf{Gen}_{\text{inj}}(1^\lambda)$, and every $x \in \{0,1\}^n$,*

$$\mathsf{Eval}(s, x) = f_s(x).$$

In this work, we need the lossy family $\mathcal{F}$ to have the property that for every $f_s \in \mathcal{F}$, the output length is almost the same as the input length. Indeed, we assume for the sake of simplicity of exposition, that our lossy function family has the property that the domain and the range are the same.

We also consider an additional property, which we call the *extraction* property, and is defined below. This property is not needed to obtain our main results, and is only used to show that our constructions satisfy the standard PRF definition (see Section 4 for more details).

Definition 2.2. *A lossy function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ is a (strong)* **α-computational extractor** *if for every sequence of random variables $\{X_{n(\lambda)}\}_{\lambda\in\mathbb{N}}$ with min-entropy $\alpha(n)$,*

$$\{s, f_s(X_{n(\lambda)})\}_{s\leftarrow\mathsf{Gen}_{\text{loss}}(1^\lambda)} \approx \{s, U_{n(\lambda)}\}_{s\leftarrow\mathsf{Gen}_{\text{loss}}(1^\lambda)}$$

2.2 Pseudo-random functions

We recall the standard definition of a pseudo-random function family (PRF).

[7] According to the definition of conditional min-entropy (See Equation 1), this implies that for every $s \in \mathsf{Gen}_{\text{loss}}(1^\lambda)$, $\tilde{H}_\infty(X|s, f_s(X)) \geq n - \omega$, where $X \leftarrow \{0,1\}^n$.

Definition 2.3. *A function family $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$ is said to be* **pseudo-random** *if the following holds:*

1. *The exists a seed generation* PPT *algorithm* Gen *that takes as input a security parameter 1^λ, and generates a seed $s \leftarrow \mathsf{Gen}(1^\lambda)$.*

2. *There exists a poly-time evaluation algorithm* Eval *such that for every $\lambda \in \mathbb{N}$, every $s \in \mathsf{Gen}(1^\lambda)$, and every $x \in \{0,1\}^k$, it holds that $\mathsf{Eval}(s, x) = g_s(x)$.*

3. *For every* PPT *adversary $\mathcal{A}$,*

$$\left|\Pr_{s\leftarrow\mathsf{Gen}(1^\lambda)}[\mathcal{A}^{g_s}(1^\lambda) = 1] - \Pr[\mathcal{A}^O(1^\lambda) = 1]\right| = \text{negl}(\lambda)$$

where O is a truly random function with the same domain and range as g_s.

3 Pseudo-entropy functions (PEFs)

Our goal in this paper is to construct a PRF family that is robust to leakage of the secret seed. Clearly, this is impossible since the leakage can contain the least-significant-bit of $g_s(0)$. In such a case, it is easy to distinguish $g_s(0)$ from uniform. Thus, when dealing with leakage, we must relax the definition of a PRF.

We consider the following relaxed definition: Rather than requiring that for each input x, the value $g_s(x)$ is indistinguishable from being *truly random*, we require that $g_s(x)$ is indistinguishable from having high min-entropy, even given oracle access to g_s everywhere except point x. We call such a family a *pseudo-entropy function* (PEF) family. We consider two definitions: The first is *selective security*, which requires that the input x should be selected by the adversary *before* getting oracle access to g_s and before seeing the leakage. The second is *adaptive security* where the input x can be chosen by the adversary after getting the leakage and oracle access to g_s.

Definition 3.1. *A function family $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$ is said to be an ℓ-leaky β-pseudo-entropy function (β-PEF) family with* **selective security** *if the following holds:*

1. Gen *is a* PPT *algorithm that takes as input the security parameter 1^λ, and generates a pair $(pp, s) \leftarrow \mathsf{Gen}(1^\lambda)$, where pp is the public part of the seed (which does not need to be kept secret) and s is the secret seed.*

2. *$\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$ is a function family, where each*

$g \in \mathcal{G}_\lambda$ is parameterized by public parameters pp and a secret seed $s \in \{0,1\}^n$. For the sake of clarity, we denote the function by g_s, and omit the dependence on pp from the notation.

3. *There exists a modified* PPT *algorithm* Gen' *such that for any sequence of inputs $x = x(\lambda) \in \{0,1\}^{k(\lambda)}$, the following holds:*

(a) $\left\{\mathsf{Gen}'(1^\lambda, x)\right\}_{\lambda\in\mathbb{N}} \approx \left\{\mathsf{Gen}(1^\lambda)\right\}_{\lambda\in\mathbb{N}}$

(b) *For any poly-time computable leakage function L with output length $\ell(n)$,*

$$\tilde{H}_\infty\left(g_s(x)|, \ldots, , L(pp, s), \{g_s(x')\}_{x'\neq x}\right) \geq \beta(n),$$

where $(pp, s) \leftarrow \mathsf{Gen}'(1^\lambda, x)$, and $n = |s|$.

Definition 3.2. *A function family $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$ is said to be an ℓ-leaky β-PEF family with* **adaptive security** *if properties (1) and (2) from above hold, and property (3) is strengthened as follows:*

There exists a modified PPT *algorithm* Gen' *such that*

$$\left\{\mathsf{Gen}'(1^\lambda)\right\}_{\lambda\in\mathbb{N}} \approx \left\{\mathsf{Gen}(1^\lambda)\right\}_{\lambda\in\mathbb{N}}$$

and for any PPT *adversary $\mathcal{A}$, for every poly-time leakage function L with output size $\ell(n)$, and for*

$$x = \mathcal{A}^{g_s}(pp, L(pp, s))$$

that was not sent by $\mathcal{A}$ as an oracle query,

$$\tilde{H}_\infty\left(g_s(x)|, \ldots, , L(pp, s)\{g_s(x')\}_{x'\neq x}\right) \geq \beta(n),$$

where $(pp, s) \leftarrow \mathsf{Gen}'(1^\lambda)$, and $n = |s|$.

4 A construction of a PEF family

In this section we construct a leaky PEF family $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$. We first construct such a family with selective security, and then show how to obtain adaptive security from any selective secure family by strengthening the assumptions. We note that the latter follows using a standard black-box reduction, and does not contain new ideas (but does require strengthening the assumptions).

In our construction, each secret seed is of length n, the inputs are of length k, and the outputs are of length n, where n and k are polynomially related to the security parameter λ, and $k = n^\delta$ for some small constant $\delta \in (0,1)$, and thus is significantly smaller than n.

Our construction makes use of an ω-lossy function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$. Each function in $\mathcal{F}_\lambda$ has inputs and outputs of length n. We set the parameter ω to be: $\omega = n - n^\epsilon$ for some constant $0 < \epsilon < 1 - \delta$, and we get an ℓ-leaky β-PEF, where $\ell = n - 2n^{\delta+\epsilon}$ and $\beta = n^{\delta+\epsilon}$.

We also show that if the underlying lossy family $\mathcal{F}$ is a γ-computational extractor for $\gamma = n - n^{\delta+\epsilon}$, then our function family $\mathcal{G}$ is a PRF family (in the standard sense).

4.1 The construction

- Gen. The seed generation algorithm takes as input a security parameter 1^λ, and generates as public parameters $2k$ random functions using $\mathsf{Gen}_{\text{inj}}(1^\lambda)$,[8]

$$pp = \begin{pmatrix} f_{1,0}, f_{2,0}, \ldots, f_{k,0} \\ f_{1,1}, f_{2,1}, \ldots, f_{k,1} \end{pmatrix}$$

and a secret seed $s \leftarrow \{0,1\}^n$.

- Eval. The Eval function takes as input public parameters pp, a seed $s \in \{0,1\}^n$, and an input $x \in \{0,1\}^k$, where $x = (x_1, \ldots, x_k)$, and outputs

$$g_s(x) = f_{1,x_1} \circ f_{2,x_2} \circ \ldots \circ f_{k,x_k}(s)$$

Notice that for the definition of g_s to make sense, we need the output of f_{i,x_i} to be a valid input to $f_{i-1,x_{i-1}}$. This is why we need the restriction that the output size of f_{i,x_i} is not significantly larger than its input size, since otherwise this may result in an exponential blowup. For the sake of simplicity, we assume that $f_{i,x_i} : \{0,1\}^n \to \{0,1\}^n$.

Theorem 4.1. *The function family $\mathcal{G} = \{\mathcal{G}_\lambda\}$ described above is an ℓ-leaky β-PEF family with* selective *security, assuming the underlying lossy function family $\mathcal{F} = \{\mathcal{F}_\lambda\}$ is ω-lossy, where*

$$\omega = n - n^\epsilon,\ k = n^\delta,\ \ell = n - 2n^{\delta+\epsilon},\ \beta = n^{\delta+\epsilon}.$$

Theorem 4.2 *The function family $\mathcal{G} = \{\mathcal{G}_\lambda\}$ described above is an ℓ-leaky β-PEF family with* adaptive *security, assuming the underlying lossy function family $\mathcal{F} = \{\mathcal{F}_\lambda\}$ is ω-lossy, with the strengthened requirement that no* PPT *adversary can distinguish between a*

[8] Alternatively, we could have generated the functions using $\mathsf{Gen}_{\text{loss}}(1^k)$. These two cases are computationally indistinguishable.

random lossy seed and a random injective seed with probability $2^{-n^\delta}/\mathrm{poly}(\lambda)$*, where*

$$\omega = n - n^\epsilon,\ k = n^\delta,\ \ell = n - 2n^{\delta+\epsilon},\ \beta = n^{\delta+\epsilon}.$$

We also prove that $\mathcal{G} = \{\mathcal{G}_\lambda\}$ is a (standard) PRF family.

Lemma 4.3. *The function family* $\mathcal{G} = \{\mathcal{G}_\lambda\}$ *described above is a PRF family, assuming the underlying lossy function family* $\mathcal{F} = \{\mathcal{F}_\lambda\}$ *is* ω*-lossy and is a* γ*-computational extractor, where*

$$\omega = n - n^\epsilon,\ \gamma = n - n^{\delta+\epsilon},\ k = n^\delta.$$

In what follows we prove Lemma 4.3, Theorem 4.1 and Theorem 4.2. We refer the reader to Section 1.3 for a high-level overview of the proofs.

1) Proof of Lemma 4.3.

It is easy to see that properties (1) and (2) of definition 2.3 are satisfied. As for property (3), we first show that for every PPT adversary $\mathcal{A}$ and for every sequence of inputs $x = x(\lambda) \in \{0,1\}^{k(\lambda)}$,

$$|\Pr[\mathcal{A}^{g_s}(1^\lambda, pp, x, g_s(x)) = 1] - \Pr[\mathcal{A}^{g_s}(1^\lambda, pp, x, U_n) = 1]| = \mathrm{negl}(\lambda)$$

assuming $\mathcal{A}$ does not query its oracle on input x, and where the probabilities are over $(pp, s) \leftarrow \mathsf{Gen}(1^\lambda)$, $U_n \leftarrow \{0,1\}^n$, and the random coin tosses of $\mathcal{A}$.

To prove this, suppose for the sake of contradiction that there exists a PPT adversary $\mathcal{A}$, a sequence of inputs $x = x(\lambda) \in \{0,1\}^{k(\lambda)}$ which $\mathcal{A}$ does not send as oracle queries, and a polynomial p, such that for infinitely many λ's

$$|\Pr[\mathcal{A}^{g_s}(1^\lambda, pp, x, g_s(x)) = 1] - \Pr[\mathcal{A}^{g_s}(1^\lambda, pp, x, U_n) = 1]| \geqslant \frac{1}{p(\lambda)}$$

Consider a new distribution of the public parameters pp, denoted by pp', where the functions $f_{1,x_1}, \ldots, f_{k,x_k}$ are generated according to the injective seed generation algorithm $\mathsf{Gen}_{\mathrm{inj}}$, and the functions $f_{1,1-x_1}, \ldots, f_{k,1-x_k}$ are generated according to the lossy seed generation algorithm $\mathsf{Gen}_{\mathrm{loss}}$. Property (a) of the lossy function family implies that

$$\{x, pp\} \approx \{x, pp'\}.$$

Thus, for infinitely many λ's,

$$|\Pr[\mathcal{A}^{g_s}(1^\lambda, pp', x, g_s(x)) = 1] - \Pr[\mathcal{A}^{g_s}(1^\lambda, pp', x, U_n) = 1]| \geqslant \frac{1}{p(\lambda)} - \mathrm{negl}(\lambda)$$

Next, note that the oracle g_s can be simulated everywhere except at point x, given only pp' and given the sequence

$$\begin{aligned} y_k &\triangleq f_{k,1-x_k}(s) \\ y_{k-1} &\triangleq f_{k-1,1-x_{k-1}}(f_{k,x_k}(s)) \\ &\vdots \\ y_1 &\triangleq f_{1,1-x_1}(f_{2,x_2}(\ldots f_{k,x_k}(s))) \end{aligned}$$

This is the case, since for any $x' \neq x$, let $i \in [k]$ be the largest index for which $x'_i \neq x_i$. Then one can efficiently compute $g_s(x')$ given only y_i and pp'. This, together with the equation above, implies that

$$\{(y_1, \ldots, y_k), pp', g_s(x)\}_{\lambda\in\mathbb{N}, pp', s\leftarrow\mathsf{Gen}(1^\lambda)} \not\approx \quad (2)$$

$$\{(y_1, \ldots, y_k), pp', U_n\}_{\lambda\in\mathbb{N}, pp', s\leftarrow\mathsf{Gen}(1^\lambda)}$$

Recall that the functions $f_{1,1-x_1}, \ldots, f_{k,1-x_k}$ were generated according to the lossy seed generation algorithm $\mathsf{Gen}_{\mathrm{loss}}(1^\lambda)$. Thus, the string $(y_1, \ldots, y_k)$ contains only $k \cdot (n - \omega)$ bits of information about $s \in \{0,1\}^n$. Namely,

$$\begin{aligned} \tilde{H}_\infty(s | pp', (y_1, \ldots, y_k)) \geq n - k(n-\omega) &= n - kn^\epsilon \\ &= n - n^{\delta+\epsilon} \\ &= \gamma(n) \end{aligned}$$

Moreover, the string $(y_1, \ldots, y_k)$ is independent of the function f_{1,x_1}, and thus we can assume that this string was computed before the function f_{1,x_1} was chosen. Therefore, conditioned on all the functions in pp' except for f_{1,x_1}, and conditioned on $(y_1, \ldots, y_k)$, the string s is a random variable with min-entropy $\gamma(n)$, and is independent of f_{1,x_1}. Therefore, $f_{2,x_2}(f_{3,x_3}(\ldots(f_{k,x_k}(s))))$ is a random variable with min-entropy $\gamma(n)$ that is independent of f_{1,x_1}. The fact that $\mathcal{F}$ is a γ-computational extractor implies that $g_s(x) = f_{1,x_1}(f_{2,x_2}(\ldots(f_{k,x_k}(s))))$ is computationally indistinguishable from uniform, even given $(y_1, \ldots, y_k)$ and pp', contradicting Equation (2).

We conclude that for every PPT adversary $\mathcal{A}$ and for every sequence of inputs $x = x(\lambda) \in \{0,1\}^{k(\lambda)}$,

$$|\Pr[\mathcal{A}^{g_s}(1^\lambda, pp, x, g_s(x)) = 1] - \Pr[\mathcal{A}^{g_s}(1^\lambda, pp, x, U_n) = 1]| = \mathrm{negl}(\lambda) \quad (3)$$

assuming x is not an oracle query, where the probabilities are over $(pp, s) \leftarrow \mathsf{Gen}(1^\lambda)$, $U_n \leftarrow \{0,1\}^n$, and over the random coin tosses $\mathcal{A}$.

Now, to prove that $\mathcal{G}$ satisfies property (3) of a PRF family, suppose for the sake of contradiction that there

exists a PPT adversary $\mathcal{A}$ such that for infinitely many λ's,

$$\left| \Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}^{g_s}(1^\lambda, pp) = 1] - \Pr[\mathcal{A}^O(1^\lambda, pp) = 1] \right| \geqslant \frac{1}{\text{poly}(\lambda)}$$

Let Q be an upper-bound on the number of queries that $\mathcal{A}$ makes, and assume without loss of generality that all these queries are distinct. Let $g_s^{(i)}$ be a "hybrid" oracle that answers the first i queries using a truly random function, and the last $Q - i$ queries according to g_s. Note that $g_s^{(0)}$ is the same as g_s, and that the oracle $g_s^{(Q)}$ is the same as a random oracle, assuming the number of oracle queries is at most Q. Using a standard hybrid argument, the equation above implies that there exists $i \in [Q]$ such that

$$\left| \Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}^{g_s^{(i)}}(1^\lambda, pp) = 1] - \Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}^{g_s^{(i-1)}}(1^\lambda, pp) = 1] \right| \geqslant \frac{1}{(Q \cdot \text{poly}(\lambda))} \geqslant \frac{1}{\text{poly}(\lambda)}$$

where the latter inequality follows from the fact that Q is polynomially related to λ.

Denote by x the i'th oracle query of $\mathcal{A}$. Note that x is a random variable that is independent of g_s, since all the previous oracle queries were answered via the random oracle. Moreover, $\mathcal{A}$ can distinguish between $(x, g_s(x))$ and (x, U_n) given oracle access to g_s (without querying on x). Thus, there exists an efficiently generated random variable $x \in \{0,1\}^n$ and there exists a PPT adversary $\mathcal{B}$ such that

$$\left| \Pr_{s\leftarrow\{0,1\}^n}[\mathcal{B}^{g_s}(1^\lambda, pp, x, g_s(x)) = 1] - \Pr_{s\leftarrow\{0,1\}^n}[\mathcal{B}^{g_s}(1^\lambda, pp, x, U_n) = 1] \right| \geqslant \frac{1}{\text{poly}(\lambda)}$$

Taking the string $x \in \{0,1\}^n$ that maximizes the gap in the equation above, we get a contradiction to Equation (3). □

2) Proof of Theorem 4.1.

Fix any PPT leakage function L with output length $\ell(n)$. As in the proof of Lemma 4.3, we consider a new Gen algorithm, denoted by Gen'. The algorithm Gen' is a PPT algorithm that takes as input the security parameter 1^λ and an input $x \in \{0,1\}^{k(\lambda)}$, and outputs a random secret seed $s \leftarrow \{0,1\}^n$ (distributed as the secret seed generated by Gen) and public paramters pp' which are distributed differently than the ones generated by Gen. Here pp' is a sequence of $2k$ functions, where the functions $f_{1,x_1}, \ldots, f_{k,x_k}$ are generated according to the injective seed generation algorithm $\mathsf{Gen}_{\text{inj}}(1^\lambda)$, and the functions $f_{1,1-x_1}, \ldots, f_{k,1-x_k}$ are generated according to the lossy seed generation algorithm $\mathsf{Gen}_{\text{loss}}(1^\lambda)$.

Fix any sequence of inputs $x = x(\lambda) \in \{0,1\}^{k(\lambda)}$. Property (a) of the lossy function family implies that

$$\{x, pp, s\} \approx \{x, pp', s\} \tag{4}$$

where $(pp, s) \leftarrow \mathsf{Gen}(1^\lambda)$ and $(pp', s) \leftarrow \mathsf{Gen}'(1^\lambda, x)$.

As in the proof of Lemma 4.3, notice that for $(pp', s) \leftarrow \mathsf{Gen}'(1^\lambda, x)$, the oracle g_s can be simulated everywhere except at point x, given only pp', x, and given the sequence

$$\begin{aligned} y_k &\triangleq f_{k,1-x_k}(s) \\ y_{k-1} &\triangleq f_{k-1,1-x_{k-1}}(f_{k,x_k}(s)) \\ &\vdots \\ y_1 &\triangleq f_{1,1-x_1}(f_{2,x_2}(\ldots f_{k,x_k}(s))) \end{aligned}$$

This is the case, since for any $x' \neq x$, let $i \in [k]$ be the largest index for which $x'_i \neq x_i$. Then one can efficiently compute $g_s(x')$ given only y_i and pp'. Therefore,

$$\tilde{H}_\infty\left(g_s(x) | pp', \{g_s(x')\}_{x'\neq x}\right) \geqslant \tilde{H}_\infty\left(g_s(x) | pp', x, y_1, \ldots, y_k\right)$$

Recall that the functions $f_{1,1-x_1}, \ldots, f_{k,1-x_k}$ were generated according to the lossy seed generation algorithm $\mathsf{Gen}_{\text{loss}}(1^\lambda)$. Thus, $(y_1, \ldots, y_k)$ contains only $k \cdot (n - \omega)$ bits of information about $s \in \{0,1\}^n$. Namely,

$$\begin{aligned} \tilde{H}_\infty\left(s | pp', y_1, \ldots, y_k\right) &= n - k(n - \omega) \\ &= n - kn^\epsilon \\ &= n - n^{\delta+\epsilon} \\ &= n - \beta(n). \end{aligned}$$

Recall that

$$g_s(x) = f_{1,x_1}(\ldots(f_{k,x_k}(s))).$$

The fact that all the functions $f_{1,x_1}, \ldots, f_{k,x_k}$ are injective, together with the equation above, implies that

$$\tilde{H}_\infty\left(g_s(x) | pp', y_1, \ldots, y_k\right) \geqslant n - \beta(n),$$

and thus

$$\tilde{H}_\infty\left(g_s(x) | pp', L(pp', s), \{g_s(x')\}_{x'\neq x}\right) \geqslant n - \beta(n) - \ell = n - \beta - (n - 2\beta) = \beta$$

as desired.

□

We next prove Theorem 4.2. We do not provide a formal proof, as this type of conversion of selective security to adaptive security is quite standard. Instead we give a proof sketch.

3) Proof Sketch of Theorem 4.2.

Define $\mathsf{Gen}'(1^\lambda)$ that simply guesses the input $x \leftarrow \{0,1\}^k$ in advance, and runs the selective $\mathsf{Gen}'(1^\lambda, x)$. Thus, in our case, it generates the functions $f_{1,x_1}, \ldots, f_{k,x_k}$ using the injective seed, and generates the functions $f_{1,1-x_1}, \ldots, f_{k,1-x_k}$ using the lossy seed. Assume that the function family $\mathcal{G}$ is not adaptively secure. Then there exists a PPT adversary $\mathcal{A}$ that breaks the adaptive security. We construct a PPT adversary $\mathcal{B}$ that breaks the selective security, as follows: $\mathcal{B}$ simply emulates $\mathcal{A}$; if $\mathcal{A}$ chooses an input that is different than the input x guessed by our Gen', then $\mathcal{B}$ simply outputs a random bit. Otherwise, $\mathcal{B}$ continues the emulation. The probability that $\mathcal{B}$ continues the emulation (i.e., the probability that Gen' guessed the input x correctly) is $2^{-k} = 2^{-n^\delta}$. Thus, $\mathcal{B}$ breaks selective security with probability $2^{-k}/\mathrm{poly}(\lambda)$. Therefore, $\mathcal{B}$ can be used to distinguish between a random lossy seed and a random injective seed with probability $2^{-k}/\mathrm{poly}(\lambda)$.

5 Leakage resilient random-input PRFs

In this section, we show how PEFs can be used to construct another relaxed version of leakage resilient PRFs. Here, we relax the requirement that *for every* input x the value $g_s(x)$ looks random, and require that for a *random* input r the value $g_s(r)$ looks random, even given oracle access to g_s everywhere except point r. We call such a family a *random-input PRF* family.

Definition 5.1. *A function family $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$ is said to be an ℓ-**leaky random-input PRF** family if for any poly-time leakage function L such that $|L(s)| = \ell(|s|)$, and for every* PPT *adversary $\mathcal{A}$ that does not query the oracle at point r,*

$$\left\{\mathcal{A}^{g_s}\left(1^\lambda, L(s), r, g_s(r)\right)\right\}_{s\leftarrow\mathsf{Gen}(1^\lambda), r\leftarrow\{0,1\}^k} \approx \left\{\mathcal{A}^{g_s}\left(1^\lambda, L(s), r, U\right)\right\}_{s\leftarrow\mathsf{Gen}(1^\lambda), r\leftarrow\{0,1\}^k, U\leftarrow\{0,1\}^{|g_s(r)|}}$$

In this section, we show how to convert any ℓ-leaky β-PEF family $\mathcal{G} = \{\mathcal{G}_\lambda\}$ with selective security into an ℓ-leaky random-input PRF family $\mathcal{G}^* = \{\mathcal{G}^*_\lambda\}$, as long as $\beta \geqslant \omega(\log\lambda)$. The idea is simply to apply an extractor to the output of $\mathcal{G}$.

Recall that an ℓ-leaky β-PEF family with selective security has the property that for every input $x \in \{0,1\}^k$, the value $g_s(x)$ is indistinguishable from having min-entropy β, given the public parameters pp, leakage $L(pp, s)$, and oracle access to g_s everywhere except at the point x. Therefore, as long as $\beta \geqslant \omega(\log n) = \omega(\log\lambda)$, where $n = |g_s(x)| = \mathrm{poly}(\lambda)$, applying an extractor to $g_s(x)$ results with a string that looks truly random even given pp, $L(pp, s)$, and oracle access to g_s everywhere except at the point x.

Unfortunately, we do not have deterministic randomness extractors, and to extract randomness from $g_s(x)$ we need fresh randomness or an independent weak source of randomness. The question is: where do we get such (weak) independent randomness from? To this end, we use a collision-resistant hash function h, and define:

$$g^*_s(x) = \mathsf{Ext}\left(g_s(h(x)); x\right),$$

where Ext is a 2-source extractor, where one source is required to have min-entropy rate greater than $1/2$ and the other min-entropy β. The 2-source extractor defined by Raz [34] has this property. The idea would be to think of $h(x)$ as fixed, and thus the sources $g_s(h(x))$ and x are independent. Moreover, we take a hash function h that is shrinking enough, so that x has min-entropy rate greater than $1/2$ even conditioned on $h(x)$, and $g_s(h(x))$ has min-entropy β even conditioned on $h(x)$ and on $pp, L(pp, s)$.

5.1 The construction

We start with an ℓ-leaky β-PEF $\mathcal{G} = \{\mathcal{G}_\lambda\}$ which is selectively secure. Let $\mathcal{H} = \{\mathcal{H}_\lambda\}$ be a family of collision resistant hash-functions, where each $h \in \mathcal{H}_\lambda$ is a function

$$h : \{0,1\}^{3k} \to \{0,1\}^k.$$

Let

$$\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^{3k} \to \{0,1\}^m$$

be a strong 2-source extractor with the property that if the first source has min-entropy at least β, and the second source has min-entropy rate at least $2/3$, then the output is statistically close to random. As noted, the extractor of Raz [34] satisfies this property.

We use $\mathcal{G}$, $\mathcal{H}$ and Ext, to construct an ℓ-leaky random-input PRF family $\mathcal{G}^*$, as follows:

1. The seed generation algorithm Gen^* takes as input the security parameter 1^λ, and does the following:

(a) Run the seed generation algorithm Gen corresponding to the PEF family $\mathcal{G}$, and compute a pair $(pp, s) \leftarrow \mathsf{Gen}(1^\lambda)$.
(b) Choose a random collision resistant hash function $h \leftarrow \mathcal{H}_\lambda$.
Output the public parameters $pp^* = (pp, h)$ and the secret seed s.

2. For each $pp^* = (pp, h)$ and each secret seed s, let

$$g_s^*(x) \triangleq \mathsf{Ext}(g_s(h(x)), x).$$

where for the sake of clarity, we eliminate from the notation the dependance of g_s^* and g_s on pp and h.

Theorem 5.1 *The function family $\mathcal{G}^* = \{\mathcal{G}_\lambda^*\}$ described above is an ℓ-leaky random-input PRF family, assuming the underlying function family $\mathcal{G} = \{\mathcal{G}_\lambda\}$ is an ℓ-leaky β-PEF family with selective security for $\beta \geqslant \omega(\log \lambda)$, and assuming the underlying family $\mathcal{H} = \{\mathcal{H}_\lambda\}$ is a collision resistant hash family.*

4) Proof of Theorem 5.1.

Suppose for the sake of contradiction that there exists a poly-time leakage function L such that $|L(pp^*, s)| = \ell(|s|)$, and there exists a PPT adversary $\mathcal{A}$ that does not query its oracle at point r, such that

$$\left\{\mathcal{A}^{g_s^*}\left(1^\lambda, pp^*, L(pp^*, s), r, g_s^*(r)\right)\right\} \not\approx \left\{\mathcal{A}^{g_s^*}\left(1^\lambda, pp^*, L(pp^*, s), r, U\right)\right\}$$

where the distributions are over $(pp^*, s) \leftarrow \mathsf{Gen}^*(1^\lambda)$, $r \leftarrow \{0,1\}^{3k}$, $U \leftarrow \{0,1\}^m$, and over the random coin tosses of $\mathcal{A}$. Note that the oracle g_s^* can be simulated given oracle access to g_s. Moreover, the fact that $\mathcal{H}$ is collision resistant implies that for any PPT adversary, given a random r and oracle access to g_s^* everywhere except at point r, the oracle can be simulated by oracle access to g_s everywhere except at point $h(r)$. This is the case since if the adversary finds $x \neq r$ such that $h(x) = h(r)$ then it breaks the collision resistant property of $\mathcal{H}$. Therefore, there exists a PPT adversary $\mathcal{B}$ that does not query its oracle at the point $h(r)$, such that

$$\left\{\mathcal{B}^{g_s}\left(1^\lambda, pp^*, L(pp^*, s), r, \mathsf{Ext}(g_s(h(r)), r)\right)\right\} \not\approx \left\{\mathcal{B}^{g_s}\left(1^\lambda, pp^*, L(pp^*, s), r, U\right)\right\} \quad (5)$$

where the distributions are over $(pp^*, s) \leftarrow \mathsf{Gen}^*(1^\lambda)$, $r \leftarrow \{0,1\}^{3k}$, $U \leftarrow \{0,1\}^m$, and over the random coin tosses of $\mathcal{B}$.

Fix a random collision resistant hash function $h \leftarrow \mathcal{H}_\lambda$. The fact that $\mathcal{G}$ is an ℓ-leaky β-PEF with selective security implies that there exists a PPT algorithm Gen' such that for every $r \in \{0,1\}^{3k}$,

$$\mathsf{Gen}'(1^\lambda, h(r)) \approx \mathsf{Gen}(1^\lambda) \quad (6)$$

and for every poly-time leakage function L (which may depend on h) with output size $\ell(|s|)$, and for $(pp, s) \leftarrow \mathsf{Gen}'(1^\lambda, h(r))$

$$\tilde{H}_\infty\left(g_s(h(r)) | r, pp, h, L(pp, s), \{g_s(x)\}_{x \neq h(r)}\right) \geqslant \beta(n).$$

This, together with the fact that Ext is a strong two-source extractor, and together with the fact that $r|h(r)$ has min-entropy rate 2/3, implies that

$$\left\{pp, h, L(pp, s), \{g_s(x)\}_{x \neq h(r)}, r, \mathsf{Ext}(g_s(h(r)), r)\right\} \approx \left\{pp, h, L(pp, s), \{g_s(x)\}_{x \neq h(r)}, r, U\right\}$$

Therefore, for every PPT algorithm $\mathcal{M}$,

$$\left\{\mathcal{M}^{g_s}\left(1^\lambda, pp^*, L(pp^*, s), r, \mathsf{Ext}(g_s(h(r)), r)\right)\right\}_{\lambda \in \mathbb{N}} \approx \left\{\mathcal{M}^{g_s}\left(1^\lambda, pp^*, L(pp^*, s), r, U\right)\right\}_{\lambda \in \mathbb{N}}$$

where the distributions are over (pp^*, s) where $pp^* = (pp, h)$, $h \leftarrow \mathcal{H}_\lambda$, $(pp, s) \leftarrow \mathsf{Gen}'(1^\lambda, h(r))$, and over $r \leftarrow \{0,1\}^{3k}$ and the random coin tosses of $\mathcal{M}$. This, together with Equation (6), contradicts Equation (5).

Acknowledgments

We would like to greatly thank Daniel Wich and Ran Canetti for many useful discussions. We would also like to thank the anonymous reviewers for useful comments. □

References

[1] A.Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474-495, 2009.

[2] J. Alwen, Y. Dodis, M. Naor, G. Segev, S. Walfish, and D. Wichs. Public-key encryption in the bounded-retrieval model. To Appear in Eurocrypt 2010, 2010.

[3] J. Alwen, Y. Dodis, and D. Wichs.Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36-54, 2009.

[4] V. Boyko. On the security properties of oaep as an all-or-nothing transform. In *CRYPTO,* pages 503-518, 1999.

[5] E. Boyle, G. Segev, and D. Wichs. Fully leakage-resilient signatures. Cryptology ePrint Archive, Report 2010/488, 2010. http://eprint.iacr.org/.

[6] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikun-tanathan. Overcoming the hole in the bucket: Public-key cryptog-raphy resilient to continual memory leakage. In *FOCS*, 2010.

[7] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *EUROCRYPT*, pages 453-469, 2000.

[8] G. D. Crescenzo, R. J. Lipton, and S. Walfish. Per-fectly secure password protocols in the bounded retrieval model. In TCC, pages 225-244, 2006.

[9] Y. Dodis, S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, 2010.

[10] Y. Dodis, K. Haralambiev, A. Lopez-Alt, and D. Wichs. Overcoming the hole in the bucket: Public-key cryptog-raphy resilient to continual memory leakage. In *FOCS*, 2010.

[11] Y. Dodis, Y. T. Kalai, and S. Lovett.On cryptography with auxiliary input. In *STOC*, pages 621-630, 2009.

[12] Y. Dodis and K. Pietrzak.Leakage-resilient pseudoran-dom functions and side-channel attacks on feistel net-works. In *CRYPTO*, 2010.

[13] S. Dziembowski.Intrusion-resilience via the bounded-storage model. In TCC, pages 207-224, 2006.

[14] S.Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293-302, 2008.

[15] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343-360, 2010.

[16] S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikun-tanathan. Protecting against leakage: the computation-ally bounded and noisy cases. To Appear in Eurocrypt 2010, 2010.

[17] D. M. Freeman, O. Goldreich, E. Kiltz, A. Rosen, and G. Segev. More constructions of lossy and correlation-secure trapdoor functions. Cryptology ePrint Archive, Report 2009/590, 2009. http://eprint.iacr.org/.

[18] O. Goldreich, S. Goldwasser, and S. Micali.How to construct random functions. *J. ACM.*, 33(4):792-807, 1986.

[19] S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikun-tanathan. Robustness of the learning with errors assump-tion. In *ICS*, 2010.

[20] S. Goldwasser and G. Rothblum. How to play mental solitaire under continuous side-channels: A complete-ness theorem using secure hardware. Manuscript, 2010.

[21] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clark-son, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appel-baum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Sympo-sium*, pages 45-60, 2008.

[22] B. Hemenway and R. Ostrovsky.Lossy trapdoor func-tions from smooth homomorphic hash proof systems. In *Electronic Colloquium on Computational Complex-ity, Report TR09-127*, 2009.

[23] Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Pri-vate circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308-327, 2006.

[24] Y. Ishai, A. Sahai, and D. Wagner.Private circuits: Se-curing hardware against probing attacks. In *CRYPTO*, pages 463-481, 2003.

[25] A. Juma and Y. Vahlis.Leakage-resilient key proxies. Manuscript, 2010.

[26] J. Katz and V. Vaikuntanathan.Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703- 720, 2009.

[27] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104-113, 1996.

[28] P. C. Kocher, J. Jaffe, and B. Jun.Differential power analysis. In *CRYPTO*, pages 388-397, 1999.

[29] S. Micali and L. Reyzin.Physically observable cryp-tography (extended abstract). In *TCC*, pages 278-296, 2004.

[30] M. Naor and G. Segev.Public-key cryptosystems re-silient to key leakage. In *CRYPTO*, pages 18-35, 2009.

[31] D. A. Osvik, A. Shamir, and E. Tromer.Cache attacks and countermeasures: The case of aes. In *CT-RSA*,pages 1-20, 2006.

[32] C. Peikert and B. Waters.Lossy trapdoor functions and their applications. In *STOC*, pages 187-196, 2008.

[33] K. Pietrzak.A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462-482, 2009.

[34] R. Raz.Extractors with weak random seeds. In *STOC*, pages 11-20, 2005.

[35] R. L. Rivest. All-or-nothing encryption and the package transform. In *FSE*, pages 210-218, 1997.

A Additional leakage models

In this section we mention two additional bounded leakage models: The auxiliary input leakage model of Dodis *et. al.* [11], and the bounded retrieval model of [8,13]. Both are more general than the shrinking leakage model of Akavia *et. al.* [1] and the model of Naor and Segev [30].

The auxiliary input leakage model considers the class of all leakage functions that are (computationally) hard to invert; namely all functions L such that given $L(s)$ it is computationally hard to find s. It is impossible to construct a leakage resilient PEF family in this model, since it is impossible to prove that for every x that was not queried, the value of $f_s(x)$ has (computational) entropy, as the leakage function can simply leak the value of $f_s(x)$ (or a hard-to-invert function of $f_s(x)$ that leaves $f_s(x)$ with no computational entropy).

The bounded retrieval model considers a (big) absolute bound N on the leakage size, and the goal is to construct a scheme that is secure against leakage of size N, where the requirement is that the efficiency of the scheme depends only logarithmically on N, though the secret key is of size poly(N). It is impossible to construct a leakage-resilient PEF family in this model, since the requirement is that the time it takes to evaluate f_s depends only logarithmically on the size of s, and thus can depend only on a few bits of s. Therefore, for any input x, the leakage can contain the bits in s that are used in the evaluation of $f_s(x)$. Notice that the fact that f_s is deterministic implies that the same bits of s are used in each evaluation of $f_s(x)$.

B Constructions of lossy functions

In this section we construct a lossy function family with the desired property, that the functions have the same domain and range. The construction is based on the DDH assumption, and is reminiscent to constructions that were given in previous work on lossy trapdoor functions [17,22,32]. However, all the constructions in these works, albeit having a trapdoor, did not have the property that the image and the range are of the same size, a property that is essential for us. Recall that we use lossy functions to construct pseudo-entropy functions (PEFs), and our PEFs compose k lossy functions. This makes it essential that the underlying lossy functions do not stretch their input (even by a small constant multiplicative factor), and all previous constructions do stretch the input by at least a constant multiplicative factor.

5) Notation.

Let $\mathbf{A} = (a_{i,j})_{i,j\in[n]} \in \mathbb{Z}_q^{n\times n}$ be a matrix over the group Z_q, and let g be a generator of a multiplicative group of prime order q. We denote by $g^{\mathbf{A}}$ the n-by-n matrix whose (i,j) entry is $g^{a_{i,j}}$.

6) The DDH assumption.

The DDH assumption asserts that for any PPT adversary $\mathcal{A}$, and for any constant c,

$$|\Pr[\mathcal{A}(g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g^a, g^b, g^c) = 1]| \leqslant 1/n^c$$

where the probabilities are over $a, b, c \leftarrow \mathbb{Z}_q$ and the randomness of $\mathcal{A}$.

The matrix form of the DDH assumption, which is equivalent to the standard form of the DDH assumption, asserts that it is hard to distinguish between $g^{\mathbf{A}}$, where $\mathbf{A}$ is a random rank-n matrix, and $g^{\mathbf{B}}$, where $\mathbf{B}$ is a random rank-1 matrix, and where $q \approx 2^\lambda$ and λ is the security parameter.

7) The construction.

Consider the family $\mathcal{F}$ which is associated with two seed generation algorithms: the lossy algorithm $\mathsf{Gen}_{\text{loss}}$ and the injective algorithm $\mathsf{Gen}_{\text{inj}}$. These algorithms take as input a security parameter 1^λ and they choose a random safe prime $p \in [2^{\lambda-1}, 2^\lambda]$; i.e., a prime p such that $q = \frac{p-1}{2}$ is also a prime.[9] These al-

[9] We rely here on the assumption that the safe primes are dense. Alternatively, one can think of these algorithms as non-uniform algorithms, where the safe prime p is hard-wired into the seed generation circuit, in which case we only need to assume that there exists a safe prime in the range $[2^{\lambda-1}, 2^\lambda]$.

gorithms also choose a quadratic-residue $g \in \mathbb{Z}_p^*$ of order q, by simply choosing any element $h \in \mathbb{Z}_p^* \setminus \{1, -1\}$ and setting $g = h^2$. Notice that the group $\mathbb{Z}_p^*$ is of order $p - 1 = 2q$. The fact that q is prime implies that each group element is of order 1,2,q, or $2q$, and hence all the quadratic residues are of order 1 or q. Thus, all the quadratic residues, except the unit element 1, have order q.

The lossy seed generation algorithm $\mathsf{Gen}_{\text{loss}}$ generates a random rank-1 matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and outputs the seed $s = g^{\mathbf{A}}$. The injective seed generation algorithm $\mathsf{Gen}_{\text{inj}}$ generates a random rank-n matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and outputs the seed $s = g^{\mathbf{A}}$.

The function $f_{g^{\mathbf{A}}}$ takes as input $\mathbf{x} \in \mathbb{Z}_q^n$ and computes $g^{\mathbf{Ax}}$. Note that this output $\mathbf{y} = (y_1, \ldots, y_n)$ is in $(\mathbb{Z}_p^*)^n$. In order to map the output $\mathbf{y}$ to $\mathbb{Z}_q^n$, we simply output a square-root of each y_i. Note that since p is a prime and each y_i is a quadratic residue, each y_i has exactly 2 square-roots ω_i and $p - \omega_i$. The function $f_{g^{\mathbf{A}}}(\mathbf{x})$ will output for each coordinate i, the smaller square-root among the two, and thus all the square-roots will be in the set $\{1, \ldots, \frac{p-1}{2}\}$, and hence the output can be interpreted as being in $\mathbb{Z}_q^n$, as desired.

The properties we require are summarized in the following theorems. The first uses the standard DDH assumption required for Theorem 4.1 and Lemma 4.3, and the second makes the stronger assumption required for Theorem 4.2.

Theorem B.1. *Suppose that for every* PPT *algorithm* $\mathcal{A}$ *and for every constant* C *we have*

$$|\Pr[\mathcal{A}(g^a, g^b g^{ab}) = 1] - \Pr[\mathcal{A}(g^a, g^b g^c) = 1]| < 1/n^C$$

where $a, b, c \leftarrow \mathbb{Z}_q$. *Then the construction yields an* $\omega = n \log q - \log q$ *lossy function family which is an* $\alpha = \log q + 2 \log^2 \lambda$ *computational extractor.*

Theorem B.2. *Fix any* $\delta \in (0, 1)$. *Suppose that for every* PPT *algorithm* $\mathcal{A}$ *we have*

$$|\Pr[\mathcal{A}(g^a, g^b g^{ab}) = 1] - \Pr[\mathcal{A}(g^a, g^b g^c) = 1]| < 1/2^{n^\delta}$$

where $a, b, c \leftarrow \mathbb{Z}_q$. *Then the construction yields an* $\omega = n \log q - \log q$ *lossy function family which is an* $\alpha = \log q + 2 \log^2 \lambda$ *computational extractor, where the adversary can distinguish between a random lossy function and a random injective one with probability at most* $\text{poly}(\lambda)/2^{n^\delta}$.

The proof of these theorems is straightforward. Note that if $\mathbf{A}$ is rank-1, then the value $f_{g^{\mathbf{A}}}(\mathbf{x})$ loses all the information about $\mathbf{x}$ except $\log q$ bits. On the other hand, if $\mathbf{A}$ is rank-n, then the function $f_{g^{\mathbf{A}}}$ is injective. Moreover, the matrix form of the DDH assumption, immediately implies that it is hard to distinguish between a random lossy function and a random injective function. Therefore, the family $\mathcal{F}$ is an ω-lossy function family, with $\omega = n \log q - \log q$. Taking $q = 2^{n^\epsilon}$, we get $\omega = n^{1+\epsilon} - n^\epsilon$, where $n^{1+\epsilon}$ is the input length.

Moreover, the leftover hash-lemma, together with the DDH assumption, implies that $\mathcal{F}$ is a (strong) α-computational extractor, for $\alpha = \log q + 2 \log^2 \lambda$. This is the case, since the leftover hash lemma[10] implies that for any random variable in $\mathbb{Z}_q^n$ with min-entropy $\log q + 2 \log^2 \lambda$,

$$(\mathbf{r}, \mathbf{rx} \pmod q) \approx (\mathbf{r}, U_q).$$

This, together with the DDH assumption, implies that

$$(g^{\mathbf{A}}, g^{\mathbf{Ax}}) \approx (g^{\mathbf{A}}, g^{\mathbf{u}}),$$

as desired.

[10] The leftover hash lemma asserts that for any random variable X over a set D, and for any 2-universal hash function $H : S \times D \to \{0,1\}^m$, if X has min-entropy at least $m + 2\log(1/\epsilon)$ then $\Delta(S, (H(S, X)), (S, U)) \leqslant \epsilon$, where $\Delta(\cdot, \cdot)$ is the statistical distance.