This is the Full Version of the Extended Abstract that appears
in the Proceedings of 13th International Conference on Applied Cryptography and Network Security (ACNS '15)
(2 – 5 June 2015, New York, USA)
Tal Malkin Ed., Springer-Verlag, LNCS 9092.

# Robust Pseudo-Random Number Generators with Input Secure Against Side-Channel Attacks

Michel Abdalla[1], Sonia Belaïd[1,2], David Pointcheval[1], Sylvain Ruhault[1,3], and Damien Vergnaud[1]

[1] Ecole Normale Supérieure, CNRS, INRIA, and PSL Research University, Paris, France.
[2] Thales Communications & Security, Gennevilliers, France.
[3] Oppida, Montigny-le-Bretonneux, France. {michel.abdalla,sonia.belaid,david.pointcheval,sylvain.ruhault,damien.vergnaud}@ens.fr

**Abstract.** A pseudo-random number generator (PRNG) is a deterministic algorithm that produces numbers whose distribution is indistinguishable from uniform. In this paper, we extend the formal model of PRNG with input defined by Dodis *et al.* at CCS 2013 to deal with partial leakage of sensitive information. The resulting security notion, termed *leakage-resilient robust PRNG with input*, encompasses all the previous notions, but also allows the adversary to continuously get some leakage on the manipulated data. Dodis *et al.* also proposed an efficient construction, based on simple operations in a finite field and a classical deterministic pseudo-random generator **G**. Here, we analyze this construction with respect to our new stronger security model, and prove that with a stronger **G**, it also resists leakage. We show that this stronger **G** can be obtained by tweaking some existing constructions based on AES. We also propose a new instantiation which may be better in specific cases. Eventually, we show that the resulting scheme remains quite efficient in spite of its new security properties. It can thus be recommended in contexts where side-channel resistance is required.

**Keywords:** Randomness, Entropy, Side-Channel countermeasures, Security models.

## 1 Introduction

While most cryptosystems require access to a perfect source of randomness for their security, such sources are extremely difficult to obtain in practice. For this reason, concrete implementations of cryptographic schemes often use a pseudo-random number generator (PRNG). The latter allows to generate a sequence of bits whose distribution is computationally indistinguishable from the uniform distribution, when given as input a secret short random value, called *seed*.

To get around the need for a truly random seed, Barak and Halevi [5] proposed a tweaked primitive, called PRNG with inputs, which still generates pseudo-random values but now remains secure even in presence of a potentially biased random source. Moreover, they also proposed a new security notion, called *robustness*, which states that a PRNG with inputs should meet three security properties: *resilience*, *forward security*, and *backward security*. While resilience models the inability of an adversary to predict future PRNG outputs even when manipulating the entropy source, forward and backward security ensures that an adversary cannot predict past or future outputs of the PRNG even when compromising its internal state. More recently, Dodis *et al.* [10] extended the work of Barak and Halevi to integrate the process of accumulation of entropy into the internal state. For this purpose, they refined the notion of robustness and proposed a very practical scheme satisfying it. Under the robustness security notion, an adversary can observe the inputs and outputs of a PRNG, manipulate its entropy source, and compromise its internal state.

**Side-Channel Resistance for PRNGs.** While the notion of robustness seems reasonably strong for practical purposes, it still does not fully consider the reality of embedded devices, which may be subject to *side-channel attacks*. In these attacks, an attacker can exploit the physical leakage of a device by several means such as power consumption, execution time or electromagnetic radiation. In order to consider such attacks, a first and important step was made

by Micali and Reyzin [18] who proposed the framework of *physically observable cryptography*. In particular, they formally defined a classical assumption according to which *only computation leaks information*. Later, Dziembowski and Pietrzak went a step further by defining the *leakage-resilient cryptography model* [13]. In the latter, every computation leaks a limited amount of information whose size is bounded by some parameter $\lambda$. It benefits from capturing most of the known side-channel attacks and was consequently used to build many recent primitives [19,14,15]. In a different direction, we should mention a recent and important work which was proposed by Prouff and Rivain [20] and then extended by Duc et al. [12] to formally prove the security of masking implementations. In the latter works, the sensitive variables are split into different share and the adversary needs to recover all of them to reconstruct the secret.

In the specific context of PRNGs and stream ciphers, several constructions have been proposed so far and proved secure in the leakage-resilient cryptography model (e,g., [24,22,25]). The work of Yu *et al.* [24], for instance, proposes a very efficient construction of a leakage-resilient PRNG. Likewise, the work of Standaert *et al.* [22] shows how to obtain very efficient constructions of leakage-resilient PRNGs by relying on empirically verifiable assumptions. None of these works, however, consider potentially biased random sources, which is our main goal here.

**Our Contributions.** In this paper, we aim to build a practical and robust PRNG with input that can resist side-channel attacks. Since the construction proposed by Dodis et al. [10] seems to be a good candidate, we use it as the basis of our work. In doing so, we extend its security model to include the leakage-resilient security and we prove the whole construction secure under stronger requirements for the underlying deterministic pseudo-random generator[1]. Since it is not obvious how to instantiate the construction to meet our stronger needs, we propose three solutions based on AES in counter mode that are only slightly less efficient than the original instantiation proposed in [10]. Two of them are tweaked existing constructions and the third one is a new proposal which may be better in specific cases. All three instantiations only require that the implementation of AES in counter mode is secure against Simple Power Analysis attacks since very few calls are made with the same secret key.

**Organization.** From a theoretical side, we propose in Section 3 a new formal security model for PRNGs with input, which, in addition to encompassing all previous security notions [10], also guarantees security in the *leakage-resilient cryptography* model. In Section 4, we analyze the robust construction based on polynomial hash functions given in [10] showing why its instantiation may be vulnerable to side-channel attacks. We then prove that, under non-restrictive conditions on the underlying deterministic pseudo-random generator, the generic construction actually meets our stronger security property. Finally, in Section 5, we discuss the instantiations of this construction.

## 2 Preliminaries

### 2.1 Notations and Definitions

**Probabilities.** When $X$ is a distribution, or a random variable following this distribution, we denote $x \xleftarrow{\$} X$ when $x$ is sampled according to $X$. For a variable $X$ and a set $S$, the notation $X \xleftarrow{\$} S$ denotes both assigning $X$ a value uniformly chosen from $S$ and letting $X$ be a uniform random variable over $S$. The uniform distribution over $n$ bits is denoted $\mathcal{U}_n$.

**Indistinguishability.** Two distributions $X$ and $Y$ are said $(t, \varepsilon)$-*computationally indistinguishable* (and we denote this property by $\mathbf{CD}_t(X, Y)$), if for any distinguisher $\mathcal{A}$ running within time $t$, its advantage in distinguishing a random variable following $X$ from a random variable

[1] A recent work by Dodis *et al.* in [11] also extends the robustness model to address the *premature next attack* where the internal state has insufficient entropy and an output is generated. Our work is a different complement.

following $Y$, denoted $|\Pr[\mathcal{A}(X) = 1] - \Pr[\mathcal{A}(Y) = 1]|$ is bounded by $\varepsilon$. When $t = \infty$, meaning $\mathcal{A}$ is unbounded, we say that $X$ and $Y$ are $\varepsilon$-*close*.

**Pseudo-Random Generators.** A function $\mathbf{G} : \{0,1\}^m \to \{0,1\}^n$ is a (deterministic) $(t,\varepsilon)$-pseudo-random generator (PRG) if $\mathbf{CD}_t(\mathbf{G}(\mathcal{U}_m), \mathcal{U}_n) \leqslant \varepsilon$.

**Pseudo-Random Functions.** A keyed family of functions $\mathbf{F} : \{0,1\}^\mu \times \{0,1\}^\mu \to \{0,1\}^\mu$ is a $(t,q,\varepsilon)$-pseudo-random function (PRF) if no adversary can have an advantage greater than $\varepsilon$, within time $t$, in distinguishing, for a random key $K \xleftarrow{\$} \{0,1\}^\mu$, $q$ answers $F_K(x_i)$ for adaptively chosen inputs $(x_i)$, from $q$ random answers $y_i \xleftarrow{\$} \{0,1\}^\mu$, for $i = 1, \ldots, q$.

**Entropy.** For a discrete distribution $X$, we denote its *min-entropy* by $\mathbf{H}_\infty(X) = \min_{x \in X}\{-\log \Pr[X = x]\}$.

**Extractors.** Let $\mathcal{H} = \{h_X : \{0,1\}^n \to \{0,1\}^m\}_{X \in \{0,1\}^d}$ be a hash function family. We say that $\mathcal{H}$ is a $(k,\varepsilon)$-*extractor* if for any random variable $I$ over $\{0,1\}^n$ with $\mathbf{H}_\infty(I) \geqslant k$, the distributions $(X, h_X(I))$ and $(X,U)$ are $\varepsilon$-close where $X$ is uniformly random over $\{0,1\}^d$ and $U$ is uniformly random over $\{0,1\}^m$. We say that $\mathcal{H}$ is $\rho$-universal if for any inputs $I \neq I' \in \{0,1\}^n$ we have $\Pr_{X \xleftarrow{\$} \{0,1\}^d}[h_X(I) = h_X(I')] \leqslant \rho$.

**Lemma 1 (Leftover-Hash Lemma).** *[21, Theorem 8.37] Assume that $\mathcal{H}$ is $\rho$-universal where $\rho = (1+\alpha)2^{-m}$ for some $\alpha > 0$. Then, for any $k > 0$, it is also a $(k,\varepsilon)$-extractor for $\varepsilon = \frac{1}{2}\sqrt{2^{m-k} + \alpha}$.*

### 2.2 Basic Security Model

In this section, we recall notations and the security notion of a PRNG with input and of distribution sampler, introduced in [10]. In Section 3, we will propose a stronger security model that takes into account possible leakage of information in the context of side-channel attacks.

**Definition 2 (PRNG with Input).** *A PRNG with input is a triple of algorithms $\mathcal{G} =$ (*setup*, *refresh*, *next*), with $n$ the state length, $\ell$ the output length, and $p$ the input length:*

- setup *is a probabilistic algorithm that outputs some public parameters* seed*;*
- refresh *is a deterministic algorithm that, given* seed*, a state $S \in \{0,1\}^n$ and an additional input $I \in \{0,1\}^p$, outputs a new state $S' =$ refresh$(S, I;$ seed$) \in \{0,1\}^n$;*
- next *is a deterministic algorithm that, given* seed *and a state $S \in \{0,1\}^n$, outputs a pair $(S', R) =$ next$(S;$ seed$)$ where $S' \in \{0,1\}^n$ is the new state and $R \in \{0,1\}^\ell$ is the output randomness.*

The parameter seed is public and fixed in the system once for all. For the sake of clarity, we drop it in the notations and we write $S' =$ refresh$(S, I)$ instead of refresh$(S, I;$ seed$)$ and $(S', R) =$ next$(S)$ instead of next$(S;$ seed$)$. When a specific part of the seed will be required, we will explicitly add it as input.

In practice, the global internal state contains all the PRNG features and some structured and redundant information, such as counters, in addition to a random pool of length $n$. When the adversary will have access to the state, this will be to all this information, with both reading (get-state) and writing (set-state) capabilities. However, when we denote $S$ in this paper, for the sake of simplicity, we refer to the randomness pool, that we expect to be truly random, or with as much entropy as possible.

**Adversary.** We consider an attacker divided in two parts: a distribution sampler $\mathcal{D}$ and a classical attacker $\mathcal{A}$. The former generates seed-independent inputs that will be used by the PRNG to improve the quality of its entropy with the refresh algorithm. These inputs, potentially biased under *partial* adversarial control, are generated in practice from the device activities (e.g., system interrupts) and cannot consequently depend on the parameter seed. Note that as explained in [10] and in [11], seed-independence is necessary to achieve security of the scheme.

**Definition 3 (Distribution Sampler).** *A distribution sampler $\mathcal{D}$ is a stateful and probabilistic algorithm which, given the current state $\sigma$, outputs a tuple $(\sigma', I, \gamma, z)$ where*

- *$\sigma'$ is the new state for $\mathcal{D}$;*
- *$I \in \{0,1\}^p$ will be the next input for the* refresh *algorithm;*
- *$\gamma$ is some* entropy estimation *of $I$, as discussed below;*
- *$z$ is the possible* leakage *about $I$ given to the adversary $\mathcal{A}$[2].*

If $q$ denotes an upper bound on the number of executions of $\mathcal{D}$, such a distribution sampler is said *legitimate* if the min-entropy of every input $I_j$ is not smaller than the entropy estimate $\gamma_j$, even given all the additional information: $\mathbf{H}_\infty(I_j \mid I_1, \ldots, I_{j-1}, I_{j+1}, \ldots, I_q, z_1, \ldots, z_q, \gamma_1, \ldots, \gamma_q) \geqslant \gamma_j$, for all $j \in \{1, \ldots, q\}$ where $(\sigma_i, I_i, \gamma_i, z_i) = \mathcal{D}(\sigma_{i-1})$ for $i \in \{1, \ldots, q\}$ and $\sigma_0 = 0$.

**Robustness.** We now recall the security game $\mathsf{ROB}(\gamma^*)$, from [10], that defines the main security notion for a PRNG with input, the *robustness*. We have slightly modified the initial definition, but in an equivalent way (see Figure 1, without the leaking procedures nor the leakage function $f$ as input to the initialize procedure). In the security game,

- the parameter $\gamma^*$ defines the minimal entropy that is required in the internal state of the PRNG so that the output looks random. Under this threshold, the PRNG has not accumulated enough entropy in its internal state, and then is not considered safe for generating random-looking outputs;
- the variable $c$ is an estimation of the actual entropy collected in the internal state of the PRNG. It does not make use of any entropy estimator, but just considers the lower-bound provided by the distribution sampler on the entropy of the input. In case of a legitimate distribution sampler, this lower-bound is correct;
- the flag/function compromised is a Boolean variable that is true if the actual entropy (the parameter $c$) is under the threshold $\gamma^*$. In such a case, the PRNG is with an unsafe status, and thus the adversary may have some control on it;
- the challenge $b$ is a bit that will be used to challenge the adversary, whose goal is to guess it.

The game $\mathsf{ROB}(\gamma^*)$ starts with an initialize procedure, applies procedures to answer to oracle queries from the adversary $\mathcal{A}$, and ends with a finalize procedure. The procedure initialize sets the parameter seed with a call to algorithm setup, the internal state $S$ of the PRNG, as well as $c$ and $b$. After all oracle queries, the adversary $\mathcal{A}$ outputs a bit $b^*$, given as input to the procedure finalize, which compares the response of $\mathcal{A}$ to the challenge bit $b$. The procedures used to answer to oracle queries are the following:

- the procedures get-state/set-state allow the adversary $\mathcal{A}$ to learn or to fix the whole internal state of the PRNG, including the structured part. However, as mentioned above, we just model the impact on the random pool in this analysis;
- the procedure next-ror is used to challenge the adversary $\mathcal{A}$ on its capability to distinguish the output of the PRNG from a truly random output. In the safe case (when $c \geqslant \gamma^*$), the new estimated entropy of the internal state, in the variable $c$, is then set to the state length, that is $n$. In the unsafe case, as explained in [10], the real value for $R$, which might reveal non-trivial information about the weak internal state, is first output and then the new estimated entropy of the internal state, in the variable $c$, is reset to 0[3].
- the procedure $\mathcal{D}$-refresh allows the adversary $\mathcal{A}$ to call the distribution sampler $\mathcal{D}$ to get a new input and to run the refresh algorithm with this specific input to improve the quality of the internal state. In addition to the input $I$ for the PRNG, the distribution sampler $\mathcal{D}$ also

[2] This leakage is not related to side-channel attacks but represents the partial knowledge the adversary has on the inputs because of its control on the distribution.

[3] We could have strengthened this definition, by only reducing $c$ by $\ell$ bits in this case, but we kept the conservative notion.

outputs the leakage $z$ on input $I$ that is given to $\mathcal{A}$ and an estimate $\gamma$ of the entropy of the input (with respect to all the other inputs and the leakage information $z$). We use a more conservative definition, by considering that it really accumulates more entropy only if $c$ was below $\gamma^*$, otherwise, it stays unchanged. The new estimated entropy of the internal state, in the variable $c$, is thus set to $c + \gamma$ if $c$ was below $\gamma^*$, but of course with a maximum of $n$.

Note that we dropped the get-next procedure from [10], but as noted by [8,3], multiple calls to next-ror are enough to capture a similar security level.

**Definition 4 (Robustness of PRNG with Input).** *A pseudo-random number generator with input* $\mathcal{G} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ *is called* $(t, q_r, q_n, q_s, \gamma^*, \varepsilon)$-robust*, if for any adversary* $\mathcal{A}$ *running within time* $t$*, that first generates a legitimate distribution sampler* $\mathcal{D}$ *(for the* $\mathcal{D}$-refresh *procedure), that thereafter makes at most* $q_r$ *calls to* $\mathcal{D}$-refresh*,* $q_n$ *calls to* next-ror*, and* $q_s$ *calls to* get-state/ set-state*, the advantage of* $\mathcal{A}$ *in game* $\mathsf{ROB}(\gamma^*)$ *is at most* $\varepsilon$*.*

### 2.3 Model for Information Leakage

In this paper, we aim to protect the construction of Dodis et al. against side-channel attacks. In this purpose, we describe hereafter the leakage model.

**Only Computation Leaks.** From the axiom "Only Computation Leaks" of Micali and Reyzin [18], we assume that only the data being manipulated in a computation can leak during this computation. This assumption actually well fits the reality of practical observations. As a consequence, we can split our cryptographic primitives into small blocks that independently leak functions of their inputs. We also authorize the adversary to choose different leakage functions for each block.

**Bounded Leakage per Iteration.** Since it is more suited for a PRNG [7], we follow the model of leakage-resilient cryptography, with the strong requirement to preserve reasonable performances. We let the adversary the choice of the polynomial time leakage functions with a bound $\lambda$ on their output length. This parameter is closely related to the security parameter of the underlying cryptographic primitives and will be part of global scheme's security bound.

**Non-Adaptive Leakage.** The choice of leakage functions left to the adversary reveals the desire to consider every possible component whatever its way of leaking. However, we based our work on the practical observation whereby leakage functions completely depend on the inherent device. On the contrary, a few works (e.g., [13,19]) give the adversary the possibility to modify its leakage functions according to its current knowledge. Even if this model aims to be more general, it leads to unrealistic scenarios since the adversary is then able to predict further steps of the algorithm through impossible leakage functions. For these reasons, this work, as many others before [24,15,25,2], only consider non-adaptive leakage functions.

## 3 Leakage-Resilient Robustness of a PRNG with Input

In the security model of [10], recalled in the previous section, the distribution sampler $\mathcal{D}$ generates the external inputs used to refresh the PRNG and already gives the adversary $\mathcal{A}$ some information about how the environment of the PRNG behaves when it generates these inputs. This information is modeled by $z$. In order to model information leakage during the executions of the PRNG algorithms refresh and next, we give the adversary the choice of the leakage functions, that we globally name $f$, associated to each algorithm, or even each small block. Since we restrict our model to non-adaptive leakage, we ask the adversary to choose them beforehand. So they are provided as input to the initialize procedure by the adversary (see Figure 1). Then, each leakage function will be implicitly used by our two new procedures named leak-refresh and leak-next that, in addition to the usual outputs, also provide some leakage $L$ about the manipulated data, as

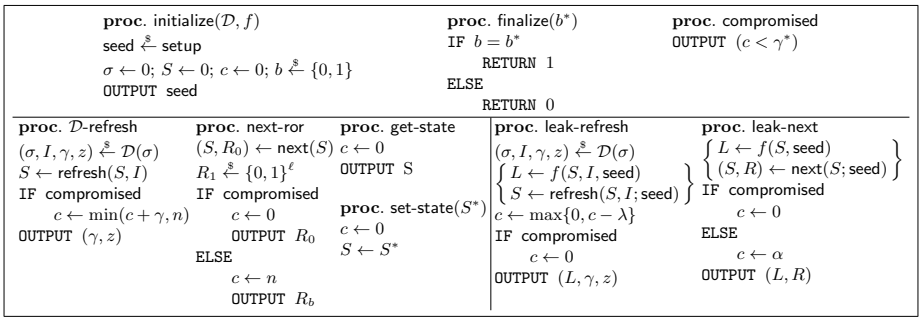


**Fig. 1.** Procedures in the Leakage-Resilient Robusteness Security Game $\mathsf{LROB}(\gamma^*, \lambda)$

described in Section 2.3. We thus have a new parameter $\lambda$, that bounds the output length of the leakage function. Our new *Leakage-Resilient Robustness* security game $\mathsf{LROB}(\gamma^*, \lambda)$ makes use of the procedures described in Figure 1 and is described in details below:

- the parameter $\gamma^*$, the variable $c$, and the Boolean flag/function compromised are the same as in Section 2.2 for the basic robustness;
- the new parameter $\lambda$ fixes the maximal information leakage which can be collected during the execution of operations refresh and next. Namely, for each operation (refresh or next), the leakage functions globally output at most $\lambda$ bits. Such a leakage will be available when querying the leaking procedures leak-refresh and leak-next below;
- the new parameter $\alpha$ is an integer that models the minimal expected entropy of $S$ after a leak-next (next with leakage) call, in a safe case (compromised is false), that is when the entropy of the internal state was assumed greater than $\gamma^*$. This captures both the creation of computational entropy during a next execution and the smaller loss of entropy caused by the leakage. We could expect $\alpha = n - \lambda$, but it may depend on the explicit construction;
- the procedures initialize$(\mathcal{D}, f)$/finalize$(b^*)$ initiate the security game with the additional leakage function $f$, check whether the adversary has won the game and output 1 in this case or 0 otherwise. Contrary to the choice made in [10] (which is also valid), the initial state $S$ is here set to zero (as well as the entropy counter) so that no assumption needs to be made on its initialization;
- the procedures get-state/set-state, $\mathcal{D}$-refresh, and next-ror are the same as for the basic robustness;
- the procedure leak-refresh runs the refresh algorithm but additionally provides some information leakage $L$ on the input $(S, I)$ and seed, as above. As for the next-ror-queries, the leakage can reveal non-trivial information about a weak internal state even before the effectiveness of the refresh, and then we reduce $c$ by $\lambda$ bits. And if it drops below the threshold $\gamma^*$, it is reset to 0. Again, we could have strengthened this definition, but we preferred to keep a conservative notion. Furthermore, this strict notion is important w.r.t. our new definitions of recovering and preserving security with leakage. Note that if the $\mathcal{D}$-refresh algorithm is complex, several leakage functions can be defined at every step, but the global leakage is limited to $\lambda$, hence the notation $\{\ldots\}$, since they can be interleaved.
- the procedure leak-next runs the next algorithm but additionally provides some information leakage $L$ on the input $S$ and seed, according to the leakage function $f$ provided to the initialize procedure. If the status was safe, then the new entropy estimate $c$ is set to $\alpha$, otherwise, it is reset to 0 (as for the next-ror). As above, if the next algorithm is complex, several leakage functions can be defined at each step, but the global leakage is limited to $\lambda$.

As in [10], attackers have two parts: a distribution sampler and a classical attacker with the former only used to generate seed-independent inputs (potentially *partially* biased) from device activities. Examples of the entropy's traces for the procedures defined in [10] and in our new model are provided in Figure 2. The threshold $\gamma^*$ has to be slightly higher in our new model, because for a similar next algorithm, we need to accumulate a bit more of entropy to maintain

security even in presence of leakage. Typically, it has to be increased by $\lambda$. Now we detailed the new security game, we can define the notion of leakage-resilient robustness of a PRNG with input.

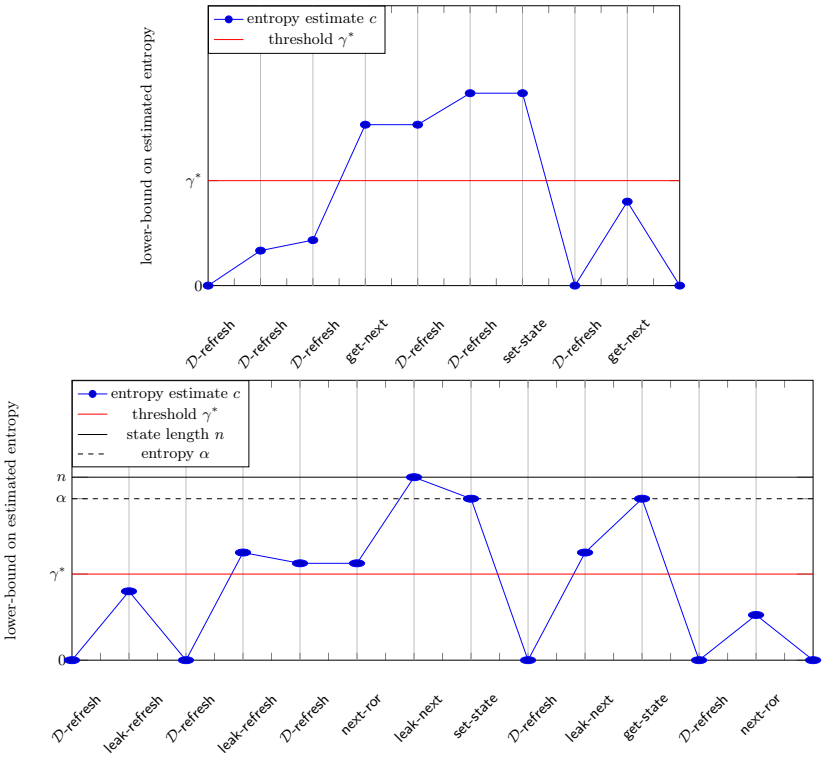


**Fig. 2.** Traces of entropy estimates in the model [10] (up) and in our new model (down)

**Definition 5 (Leakage-Resilient Robustness of PRNG with Input).** *A pseudo-random number generator with input* $\mathcal{G} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ *is called* $(t, q_r, q_n, q_s, \gamma^*, \lambda, \varepsilon)$-leakage-resilient robust*, if for any adversary* $\mathcal{A}$ *running in time* $t$*, that first generates a legitimate distribution sampler* $\mathcal{D}$ *(for the* $\mathcal{D}$-refresh*/* leak-refresh *procedure), that after makes at most* $q_r$ *calls to* $\mathcal{D}$-refresh/leak-refresh*,* $q_n$ *calls to* next-ror/leak-next*, and* $q_s$ *calls to* get-state/set-state *with a leakage bounded by* $\lambda$*, the advantage of* $\mathcal{A}$ *in game* $\mathsf{LROB}(\gamma^*, \lambda)$ *is at most* $\varepsilon$*.*

## 4 New Construction

In this section, we show how to modify the original construction of [10] to achieve the robustness together with the resistance against side-channel attacks.

### 4.1 Original Construction

We first recall the *robust* PRNG construction of [10], named $\mathcal{G}$. It makes use of a $(t, \varepsilon)$-secure pseudo-random generator (PRG) $\mathbf{G} : \{0,1\}^m \to \{0,1\}^{n+\ell}$. The seed is a pair $(X, X')$, $n$ is the state length, $\ell$ is the output length, and $p = n$ is the input length. This construction uses multiplication because it gives a proven seeded extractor that accumulates entropy, which we do not know how to do with a hash function. Plus, it is more efficient:

- $\mathsf{setup}()$ outputs $\mathsf{seed} = (X, X') \leftarrow \{0,1\}^{2n}$;
- $S' = \mathsf{refresh}(S, I; X) = S \cdot X + I$, where all operations are over $\mathbb{F}_{2^n}$;

– $(S', R) = \mathsf{next}(S; X') = \mathbf{G}(U)$, where $U = [X' \cdot S]_1^m$, the truncation of $X' \cdot S$.

Unfortunately, even a secure PRG is not enough to resist information leakage. As shown below, the instantiation proposed in [10] is vulnerable to side-channel attacks. However, with a secure *and* leakage-resilient PRG, we prove that the whole construction remains secure even in the presence of leakage.

### 4.2 Limitations of the Original Construction

In the original paper [10], $\mathbf{G}$ is instantiated with the pseudo-random function AES in counter mode with the truncated product $U$ as the secret key. Depending on the parameters, several calls to the PRF are required. We show hereafter that when the implementation is leaking, this construction faces vulnerabilities.

As shown in [6], on Figures 8 (left) and 13 (right) for software implementations of AES and on Figure 10 (left) for hardware implementations of AES, several calls to AES with known inputs and one single secret key may lead to very efficient side-channel attacks that can help to recover the secret key. Because of the numerous executions of AES with the same key, one essentially performs a differential power analysis (DPA) attack. Then, for the above construction, during a leak-next, even with a safe state, the DPA can reveal the secret key of the internal AES, that is also used to generate the new internal state from public plaintexts. This internal state, after the leak-next, can thus be recovered, whereas it is considered as safe in the security game. A next-ror challenge can then be easily broken.

Furthermore, even if we only make a few calls with the same key, with a counter as input, the adversary can predict future randomness. This vulnerability applies to AES with predictable inputs. As determined by the security games, the adversary chooses a leakage function $f$ to further collect the leakage during the product and the truncation between the internal state $S$ and the public seed $X'$. Assume that this function is $f(S, X') =$

$$\left[\mathsf{AES}_{\left(\left[X' \cdot \left(\mathsf{AES}_{[X' \cdot S]_1^m}(C_0) || \ldots || \mathsf{AES}_{[X' \cdot S]_1^m}(C_0 + \lceil \frac{n}{m} \rceil - 1)\right)\right]_1^m\right)} \left(C_0 + \left\lceil \frac{n+\ell}{m} \right\rceil\right)\right]_1^\lambda$$

with $C_0$ an integer arbitrarily chosen by the attacker. With this leakage function set, the adversary can make a set-state-call and fix the counter $C$ to $C_0$. Indeed, this counter is a part of the global internal state which can be compromised by the adversary. Following this compromission, sufficient calls to $\mathcal{D}$-refresh are made to refresh $S$ so that its entropy increases above the threshold $\gamma^*$. Then, the attacker can ask a leak-next-query and gets back the leakage $f(S, X')$ described above. Eventually, the attacker asks a challenge next-ror-query, and either gets the real output or a random one. The $\lambda$ bits it got from the leakage are exactly the first $\lambda$ bits of the real output. The attacker has consequently a significant advantage in the next-ror challenge.

### 4.3 New Assumption

We slightly modify the requirements of [10] on the PRG $\mathbf{G}$, to keep the PRNG secure even in the presence of leakage: The PRG $\mathbf{G} : \{0,1\}^m \to \{0,1\}^{n+\ell}$ instantiated with the truncated product $U = [X' \cdot S]_1^m$ is now required to be a $(\alpha, \lambda)$-leakage-resilient and $(t, \varepsilon)$-secure PRG according to Definition 6. In that definition, $\lambda$ denotes the leakage during one execution of $\mathbf{G}$, and $\alpha$ is the expected entropy of the output, even given the leakage.

**Definition 6.** *A PRG $\mathbf{G} : \{0,1\}^m \to \{0,1\}^N$ is $(\alpha, \lambda)$-leakage-resilient and $(t, \varepsilon)$-secure if it is first a $(t, \varepsilon)$-secure PRG, but in addition, for any adversary $\mathcal{A}$, running within time $t$, that first outputs a leakage $f$ with $\lambda$-bit outputs, there exists a source $\mathcal{S}$ that outputs couples $(L, T) \in \{0,1\}^\lambda \times \{0,1\}^N$, so that the entropy of $T$, conditioned on $L$ being greater than $\alpha$, and the*

*advantage with which $\mathcal{A}$ can distinguish $(f(\mathcal{U}_m), \mathbf{G}(\mathcal{U}_m))$ from $(L, T)$ is bounded by $\varepsilon$. Note that $f(\mathcal{U}_m)$ denotes the information leakage generated by $f$ during this execution of $\mathbf{G}$ (on the inputs at the various atomic steps of the computation, that includes $\mathcal{U}_m$ and possibly some internal values).*

This definition ensures that for one execution of $\mathbf{G}$, its output is indistinguishable from a source of min-entropy $\alpha$, with a leakage of size $\lambda$ on the input of $\mathbf{G}$.

### 4.4 Security Analysis

Theorem 7 shows that the PRNG $\mathcal{G}$ is leakage-resilient robust.

**Theorem 7.** *Let $m$, $n$, $\alpha$, and $\gamma^*$ be integers, such that $n > m$ and $\alpha > \gamma^*$, and $\mathbf{G} : \{0,1\}^m \to \{0,1\}^{n+\ell}$ an $(\alpha + \ell, \lambda)$-leakage-resilient and $(t, \varepsilon_{\mathbf{G}})$-secure PRG. Then, the PRNG $\mathcal{G}$ previously defined and instantiated with $\mathbf{G}$ is $(t', q_r, q_n, q_s, \gamma^*, \lambda, \varepsilon)$-leakage-resilient robust where $t' \approx t$, after at most $q = q_r + q_n + q_s$ queries, where $q_r$ is the number of $\mathcal{D}$-refresh/ leak-refresh-queries, $q_n$ the number of next-ror/leak-next-queries, and $q_s$ the number of get-state/set-state-queries, where $\varepsilon \leq q q_n \cdot \left( ({q_r}^2 + 1) \cdot \varepsilon_{ext} + 3\varepsilon_{\mathbf{G}} \right)$ and $\varepsilon_{ext} = \sqrt{2^{m+1-\delta}}$ for $\delta = \min\{n - \log q_r, \gamma^* - \lambda\}$.*

To prove Theorem 7, we need to adapt the notions of *recovering* and *preserving* introduced in [10] to also capture information leakage. We then prove an intermediate result (Theorem 13, in Appendix A.1) which states that the combination of recovering and preserving, both with leakage, imply leakage-resilient robustness. Finally, we show that the PRNG $\mathcal{G}$ satisfies both the recovering security with leakage and the preserving security with leakage. Full details of the proof are given in Appendix A.2.

## 5 Instantiations of the PRG G

In the previous section, we explained that the original instantiation in [10] was vulnerable to side-channel attacks, and needs a stronger PRG $\mathbf{G}$, namely a leakage-resilient PRG which takes as input a perfectly random $m$-bit string $U$, and generates an $(n+\ell)$-bit output $T = (S, R)$ that looks random. Even in case of leakage, $S$ should have enough entropy. In this section, we first discuss the use of existing primitives for such a leakage-resilient PRG $\mathbf{G}$. Then, we propose a new concrete instantiation that may achieve better performances in specific scenarios by taking advantage of the PRNG design. Eventually, we provide a security analysis of our solution and we implement it to give some benchmarks.

### 5.1 Existing Constructions

To instantiate the PRG $\mathbf{G}$, we need a leakage-resilient construction which can get use of a bounded part of the internal state. We recall here two leakage-resilient constructions which can be tweaked to fit these requirements at a reasonable cost. The first one is a binary tree PRF introduced by Faust, Pietzrak and Schipper at CHES 2012 [15] and the second one is a sequential PRNG with minimum public randomness proposed by Yu and Standaert at CT-RSA 2013 [25]. We voluntarily ignore the chronological order and start the description with the second instantiation since it will be used to complete the first one.

**Sequential PRNG from [25].** The PRNG of Yu and Standaert comes with an internal state made of two randomly chosen values : a secret key $K_0 \in \{0,1\}^\mu$ and a public seed $s \in \{0,1\}^\mu$. The construction is made of two stages. In the upper stage, a (non leakage-resilient) generator $\mathbf{F}'$ is processed in counter mode to expand the seed $s$ into uniformly random values $p_0, p_1, \ldots$. In the lower stage, a (non leakage-resilient) PRF $\mathbf{F}$ generates outputs with public values $p_i$ and updates the secret so it is never used more than twice. The parameter $s$ can be included in our PRNG seed (under the notation $X''$) since it shares the same properties than $X$ and $X'$.

However, the current counter is varying and thus need to be stored in the deterministic part of the internal state. In the proof of [25], the counter is implicitly required to be different at each use since the public values $p_i$ need to be independent. But in our model of leakage-resilient robustness, the deterministic part of the internal state can be definitively compromised. Attacker could, in this case, set the counter to a previous value, making the public $p_i$ not independent anymore. To thwart this issue, we suggest to extend the internal state so that the truncated part of full entropy can contain both the secret key $K_0$ and a uniformly random counter used only for a single execution of next. Hence, no parameter can be compromised and we are back to the context of the original proof. The only difference in the security comes from the probability of collisions when using a uniformly random counter at each call.

This two-stage instantiation is illustrated in Figure 3. One can note that the input $U$ is split in two slices, to initiate the secret key $K_0$ and the counter $C$, each of size $\mu$. In order to relate these parameters with the parameters of our PRNG from Section 4 that provides an $m$-bit random string $U$ as input to the PRG $\mathbf{G}$, and wants to receive back an $N$-bit string, where $N = n + \ell$, that is $\kappa = N/\mu$ blocks generated with $\kappa$ keys. The $\kappa$ blocks of output and new internal state are all generated using $2\kappa - 1$ calls to $\mathbf{F}'$ and $2\kappa - 1$ calls to $\mathbf{F}$.

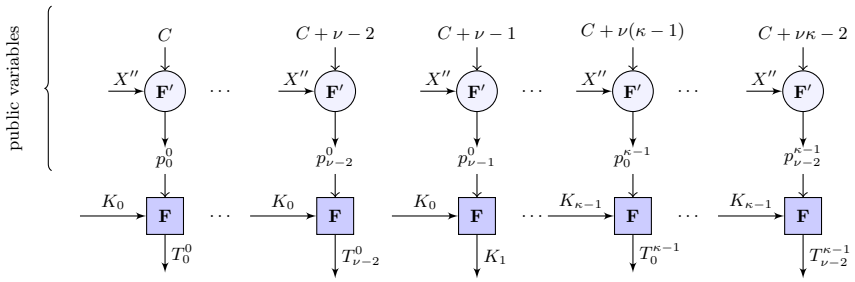


**Fig. 3.** Instantiation of Generator $\mathbf{G}$ from [25] with Random Input $U = (C, K_0)$

**Tweaked Binary Tree PRF from [15].** The second solution was proposed by Faust et al. at CHES 2012 [15]. Thanks to its structure of binary tree, it requires less calls to $\mathbf{F}$ and consequently overtakes the performances of the first solution. However, the original construction does not provide sources for the required randomness. That is why we suggest to use the same upper stage than [25] recommended in and proven secure in [25,2]. Plus, with the advantageous reuse of the two same random values at each layer, the number of calls to $\mathbf{F}'$ is limited to $2\log_2(\kappa)$ which is less than the the number of $\mathbf{F}$ calls. Eventually, for the reasons depicted above, we also need to use a uniformly random counter, updated at each call to next. The tweaked construction is depicted in Figure 4.

### 5.2 New Proposal

To thwart the first attack of Section 4.2, we still make use of a PRF with a regular re-keying whose frequency depends on the parameters of the inherent device. To thwart the second attack and for the needs of the proof, we continue to make use of unpredictable values as inputs of the PRF. Combining these two solutions, we get close to the two stages exhibited by existing constructions. However, while we keep the same upper stage, we modify the lower one to try to achieve better performances in function next. The latter still makes several calls to the PRF $\mathbf{F} : \{0,1\}^\mu \times \{0,1\}^\mu \to \{0,1\}^\mu$, with public but uniformly distributed inputs and $\kappa$ distinct secret keys (as in the second existing construction). However, the secret keys are all directly extracted from the input value $U = [X' \cdot S]_1^m$ together with the counter $C$. In this way, there is no need to derive

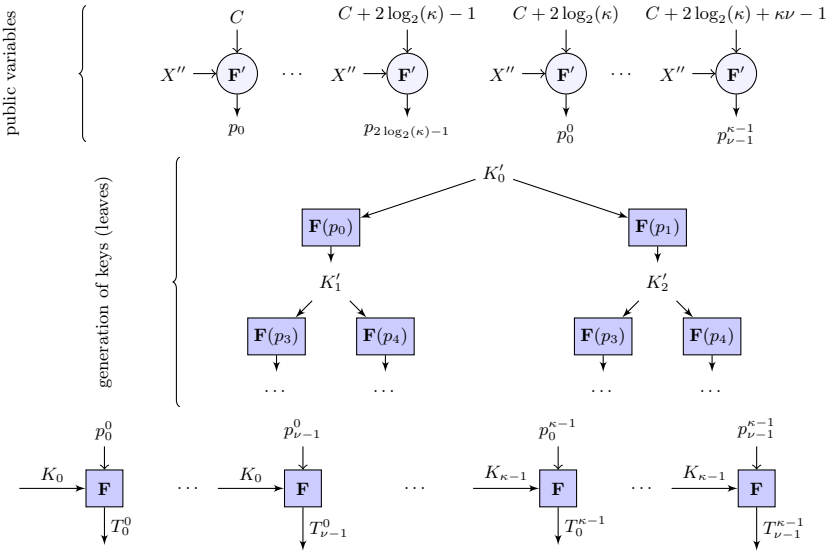


**Fig. 4.** Instantiation of Generator **G** from [15] with Random Input $U = (C, K'_0)$

the next keys in function next but the internal state is much more larger: the extracted value $U$ is of length $(\kappa + 1)\mu$ instead of $2\mu$ in previous constructions. The precise security requirements are formalized in Definition 8 and the performances comparison with existing solutions is given in Table 1, Section 5.3.

**Definition 8 (leakage-resilient PRF).** *A PRF* $\mathbf{F} : \{0,1\}^\mu \times \{0,1\}^\mu \to \{0,1\}^\mu$ *is* $(\alpha, \lambda)$*-leakage-resilient and* $(t, q, \varepsilon)$*-secure if it is a* $(t, q, \varepsilon)$*-PRF and if, for any adversary* $\mathcal{A}$*, running within time* $t$*, that first outputs a leakage* $f$ *with* $\lambda$*-bit outputs, there exists a source* $\mathcal{S}$ *that outputs* $(L_i, P_i, T_i)_i \in (\{0,1\}^\lambda \times \{0,1\}^\mu \times \{0,1\}^\mu)^q$*, with a uniform distribution for the* $P$*'s, so that the entropy of* $(T_i)_i$*, conditioned to* $(L_i, P_i)_i$*, is greater than* $\alpha$*, and the advantage with which* $\mathcal{A}$ *can distinguish the tuple* $(f(K_i, P_i), P_i, \mathbf{F}_K(P_i))_i$ *from* $(L_i, P_i, T_i)_i$ *is bounded by* $\varepsilon$*.*

When $q$ is large, such a requirement implies security against DPA, but when $q$ is small only SPA is available which is limited in practice. Such an assumption is implicitly done in [25] with $\alpha = \mu - \lambda$, since the loss of entropy in the output corresponds to the leakage. This new two-stage instantiation is illustrated in Figure 5. The input $U$ is split in $\kappa + 1$ slices, to initiate the $\kappa$ keys $\{K_i\}_{0 \leqslant i \leqslant \kappa - 1}$ and the counter $C$, each of size $\mu$. Theorem 9 shows that this proposal achieves the

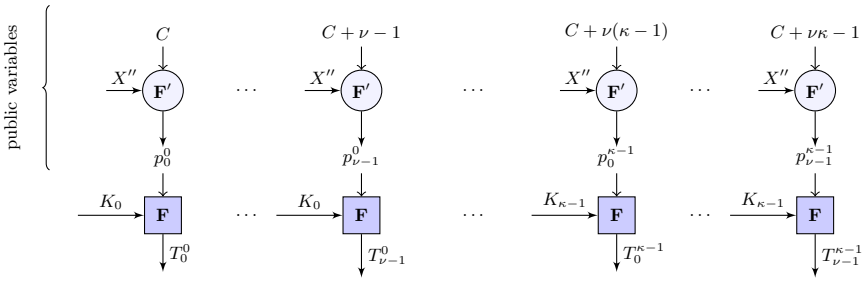


**Fig. 5.** New Instantiation of Generator **G** with Random Input $U = (C, K_0, \ldots, K_{\kappa-1})$

security requirements in Definition 6.

**Theorem 9.** *Let* $\mu$ *and* $\kappa$ *be paramaters such that* $\nu\kappa\mu = N$*. Let* $\mathbf{F} : \{0,1\}^\mu \times \{0,1\}^\mu \to \{0,1\}^\mu$ *be a* $(\alpha/\kappa, \lambda)$*-leakage-resilient and* $(t, \nu, \varepsilon_{\mathbf{F}})$*-secure PRF and* $\mathbf{F'} : \{0,1\}^\mu \times \{0,1\}^\mu \to \{0,1\}^\mu$ *be*

*a* $(t, q\nu\kappa, \varepsilon_{\mathbf{F}'})$*-secure PRF, where* $q$ *is a bound on the global number of executions of* $\mathbf{G}$*. The instantiation proposed for* $\mathbf{G}$ *in Section 5.2 with* $\mathbf{F}$ *and* $\mathbf{F}'$ *provides an* $(\alpha, \lambda)$*-leakage-resilient and* $(t, \varepsilon_{\mathbf{G}})$*-secure PRG where* $\varepsilon_{\mathbf{G}} \leq \kappa \cdot \varepsilon_{\mathbf{F}} + \varepsilon_{\mathbf{F}'} + q^2\nu\kappa/2^{\mu}$*.*

In the proposal, each call to $\mathbf{G}$ makes $\nu\kappa$ calls to the PRF $\mathbf{F}$: $\kappa$ keys are used at most $\nu$ times. The inputs of $\mathbf{F}$ are generated by $\mathbf{F}'$ with the key $X''$ (randomly set in $\mathsf{seed}$) on a counter $C$ randomly initialized, and then incremented for each $\mathbf{F}'$ call in an execution of $\mathbf{G}$. The details of the proof which uses a similar argument as [25] in the minicrypt world [16], can be found in Appendix A.3. However, for the global security, we need all the intermediate values $(p_j^i)$ to be distinct and unpredictable to avoid the aforementioned attack. We thus require $\mathbf{F}'$ to be secure after $q_n\nu\kappa$ queries and the inputs to be all distinct: by setting the $\log(\nu\kappa)$ least significant bits of $C$ to zero, we just have to avoid collisions on the $\mu - \log(\nu\kappa)$ most significant bits for the $q_n$ queries. The probability of collision is thus less than ${q_n}^2\nu\kappa/2^{\mu}$ and can appear once and for all in the global security:

**Corollary 10.** *Let us consider parameters* $n$*,* $m$*, and* $\ell$ *in the construction of the PRNG with input* $\mathcal{G}$ *from Section 4.1, using the generator* $\mathbf{G}$ *as described in this section. Let* $\mu$ *and* $\kappa$ *be paramaters such that* $\nu\kappa\mu = n + \ell$*, and* $\alpha > \gamma^*$*. Let* $\mathbf{F} : \{0,1\}^{\mu} \times \{0,1\}^{\mu} \to \{0,1\}^{\mu}$ *be a* $(\alpha/\kappa, \lambda)$*-leakage-resilient and* $(t, \nu, \varepsilon_{\mathbf{F}})$*-secure PRF, and* $\mathbf{F}' : \{0,1\}^{\mu} \times \{0,1\}^{\mu} \to \{0,1\}^{\mu}$ *be a* $(t, q_n\nu\kappa, \varepsilon_{\mathbf{F}'})$*-secure PRF. Then,* $\mathcal{G}$ *is* $(t, q_r, q_n, q_s, \gamma^*, \lambda, \varepsilon)$*-leakage-resilient robust after at most* $q = q_r + q_n + q_s$ *queries, where* $q_r$ *is the number of* $\mathcal{D}$-$\mathsf{refresh}$/$\mathsf{leak\text{-}refresh}$*-queries,* $q_n$ *the number of* $\mathsf{next\text{-}ror}$/$\mathsf{leak\text{-}next}$*-queries, and* $q_s$ *the number of* $\mathsf{get\text{-}state}$/$\mathsf{set\text{-}state}$*-queries, where* $\varepsilon \leq qq_n \cdot \left((q_r{}^2 + 1) \cdot \sqrt{2^{m+1-\delta}} + 3(\kappa \cdot \varepsilon_{\mathbf{F}} + \varepsilon_{\mathbf{F}'})\right) + {q_n}^2\nu\kappa/2^{\mu}$*, for* $\delta = \min\{n - \log q_r, \gamma^* - \lambda\}$*.*

It seems reasonable to have $(\alpha, \lambda)$-leakage resilience with $\alpha = n + \ell - \nu\kappa\lambda$: with a large $\gamma^*$, $\varepsilon$ can be made small.

### 5.3 Practical Analysis: Implementation and Benchmarks

We present some benchmarks of the construction of [10] and the three instantiations. Since our leakage-resilient construction is based on [10], we use the latter as a reference when measuring efficiency. Thus, we simply implemented them on an Intel Core i7 processor to show that the new property does not significantly impact the performances. This is mainly due to the use of SPA-resistant AES implementations instead of DPA-resistant (*e.g.*, masked) ones. We used the same public cryptographic libraries that in [10] and to achieve a similar security level as the construction of [10], our experiments show that the tweaked binary tree construction is only less than 4 times slower.

**General Benchmarks.** We recall that our construction is based on the construction of [10]: $\mathsf{refresh}(S, I) = S \cdot X + I \in \mathbb{F}_{2^n}$ and $\mathsf{next}(S) = \mathbf{G}(U)$, with $U = [X' \cdot S]_1^m$. In [10], the PRG $\mathbf{G}$ is defined by $\mathbf{G}(U) = \mathsf{AES}_U(0) \| \ldots \| \mathsf{AES}_U(\nu - 1)$, where $\nu$ is the number of calls to $\mathsf{AES}$ with a 128-bit key $U$, and thus $m = 128$. For a security parameter $k = 40$, the security analysis leads to $n = 489$, $\gamma^* = 449$, and $\nu = 5$. To acheive leakage-resilience, we need additional security requirements for the PRG $\mathbf{G}$. The three instantiations split $\mathbf{G}$ between two PRFs $\mathbf{F}$ and $\mathbf{F}'$, where $\mathbf{F}$ is used with public uniformly distributed inputs and $\kappa$ different secret keys. In the existing constructions, a first key is extracted from the truncated product $U$ and the other ones are derived through a re-keying process. In the new instantiation, all the secret keys are extracted from $U$. The public inputs of $\mathbf{F}$ are generated by the PRF $\mathbf{F}'$ in counter mode, with a secret initial value for the counter also extracted from $U$: $m = 2 \cdot 128$ for the existing constructions or $m = 128(\kappa + 1)$ for the new instantiation if both $\mathbf{F} = \mathbf{F}' = \mathsf{AES}$ with 128-bit keys. To provide the security bounds of the three constructions, we need to fix the security bounds of functions $\mathbf{F}$ and $\mathbf{F}'$. As far as we know, the best key recovery attacks on $\mathsf{AES}$ without leakage [9] require a complexity of $2^{162.1}$ with $2^{88}$ data. However, our functions being executed at most twice (resp.

6 times) with the same secret keys for $2^{-40}$ security (resp. for $2^{-64}$ security), such a complexity is unreachable. As for the leakage, we give the adversary $\lambda$ bits of useful information by leaking query. Nevertheless, until now it remains unclear how these $\lambda$ bits of information in a single trace may reduce the security bound of the AES. In [23] for instance, the authors show that a single trace on the AES might give the adversary all the required knowledge to recover the secret key, namely, when a sufficient number of noisy Hamming Weight values are available. But summing the useful information of these noisy Hamming Weight values would give a very large $\lambda$ for which we cannot guarantee anything. However, we can expect either a larger amount of noise, a desynchronization of the traces or a low leaking from the inherent component which would result in a reasonable value for $\lambda$. In this case, we can fix $\varepsilon_{\mathbf{F}} = \varepsilon_{\mathbf{F}'} \approx 2^{-127}$. The resulting security bounds are given in Table 1 with the size $n$ of the internal state, the number of 128 or 256-bit keys and the number of AES calls in function next, for $2^{-40}$ and $2^{-64}$ security.

**Table 1.** Security bounds and complexity of the three instantiations

| Refs | Security Bound $\epsilon_G$ | $2^{-40}$ Security | | | $2^{-64}$ Security | | |
|---|---|---|---|---|---|---|---|
| | | $n$ | keys (128) | AES calls | $n$ | keys (256) | AES calls |
| [25] | $\kappa\epsilon_F + \epsilon'_F + q^2(\nu\kappa - 1)/2^\mu$ | 768 | 7 | 26 | 1152 | 5 | 30 |
| [15] | $2\kappa\epsilon_F + \epsilon'_F + q^2(\nu\kappa + 2\log_2(\kappa))/2^\mu$ | 896 | 4 | 20 | 1408 | 4 | 24 |
| New | $\kappa\varepsilon_{\mathbf{F}} + \varepsilon_{\mathbf{F}'} + q^2\nu\kappa/2^\mu$ | 1408 | 6 | 24 | 1792 | 5 | 30 |

The best instantiation in terms of complexity is the construction from [15]. This is not surprising considering the advantageous binary shape of this function. However, if we relax the security assumptions on the AES with $\varepsilon_F = \varepsilon'_F = 2^{-126}$, the conditions of the security proof are not met and therefore we cannot guarantee its security based on Corollary 10. In these specific cases, our construction seems to be the best one to use since it guarantees that the conditions of the security proof are met. Note that for $2^{-64}$ security, as explained in Section 5.3, we cannot get a provable security with 128-bits input blocks, and we need $\varepsilon_{\mathbf{F}}$ and $\varepsilon_{\mathbf{F}'}$ to be smaller than $2^{-200}$, and then use AES with 256-bit keys. Since the implementation built from [15] appears to be the best one in the general case, we implement it to compare it with the instantiation of [10]. As in [10], we use fb_mul_lodah and fb_add from RELIC open source library [4], extended with the necessary fields ($\mathbb{F}_{2^{489}}$, defined with $X^{489} + X^{83} + 1$ and $\mathbb{F}_{2^{896}}$, defined with $X^{896} + X^7 + X^5 + X^3 + 1$). We use public functions aes_setkey_enc and aes_crypt_ctr from PolarSSL open source library [1]. As in [10], we measure the number of CPU cycles for a recovering process and a key generation process. The CPU cycles count is done using ASM instruction RDTSC, our C code is optimized with O2 flag. We simulate a full recovery of the PRNG for [15] and [10] implementations, with an input containing one bit of entropy per byte. Then, 8 inputs of size 489 bits are necessary to recover from a compromise for [10], whereas, for [15], 8 inputs of size 896 bits are necessary. Then we simulate the generation of 2048-bit keys that each requires 16 calls to next, as every call outputs 128 bits. Figure 6 gives the numbers of CPU cycles for 100 complete recovering experiments (left) and 100 key generations (right) for [10] and [15]. Both processes require on average 4 times less CPU cycles to perform for [15] implementation than for [10] implementation.

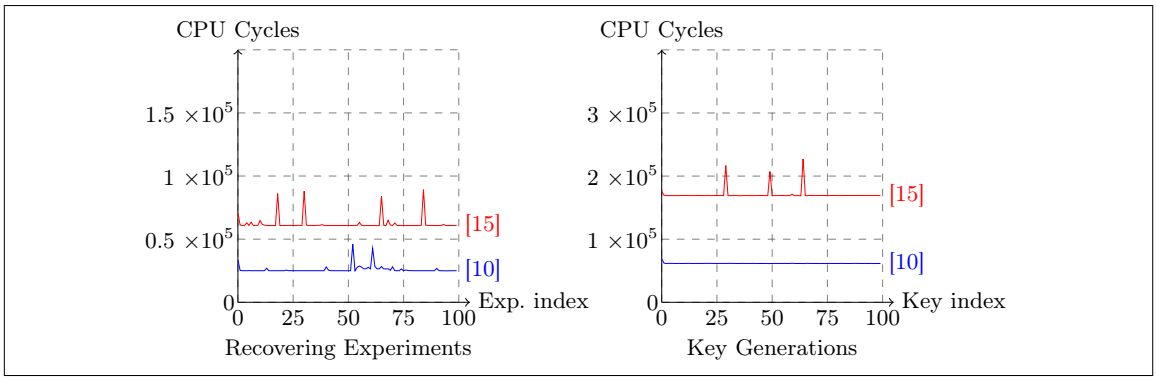


**Fig. 6.** Benchmarks Between [15] and [10]

**The Tweaked Binary Tree Instantiation.** We first recall the constraints (similar to Corollary 10): the quality of the pseudo-random number generator is measured by

$$\varepsilon \leq qq_n \cdot \left( ({q_r}^2 + 1) \cdot \sqrt{2^{m+1-\delta}} + 3(2\kappa \cdot \varepsilon_{\mathbf{F}} + \varepsilon_{\mathbf{F}'}) \right) + {q_n}^2 (2\log_2(\kappa) + \nu\kappa)/2^{\mu},$$

for $\delta = \min\{n - \log q_r, \gamma^* - \lambda\}$. With $q_r = q_n = q_s = 2^k$, we get:

$$\begin{aligned} \varepsilon &\leq 3 \cdot 2^{2k} \cdot \left( (2^{2k} + 1) \cdot \sqrt{2^{m+1-\delta}} + 3(2\kappa \cdot \varepsilon_{\mathbf{F}} + \varepsilon_{\mathbf{F}'}) \right) + (2\log_2(\kappa) + \nu\kappa) \cdot 2^{2k}/2^{\mu} \\ &\leq \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4 \end{aligned}$$

with

- $\varepsilon_1 = 2^{4k+2+(m+1-\delta)/2}$;
- $\varepsilon_2 = 18\kappa \cdot 2^{2k} \cdot \varepsilon_{\mathbf{F}}$;
- $\varepsilon_3 = 9 \cdot 2^{2k} \cdot \varepsilon_{\mathbf{F}'}$;
- $\varepsilon_4 = 2^{2k-\mu} \cdot (2\log_2(\kappa) + \nu\kappa)$.

**$2^{-v}$ Security.** With $m = 256$, $\mu = 128$, $\varepsilon_{\mathbf{F}} = \varepsilon_{\mathbf{F}'} \approx 2^{-127}$:

- $\varepsilon_1 < 2^{-v}$, as soon as $8k + 2v + 5 + m < \delta$, which is verified for $n > 9k + 2v + 5 + m$ and $\gamma^* > n + \lambda - k$;
- $\varepsilon_2 < 2^{-v}$, as soon as $2k + v < 127 - \log_2(18\kappa)$;
- $\varepsilon_3 < 2^{-v}$, as soon as $2k + v < 127 - \log_2(9) < 123$;
- $\varepsilon_4 < 2^{-v}$, as soon as $2k + v < 128 - \log_2(2\log_2(\kappa) + \nu\kappa)$.

**$2^{-40}$ Security.** For $k = v = 40$, the constraint on $\varepsilon_3$ is satisfied. The constraints on $\varepsilon_1$ are satisfied as soon as $n > 701$ and $\gamma^* > n + \lambda - 40$. With $\nu = 2$, we need $n = 256\kappa - 128 > 701$ and thus $\kappa = 4$, which ensures that the contraints on $\varepsilon_2$ and $\varepsilon_4$ are satisfied . Finally, $n = 896$ and $\gamma^* = 858$ for $\lambda \approx 2$.

**$2^{-64}$ Security.** Unfortunately, for $k = v = 64$, one cannot get a provable security with the size of the input block $\mu = 128$, because of the collisions on the counters. In order to increase the size of the input blocks, one can XOR PRPs to get a PRF on larger inputs [17]. This makes $\varepsilon_4$ negligible: $2^{2k-2\mu} = 2^{-128}$, and thus the factor $\nu\kappa$ will not affect it. On the other hand, to make $\varepsilon_2$ and $\varepsilon_3$ small enough, we need $\varepsilon_{\mathbf{F}}$ and $\varepsilon_{\mathbf{F}'}$ to be smaller than $2^{-200}$, and then use AES with 256-bit keys. But then we have to use the same key 6 times in order to extract 384 bits (see a 3-block extraction in Figure 7), where $\kappa$ keys are used $\nu = 6$ times, and two counters $C_0$ and $C_1$ are extracted: $m = 3 \cdot 128 = 384$, $n = 3 \times 128 \times \kappa - 128 = 384\kappa - 128$. As for the constraint on $\varepsilon_1$, we need $384\kappa > 1221$. We can take $\kappa = 4$. Then, $n = 1408$ and $\gamma^* = 1346$.

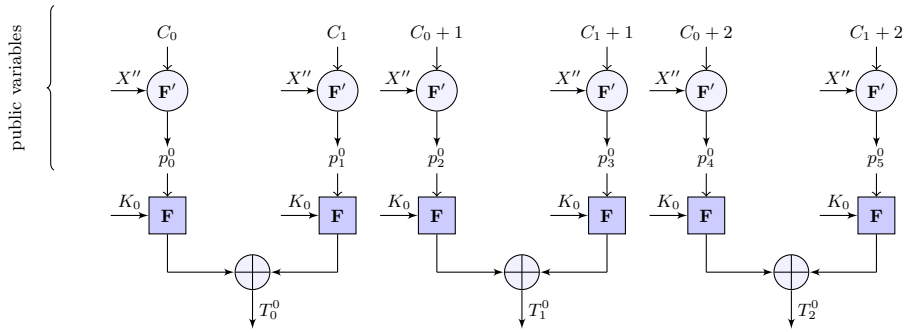


**Fig. 7.** Example of Instantiation of Generator **G** for Higher Security Bounds

## 6 Conclusion

We have put forward a new property for PRNGs with input, that captures security in a setting where partial sensitive information may leak. Then, we have tweaked the PRNG with input proposed by Dodis *et al.* to meet our new property of leakage-resilient robustness. Finally, we have proposed three secure instantiations of the new PRNG with input including a new one which provide the same level of security as the construction of [10] for a limited additional cost in efficiency and size.

As further work, the security bounds could be made tighter if the construction was proven robust with leakage without going through the preserving-with-leakage and recovering-with-leakage steps. Another interesting future work would be to implement the construction on constrained devices.

*Acknowledgments:* This research was supported in part by the French ANR-12-JS02-0004 ROMAnTIC Project and the French ANR-10-SEGI-015 PRINCE Project.

## References

1. PolarSSL is an open source and commercial SSL library licensed by Offspark B.V. https://polarssl.org.
2. Michel Abdalla, Sonia Belaïd, and Pierre-Alain Fouque. Leakage-resilient symmetric encryption via re-keying. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 471–488. Springer, August 2013.
3. Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, January 2005.
4. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. http://code.google.com/p/relic-toolkit/.
5. Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05*, pages 203–212. ACM Press, November 2005.
6. Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? Cryptology ePrint Archive, Report 2014/053, 2014. `http://eprint.iacr.org/2014/053`.
7. Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? *Cryptography and Communications*, 7(1):163–184, 2015.
8. Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997.
9. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 344–371. Springer, December 2011.
10. Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergnaud, and Daniel Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 647–658. ACM Press, November 2013.

11. Yevgeniy Dodis, Adi Shamir, Noah Stephens-Davidowitz, and Daniel Wichs. How to eat your entropy and have it too - optimal recovery strategies for compromised RNGs. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 37–54. Springer, August 2014.
12. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, May 2014.
13. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *49th FOCS*, pages 293–302. IEEE Computer Society Press, October 2008.
14. Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 343–360. Springer, February 2010.
15. Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. Practical leakage-resilient symmetric cryptography. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 213–232. Springer, September 2012.
16. Russell Impagliazzo. A Personal View of Average-Case Complexity. In *Structure in Complexity Theory Conference*, pages 134–147, 1995.
17. Stefan Lucks. The sum of PRPs is a secure PRF. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 470–484. Springer, May 2000.
18. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 278–296. Springer, February 2004.
19. Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 462–482. Springer, April 2009.
20. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, May 2013.
21. Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2006.
22. François-Xavier Standaert, Olivier Pereira, and Yu Yu. Leakage-resilient symmetric cryptography under empirically verifiable assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 335–352. Springer, August 2013.
23. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 282–296. Springer, December 2014.
24. Yu Yu, François-Xavier Standaert, Olivier Pereira, and Moti Yung. Practical leakage-resilient pseudorandom generators. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10*, pages 141–151. ACM Press, October 2010.
25. Yu Yu and François-Xavier Standaert. Practical leakage-resilient pseudorandom objects with minimum public randomness. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 223–238. Springer, February / March 2013.

# A Appendices

## A.1 Recovering and Preserving Security with Leakage

To prove Theorem 7, we first adapt the notions of *recovering* and *preserving* introduced in [10] to also capture information leakage. The former essentially deals with the capacity for the PRNG to accumulate the entropy from the inputs in the internal state, with the refresh algorithm, and then to recover a safe state even after being compromised. The latter deals with the quality of the internal state, even with adversarially chosen and known inputs. The quality of the internal state will then be measured by the ability of the adversary to distinguish the output randomness (by the next algorithm) from a truly random output, using one next-ror query. Since more oracles are available, contrarily to [10], our security games will be interactive and adaptive: the leak-refresh and leak-next oracles are available during the recovering and preserving sequences, and not just the $\mathcal{D}$-refresh oracle.

**Recovering Security with Leakage.** It considers an adversary that compromises the state to some arbitrary value $S_0$, either by asking for the state (get-state), setting it (set-state) or learning information with the collected leakage or with the output (leak-refresh, leak-next or next-ror) when the internal state is unsafe. Afterwards, sufficient calls to $\mathcal{D}$-refresh are made to increase the entropy estimate $c$ above the threshold $\gamma^*$. This is the *recovering* process, which should make the bit $b$ involved in the next-ror procedure indistinguishable: when the internal state is considered as safe, the output randomness $R$ should look indistinguishable from random. The security game is defined hereafter, where $\mathcal{D}$ is the distribution sampler, and $f = (f_{\mathsf{refresh}}, f_{\mathsf{next}})$ denotes the union of the leakage functions related to the execution of refresh and next, both proposed by the adversary. Note that even if leak-refresh and leak-next-queries are possible, they are not allowed with compromised states, since it would make $c$ drop to 0. In the recovery process, the entropy should almost always increase, but never drop to 0.

1. The challenger generates a seed $\mathsf{seed} \xleftarrow{\$} \mathsf{setup}$ and a bit $b \xleftarrow{\$} \{0,1\}$ uniformly at random. It sets $\sigma_0 = 0$ and for $k = 1, \ldots, q_{\mathcal{D}}$, it computes $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow \mathcal{D}(\sigma_{k-1})$. It then gives back the seed and the values $\gamma_1, \ldots, \gamma_{q_{\mathcal{D}}}$ and $z_1, \ldots, z_{q_{\mathcal{D}}}$ to the adversary.
2. The adversary can ask as many $\mathcal{D}$-refresh queries as it wants, and gets back for each the input $I_k$ generated by the distribution sampler. Then, the adversary outputs a value $S_0 \in \{0,1\}^n$, to be the new internal state.
3. The challenger sets the new internal state to $S_0$, sets $c = 0$, and allows queries to the oracles $\mathcal{D}$-refresh, leak-refresh, and leak-next. These queries are processed, respectively, as $\{S_j = \mathsf{refresh}(S_{j-1}, I_k);$ if $c < \gamma^*$, then $c = c + \gamma_k\}$, $\{L_j = f_{\mathsf{refresh}}(S_{j-1}, I_k, \mathsf{seed}); S_j = \mathsf{refresh}(S_{j-1}, I_k); c = c - \lambda;$ if $c < \gamma^*$, then $c = 0\}$, and $\{L_j = f_{\mathsf{next}}(S_{j-1}, \mathsf{seed}); (S_j, R_j) = \mathsf{next}(S_{j-1});$ if $c < \gamma^*$, then $c = 0$, else $c = \alpha\}$, with the new input $I_k$ provided by the distribution sampler for the $k$-th refresh-query.
   These queries are answered, respectively, by nothing, by the information leakage $L_j$, and by the information leakage $L_j$ together with the randomness $R_j$;
4. Eventually, under the restriction that $c$ never dropped to 0, the challenger sets $(S^{(0)}, R^{(0)}) \leftarrow \mathsf{next}(S)$ and generates $(S^{(1)}, R^{(1)}) \xleftarrow{\$} \{0,1\}^{n+\ell}$. It then gives $(S^{(b)}, R^{(b)})$ to the adversary, together with the next inputs $I_{k+1}, \ldots, I_{q_{\mathcal{D}}}$ (if $k$ was the number of refresh-queries asked up to this point);
5. The adversary $\mathcal{A}$ outputs a bit $b^*$.

In this game, we define the advantage of the adversary $\mathcal{A}$ as $|2\Pr[b^* = b] - 1|$. Note that we restrict our game to executions where $c$ never dropped to 0, but one could have answered independently to $b$ otherwise (e.g., always using $(S^{(0)}, R^{(0)})$).

**Definition 11 (Recovering Security with Leakage).** *A PRNG with input is said* $(t, q_r, q_n, \gamma^*, \lambda, \varepsilon)$*-recovering with leakage if for any adversary* $\mathcal{A}$ *running within time* $t$*, its advantage in the*

*above game with parameters $q_r$ (number of $\mathcal{D}$-refresh and leak-refresh-queries), $q_n$ (number of leak-next-queries), $\gamma^*$, and $\lambda$ is at most $\varepsilon$.*

**Preserving Security with Leakage.** This security notion considers a safe internal state. After several calls to $\mathcal{D}$-refresh and leak-refresh with known (and even chosen) inputs, the internal state should remain safe. An initial state $S_0$ is generated with entropy $n$. Then it is refreshed with arbitrary many calls to either $\mathcal{D}$-refresh or leak-refresh, as long as the leakage does not decrease the entropy below the threshold $\gamma^*$. This is the *preserving* process, which should make the bit $b$ involved in the next-ror procedure indistinguishable: since the internal state is considered as safe, the output randomness $R$ should look indistinguishable from random. The security game is the following, where $\mathcal{D}$ is the distribution sampler, and $f = (f_{\mathsf{refresh}}, f_{\mathsf{next}})$ denotes the union of the leakage functions during the execution of refresh and next, both proposed by the adversary.

1. The challenger generates an initial state $S_0 \xleftarrow{\$} \{0,1\}^n$, a seed $\mathsf{seed} \leftarrow \mathsf{setup}$, and a bit $b \xleftarrow{\$} \{0,1\}$ uniformly at random. It sets $c = n$ and then gives back the seed to the adversary;
2. The adversary $\mathcal{A}$ gets seed and can ask as many queries as it wants to the oracles $\mathcal{D}$-refresh, leak-refresh, and leak-next, but with chosen inputs $I$ to the refresh-queries. These queries are thus processed, respectively, as
   - $\{S_j = \mathsf{refresh}(S_{j-1}, I)\}$,
   - $\{L_j = f_{\mathsf{refresh}}(S_{j-1}, I, \mathsf{seed}); S_j = \mathsf{refresh}(S_{j-1}, I); c = c - \lambda;$ if $c < \gamma^*,$ then $c = 0\}$, and
   - $\{L_j = f_{\mathsf{next}}(S_{j-1}, \mathsf{seed}); (S_j, R_j) = \mathsf{next}(S_{j-1});$ if $c < \gamma^*,$ then $c = 0,$ else $c = \alpha\}$,

   with the input $I$ provided by the adversary. Theses queries are answered, respectively, by nothing, by the information leakage $L_j$, and by the information leakage $L_j$ together with the randomness $R_j$;
3. Eventually, under the restriction that $c$ never dropped to 0, the challenger sets $(S^{(0)}, R^{(0)}) \leftarrow \mathsf{next}(S)$, and generates $(S^{(1)}, R^{(1)}) \xleftarrow{\$} \{0,1\}^{n+\ell}$. It then gives $(S^{(b)}, R^{(b)})$ to the adversary;
4. The adversary $\mathcal{A}$ outputs a bit $b^*$.

As above, we define the advantage of the adversary $\mathcal{A}$ as $|2\Pr[b^* = b] - 1|$, and the restriction that $c$ never dropped to 0 could have been dealt another way.

**Definition 12 (Preserving Security with Leakage).** *A PRNG with input is said $(t, q_r, q_n, \gamma^*, \lambda, \varepsilon)$-preserving with leakage if for any adversary $\mathcal{A}$ running within time $t$, its advantage in the above game with parameters $q_r$, $q_n$, $\gamma^*$, and $\lambda$ is at most $\varepsilon$.*

From these two security notions, one can prove the following theorem, inspired from [10].

**Theorem 13.** *If a PRNG with input is $(t, q_r, q_n, \gamma^*, \lambda, \varepsilon_r)$-recovering with leakage and $(t, q_r, q_n, \gamma^*, \lambda, \varepsilon_p)$-preserving with leakage then it is also $(t', q_r, q_n, q_s, \gamma^*, \lambda, q_n \cdot (\varepsilon_r + q \cdot \varepsilon_p))$-leakage-resilient robust where $t' \approx t$, where the adversary can ask at most $q = q_r + q_n + q_s$ queries, where $q_r$ is the number of calls to $\mathcal{D}$-refresh/leak-refresh, $q_n$ the number of calls to next-ror/leak-next, and $q_s$ the number of calls to get-state/set-state.*

*Proof.* This proof follows the one from [10]. It splits the leakage-resilient robustness game in preserving with leakage steps and recovering with leakage steps.

*Queries.* In the game of leakage-resilient robustness, we term next-*queries* the calls to the oracle next-ror. Since $q_n$ is a bound on the next-ror and leak-next queries, this is also a bound on the next-queries. Actually, there are unsafe/compromised next-queries, when the internal state is unsafe and so the entropy estimate $c$ is below the threshold $\gamma^*$ before the query and reset to 0 after the query, and safe/uncompromised next-queries, when the internal state is safe before the next-ror-query. For uncompromised next-queries, the output randomness $R$ should look indistinguishable from truly random, while for compromised next-queries, there is no guarantee. As in [10], we split the uncompromised next-queries in two sets: the *preserving with leakage* queries, if the entropy estimate is above the threshold $\gamma^*$ since the previous next-query; and the *recovering with leakage*

queries, if the entropy estimate dropped below the threshold $\gamma^*$ and has thus been reset to 0. For the latter queries, we reuse the notion of mRED (most recent entropy drain) to define the most recent query to one of the oracles get-state, set-state, leak-refresh or leak-next that reset $c$ to 0.

*Sequence of Games.* Let us now define the sequence of games. Let game $G_0$ be the initial real-or-random game. Game $G_i$ modifies the first $i$ uncompromised next-queries by outputting a uniformly random $R$, and by setting the internal state $S$ uniformly at random. We note that $G_{q_n}$ is then independent of $b$, since all the safe next-ror-queries are answered randomly, while the unsafe next-ror-queries are anyway always answered by the real value of $R$. Game $G_{i+1/2}$ acts according to the nature of the $(i+1)^{th}$ uncompromised next-query: If it is preserving, then it acts as $G_{i+1}$, if it is recovering, it acts as $G_i$. Then, as in [10], one can show that:

- If the PRNG is $(t, q_r, q_n, \gamma^*, \lambda, \varepsilon_p)$-preserving with leakage, then $|\Pr[G_i = 1] - \Pr[G_{i+1/2} = 1]| \leq \varepsilon_p$. This applies thanks to our security games that make $c$ evolve the same way if the entropy of the input is assumed to be zero, when $c$ is above $\gamma^*$;
- If the PRNG is $(t, q_r, q_n, \gamma^*, \lambda, \varepsilon_r)$-recovering with leakage, then $|\Pr[G_{i+1/2} = 1] - \Pr[G_{i+1} = 1]| \leq (q_{ri} + q_{ni} + q_{si}) \cdot \varepsilon_r$, where $q_{ri}$, $q_{ni}$, and $q_{si}$ are numbers of refresh, next and state queries between the $i^{th}$ uncompromised next-queries and the $(i+1)^{th}$ uncompromised next-query. Again, this applies since $c$ drops to zero as soon as it decreases below $\gamma^*$,and so this cannot happen in a recovering phase.

Combining both results, we have $|\Pr[G_0 = 1] - \Pr[G_{q_n} = 1]| \leq q_n \cdot (\varepsilon_p + (q_r + q_n + q_s) \cdot \varepsilon_r)$, while the former game $G_0$ is the leakage-resilient robustness security game and the latter game $G_{q_n}$ is independent of $b$.

Let prove now that if the PRNG is $(t, q_r, q_n, \gamma^*, \lambda, \varepsilon_p)$-preserving with leakage, then $|\Pr[G_i = 1] - \Pr[G_{i+1/2} = 1]| \leq \varepsilon_p$. We therefore consider the distance between the games $G_i$ and $G_{i+1/2}$. We assume that we are in a *preserving with leakage* case, otherwise the two games are identical. We build a reduction to the preserving with leakage security notion. We will thus define an adversary $\mathcal{A}'$ against the preserving with leakage security game. The adversary $\mathcal{A}'$ emulates the challenger for $\mathcal{A}$. It thus runs the adversary $\mathcal{A}$, to get a distribution sampler $\mathcal{D}$ and the leakage functions $f$. $\mathcal{A}'$ initiates its security game, and receives back the seed it transfers to $\mathcal{A}$. Then $\mathcal{A}'$ sets $S = 0^n$ and $\sigma = 0$, and simulates the answers to all the oracle calls that $\mathcal{A}$ makes in the leakage-resilient robustness security game, but altered as in $G_i$, until the $(i+1)^{th}$ uncompromised next-query. Note that after the $i^{th}$ uncompromised next-query, the internal state is assumed to be the state $S_0$ provided by the challenger, since it has full entropy $n$.
Since $\mathcal{A}'$ knows the leakage functions, controls the internal state of the PRNG and has access to the distribution sampler $\mathcal{D}$, with the knowledge of its state, it can simulate all the calls to get-state, set-state, $\mathcal{D}$-refresh, leak-refresh, leak-next, and unsafe/compromised next-ror. The uncompromised next-queries are answered with a truly random $R$ (as in $G_i$) and the internal state is renewed with a truly random $S$. All these new states are chosen, and thus known to $\mathcal{A}'$, until the $i^{th}$ uncompromised next-queries. After this last query the internal state is not known any more to $\mathcal{A}'$, but is assumed to be the state $S_0$ provided by the challenger, which is also uniformly random.
Between the $i^{th}$ uncompromised next-queries and the $(i+1)^{th}$ uncompromised next-query, $\mathcal{A}'$ can use its challenger to answer the queries asked by $\mathcal{A}$, but providing the inputs for the refresh queries (with or without leakage), using the distribution sampler $\mathcal{D}$, since it still knows its state $\sigma$. Indeed, during the preserving sequence, only oracles $\mathcal{D}$-refresh, leak-refresh, and leak-next are possible. At the end of this sequence, $\mathcal{A}'$ receives the challenge $(S^{(b)}, R^{(b)})$ it uses for the $(i+1)^{th}$ uncompromised next-query: it answers $\mathcal{A}$ with $R^{(b)}$ and updates the internal state of the PRNG with $S^{(b)}$ and continues to simulate the oracle calls made by $\mathcal{A}$ as in $G_i$, since it knows again the internal state of the PRNG and has always known the state of the distribution sampler. Eventually, $\mathcal{A}$ outputs the bit $b^*$. If the challenge bit $b$ was 0 then $(S^{(b)}, R^{(b)})$ is the real value so

it perfectly simulates $G_i$ for $\mathcal{A}$. Otherwise, $(S^{(1)}, R^{(1)})$ is a random value, as in $G_{i+1}$. Therefore, the challenge of $\mathcal{A}'$ is exactly the same as the challenge consisting in distinguishing both games and we have, $|\Pr[G_i = 1] - \Pr[G_{i+1/2} = 1]| \leq \varepsilon_p$.

Let prove now that if the PRNG is $(t, q_r, q_n, \gamma^*, \lambda, \varepsilon_p)$-preserving with leakage, then $|\Pr[G_i = 1] - \Pr[G_{i+1/2} = 1]| \leq \varepsilon_p$. We therefore consider the distance between the games $G_{i+1/2}$ and $G_{i+1}$. We assume that we are in a *recovering with leakage* case, otherwise the two games are identical.
We build a reduction to the recovering with leakage security notion. We will thus define an adversary $\mathcal{A}'$ against the recovering with leakage security game. The adversary $\mathcal{A}'$ emulates the challenger for $\mathcal{A}$. It thus runs the adversary $\mathcal{A}$, to get a distribution sampler $\mathcal{D}$ and the leakage functions $f$. $\mathcal{A}'$ initiates its security game, and receives back the seed it transfers to $\mathcal{A}$, as well as the values $\gamma_1, \ldots, \gamma_{q_\mathcal{D}}$ and $z_1, \ldots, z_{q_\mathcal{D}}$. Then $\mathcal{A}'$ sets $S = 0^n$, and simulates the answers to all the oracle calls that $\mathcal{A}$ makes in the leakage-resilient robustness security game, but altered as in $G_i$, until the $(i+1)^{th}$ uncompromised next-query. Actually, as above, the uncompromised next-queries are answered with truly random $R$ and the internal state is renewed with a truly random $S$. To simulate the calls to $\mathcal{D}$-refresh and leak-refresh queries from $\mathcal{A}$, $\mathcal{A}'$ asks for a $\mathcal{D}$-refresh query and gets back the input $I_k$, which allows it to evaluate the refresh algorithm itself, and even compute the leakage information. Together with the values $\gamma_k$ and $z_k$ it received above from the challenger, it can answer appropriately to $\mathcal{A}$. Since $\mathcal{A}'$ knows the internal state of the PRNG (and even controls it during the uncompromised next-query), it can easily simulate get-state, set-state, unsafe/compromised next-ror, and leak-next queries.
After the $i^{th}$ uncompromised next-query, it continues the same way until the mRED query, it has to guess among the possible queries (whose number is bounded by $q_{ri} + q_{ni} + q_{si}$, the sum of the refresh, next and $\sigma$ queries between the $i^{th}$ uncompromised next-queries and the $(i+1)^{th}$ uncompromised next-query). For this guess (which might later revealed to be incorrect), $\mathcal{A}'$ provides the current internal state, right after the mRED, as $S_0$ to its challenger. $\mathcal{A}'$ can use its challenger to answer the queries asked by $\mathcal{A}$. Indeed, during the recovering sequence, only oracles $\mathcal{D}$-refresh, leak-refresh, and leak-next are possible (without making $c$ drop to 0). At the end of this sequence, $\mathcal{A}'$ receives the challenge $(S^{(b)}, R^{(b)})$ it uses for the $(i+1)^{th}$ uncompromised next-query, together the sequence of the next inputs: it answers $\mathcal{A}$ with $R^{(b)}$ and updates the internal state of the PRNG with $S^{(b)}$ and continues to simulate the oracle calls made by $\mathcal{A}$ as in $G_i$, since it knows again the internal state of the PRNG and knows the inputs to update it (as in the first part of the simulation). Eventually, $\mathcal{A}$ outputs the bit $b^*$. If the challenge bit $b$ was 0 then $(S^{(b)}, R^{(b)})$ is the real value so it perfectly simulates $G_i$ for $\mathcal{A}$. Otherwise, $(S^{(1)}, R^{(1)})$ is a random value, as in $G_{i+1}$. Therefore, the challenge of $\mathcal{A}'$ is exactly the same as the challenge consisting in distinguishing both games and we have, $|\Pr[G_{i+1/2} = 1] - \Pr[G_{i+1} = 1]| \leq (q_{ri} + q_{ni} + q_{si}) \cdot \varepsilon_r$. □

### A.2 Proof of Theorem 7

Following Theorem 13, we show that the PRNG $\mathcal{G}$ satisfies both the recovering security with leakage and the preserving security with leakage. We also denote $\varepsilon_{ext}$ the bias of the distribution of $U = [X' \cdot S]_1^m$ from uniform when the min-entropy of $S$ is greater than $\gamma^* - \lambda$, and show that it can be any value greater than $\sqrt{2^{m+1-\delta}}$ for $\delta = \min\{n - \log q_r, \gamma^* - \lambda\}$. Let us recall that we denote $q_r$ the number of calls to $\mathcal{D}$-refresh/leak-refresh, $q_n$ the number of calls to next-ror/leak-next, and $q_s$ the number of calls to get-state/set-state. We also denote $q = q_r + q_n + q_s$, the global number of queries.

As explained in Section 2.3 under the term *granular* model, we split the algorithm in atomic procedures, with leakage on their manipulated data. In particular, in the above construction, the refresh procedure can be considered atomic, while the next procedure should be split in two: the truncation of the product, and the PRG evaluation. As a consequence, we consider three leakage functions:

– $f_{\mathsf{refresh}}$ collects the leakage during the computation of the algorithm refresh, and thus takes as inputs the internal state $S$, the sample $I$ and the part $X$ of the seed;
– $f_{\mathsf{next},\Pi}$ collects the leakage in algorithm next, during the computation of the truncation of the product, and thus takes as inputs the internal state $S$ and the part $X'$ of the seed;
– $f_{\mathsf{next},\mathbf{G}}$ collects the leakage during the PRG call. It takes as input the intermediate variable $U = [X' \cdot S]_1^m$.

**Lemma 14.** *The PRNG $\mathcal{G}$ satisfies the $(t, q_r, q_n, \gamma^*, \lambda, q_n \cdot ({q_r}^2 \cdot \varepsilon_{ext} + \varepsilon_{\mathbf{G}}))$-recovering security with leakage.*

*Proof of Lemma 14.* The proof extends the one built in [10] to integrate the impact of the leakage.

**Game 0 [Recovering with Leakage Security Game].** This game is the original attack game described in Section 3, where $f$ is described by three leakage functions $f_{\mathsf{refresh}}$, $f_{\mathsf{next},\Pi}$ and $f_{\mathsf{next},G}$. Because of the restriction for the estimated entropy not to drop to 0, a first sequence includes only $\mathcal{D}$-refresh-queries, until $c$ gets larger than $\gamma^*$. Thereafter, the leaking procedures leak-refresh and leak-next are also allowed, in addition to the $\mathcal{D}$-refresh, as soon as $c$ remains above $\gamma^*$. With the answer to the challenge next-ror, this game eventually outputs 1 if $b^* = b$, and we want to show that $\Pr[G_0 = 1]$ is close to 1/2.

**Game 1.a [First leak-next Query: Random $U$].** In the first call to leak-next, we replace the truncated product $U$ by a truly random value. Using the same approach as in [10] with Lemma 1, we can show that the sequence of inputs $(I_k)_{k=1}^d$ generated by the distribution sampler, and the polynomial evaluation followed by the $m$-truncation leads to a $(N, \varepsilon)$-randomness extractor as long as the entropy in the source $N \geq m + 2\log(1/\varepsilon) + 1$ and $n \geq m + 2\log(1/\varepsilon) + \log(d) + 1$. With the possible information leakage $z_k$ and $L_k$, the sequence $(I_k)_{k=1}^d$ has a min-entropy larger than $\gamma^* - \lambda$ (because of the possible additional $f_{\mathsf{next},\Pi}(S, X')$), so we just need $m \leq \gamma^* - \lambda - 2\log(1/\varepsilon_{ext}) - 1$ and $m \leq n - 2\log(1/\varepsilon_{ext}) - \log(q_r) - 1$ to guarantee $\varepsilon_{ext}$ indistinguishability between the real $U$ and a random value, with this sequence $(I_k)$.
However, since the adversary can choose when it starts (among $q_r$ possibility), and how long it lasts (again, $q_r$ possibilities), there is a factor loss ${q_r}^2$. Then, we then have $|\Pr[G_0 = 1] - \Pr[G_{1.a} = 1]| \leq {q_r}^2 \cdot \varepsilon_{ext}$.

**Game 1.b [First leak-next Query: Random State and Output].** In the first call to leak-next, since $f_{\mathsf{next},G}$ is fixed, and $\mathbf{G}$ is a leakage-resilient PRG, the source $\mathcal{S}$ generates indistinguishable leakage, state and random as in the previous game with a truly random $U$. Then, we then have $|\Pr[G_{1.a} = 1] - \Pr[G_{1.b} = 1]| \leq \varepsilon_{\mathbf{G}}$.

**Game 2 [All leak-next Queries: Random States].** In an hybrid way, we replace all the leak-next outputs by $\mathcal{S}$. Then, we have $|\Pr[G_2 = 1] - \Pr[G_0 = 1]| \leq (q_n - 1) \cdot ({q_r}^2 \cdot \varepsilon_{ext} + \varepsilon_{\mathbf{G}})$.

**Game 3 [The next-ror Query: Random $U$].** If this was the first next-query, we have already shown that $U$ can be replaced by a truly random value. If it happens after a leak-next, the state $S$ has enough entropy for the extractor (the global output $(S, R)$ from the source $\mathcal{S}$ had entropy $\alpha + \ell$ when knowing the leakage, then knowing the $\ell$-bit randomness $R$, the remaining entropy for $S$ is above $\alpha \geq \gamma^* \geq \gamma^* - \lambda$): the truncated product in the field is a $2^{-m}(1 + 2^{m-n})$-universal, and thus a $(N, \varepsilon)$-randomness extractor as long as the entropy in the source is $N \geq m + 2\log(1/\varepsilon) + 1$. The above constraint is enough for a bias bounded by $\varepsilon_{ext}$ between the real $U$ and a random value. We then have $|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \leq \varepsilon_{ext}$.

**Game 4 [The next-ror Query: Random Output].** Since $\mathbf{G}$ is a $(t, \varepsilon_{\mathbf{G}})$-secure PRG, we can replace both the output and the random state by truly random values. And we have $|\Pr[G_3 = 1] - \Pr[G_4 = 1]| \leq \varepsilon_{\mathbf{G}}$.

From the above games, one gets, $|\Pr[G_0 = 1] - \Pr[G_4 = 1]| \leq q_n \cdot ({q_r}^2 \cdot \varepsilon_{ext} + \varepsilon_{\mathbf{G}})$, while $\Pr[G_4 = 1] = 1/2$, for any $\varepsilon_{ext} \geq \sqrt{2^{m+1-\delta}}$ for $\delta = \min\{n - \log q_r, \gamma^* - \lambda\}$. □

*Remark 15.* This proof with leakage shows the relevance of the adaptation of the generic construction. Concretely, to ensure an internal state with enough entropy at the input of the final next-ror, we established two measures to limit the negative impact of the leak-next calls. First, the threshold $\gamma^*$ was set voluntary higher than the original one in [10] to capture the leakage in the truncated product, given by the leakage function $f_{\mathsf{next},\Pi}$. Then, the generator $\mathbf{G}$ was defined with security properties whereby, in a leak-next call, the final output comes with an entropy at least equal to $\alpha$ despite the leakage.

**Lemma 16.** *The PRNG $\mathcal{G}$ has $(t, q_r, q_n, \gamma^*, \lambda, q_n \cdot (\varepsilon_{ext} + \varepsilon_{\mathbf{G}}) + 2^{-n})$-preserving security with leakage.*

*Proof.* This proof also extends the one in [10], with leaking queries.

**Game 0 [Preserving with Leakage Security Game].** This is the original preserving with leakage security game described in Section 3. The internal state starts uniformly at random, and then the adversary can ask $\mathcal{D}$-refresh and leak-refresh-queries with chosen inputs, and leak-next-queries before the challenge next-ror, as soon as $c$ remains above $\gamma^*$. With the answer to this, this game eventually outputs 1 if $b^* = b$, and we want to show that $\Pr[G_0 = 1]$ is close to 1/2.

**Game 1.a [First leak-next Query: Random $U$].** As above, in the first call to leak-next, we replace the truncated product $U$ by a truly random value. But since the internal state started full of randomness (but it would be true with any entropy level), following $\mathcal{D}$-refresh and leak-refresh-query maintain entropy or reduce it by $\lambda$ at most, but remaining above $\gamma^*$, unless $X = 0$. Then, since the truncated product in the field is a $(\gamma^*, \varepsilon)$-randomness extractor (with above constraints), the bias is bounded by $\varepsilon_{ext}$ between the real $U$ and a random value. Then, we have $|\Pr[G_{1.a} = 1] - \Pr[G_0 = 1]| \leq \varepsilon_{ext} + 2^{-n}$.

**Game 1.b [First leak-next Query: Random State and Output].** In the first call to leak-next, since $f_{\mathsf{next},G}$ is fixed, and $\mathbf{G}$ is a leakage-resilient PRG, the source $\mathcal{S}$ generates indistinguishable leakage, state and random as in the previous game with a truly random $U$. We then have $|\Pr[G_{1.a} = 1] - \Pr[G_{1.b} = 1]| \leq \varepsilon_{\mathbf{G}}$.

**Game 2 [All leak-next Queries: Random States].** In an hybrid way, we replace all the leak-next outputs by $\mathcal{S}$. Then, we then have $|\Pr[G_2 = 1] - \Pr[G_0 = 1]| \leq (q_n - 1) \cdot (\varepsilon_{ext} + \varepsilon_{\mathbf{G}}) + 2^{-n}$.

**Game 3 [The next-ror Query: Random $U$].** If this was the first next-query, we have already shown that $U$ can be replaced by a truly random value. If it happens after a leak-next, the state $S$ has enough entropy for the extractor: the truncated product in the field is a $(\gamma^*, \varepsilon)$-randomness extractor. Since the state $S$ has an entropy larger than $\gamma^*$, the above constraint is enough for a bias bounded by $\varepsilon_{ext}$ between the real $U$ and a random value. Then, we have $|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \leq \varepsilon_{ext}$.

**Game 4 [The next-ror Query: Random Output].** Since $\mathbf{G}$ is $(t, \varepsilon_{\mathbf{G}})$-secure, we can replace both the output and the random state by truly random values. And we have $|\Pr[G_3 = 1] - \Pr[G_4 = 1]| \leq \varepsilon_{\mathbf{G}}$.

From the above games, one gets, $|\Pr[G_0 = 1] - \Pr[G_4 = 1]| \leq q_n \cdot (\varepsilon_{ext} + \varepsilon_{\mathbf{G}}) + 2^{-n}$, while $\Pr[G_4 = 1] = 1/2$. □

From above Lemma 14 and Lemma 16, we conclude that the PRNG $\mathcal{G}$ satisfies $(t', q_r, q_n, q_s, \gamma^*, \lambda, \varepsilon)$-leakage-resilient robustness where $t' \approx t$, and

$$\varepsilon = q_n \cdot \left( \left( q_n \cdot ({q_r}^2 \cdot \varepsilon_{ext} + \varepsilon_{\mathbf{G}}) \right) + (q_r + q_n + q_s) \cdot \left( q_n \cdot (\varepsilon_{ext} + \varepsilon_{\mathbf{G}}) + 2^{-n} \right) \right),$$

which proves Theorem 7, since $q_n \leq q$ and $2^{-n} \leq \varepsilon_{ext}$.

### A.3 Proof of Theorem 9

We split the proof in two parts. We first show the security without leakage and then with leakage.

**Game 0 [Real Game without Leakage].** This game is referred to as the *real* game against a $(t, \varepsilon)$-secure PRG. The input $U$ is split to set the secret keys $(K_i)$ and the counter $C$. $C$ is made public, and is used to generate the $(p_j^i)$, and then the keys are used to generate the outputs $(T_j^i)$. According to the challenge bit $b$, either the $(T_j^i)$ are provided, or truly random values $\mathcal{S}$.

**Game 1 [Random Outputs $(T_j^i)$].** Since $\mathbf{F}$ is a $(t, \nu, \varepsilon_{\mathbf{F}})$-secure PRF, used with truly random keys and distinct inputs, with an hybrid proof, replacing the first sequences $(T_0^i, \ldots, T_{\nu-1}^i)$, for $i = 1, \ldots, k$, by random sequences, in the $k$-th hybrid game, we can show that $|\Pr[G_1 = 1] - \Pr[G_0 = 1]| \leq \kappa \cdot \varepsilon_{\mathbf{F}}$, and $|\Pr[G_1 = 1] = 1/2$.

**Game 0 [Real Game with Leakage].** This game is referred to as the *real* game implicitly defined in Definition 6. First, the adversary outputs a leakage function $f$, and it is given back either $(f(U), \mathbf{G}(U))$ for a truly random string $U$, or a pair $(L, T)$ from a random source $\mathcal{S}$. It has to guess in which case it is.

*Leakage Functions.* The adversary first chooses $\kappa$ leakage functions $(f_{\mathbf{F},i})_0^{\kappa-1}$ with $\lambda$-bit global output. Actually, the leakage function $f_{\mathbf{F},i}$ collects the leakage during the calls of the PRF $\mathbf{F}$ instantiated with the secret key $K_i$ and thus takes as inputs this secret key and the public inputs $(p_0^i, \ldots, p_{\nu-1}^i)$. Since the adversary can choose to get all the leakage from one block, we have to protect all the blocks for a $\lambda$-leakage. But globally, the sum of the leakages is bounded by $\lambda$. Note that since the upper-level is public, no leakage can be asked on $\mathbf{F}'$-executions.

*Challenge.* The input $U$ is split to set the secret keys $(K_i)$ and the counter $C$. $C$ is made public, and is used to generate the $p_j^i$, and then the keys are used to generate the outputs $S_j^i$. According to the challenge bit $b$, either the real leakage and outputs are provided, or the values from a random source $\mathcal{S}$.

**Game 1 [Secret $X''$].** In this game, instead of publishing $X''$, it is kept secret. Using a similar proof as in [25] in the minicrypt world, if an efficient distinguisher would exist in $G_0$ while $G_1$ cannot be broken, then one could build a public-key encryption scheme from these PRFs, which is very unlikely.
To conclude, we thus have to prove that $\Pr[G_1 = 1]$ is close to $1/2$.

**Game 2 [Random Values $p_j^i$].** Instead of generating the inputs $p_j^i$ with the PRF $\mathbf{F}'$ with secret key $X''$ in counter mode, we generate them uniformly at random. Since $\mathbf{F}'$ is a $(t, q_n\nu\kappa, \varepsilon_{\mathbf{F}'})$-secure PRF, where $q_n\nu\kappa$ is the global number of calls to $\mathbf{F}'$ during the $q_n$ next-queries, and there is no leakage at this level (everything was public in the original scheme) then, unless there is a collision on the inputs to $\mathbf{F}$, then the two games are indistinguishable: $|\Pr[G_2 = 1] - \Pr[G_1 = 1]| \leq \frac{{q_n}^2\nu\kappa}{2^\mu} + \varepsilon_{\mathbf{F}'}$. The collision probability is indeed bounded by ${q_n}^2\nu\kappa/2^\mu$ as explained in Section 5.2.

**Game 3 [High Entropy Outputs $T_j^i$].** When there is some leakage, we want to show that $\mathbf{G}$ outputs strings that are computationally indistinguishable from $\alpha$-sources, even with the leakage. Then, we use the $(\alpha/\kappa, \lambda)$-leakage-resilience of $\mathbf{F}$: for each leakage function with $\lambda$-bit outputs, there are sources that generate random inputs (indistinguishable from the $(p_j^i)$), as well as leakage and outputs $(T_j^i)_j$ with entropy $\alpha/\kappa$ for each $i$. Then, for a maximal leakage of $\lambda$ (since this is the global allowed leakage) the global output has a minimal entropy of $\alpha$. The concatenation of these sources thus helps to conclude, with again an hybrid sequence, since the final game uses the final source for both $b = 0$ and $b = 1$: $|\Pr[G_3 = 1] - \Pr[G_2 = 1]| \leq \kappa \cdot \varepsilon_{\mathbf{F}}$, and $|\Pr[G_3 = 1] = 1/2$. For an appropriate choice for $\mathbf{F}$, we can say that $\Pr[G_1 = 1]$ is close to $1/2$, and so this is the same for $\Pr[G_0 = 1]$ in the minicrypt world.

**Conclusion.** In any case (with or without leakage), the global advantage is bounded by $\kappa \cdot \varepsilon_{\mathbf{F}} + \varepsilon_{\mathbf{F}'} + {q_n}^2\nu\kappa/2^\mu$.