Side-channel Masking with Pseudo-Random Generator

Jean-Sébastien Coron[1], Aurélien Greuet[2], and Rina Zeitoun[2]

[1] University of Luxembourg
jean-sebastien.coron@uni.lu
[2] IDEMIA, France
aurelien.greuet@idemia.com, rina.zeitoun@idemia.com

Abstract High-order masking countermeasures against side-channel attacks usually require plenty of randomness during their execution. For security against t probes, the classical ISW countermeasure requires $\mathcal{O}(t^2 s)$ random bits, where s is the circuit size. However running a True Random Number Generator (TRNG) can be costly in practice and become a bottleneck on embedded devices. In [IKL+13] the authors introduced the notion of *robust* pseudo-random number generator (PRG), which must remain secure even against an adversary who can probe at most t wires. They showed that when embedding a robust PRG within a private circuit, the number of random bits can be reduced to $\tilde{\mathcal{O}}(t^4)$, that is independent of the circuit size s (up to a logarithmic factor). Using bipartite expander graphs, this can be further reduced to $\tilde{\mathcal{O}}(t^{3+\varepsilon})$; however the resulting construction is impractical.
In this paper we describe a construction where the number of random bits is only $\tilde{\mathcal{O}}(t^2)$ for security against t probes, without expander graphs; moreover the running time of each pseudo-random generation goes down from $\tilde{\mathcal{O}}(t^4)$ to $\tilde{\mathcal{O}}(t)$. Our technique consists in using multiple independent PRGs instead of a single one. We show that for ISW circuits, the robustness property of the PRG is not required anymore, which leads to simple and efficient constructions. For example, for AES we only need 48 bytes of randomness to get second-order security ($t = 2$), instead of 2880 in the original Rivain-Prouff countermeasure. As a first feasibility result, we have implemented our countermeasure on an ARM-based embedded device with a relatively slow TRNG, and obtained a 50% speed-up compared to Rivain-Prouff.

1 Introduction

High-order masking. Side-channel analysis is a class of attacks which exploits the physical environment of a cryptosystem during its execution, to reveal the secrets being manipulated. The masking countermeasure is an efficient technique to protect sensitive data against this threat. To protect a sensitive data x, the masking technique consists in generating a random variable r and manipulating the masked variable $x' = x \bigoplus r$ and the random r separately, instead of x directly. In that case, every intermediate variable has the uniform distribution

and any first-order attack is thwarted. However by combining information from both leakage points x' and r, a second-order attack can still be feasible (see for example [OMHT06]).

A natural countermeasure against high-order attacks is to use a high-order masking, where each variable x is split into n Boolean shares $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$, with $n > t$ for security against t probes. Initially the shares are generated uniformly at random under this condition; for example one can generate $x_1, \ldots, x_{n-1}$ randomly and let $x_n = x \oplus x_1 \oplus \cdots \oplus x_{n-1}$. The shares are then processed separately in masked operations (also called gadgets) that enable to compute the underlying secret variables in a secure way.

The study of circuits resistant against probing attacks was initiated by Ishai, Sahai and Wagner in [ISW03]. They showed how to transform any circuit of size s into a circuit of size $\mathcal{O}(t^2 s)$ secure against any adversary who can probe at most t wires. The ISW construction is based on secret sharing every variable x into $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ as above, with $n = 2t+1$ shares to guarantee security against t probes. Processing a XOR gate is straightforward as the shares can be xored separately. For processing an AND gate $z = xy$, one computes all cross-products $x_i y_j$ in Equation (1) below, and then uses a randomized algorithm to recombine the n^2 cross-products into an n-sharing of the output z.

$$z = xy = \left(\bigoplus_{i=1}^{n} x_i\right) \cdot \left(\bigoplus_{i=1}^{n} y_i\right) = \bigoplus_{1 \le i,j \le n} x_i y_j \tag{1}$$

Every AND gate is then expanded into a gadget of size $\mathcal{O}(t^2)$ and the resulting circuit has size $\mathcal{O}(t^2 s)$.

The ISW construction was adapted to AES by Rivain and Prouff in [RP10], by working in $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$. The authors observed that the non-linear part $S(x) = x^{254}$ of the AES SBox can be efficiently evaluated with only 4 non-linear multiplications over $\mathbb{F}_{2^8}$, and a few linear squarings. Each of those 4 multiplications can in turn be evaluated with the previous ISW gadget based on Equation (1), by working over $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$.

Proving security. The approach initiated in [ISW03] for proving security against a t-probing adversary is based on simulation; one must show that the view of an adversary probing at most t wires can be perfectly simulated without knowing the secret variables from the original circuit. To this aim, one shows that any set of t probed variables can be perfectly simulated from the knowledge of at most $n-1$ input shares. Since any subset of $n-1$ input shares is uniformly and independently distributed, this ensures that the adversary learns nothing from the t probes, since he could simulate them by himself. It was shown in [DDF14] that security against t probes implies security against noisy leakage, under the assumption that every variable leaks independently.

Recently, the notions of (Strong) Non-Interference (NI/SNI) were introduced by Barthe *et al.* in [BBD+16], to allow easy composition of gadgets. The authors showed that the ISW multiplication gadget does satisfy the stronger t-SNI security definition. They also showed that with some additional mask refreshing, the

Rivain-Prouff countermeasure for the full AES can be made secure with $n = t+1$ shares only, instead of $n = 2t + 1$ shares in [ISW03].

More recently, a new security notion was introduced by Cassiers and Standaert in [CS18], called PINI, that allows even simpler composition of gadgets. Namely it suffices to ensure that all gadgets are PINI, and the composite gadget is then also PINI, which also implies security against t probes. With its power and simplicity, the PINI definition appears to be the "right" notion for gadget security and composition; therefore we will use this definition in this paper, either by proving the PINI property of a gadget directly, or by first proving the t-SNI property and then PINI.

Minimizing randomness complexity. High-order masking countermeasures against side-channel attacks usually require plenty of randomness during their execution. The secure AND operation from [ISW03] with $t + 1$ shares requires $t(t + 1)/2$ random bits, and therefore the randomness complexity of the ISW countermeasure is $\mathcal{O}(t^2 s)$, where s is the circuit size. More concretely, the evaluation of the AES SBox in Rivain-Prouff [RP10] requires the execution of 4 secure multiplications and 2 mask refreshing; each of those 6 gadgets requires $t(t+1)/2$ fresh random bytes. For the 16 SBoxes and the 10 rounds of the AES, this amounts to generating $6 \times 16 \times 10 \times t(t+1)/2 = 480t(t+1)$ random bytes, which gives 2880 bytes for second-order security ($t = 2$).

However running a True Random Number Generator (TRNG) can be costly in practice and become a major bottleneck on embedded devices such as smartcards. Thus, high-order resistant algorithms can rapidly become impractical when the number of shares grows. The main question is therefore how to minimize the number of TRNG calls while still guaranteeing t-probing security as in [ISW03].

Several attempts have been made to reduce the randomness complexity of private circuits. In [BBP+16], the authors showed a variant of the ISW multiplication with roughly $t^2/4$ randoms instead of $t^2/2$ in ISW. In [FPS17], the authors showed how to re-use randomness within several gadgets, thereby reducing the total amount of randomness needed, for small values of t ($t \leq 7$). However the two above approaches only reduce the randomness complexity by a constant factor; that is, their asymptotic complexity is still $\mathcal{O}(t^2 s)$ for circuit size s, as in the original ISW countermeasure.

A natural idea to reduce the number of calls to the TRNG is to use a pseudo-random generator (PRG) to generate all randoms in the circuit, while only a small seed will be generated by the TRNG. Obviously the PRG circuit should also be secure against probing attacks. We recall below that such approach, initiated by Ishai *et al.* in [IKL+13] with the concept of robust PRG, enables to reduce the randomness complexity of t-private circuits from $\mathcal{O}(t^2 s)$ to $\mathcal{O}(t^4(\log s + \log t))$; with respect to the circuit size s, this is therefore an exponential improvement. Our main contribution is this paper will be to reduce this complexity further down to $\mathcal{O}(t^2(\log s + \log t))$, and to describe a concrete implementation of AES based on this approach. We refer to Table 2 below for the

number of bytes required to protect AES against t-th order attacks; we see that for small values of t, we obtain almost two orders of magnitude improvement compared to previous methods.

Robust PRGs and private circuits. In [IKL+13], the authors introduced the notion of *robust* pseudo-random number generator (PRG). A robust PRG must remain secure even if an adversary can probe at most t intermediate variables in the PRG circuit. The authors showed that such robust PRG can be used in the ISW countermeasure to minimize the randomness complexity. Namely the resulting circuit uses a short random seed only, and remains secure against t-th order attacks.

Recall that the original ISW countermeasure requires $\mathcal{O}(t^2 s)$ bits of randomness, where s is the circuit size. Following [IKL+13], we first recall how this can be reduced to $\mathcal{O}(t^4(\log t + \log s))$, using a trivial construction of robust PRG. More precisely, the construction is based on r-wise independent PRG. A PRG is said to be r-wise independent if any subset of at most r output bits of the PRG is uniformly and independently distributed. The authors show that the ISW countermeasure can be adapted so that any wire in the ISW circuit depends on at most $\ell = \mathcal{O}(t^2)$ bits of randomness; such parameter ℓ is called the *locality* of the randomness and will play a crucial role in this paper. Since the adversary can probe at most t wires, the adversary's side-channel observation can then depend on at most $t \cdot \ell = \mathcal{O}(t^3)$ bits of randomness. Therefore, instead of using a TRNG, it is sufficient to use an r-wise independent PRG with parameter $r = t \cdot \ell = \mathcal{O}(t^3)$; if the r-wise PRG is secure against t probes, as shown in [IKL+13] the resulting circuit will remain secure against t probes.

It is easy to obtain an r-wise independent PRG by evaluating a degree $r-1$ polynomial on distinct inputs in a finite field $\mathbb{F}$; the r coefficients of the polynomials are initially generated at random in $\mathbb{F}$; this is the seed of the PRG. From r fresh randoms in $\mathbb{F}$, one can then obtain m pseudo-randoms with the r-wise independence property, as long as $m \leq |\mathbb{F}|$. To obtain an r-wise independent PRG with robustness against t probes, as observed in [IKL+13] a trivial construction consists in xoring the output of $t+1$ PRGs, so that at least one PRG has not been probed. One can therefore obtain an r-wise independent PRG robust against t probes by using $r \cdot (t+1) = \mathcal{O}(t^4)$ fresh randoms in $\mathbb{F}$ as input, and such PRG can then generate $m \leq |\mathbb{F}|$ pseudo-randoms in $\mathbb{F}$. Since the original ISW countermeasure requires $m = \mathcal{O}(t^2 s)$ randoms (where s is the circuit size), using $\mathbb{F} = \mathbb{F}_{2^k}$ one can take $k = \mathcal{O}(\log m) = \mathcal{O}(\log t + \log s)$. One therefore needs $\mathcal{O}(t^4(\log t + \log s)) = \tilde{\mathcal{O}}(t^4)$ bits of randomness [1], instead of $\mathcal{O}(t^2 s)$. The number of input random bits is then independent of the circuit size s (up to some logarithmic factor). In summary, any t-private circuit in which each wire depends on at most ℓ bits of randomness can be converted into a t-private circuit using roughly $t^2\ell$ bits of randomness via the use of robust r-wise PRGs. As written by the authors: "Improving the randomness locality ℓ of private circuits would

[1] We use the notation $f(\lambda) = \tilde{\mathcal{O}}(g(\lambda))$ if $f(\lambda) = \mathcal{O}(g(\lambda)\log^k \lambda)$ for some $k \in \mathbb{N}$.

immediately yield a corresponding improvement [in the number of input random bits].".

In [IKL+13], the authors describe an improved construction of robust PRG, based on unbalanced bipartite expander graphs. Using the Guruswami-Umans-Vadhan construction of expander graphs [GUV09], they obtain r-wise independence and resistance against $t = r$ probes with $r^{1+\eta}$ bits of true randomness as input, for any $\eta > 0$. In the context of the ISW countermeasure, this enables to use $\tilde{\mathcal{O}}(t^{3+\varepsilon})$ random bits as input for any $\varepsilon > 0$, instead $\tilde{\mathcal{O}}(t^4)$.

Our contribution. Our main contribution is a countermeasure against side-channel attacks where the number of random bits is only $\tilde{\mathcal{O}}(t^2)$ for security against t probes, independently of the circuit size (up to a logarithmic factor), and without using expander graphs. Moreover the running time of pseudo-random generation goes down from $\tilde{\mathcal{O}}(t^4)$ to $\tilde{\mathcal{O}}(t)$. We summarize in Table 1 below the asymptotic complexities of existing techniques and our new techniques. We proceed in two steps.

In the first step, we show how to improve the locality ℓ of private circuits from $\ell = \mathcal{O}(t^2)$ down to $\ell = \mathcal{O}(t)$. As illustrated in the third line of Table 1 below, reducing ℓ from $\mathcal{O}(t^2)$ to $\mathcal{O}(t)$ enables to reduce the r-wise independence parameter from $r = \mathcal{O}(t^3)$ down to $r = \mathcal{O}(t^2)$; the number of input random bits is then now decreased from $\tilde{\mathcal{O}}(t^4)$ to $\tilde{\mathcal{O}}(t^3)$ with the trivial construction (and from $\tilde{\mathcal{O}}(t^{3+\varepsilon})$ to $\tilde{\mathcal{O}}(t^{2+\varepsilon})$ with expander graphs). Our technique is as follows. The authors of [IKL+13] obtain $\ell = \mathcal{O}(t^2)$ by performing a mask locality refreshing at the end of each ISW multiplication gadget. Instead we modify the ISW multiplication by performing a series of internal locality refreshing. For this we consider successive $i \times i$ ISW submatrices and perform a mask refreshing after the processing of each submatrix; these internal mask refreshing enable to bring the locality down to $\ell = \mathcal{O}(t)$. We have also performed a formal verification of our new algorithms, using the CheckMasks tool [Cor18], for both the locality and the security properties; we provide the source code in [Cor19a]. This first step is described in Section 3.

In the second step, our technique consists in using multiple independent PRGs instead of a single one. This has two main advantages. The first advantage is that for ISW circuits, one can show that the robustness property of the PRG is not required anymore; this implies that we can use a very simple PRG based on polynomial evaluation as above. The second advantage is that the locality with respect to each subset of randoms generated by each PRG becomes $\ell = \mathcal{O}(1)$. Therefore each independent PRG can be r-wise independent with a much smaller parameter $r = \mathcal{O}(t)$ instead of $r = \mathcal{O}(t^3)$, and therefore requires only $r = \mathcal{O}(t)$ randoms in the finite field (since robustness is not needed). In that case, we need $\mathcal{O}(t^2)$ independent PRGs and therefore the size of the input randomness is $\tilde{\mathcal{O}}(t^3)$; see Line 4 of Table 1. Finally, when using internal locality refreshing as in the first step above, we only need $\mathcal{O}(t)$ independent PRGs, and eventually the number of input random bits is reduced to $\tilde{\mathcal{O}}(t^2)$, instead of $\tilde{\mathcal{O}}(t^{3+\varepsilon})$ with expander graphs in [IKL+13] (see Line 5 of Table 1). We stress that this asymptotic improvement

over [IKL+13] is obtained *without* using expander graphs, that is we can use a simple PRG based on polynomial evaluation in a finite field (see Section 4). [2]

As mentioned previously, we found that expander graphs PRG are impractical for minimizing the amount of input randomness. However expander graphs can still be useful for optimizing the time generation of each pseudo-random; namely the output locality of an expander graph PRG (i.e., the number of inputs on which each output depends) can be at most polylogarithmic in the seed length (as opposed to linear for a PRG based on polynomial evaluation); hence in Table 1 the pseudo-random time generation is always $\tilde{\mathcal{O}}(1)$. In Section 2.3 we give an example of a simple construction based on expander graph that achieves very fast pseudo-random generation, at the cost of significantly more input randomness.

	#PRG	loc. ℓ	r-wise	PRG	TRNG	time PRG
ISW without PRG [ISW03]	–	–	–	–	$\mathcal{O}(t^2 s)$	–
ISW with Final LR, single PRG [IKL+13]	1	$\mathcal{O}(t^2)$	$\mathcal{O}(t^3)$	Trivial	$\tilde{\mathcal{O}}(t^4)$	$\tilde{\mathcal{O}}(t^4)$
				EG	$\tilde{\mathcal{O}}(t^{3+\varepsilon})$	$\tilde{\mathcal{O}}(1)$
ISW with Internal LR, single PRG (Sec. 3)	1	$\mathcal{O}(t)$	$\mathcal{O}(t^2)$	Trivial	$\tilde{\mathcal{O}}(t^3)$	$\tilde{\mathcal{O}}(t^3)$
				EG	$\tilde{\mathcal{O}}(t^{2+\varepsilon})$	$\tilde{\mathcal{O}}(1)$
ISW with Final LR, multiple PRGs (Sec. 4)	$\mathcal{O}(t^2)$	$\mathcal{O}(1)$	$\mathcal{O}(t)$	Linear	$\tilde{\mathcal{O}}(t^3)$	$\tilde{\mathcal{O}}(t)$
				EG	$\tilde{\mathcal{O}}(t^{3+\varepsilon})$	$\tilde{\mathcal{O}}(1)$
ISW with Internal LR, multiple PRGs (Sec. 4)	$\mathcal{O}(t)$	$\mathcal{O}(1)$	$\mathcal{O}(t)$	Linear	$\tilde{\mathcal{O}}(t^2)$	$\tilde{\mathcal{O}}(t)$
				EG	$\tilde{\mathcal{O}}(t^{2+\epsilon})$	$\tilde{\mathcal{O}}(1)$

Table 1. Asymptotic efficiency of various constructions. The Locality Refreshing (LR) is performed either at the end of each gadget (Line 2 and Line 4), or sequentially within each gadget (Line 3 and Line 5). The trivial construction of PRG is based on xoring $t+1$ linear PRGs to get robustness against t probes.

Finally, we describe in Section 5 an application of our countermeasure to AES. We show that for AES we only need 48 bytes of randomness to get second-order security ($t = 2$), instead of 2880 in the original Rivain-Prouff countermeasure. We see in Table 2 below that for small values of t, our construction reduces the randomness complexity of masking AES by almost 2 orders of magnitude. In Section 5, we also provide the results of a concrete implementation. When implemented on an ARM-based embedded device with a relatively slow TRNG, we obtain a 50% speed-up compared to Rivain-Prouff for $t = 2$. We provide the source code in C in [Cor19b]. Needless to say, we do not claim that in practice our implementation would be secure against a t-th order attack. Namely the implementation is only provided for illustrative purpose, and timing comparisons. Obtaining a secure implementation would require to (at least) carefully examine

[2] An earlier version of [AIS18] claimed to achieve randomness complexity $\mathcal{O}(t^{1+\varepsilon})$, but the claim was later retracted in the final version.

the assembly code, and perform a leakage test with concrete acquisitions from an oscilloscope.

	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$
Rivain-Prouff [RP10]	2880	5760	9600	14400	20160	26880
Belaïd *et. al* [BBP+16]	2560	5120	8000	13120	18240	24000
Faust *et. al* [FPS17]	1415	2530	6082	6699	20712	20726
This paper	48	108	192	300	432	588

Table 2. Number of bytes of randomness to get t-th order security for AES.

2 Definitions and Previous Work

2.1 Private circuits

In 2003, Ishai, Sahai and Wagner [ISW03] initiated the study of securing circuits against an attacker who can probe a fraction of its wires. They showed how to transform any circuit of size $|C|$ into a larger circuit of size $O(|C| \cdot t^2)$ with the same functionality but secure against a t-probing adversary, based on splitting each variable x into $n = 2t + 1$ shares with $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$.

Definition 1 (Private circuit). *A private circuit for $f : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$ is a triple (I, C, O) where $I : \{0,1\}^{n_i} \to \{0,1\}^{\hat{n}_i}$ is a randomized input encoder, C is a randomized boolean circuit with input $\hat{\omega} \in \{0,1\}^{\hat{n}_i}$, output $\hat{y} \in \{0,1\}^{\hat{n}_o}$, and randomness $\rho \in \{0,1\}^m$, and $O : \{0,1\}^{\hat{n}_o} \to \{0,1\}^{n_o}$ is an output decoder, such that for any input $\omega \in \{0,1\}^{n_i}$ we have $\Pr[O(C(I(\omega), \rho)) = f(\omega)] = 1$, where the probability is over the randomness of I and ρ.*

For I and O we consider the canonical encoder and decoder: I encodes each input bit ω_i by a vector of $2t + 1$ random bits with parity ω_i, and O takes the parity of each block of $2t + 1$ bits.

Definition 2 (t-privacy). *We say that C is a t-private implementation of f with encoder I and decoder O is t-private (or t-probing secure) if for any $\omega, \omega' \in \{0,1\}^{n_i}$ and any set P of t wires in C, the distributions $C_P(I(\omega), \rho)$ and $C_P(I(\omega'), \rho)$ are identical, where C_P denotes the set of t values on the wires from P.*

2.2 PINI and t-SNI security

The Probe Isolating Non-Interference (PINI) security notion was introduced in [CS18] to enable easy composition of gadgets. Let n be the number of shares. We let $x_\star = (x_i)_{i=1,\ldots n}$ be an n-sharing of x if $x = \bigoplus_{i=1}^{n} x_i$. Given a subset

$I \subset [1, n]$ of share indices, we denote by $x_{|I} := \{x_i : i \in I\}$ the corresponding subset of shares. A gadget with m inputs and ℓ outputs is a circuit with mn input shares grouped into m n-sharings denoted $(x_{\star,1}, \dots x_{\star,m})$, and similarly ℓn output shares denoted $(y_{\star,1}, \dots y_{\star,\ell})$. For a given share index i, we also use the notation $x_{i,\star} = \{x_{i,j} : 1 \leq j \leq m\}$ to denote all shares with index $1 \leq i \leq n$; similarly, we also write $x_{|I,\star} = \{x_{i,\star} : i \in I\}$. Below we recall the Probe Isolating Non-Interference (PINI) definition from [CS18]; we actually use a slightly simplified (and equivalent) definition compared to [CS18]; we explain the difference in the full version of our paper [CGZ19].

Definition 3 (PINI [CS18] (adapted)). *Let G be a gadget with input shares $x_{i,\star}$ and output shares $y_{i,\star}$ for $1 \leq i \leq n$. The gadget G is PINI if for any $t_1 \in \mathbb{N}$, any set of t_1 intermediate variables and any subset $\mathcal{O}$ of output indices, there exists a subset $I \subset [1, n]$ of input indices with $|I| \leq t_1$ such that the t_1 intermediate variables and the output shares $y_{|\mathcal{O},\star}$ can be perfectly simulated from the input shares $x_{|I \cup \mathcal{O},\star}$.*

It is straightforward to show that a PINI gadget with n shares is secure against $t = n - 1$ probes. We recall the proof of PINI composition (under our slightly modified definition) in the full version of our paper [CGZ19].

Proposition 1 (PINI security [CS18]). *Any PINI gadget with n shares is $(n-1)$-probing secure.*

Proposition 2 (PINI composition [CS18]). *Any composite gadget made of PINI composing gadgets is PINI.*

Below we recall the SNI security notion introduced in [BBD+15]. We consider a gadget taking as input two n-tuples $(x_i)_{1 \leq i \leq n}$ and $(y_i)_{1 \leq i \leq n}$ of shares, and outputting a single n-tuple $(z_i)_{1 \leq i \leq n}$. As previously, given a subset $I \subset [1, n]$, we denote by $x_{|I}$ all elements x_i such that $i \in I$.

Definition 4 (t-SNI security). *Let G be a gadget taking as input n shares $(x_i)_{1 \leq i \leq n}$ and n shares $(y_i)_{1 \leq i \leq n}$, and outputting n shares $(z_i)_{1 \leq i \leq n}$. The gadget G is said to be t-SNI secure if for any set of t_1 probed intermediate variables and any subset $\mathcal{O}$ of output indices, such that $t_1 + |\mathcal{O}| \leq t$, there exist two subsets I and J of input indices which satisfy $|I| \leq t_1$ and $|J| \leq t_1$, such that the t_1 intermediate variables and the output variables $z_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$ and $y_{|J}$*

Intuitively, the t-SNI security definition provides an "isolation" between the output shares and the input shares, so that the number of input variables required for the simulation is upper-bounded by the number of internal probes t_1, and does not depend on the number of output variables that must be simulated, as long as $t_1 + |\mathcal{O}| \leq t$. There is an analogous definition for a gadget with a single input $(x_i)_{1 \leq i \leq n}$; in that case, the simulation is performed from $x_{|I}$ with $|I| \leq t_1$.

It is easy to see that for a single input gadget, $(n-1)$-SNI security implies PINI security. Moreover, for a 2-input $(n-1)$-SNI gadget as considered in Definition 4, as shown in [CS18] we can obtain a PINI gadget by pre-refreshing one of the inputs with a $(n-1)$-SNI mask refreshing algorithm; this is the double-SNI approach (see Fig. 1). A mask refreshing gadget takes as input the n-sharing of a value x and outputs a randomized n-sharing of the same value x. Therefore, in this paper, our strategy for proving gadget security is either to directly prove the PINI property, or to first prove the t-SNI property and then apply the "double-SNI" strategy. Note that for specific circuits such as the AES SBox, one can use some optimization; for example the full SBox computation can be proven t-SNI and therefore PINI with 4 multiplications and 2 mask refreshing only (instead of 4 mask refreshing as in the naive "double-SNI" strategy).

Proposition 3 (Double-SNI [CS18]). *Let G be a $(n-1)$-SNI gadget taking as input $(a_i)_{1\leq i\leq n}$ and $(b_i)_{1\leq i\leq n}$, and outputting $(c_i)_{1\leq i\leq n}$. Let R be a $(n-1)$-SNI gadget taking as input $(x_i)_{1\leq i\leq n}$ and outputting $(y_i)_{1\leq i\leq n}$. The composite gadget G' taking as input $(x_i)_{1\leq i\leq n}$ and $(b_i)_{1\leq i\leq n}$, and outputting $(c_i)_{1\leq i\leq n}$, with $G'((x_i),(b_i)) = G(R((x_i)),(b_i))$ is PINI.*

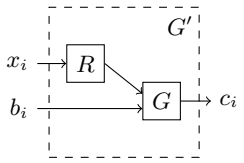

Figure 1. The double-SNI approach: when both gadgets G and R are $(n-1)$-SNI, the composite gadget G' is PINI.

Finally, we recall in Appendix B the SecMult gadget used in [RP10] for protecting AES against t-th order attacks. It is an extension to $\mathbb{F}_{2^k}$ of the original ISW countermeasure [ISW03] described in $\mathbb{F}_2$. The SecMult gadget was proven t-SNI in [BBD+16]. We also recall in the full version of our paper [CGZ19] the mask refreshing gadget FullRefresh introduced by Duc *et. al* in [DDF14], based on SecMult; it was also proven t-SNI in [BBD+16]. We can therefore use the FullRefresh gadget to apply the above "double-SNI" strategy. Moreover, in this paper, when we describe a variant of SecMult, we apply the same modifications to the FullRefresh gadget; this is straightforward, since the FullRefresh gadget can be seen as a SecMult with one input equal to $(1,0,\ldots,0)$.

2.3 r-wise independent PRG: definition and construction

We recall the definition of an r-wise independent pseudo-random generator (PRG). We denote by U_n the uniform distribution in $\{0,1\}^n$.

Definition 5 (r-wise independent PRG). *A function $G : \{0,1\}^n \to \{0,1\}^m$ is an r-wise independent pseudo-random generator if any subset of r bits of $G(x)$ is uniformly and independently distributed when $x \leftarrow U_n$*

We can construct an r-wise independent PRG via polynomial evaluation in a finite field $\mathbb{F}$. Letting $\boldsymbol{a} = (a_0, \ldots, a_{r-1}) \in \mathbb{F}^r$, we consider the polynomial:

$$h_{\boldsymbol{a}}(x) = \sum_{i=0}^{r-1} a_i x^i$$

For any $m \leq |\mathbb{F}|$, we can define the function $G : \mathbb{F}^r \to \mathbb{F}^m$ by letting:

$$G(\boldsymbol{a}) = (h_{\boldsymbol{a}}(0), \ldots, h_{\boldsymbol{a}}(m-1))$$

where we assume that we have some indexing of the field elements in $\mathbb{F}$. The function G is an r-wise independent PRG because there is a bijection between the r coefficients of a polynomial of degree at most $r-1$ and its evaluation at r distinct points x_i.

For $\mathbb{F} = \mathbb{F}_{2^k}$, this gives an r-wise independent PRG taking as input rk bits and outputting at most $k \cdot 2^k$ bits. Namely when working over $\mathbb{F}_{2^k}$ and generating k-bit pseudo-randoms, we can use each individual bit of the k-bit pseudo-random, and the PRG function remains r-wise independent. The parameter k determines the expansion factor of the PRG. For our application to AES in Section 5, for simplicity we will work over $\mathbb{F}_{2^{16}}$, using $\mathbb{F}_{2^8}$ as a subfield. For a block-cipher using single bits, one would work in $\mathbb{F}_{2^k}$ and use each of the k bits of $\mathbb{F}_{2^k}$ separately.

A simple 3-wise independent PRG. We also consider a very simple PRG that achieves 3-wise independence only. We consider a set of $2d$ random bits x_i and y_i for $1 \leq i \leq d$. We define the following function $G : \{0,1\}^{2d} \to \{0,1\}^{d^2}$:

$$G(x_1, \ldots, x_d, y_1, \ldots, y_d) = (x_i \oplus y_j)_{1 \leq i,j \leq d}$$

The function G can be seen as a PRG based on expander graph; see the full version of our paper [CGZ19].

Lemma 1. *The function G is a 3-wise independent PRG.*

Proof. We must show that any 3 variables $(x_{i_1} \oplus y_{j_1})$, $(x_{i_2} \oplus y_{j_2})$ and $(x_{i_3} \oplus y_{j_3})$ are uniformly and independently distributed.

We distinguish 3 cases. If $\#\{i_1, i_2, i_3\} = 3$, then the three values are independent thanks to randoms x_{i_1}, x_{i_2} and x_{i_3}. If $i_1 = i_2 = i_3$, then we must have $\#\{j_1, j_2, j_3\} = 3$ and the three values are independent thanks to randoms y_{j_1}, y_{j_2} and y_{j_3}. Eventually, if exactly two indices among i_1, i_2 and i_3 are equal, say wlog $i_1 = i_2 \neq i_3$, then we must have $j_1 \neq j_2$ and the randoms y_{j_1}, y_{j_2} and x_{i_3} ensure the independence of the three values. □

2.4 Robust PRG: definition and trivial construction

In [IKL+13], the authors introduced the notion of *robust* pseudo-random number generator (PRG), which should remain secure even if an adversary can probe at most k intermediate variables in the PRG circuit. We recall the definition of (strongly) robust PRG from [IKL+13] below. Under this definition, the output bits of the PRG must remain r-wise independent outside some set T of bounded size, conditioned on the values of any set S of at most k probes in the PRG circuit and the outputs in T.

In this paper we actually use a slightly weaker definition of strong robustness compared to [IKL+13], in which we allow the output bits outside the set T to be only $(r - q|S|)$-wise independent, instead of r-wise independent, where $|S| \leq k$ is the number of probes and q a parameter. In other words, we allow the r-wise independence of the PRG to degrade gracefully with the number of probes. This will give slightly more efficient constructions; in particular, the trivial construction of xoring $k + 1$ PRGs will only require the r-wise independence of each PRG, instead of the $(r + k)$-wise independence in [IKL+13]. Obviously we need to ensure that a robust PRG under our definition can still be embedded in a private circuit with the same parameters as in [IKL+13]; see Theorem 1 below.

Definition 6 (Strong robust PRG [IKL+13] (adapted)). *A circuit implementation C of a PRG $G : \{0,1\}^n \to \{0,1\}^m$ is strong (r, k, q)-robust if given $Y = G(X)$ where $X \leftarrow \{0,1\}^n$, for any set S of at most k probes in C, there is a set T of at most $q|S|$ output bits such that conditioned on any fixing of the values C_S of the wires in S and of Y_T, the values $Y_{\overline{T}}$ of the output bits not in T are $(r - q|S|)$-wise independent and uniformly distributed.*

Trivial construction. As noted in [IKL+13], we can obtain a strong $(r, k, 1)$-robust PRG by taking the xor of $k+1$ PRGs, each with the r-wise independence property. More precisely, letting $g : \{0,1\}^n \to \{0,1\}^m$, we let $G : \{0,1\}^{n\cdot(k+1)} \to \{0,1\}^m$:

$$G(x_1, \ldots, x_{k+1}) = g(x_1) \oplus g(x_2) \oplus \cdots \oplus g(x_{k+1})$$

where the xors are performed from left to right.

Lemma 2 (Strong robustness of G). *If g is an r-wise independent PRG, then G is a strong $(r, k, 1)$-robust PRG.*

Proof. Since there are at most k probes and $k + 1$ PRGs, there exists an index $i^\star$ such that $g(x_{i^\star})$ has not been probed. In the following, we fix all inputs x_i except $x_{i^\star}$.

Let $t \leq k$ be the number of probes. We consider the set T of indices $j \in [1, m]$ such that the j-th bit of any partial sum $g(x_1) \oplus \cdots \oplus g(x_i)$ is probed. We must have $|T| \leq t$. Since g is an r-wise independent PRG, by definition any set of r output bits of $g(x_i^\star)$ is uniformly and independently distributed; this implies that any set of $r - t$ output bits of $g(x_i^\star)$ with indices outside T are uniformly and independently distributed, even conditioned on the output bits in T and the other probes. Since we have fixed the inputs of all other PRGs, this also applies for the output of G. Therefore G is a strong $(r, k, 1)$-robust PRG. □

Expander graph construction. Using an explicit construction of a bipartite expander graph [GUV09], the authors of [IKL+13] obtain a construction of a strong (r, k, q)-robust PRG with $r, k = n^{1-\eta}$ where n is the number of random input bits, for any $\eta > 0$. In the full version of our paper [CGZ19] we provide a simplified proof of strong robustness for expander graph based PRG, based on the proof of weak robustness from [IKL+13]. We also argue that for minimizing the amount of input randomness, while asymptotically better than the trivial construction, expander graph based constructions are actually impractical. Namely in our analysis the expander graph PRG construction based on [GUV09] becomes better than the trivial construction only for $r \geq 2^{18}$ and at least 2^{36} random input bits.

2.5 Application to private circuits

We recall below the main theorem from [IKL+13], showing that we can plug a robust PRG in a private circuit to generate all randomness from a small random seed, and the resulting construction remains secure against probing attacks. Firstly an important parameter is the locality ℓ of the randomness in the circuit.

Definition 7 (Randomness locality [IKL+13]). *A circuit C is said to make an ℓ-local use of its randomness if the value of each of its wires is determined by its (original, unmasked) input and at most ℓ bits of the randomness used in the circuit.*

Theorem 1 (Private circuit with PRG [IKL+13] (adapted)). *Suppose $C(\hat{\omega}, \rho)$ is a qk-private implementation of f with encoder I and decoder O, where C makes an ℓ-local use of its randomness, and uses at most m bits of randomness. Let $G : \{0,1\}^n \to \{0,1\}^m$ be a strong (r, k, q)-robust linear PRG with $r \geq k \cdot \max(\ell, q)$. Then, the circuit C' defined by $C'(\hat{\omega}, \rho') = C(\hat{\omega}, G(\rho'))$ is a k-private implementation of f with encoder I and decoder O which uses n random bits.*

The proof of Theorem 1 is based on showing that the view of any adversary who attacks with t probes an implementation in which the randomness is generated by a PRG, can be simulated given the view of an adversary with at most qt probes who attacks an implementation with a true source of randomness; see Figure 3 for an illustration.

In the full version of our paper [CGZ19] we provide a proof that is essentially the same as in [IKL+13, Theorem 30], except that we use our slightly weaker definition of robustness. We recall the main steps of the proof below. We start with the following Lemma, which is similar to [IKL+13, Lemma 29]. As illustrated in Figure 2, any output of at most $r - q|S|$ bits of the robust PRG can be replaced by a TRNG and any set S of at most k probes in the PRG can be perfectly simulated using a subset T of the output with $|T| \leq q|S|$. This means that probing $|S|$ probes within the PRG is not better for the adversary than probing $q|S|$ outputs of the TRNG. To simplify notation, we will use G to denote both the function computed by a robust PRG and its circuit implementation. For a

set S of k wires in G, we denote by G_S the value of these wires; similarly, for a subset T of output bits of G, we denote by G_T the values of these output bits.

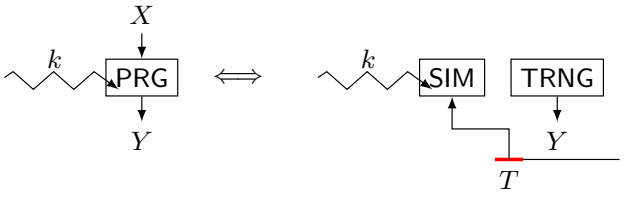

Figure 2. With a strong (r, k, q)-robust PRG, any output of at most $r - q|S|$ bits of the PRG can be replaced by a TRNG and any set S of at most k probes can be perfectly simulated using a subset T of the output with $|T| \leq q|S|$.

Lemma 3 (Robust PRG). *Let $G : \{0,1\}^n \to \{0,1\}^m$ be a strong (r, k, q)-robust linear PRG with $r \geq kq$. Let S be any set of at most k wires in G. Let $L \subset [m]$ be any subset of $r - q|S|$ bits. There exists a subset T with $|T| \leq q|S|$ such that the distribution of $Y = G_{L \cup T}(X)$ is uniform in $\{0,1\}^{|L \cup T|}$ when $X \leftarrow \{0,1\}^n$ and moreover $G_S(X)$ can be efficiently simulated given Y_T only.*

Thanks to Lemma 3 we can now prove Theorem 1. As illustrated in Figure 3, we can simulate any t probes within the PRG with a simulator SIM that uses qt random bits from the TRNG (see Fig. 2); these qt random bits can actually be queried by probing the original circuit C. This shows that when probing the PRG in C' the adversary does not learn more than by probing the circuit C with true randomness, as required; see the full version of our paper [CGZ19] for the details.

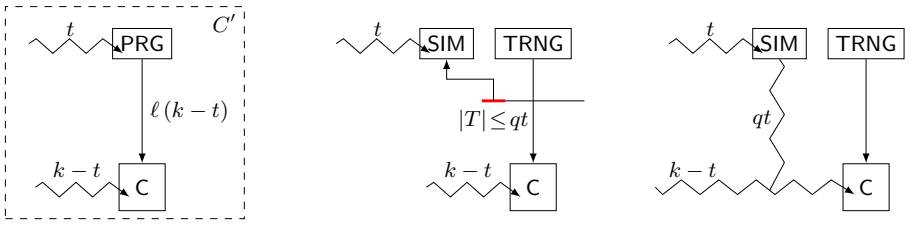

Figure 3. Security proof when plugging a PRG into a private circuit.

2.6 Locality refreshing

As recalled in Theorem 1, the r-wise independence parameter r of the PRG depends on the randomness locality ℓ of the circuit (see Definition 7). The goal is therefore to minimize the parameter ℓ. In the original ISW construction, the parameter ℓ would grow linearly with the circuit size; namely some wires can depend on almost all the randoms used in the circuit. To keep a small $\ell = \mathcal{O}(t^2)$, the authors of [IKL+13] use a mask refreshing at the end of each ISW gadget. Such locality refreshing, that we denote by LR, proceeds as described in Algorithm 1; see Figure 4 for an illustration.

Algorithm 1 Locality refreshing LR

Input: shares $x_1, \ldots, x_n$,
Output: shares $y_1, \ldots, y_n$ such that $\bigoplus_{i=1}^n y_i = \bigoplus_{i=1}^n x_i$

$y_n \leftarrow x_n$
for $i = 1$ **to** $n-1$ **do**
  $s \leftarrow \mathbb{F}_{2^k}$ # referred by s_i
  $y_i \leftarrow s$
  $y_n \leftarrow y_n \oplus (x_i \oplus s)$ # referred by $y_n^{(i)}$
end for
return $(y_1, \ldots, y_n)$

At the end of the algorithm, we have $y_i = s_i$ for all $1 \leq i \leq n-1$, and $y_n = x \oplus s_1 \oplus \cdots \oplus s_{n-1}$ for the secret $x = x_1 \oplus \cdots \oplus x_n$. Therefore one can show recursively over the circuit that the internal variables of the ISW multiplication depend on at most $\ell = \mathcal{O}(t^2)$ randoms, and this actually holds for any variable in the circuit. The following Lemma shows that the LR gadget is PINI, so that it can be included in a circuit without degrading its security.

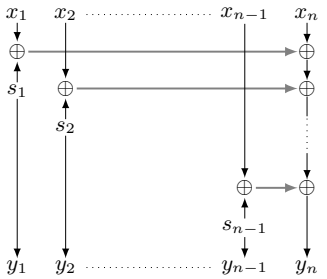

Figure 4. Locality refreshing algorithm.

Lemma 4 (PINI security of LR). *Let $(x_i)_{1 \leq i \leq n}$ be the input shares of the mask refreshing Algorithm LR. For any $t \in \mathbb{N}$, any set of t intermediate variables*

and any subset $\mathcal{O}$ of output indices, there exists a subset $I \subset [1,n]$ of indices such the t intermediate variables and the output shares $y_{|\mathcal{O}}$ can be perfectly simulated from the input shares $x_{|I\cup\mathcal{O}}$, with $|I| \leq t$.

Proof. We consider the following simple gadget G: $(x_1, x_n) \to (s_1, x_n \oplus (x_1 \oplus s_1))$, where s_1 is a random value. We start by showing that in Gadget G, we can always simulate t probes and $|\mathcal{O}|$ output variables from the input shares $x_{|I\cup\mathcal{O}}$, with $|I| \leq t$.

If $t + |\mathcal{O}| \geq 2$, we can let $I = \{1,n\} \setminus \mathcal{O}$ which gives $I \cup \mathcal{O} = \{1,n\}$ and all variables can be simulated from the input shares $x_{|I\cup\mathcal{O}}$. Moreover we have $|I| = |\{1,n\} \setminus \mathcal{O}| \leq 2 - |\mathcal{O}| \leq t$. If $t + |\mathcal{O}| = 1$, we distinguish two cases. If $|\mathcal{O}| = 1$ and $t = 0$, then we can simulate either s_1 or $x_n \oplus (x_1 \oplus s_1)$ by generating a random value. If $t = 1$ and $|\mathcal{O}| = 0$, we can simulate x_1 or x_n with $I = \{1\}$ or $I = \{n\}$; the other variables can be simulated by a random value.

We now consider the following gadget G_i for $1 \leq i \leq n-1$:

$$G_i : (x_1, \ldots, x_i, \ldots, x_n) \to (x_1, \ldots, x_{i-1}, s_i, x_{i+1}, \ldots, x_n \oplus (x_i \oplus s_i))$$

which is similar to Gadget G, but with n input shares instead of 2, and $n-2$ unmodified input shares. As previously, we can always simulate t probes and $|\mathcal{O}|$ output variables from the input shares $x_{|I\cup\mathcal{O}}$, with $|I| \leq t$. This implies that the gadget G_i is PINI. Since the LR gadget is the composition of $G_1, \ldots, G_{n-1}$, from Proposition 2 the LR gadget is also PINI. □

In [IKL+13] the LR algorithm is then applied after each ISW gadget. In particular, for the $\mathsf{SecMult}$ gadget recalled in Appendix B, we obtain the following $\mathsf{SecMultFLR}$ gadget. Since the original $\mathsf{SecMult}$ is t-SNI, the $\mathsf{SecMultFLR}$ gadget is also t-SNI. The same LR algorithm is applied after the Xor gadget and the $\mathsf{FullRefresh}$ gadgets (see the full version of our paper [CGZ19]).

Algorithm 2 $\mathsf{SecMultFLR}$

Input: shares a_i satisfying $\bigoplus_{i=1}^n a_i = a$, shares b_i satisfying $\bigoplus_{i=1}^n b_i = b$
Output: shares d_i satisfying $\bigoplus_{i=1}^n d_i = a \cdot b$
1: $c_1, \ldots, c_n \leftarrow \mathsf{SecMult}((a_i)_{1\leq i\leq n}, (b_i)_{1\leq i\leq n})$
2: $d_1, \ldots, d_n \leftarrow \mathsf{LR}(c_1, \ldots, c_n)$
3: **return** $(d_1, \ldots, d_n)$

Application to private circuits. We recall Claim 31 and Corollary 32 from [IKL+13]; we also recall the proof in the full version of our paper [CGZ19]. We use the notation $f(\lambda) = \tilde{\mathcal{O}}(g(\lambda))$ if $f(\lambda) = \mathcal{O}(g(\lambda)\log^k \lambda)$ for some $k \in \mathbb{N}$. We assume that the circuit size $s(\lambda)$ and the number of probes $t(\lambda)$ are both polynomial in the security parameter λ.

Lemma 5 (Private circuit with PRG [IKL$^+$13]). *Any function f with circuit size s admits a t-private implementation (I, C, O) with the canonical encoder I and decoder O, where C uses $\mathcal{O}(t^2 s)$ random bits and makes an $\ell = \mathcal{O}(t^2)$-local use of its randomness. Consequently, f admits a t-private implementation (I, C', O), where C' uses $\tilde{\mathcal{O}}(t^4)$ bits of randomness, and runs in time $\tilde{\mathcal{O}}(t^6 s)$, using the trivial construction. Using the expander graph construction, for any $\varepsilon > 0$, it uses $\tilde{\mathcal{O}}(t^{3+\varepsilon})$ random bits and runs in time $\tilde{\mathcal{O}}(t^2 s)$.*

2.7 Composing ℓ-local gadgets

In this section we provide an explicit definition of locality for a gadget, so that the locality property can be composed over a full circuit (as for the PINI definition for security against probing). As in [IKL$^+$13], the basic technique is to perform a locality refresh (such as Algorithm 1) of the output of each gadget. We say that a set of wires $(y_i)_{1 \leq i \leq n}$ is locality refreshed if $y_i = s_i$ for all $1 \leq i \leq n-1$, for randoms s_i, and $y_n = y \oplus s_1 \oplus \cdots \oplus s_{n-1}$, where y is the original unmasked variable. In the definition below of gadget locality, we take into account the randomness of the (locality refreshed) inputs.

Definition 8 (ℓ-local gadget). *Let G be a gadget whose output is locality refreshed. Consider the circuit C where G is given locality refreshed inputs $x_{\star,\star}$. Let ρ be the randomness used by C, including the randomness from the inputs. The gadget G is said to make an ℓ-local use of its randomness if C makes an ℓ-local use of its randomness ρ*

Theorem 2 (Composition of ℓ-local gadgets). *Any composite gadget made of ℓ-local gadgets is ℓ-local.*

Proof. We consider m gadgets $G_1, \cdots, G_m$ that we order as a direct acyclic graph from output to input in a reverse topological sort order. We assume that each gadget G_i makes an ℓ-local use of its randomness, with locality refreshed outputs. We prove by recurrence on n that the composition of ℓ-local gadgets is ℓ-local.

If $n = 1$, then there is only one gadget and this is straightforward since by assumption the gadget is ℓ-local. Now we assume that the composition of gadgets $G_1, \cdots, G_n$ is ℓ-local and we prove that the composition of gadgets $G_1, \cdots, G_{n+1}$ is still ℓ-local. Since the composition of gadgets $G_1, \cdots, G_n$ is ℓ-local, and since by definition the inputs of the gadget G_n are locality refreshed because they correspond to outputs of Gadget G_{n+1} which are locality refreshed, we get that the composition of both parts G_{n+1} and $G_1, \cdots, G_n$ does not increase the global locality. Namely, the global locality corresponds to the maximum locality between both parts. Since the composition of gadgets $G_1, \cdots, G_n$ is ℓ-local and since Gadget G_{n+1} is also ℓ-local, the maximum locality is ℓ and the composition of gadgets $G_1, \cdots, G_{n+1}$ is ℓ-local. □

In the above definition, in order to determine the locality ℓ of a gadget, we must therefore assume that it receives locality refreshed inputs, and the

randomness from this locality refreshed inputs must be taken into account when computing ℓ. Below we provide an example with the Xor gadget; the Xor gadget takes as input a_i and b_i for $1 \leq i \leq n$, and returns $c_i = a_i \oplus b_i$ for all $1 \leq i \leq n$.

Lemma 6 (Locality of Xor). *The* Xor *gadget followed by a locality refresh makes an ℓ-local use of its randomness, with $\ell = 2(n-1)$.*

Proof. The gadget takes as input a_i and b_i for $1 \leq i \leq n$, and then computes $c_i = a_i \oplus b_i$ for all $1 \leq i \leq n$, and finally $d_{n,j} = c_n \oplus (\oplus_{i=1}^{j} a_i \oplus b_i \oplus s_i)$ for $1 \leq j \leq n-1$, with outputs $d_i = s_i$ for $1 \leq i \leq n-1$ and $d_n = d_{n,n-1}$. We must consider $a_i = s_i^{(a)}$ for $1 \leq i \leq n-1$ and $a_n = a \oplus s_1^{(a)} \oplus \cdots \oplus s_{n-1}^{(a)}$, and similarly for b_i. Therefore c_n depends on $2(n-1)$ randoms, while $d_{n,j}$ depends on $2(n-1)-j$ randoms, which proves the lemma. □

We also compute the concrete locality ℓ of the SecMultFLR algorithm introduced above; in [IKL+13] only the asymptotic bound $\ell = \mathcal{O}(n^2)$ was proved. Such concrete locality computations will be important when implementing the countermeasure for AES in Section 5; namely for a locality ℓ, from Theorem 1 the r-wise independence parameter of the PRG must be set to $r = \ell t$ for security against t probes. We refer to the full version of our paper [CGZ19] for the proof.

Lemma 7 (Locality of SecMultFLR). *The* SecMult *algorithm followed by a final locality refresh (*SecMultFLR*) is an ℓ-local gadget with $\ell = n^2/4 + 5n/2 - c$, where $c = 3$ for even n, and $c = 11/4$ for odd n.*

3 Improving the locality of the multiplication gadget

In this section we describe two variants of the SecMult algorithm that improve the randomness locality of t-private circuits from $\ell = \mathcal{O}(t^2)$ to $\ell = \mathcal{O}(t)$. We show that this decreases the randomness complexity of private circuits from $\tilde{\mathcal{O}}(t^4)$ to $\tilde{\mathcal{O}}(t^3)$ using the trivial robust PRG construction. For our two new algorithms SecMultILR and SecMultILR2, we summarize in Table 3 below the number of required randoms and their locality ℓ. Since these randoms are eventually generated by a PRG, one should minimize their locality ℓ. We introduce SecMultILR first because the t-SNI proof of SecMultILR2 is significantly more complex.

	SecMult [ISW03]	SecMultFLR [IKL+13]	SecMultILR	SecMultILR2
Number of randoms	$n(n-1)/2$	$n(n-1)/2 + n - 1$	$n(n-1)$	$n(n-1)/2 + n - 1$
Locality ℓ	–	$n^2/4 + 5n/2 - c$	$4n - 5$	$4n - 6$
Security	t-SNI	t-SNI	t-SNI	t-SNI

Table 3. Summary of the multiplication gadgets, their locality and security. We have $c = 3$ for even n, and $c = 11/4$ for odd n.

3.1 First Construction with Internal Locality Refreshing (SecMultILR)

We describe below a variant of the SecMultFLR algorithm with locality $\ell = \mathcal{O}(t)$ instead of $\ell = \mathcal{O}(t^2)$. Our new SecMultILR is described below. The idea is to process the ISW matrix differently. In the original SecMult the final encoding is obtained by summing over all rows of the $n \times n$ ISW matrix. Instead we compute the partial sums over the rows of the successive $j \times j$ submatrices for $2 \leq j \leq n$. At each step we perform a locality refreshing of the j shares of the partial sum. In particular, the output of the algorithm is locality refreshed, so there is no need to apply the LR algorithm again.

Algorithm 3 SecMultILR

Input: shares a_i satisfying $\bigoplus_{i=1}^{n} a_i = a$, shares b_i satisfying $\bigoplus_{i=1}^{n} b_i = b$
Output: shares c_i satisfying $\bigoplus_{i=1}^{n} c_i = a \cdot b$

1: **for** $i = 1$ **to** n **do**
2: $\quad c_i \leftarrow a_i \cdot b_i$
3: **end for**
4: **for** $j = 2$ **to** n **do**
5: $\quad$ **for** $i = 1$ **to** $j - 1$ **do**
6: $\quad\quad r \leftarrow \mathbb{F}_{2^k}$ # referred by $r_{i,j}$
7: $\quad\quad c_i \leftarrow c_i \oplus r$ # referred by $c_{i,j}$
8: $\quad\quad r \leftarrow (a_i \cdot b_j \oplus r) \oplus a_j \cdot b_i$ # referred by $r_{j,i}$
9: $\quad\quad c_j \leftarrow c_j \oplus r$ # referred by $c_{j,i}$
10: $\quad$ **end for**
11: $\quad$ **for** $i = 1$ **to** $j - 1$ **do**
12: $\quad\quad s \leftarrow \mathbb{F}_{2^k}$ # referred by $s_{i,j}$
13: $\quad\quad c_j \leftarrow c_j \oplus (c_i \oplus s)$ # referred by $c_{j,i}$
14: $\quad\quad c_i \leftarrow s$
15: $\quad$ **end for**
16: **end for**
17: **return** $(c_1, \ldots, c_n)$

We see that lines 6 to 9 are the same as in the original SecMult (see Appendix B), except that they are processed in a different order, since the loop starts with j instead of i. This implies that at Step 10 we have processed the $j \times j$ submatrix of the ISW matrix, and therefore the first j shares c_i must satisfy the equality:

$$c_1 \oplus \cdots \oplus c_j = (a_1 \oplus \cdots \oplus a_j) \cdot (b_1 \oplus \cdots \oplus b_j) \tag{2}$$

From lines 11 to 15 we then perform a locality refresh of these j shares $(c_i)_{i=1}^{j}$ using new randoms s_{ij}; therefore after the locality refresh the new shares c_i satisfy the same equality (2), but now they only depend on the $j - 1$ randoms s_{ij} for $1 \leq i \leq j - 1$, and not on the r_{ij}'s. This implies that at the next step of the loop (for index $j + 1$), the shares c_i will only depend on a linear number of randoms r_{ij}, instead of quadratic in the original SecMult. Thanks to

these internal locality refreshings, the new locality parameter becomes $\ell = \mathcal{O}(t)$ instead of $\ell = \mathcal{O}(t^2)$.

Lemma 8 (Locality of SecMultILR). *The* SecMultILR *algorithm is an ℓ-local gadget with $\ell = 4n - 5$ for $n \geq 3$.*

Theorem 3 (Completeness of SecMultILR). *The* SecMultILR *algorithm, when taking $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ as inputs, outputs $c_1, \ldots, c_n$ such that $c_1 \oplus \cdots \oplus c_n = (a_1 \oplus \ldots \oplus a_n) \cdot (b_1 \oplus \ldots \oplus b_n)$.*

Theorem 4 (t-SNI of SecMultILR). *The* SecMultILR *algorithm is t-SNI for any $1 \leq t \leq n - 1$.*

One can therefore use a robust PRG with r-wise independence parameter $r = \ell \cdot t = \mathcal{O}(t^2)$ instead of $r = \mathcal{O}(t^3)$ in [IKL+13]. With the trivial construction of xoring $t + 1$ PRGs, the number of input randoms in the finite field becomes $r \cdot (t + 1) = \mathcal{O}(t^3)$ instead of $\mathcal{O}(t^4)$. This gives the following lemma, which improves over Lemma 5 from [IKL+13].

Lemma 9 (Efficiency properties of SecMultILR). *Any function of circuit size s admits a t-private implementation (I, C, O) with the canonic encoder I and decoder O, where C uses $\tilde{\mathcal{O}}(t^3)$ bits of randomness using the trivial construction, and runs in time $\tilde{\mathcal{O}}(s \cdot t^5)$.*

3.2 Second construction with less randomness (SecMultILR2)

We describe in the full version of our paper [CGZ19] a variant called SecMultILR2 of the previous algorithm, that achieves the same locality ℓ as SecMultILR but with roughly half as many randoms. It uses the same number of randoms as SecMultFLR from [IKL+13], but with locality $\mathcal{O}(t)$ instead of $\mathcal{O}(t^2)$. Therefore it is strictly better than both SecMultFLR and SecMultILR; see Table 3.

Lemma 10 (Locality of SecMultILR2). *The* SecMultILR2 *gadget uses ℓ-local randomness, with $\ell = 4n - 6$ for $n \geq 3$.*

Theorem 5 (Completeness of SecMultILR2). *The* SecMultILR2 *algorithm, when taking $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ as inputs, outputs $c_1, \ldots, c_n$ such that $c_1 \oplus \cdots \oplus c_n = (a_1 \oplus \ldots \oplus a_n) \cdot (b_1 \oplus \ldots \oplus b_n)$.*

Theorem 6 (t-SNI of SecMultILR2). *The* SecMultILR2 *is t-SNI for any $1 \leq t \leq n - 1$.*

3.3 Formal verification of locality and security

We have performed a formal verification of the above locality and security lemmas, using the CheckMasks tool [Cor18]. We refer to the full version of this paper [CGZ19] for the details.

4 Private Circuits with Multiple PRGs without Robustness

In the previous section we have described two variants of SecMult where following the [IKL+13] paradigm a single robust PRG is used to generate all the randoms from the circuit; by improving the locality parameter from $\ell = \mathcal{O}(t^2)$ to $\ell = \mathcal{O}(t)$, we have decreased the number of input random bits from $\tilde{\mathcal{O}}(t^4)$ to $\tilde{\mathcal{O}}(t^3)$, that is independent of the circuit size s (up to logarithmic factors). In this section, we show that by using multiple independent PRGs instead of a single one, the robustness property of the PRG is not required anymore, and therefore much more efficient PRG constructions can be used; this allows to decrease the randomness complexity of private circuits down to $\tilde{\mathcal{O}}(t^2)$.

We start with a simple observation. In the security proof of ISW, if the attacker probes a given random r_{ij} in some SecMult gadget, then it is easy to see that we could give away to the attacker not only the probed r_{ij}, but actually *all* randoms $r_{ij}^{(k)}$ for the same i, j in all other SecMult gadgets k; namely in the ISW security proof with global index I, one would have $i \in I$, and therefore each $r_{ij}^{(k)}$ would then be simulated by letting $r_{ij}^{(k)} \leftarrow \mathbb{F}$ as in the original circuit, so it could be given to the attacker without requiring the knowledge of more input shares.

Now assume that for every pair (i, j) we use an independent PRG to generate the randoms $r_{ij}^{(k)}$ for all gadgets k. In that case the attacker has no advantage in probing the intermediate variables of the PRG circuit, since in our extended probing model he could get all corresponding randoms $r_{ij}^{(k)}$ with a single probe anyway. Therefore when each r_{ij} has a dedicated PRG (see Figure 5 for an illustration), the robustness property of the PRG is not required anymore, and we can use a simple PRG with r-wise independence only, as for example the PRG based on polynomial evaluation from Section 2.3.

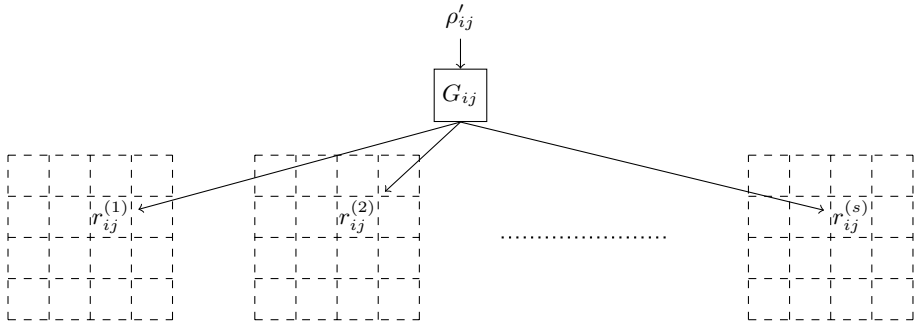

Figure 5. In Construction 1, each r_{ij} has its dedicated PRG across all gadgets, from a random seed ρ'_{ij}.

Moreover, if a mask locality refreshing is performed at the end of each multiplication gadget, it is easy to see that any intermediate variable of the circuit can depend on at most a *single* random $r_{ij}^{(k)}$ for a fixed i, j, and therefore the locality with respect to each randomness subset $\rho_{ij} = \{r_{ij}^{(k)} : 1 \leq k \leq s\}$ is $\ell = 1$; this is because the locality refresh at the end of each multiplication gadget cancels the dependence on the internal $r_{ij}^{(k)}$. In that case, with t probes on intermediate variables the adversary can get information on at most t randoms within such set. Therefore these randoms can be generated by a PRG with r-wise independence parameter $r = t$. Since the robustness property is not required, we can use a PRG based on polynomial evaluation that requires only $r = t$ coefficients in a finite field, and therefore $\tilde{\mathcal{O}}(t)$ random bits per PRG. Since there are $\mathcal{O}(t^2)$ randoms r_{ij}, we need $\mathcal{O}(t^2)$ independent PRGs to generate all of them, and the total number of input random bits is therefore $\tilde{\mathcal{O}}(t^3)$, as in our single PRG constructions from Section 3. Note that the time to generate a pseudo-random is now $\tilde{\mathcal{O}}(t)$, instead of $\tilde{\mathcal{O}}(t^3)$ in Section 3.

We can improve the above randomness complexity as follows. Firstly, we observe as previously that in the security proof of ISW, whenever the attacker probes a random r_{ij}, we can actually give to the attacker the complete row of r_{ij}'s, that is for a given i, all r_{ij} with $i < j \leq n$; and more generally, for a fixed i, all randoms $r_{ij}^{(k)}$ with $i < j \leq n$ in all SecMult gadgets k. Therefore as previously we can use for each $1 \leq i < n$ a dedicated PRG to generate all $r_{ij}^{(k)}$ for all $i < j \leq n$ in all gadgets k, without needing the robustness property. Since we generate the complete row of r_{ij}'s (see Fig. 8 for an illustration), we only need $\mathcal{O}(t)$ independent PRGs, instead of $\mathcal{O}(t^2)$.

Moreover, if we perform internal mask refreshing as in the SecMultILR algorithm from Section 3 (instead of only at the end of the SecMult gadget), then no intermediate variable can depend on two distinct r_{ij}'s in the same row i. This implies that the locality with respect to the randomness subset $\rho_i = \{r_{ij}^{(k)} : i < j \leq n, 1 \leq k \leq s\}$ is still equal to 1. Therefore a PRG can be used to generate all $r_{ij}^{(k)}$ from a given row i in all gadgets k, still with r-wise independence parameter $r = t$. Since we need only $\mathcal{O}(t)$ independent PRGs instead of $\mathcal{O}(t^2)$ previously, the number of input random bits goes down to $\tilde{\mathcal{O}}(t^2)$, while the time to generate a pseudo-random is still $\tilde{\mathcal{O}}(t)$. Asymptotically this is the most efficient technique (see Table 1), and also the most efficient in practice (see Section 5 for our implementation results on AES).

4.1 Security with multiple PRGs

The following lemma shows that the PRG robustness is not needed when the PRG generates only a subset ρ of the randomness, and the adversary can get ρ with a single probe; the lemma is analogous to Theorem 1 for a single robust PRG. We first consider a circuit C where we split the randomness in two parts ρ and $\bar{\rho}$, where only the randomness ρ will be replaced by pseudo-randoms. We consider an extended security model in which the attacker can get ρ with a single

probe. Intuitively probing the PRG that generates ρ does not help the attacker, since in the extended security model he can get ρ with a single probe.

Lemma 11 (Security from r-wise independent PRG). *Suppose C is a t-private implementation of f with encoder I and decoder O, where $C(\hat{\omega}, \rho, \bar{\rho})$ uses m random bits ρ and makes an ℓ-local use of its randomness ρ, and the adversary can obtain ρ with a single probe. Let $G : \{0,1\}^{n_r} \to \{0,1\}^m$ be a linear ℓt-wise independent PRG. Then, the circuit C' defined by $C'(\hat{\omega}, \rho', \bar{\rho}) = C(\hat{\omega}, G(\rho'), \bar{\rho})$ is a t-private implementation of f with encoder I and decoder O which uses n_r random bits ρ' and random $\bar{\rho}$.*

Proof. We show that the view of an adversary A' who attacks $C'(\hat{\omega}, \rho', \bar{\rho})$ by probing a set S of $t' \leq t$ wires in G and a set of P of $t - t'$ wires in C is independent of the secret input ω. Since C is t-private, it suffices to show that that the view of A' can be simulated given the view of an adversary A who probes at most t wires in $C(\hat{\omega}, \rho, \bar{\rho})$, and who can obtain the randomness ρ with a single probe.

Since C makes an ℓ-local use of its randomness ρ, the $t - t'$ probes from the set P in the circuit C can depend on at most $\ell(t - t') \leq \ell t$ bits of ρ. More precisely, for any $\hat{\omega}$ and $\bar{\rho}$, let $Q_{\hat{\omega},\bar{\rho}}(\rho) = C_P(\hat{\omega}, \rho, \bar{\rho})$ be the value of these probes; the function $Q_{\hat{\omega},\bar{\rho}}$ depends on at most ℓt bits of ρ. Let $T \subset [1, m]$ be the corresponding subset of bits of ρ on which $Q_{\hat{\omega},\bar{\rho}}$ depends, with $|T| \leq \ell t$; we can write $Q_{\hat{\omega},\bar{\rho}}(\rho) = Q'(\rho_T)$, where ρ_T is the corresponding subset of ρ.

We now proceed as follows. Instead of generating the PRG seed $X \leftarrow \{0,1\}^{n_r}$ and then the PRG output $G_T(X)$ corresponding to T, we can first generate the PRG output $\rho_T \leftarrow \{0,1\}^{|T|}$ and then sample the PRG seed; this is possible because G is a linear ℓt-wise independent PRG, and moreover $|T| \leq \ell t$. More precisely, since G is a linear ℓt-wise PRG, there exists a randomized simulator Sim that can perfectly sample the PRG input and therefore the probes within the PRG, given at most ℓt bits of PRG output; formally this means $(G_S(X), G_T(X)) \equiv (\mathsf{Sim}(\rho_T), \rho_T)$ where $X \leftarrow \{0,1\}^{n_r}$ and $\rho \leftarrow \{0,1\}^m$. We obtain:

$$(G_S(X), Q'(G_T(X))) \equiv (\mathsf{Sim}(\rho_T), Q'(\rho_T))$$

We now distinguish two cases. If the number of probes within the PRG is such that $t' \geq 1$, we let $\mathsf{Sim}'(\rho_T, v) = (\mathsf{Sim}(\rho_T), v)$ and we obtain:

$$(G_S(X), Q'(G_T(X))) \equiv (\mathsf{Sim}(\rho_T), Q'(\rho_T)) \equiv \mathsf{Sim}'(\rho_T, Q'(\rho_T))$$

which gives $(G_S(X), Q_{I(\omega),\bar{\rho}}(G(X))) \equiv \mathsf{Sim}'(\rho_T, Q_{I(\omega),\bar{\rho}}(\rho))$. In this case, the distribution to which Sim' is applied captures the view of an adversary A who corrupts a set $T \cup P$ of wires in C, where $|P| \leq t - t'$ and by definition ρ_T can be obtained with a single probe, which gives a total of at most $t - t' + 1 \leq t$ probes in C. Since by assumption C is t-private, this view is independent of the secret ω. Since the distribution on the left hand side captures the view of A', it follows that the view of A' is also independent of ω, as required.

In the second case, G is not probed by the adversary A'. Since G is ℓt-wise independent and the view of A' depends on at most ℓt bits of ρ, the view of A'

is the same as the view of an adversary A probing the same wires in C. More precisely, we have from $G_T(X) \equiv \rho_T$:

$$Q_{I(\omega),\bar{\rho}}(G(X)) \equiv Q_{I(\omega),\bar{\rho}}(\rho)$$

As previously, the right hand side corresponds to the view of an adversary A who corrupts a set P of at most t wires in C and the distribution of the left hand side captures the view of A'; therefore the view of A' is independent of ω also in the second case. □

We now consider the main theorem where the circuit randomness ρ can be split into $(\rho_i)_{i=1}^k$, and when considering each ρ_i separately, the circuit C makes an ℓ-local use of ρ_i; moreover we assume that C remains t-private even if the adversary can obtain each ρ_i with a single probe. The proof follows from a recursive application of Lemma 11.

Theorem 7 (Security with multiple PRGs). *Suppose C is a t-private implementation of f with encoder I and decoder O, where the circuit $C(\hat{\omega}, \rho_1, \ldots, \rho_k)$ uses for each $1 \leq i \leq k$, m random bits ρ_i, and makes an ℓ-local use of ρ_i, and the adversary can obtain each ρ_i with a single probe. Let $G : \{0,1\}^{n_r} \to \{0,1\}^m$ be a linear ℓt-wise independent PRG. Then, the circuit C' defined by $C'(\hat{\omega}, \rho'_1, \ldots, \rho'_k) = C(\hat{\omega}, G(\rho'_1), \ldots, G(\rho'_k))$ is a t-private implementation of f with encoder I and decoder O which uses $k \cdot n_r$ random bits.*

4.2 Extended Security Model: PINI-R

In Theorem 7 above we have considered an extended model of security, where the adversary can get any randomness subset ρ_i in the circuit with a single probe. Therefore, we define a variant of the PINI notion from [CS18], called PINI-R, in which the adversary can also get access to a subset of the randoms in a gadget, using a single probe.

Definition 9 (PINI-R). *Let G be a gadget with input shares $x_{i,\star}$ and output shares $y_{i,\star}$. Let $(\rho_i)_{1 \leq i \leq n}$ be a partition of the randoms used by G. The gadget G is PINI-R if for any $t_1 \in \mathbb{N}$, any set of t_1 intermediate variables, any subset $\mathcal{O}$ of output indices and any subset $R \subset [1, n]$, there exists a subset $I \subset [1, n]$ of input indices with $|I| \leq t_1$ such that the t_1 intermediate variables, the output shares $y_{|\mathcal{O} \cup R,\star}$ and the randoms ρ_i for $i \in R$ can be perfectly simulated from the input shares $x_{|I \cup \mathcal{O} \cup R,\star}$.*

The following proposition is analogous to Proposition 1. It shows that if a gadget with $n = t+1$ shares is PINI-R, then a t-probing adversary learns nothing about the underlying secrets, even in an extended model of security where the adversary can get each randomness subset ρ_i with a single probe. We provide the proof in the full version of our paper [CGZ19].

Proposition 4 (PINI-R security). *Let G be a gadget with input shares $x_{i,\star}$ and output shares $y_{i,\star}$ for $1 \leq i \leq n$. Let $(\rho_i)_{1 \leq i \leq n}$ be a partition of the randomness used by G. If G is PINI-R, then G is $(n-1)$-probing secure in an extended model of security where the adversary can get each ρ_i with a single probe.*

In the composition theorem below, the attacker can get the union of all corresponding subsets of randoms from all gadgets, still with a single probe; see the full version of our paper [CGZ19] for the proof.

Theorem 8 (Composition of PINI-R). *Any composite gadget made of PINI-R composing gadgets G_i for $i \in K$ is PINI-R, where for the composite gadget we take the randomness partition $\rho_i = \bigcup_{k \in K} \rho_i^{(k)}$ for $1 \leq i \leq n$.*

It is straightforward to prove the PINI-R property of the locality refreshing algorithm from Section 2.6, with the randomness partition $\rho_i = \{s_i\}$ for $1 \leq i \leq n-1$. In the full version of our paper [CGZ19] we consider an analogous extension of the t-SNI property, called t-SNI-R, which we prove for the SecMult and SecMultILR constructions, and the corresponding FullRefresh. More precisely, we show that those gadgets remain secure in an extended model of security where the adversary can get all randoms r_{ij} (and all randoms s_{ij} for SecMultILR) for a given i with a single probe. Moreover the "double-SNI" approach still works for the t-SNI-R and PINI-R notions. This implies that we can base our construction on t-SNI-R and PINI-R gadgets, and the resulting construction will be PINI-R. Note that the t-SNI security proof of SecMultILR2 is already complex, so we will not try to prove the t-SNI-R property of SecMultILR2; therefore we will use the multiple PRGs approach for SecMultFLR and SecMultILR only.

4.3 Constant locality with respect to a randomness subset

In this section we show that we can achieve constant locality, even $\ell = 1$, when we consider different subsets of randomness. Therefore we first provide a definition of gadget locality with respect to a subset of the gadget randomness only (and excluding the randomness of the inputs, as opposed to Section 2.7), and then a locality composition theorem as in Section 2.7.

Definition 10 (ℓ-local gadget with randomness subset). *Let G be a gadget and let ρ be a subset of the randomness used by G. The gadget G is said to make an ℓ-local use of its randomness ρ if any intermediate variable of G depends on at most ℓ bits of ρ.*

For example, the SecMult gadget makes a 1-local use of its randomness $\rho = \{r_{ij}\}$ for any $1 \leq i < j \leq n$; this is obvious, since ρ contains a single random bit. We can now state our composition theorem for locality with respect to a randomness subset. It shows that the gadget locality ℓ is kept the same in the composite gadget, while the locality of the randoms used for output refreshing is equal to 3 with respect to each subset $\{s_i^{(k)}, k \in K\}$ for $1 \leq i \leq n-1$. We refer to the full version of our paper [CGZ19] for the proof.

Theorem 9 (Locality composition with randomness subset). *Let G_k for $k \in K$ be a set of fan-in 2 gadgets which all make an ℓ-local use of a subset ρ_k of their randomness. Consider the gadgets G'_k for $k \in K$ where the output of G_k is locality refreshed with randoms $s_i^{(k)}$ for $1 \leq i \leq n-1$. Any composite*

gadget made of G'_k makes an ℓ-local use of the randomness $\bigcup_{k\in K}\rho_k$, and for any $1 \le i \le n-1$, it makes a 3-local use of the randoms in $\{s_i^{(k)} : k \in K\}$.

For example if we compose a number of SecMultFLR gadgets, in the composite gadget the locality with respect to the randoms $r_{ij}^{(k)}$ for fixed i, j is $\ell = 1$, while the locality with respect to the randoms $s_i^{(k)}$ for fixed i from the output locality refreshing is $\ell = 3$. We stress that in the final implementation all the randomness (including the randomness from the locality refreshing) will be generated by the PRGs. Finally, we show in the full version of our paper [CGZ19] that the latter locality can be brought down to 1; for this it suffices to additionally perform a locality refreshing of the two inputs of each gadget, with independent sets of PRGs for the two inputs.

4.4 First construction: multiple PRGs with SecMultFLR

Our first construction is described in Figure 6. It consists in using the SecMult algorithm and perform a locality refresh after each gadget; this includes the SecMult gadget, the Xor gadget and the FullRefresh gadget. For every $1 \le i < j \le n$, an independent PRG generates all randoms $r_{ij}^{(k)}$ in the SecMult and FullRefresh gadgets. Similarly, for each $1 \le i \le n-1$, an independent PRG generates all randoms $s_i^{(k)}$ in all locality refreshing gadgets.

Construction 1: multiple PRGs with SecMultFLR

1. Given a circuit C, generate a private circuit (I, C', O) with $n = t + 1$ shares as follows:
   - replace every AND gate by the "double-SNI" gadget with SecMult and FullRefresh. Perform a locality refreshing LR after SecMult and FullRefresh.
   - replace every XOR gate by the Xor gadget. Perform a locality refreshing LR after each Xor gadget.
2. Initialize $n(n-1)/2$ PRG functions G_{ij} for $1 \le i < j \le n$, each with r-wise independence parameter $r = t$.
3. Generate all randoms $r_{ij}^{(k)}$ in SecMult or FullRefresh gadget k with the PRG function G_{ij}.
4. Initialize $n-1$ PRG functions G'_i for $1 \le i < n$, each with r-wise independence parameter $r = 3t$.
5. Generate all randoms $s_i^{(k)}$ in the LR algorithm from gadget k using the PRG function G'_i.

Figure 6. Private circuit construction with multiple PRGs with SecMultFLR.

From the locality composition theorem (Theorem 9), in the global construction the locality with respect to the randoms $\{r_{ij}^{(k)} : k \in K\}$ is $\ell_r = 1$, while

the locality with respect to the randoms $\{s_i^{(k)} : k \in K\}$ is $\ell_s = 3$. From the PINI-R property of the gadgets and Theorem 8, the full circuit is PINI-R. Therefore, from Proposition 4, it is secure in an extended model of security in which the adversary can get the previous randomness subsets with a single probe. From Lemma 11, the PRGs for the r_{ij}'s must be t-wise independent, while the PRGs for the s_i's must be $3t$-wise independent. Since one requires $n(n-1)/2$ independent PRGs for the r_{ij}'s, and $n-1$ independent PRGs for the s_i's, the number of input randoms in the finite field is therefore, with $n = t+1$, $n_r = n(n-1)/2 \cdot t + (n-1) \cdot 3t = \mathcal{O}(t^3)$. Thus we have shown the following lemma. Compared to Lemma 9 for a single robust PRG with our SecMultILR algorithm, the randomness complexity is the same but the total running time goes down from $\tilde{\mathcal{O}}(st^5)$ to $\tilde{\mathcal{O}}(st^3)$.

Lemma 12 (multiple PRGs with SecMultFLR). *Any function of circuit size s admits a t-private implementation (I, C, O) with the canonic encoder I and decoder O, where C uses $\mathcal{O}(t^3 \cdot \log(st))$ bits of randomness, and runs in time $\mathcal{O}(s \cdot t^3 \cdot \log^2(st))$.*

4.5 Second construction: multiple PRGs with SecMultILR

Our second construction is described in Figure 7, based on the SecMultILR algorithm. As illustrated in Figure 8, a dedicated PRG generates the r_{ij}'s for a given row i, in all gadgets. We first show that the SecMultILR algorithm makes a 1-local use of each row of randoms r_{ij} and a 2-local use of each row of randoms s_{ij}; see the full version of our paper [CGZ19] for the proof.

Lemma 13 (Locality of SecMultILR). *The SecMultILR algorithm makes a 1-local use of each randomness set $\rho_i = \{r_{ij} : i < j \leq n\}$ and a 2-local use of each randomness set $\rho'_i = \{s_{ij} : i < j \leq n\}$.*

From Lemma 13 and Theorem 9, in the global construction the locality with respect to the subsets of randoms $\rho_i = \{r_{ij}^{(k)} : i < j \leq n, k \in K\}$ is equal to 1, the locality with respect to the subsets of randoms $\rho'_i = \{s_{ij}^{(k)} : i < j \leq n, k \in K\}$ is equal to 2, and the locality with respect to the subsets of randoms $\rho''_i = \{s_i^{(k)} : k \in K\}$ is still equal to 3, for each $1 \leq i < n$. As previously, from the PINI-R property of the gadgets and Proposition 8, the full circuit is PINI-R. Therefore, it is secure in an extended model of security in which the adversary can get the previous randomness subsets with a single probe. From Lemma 11, the corresponding PRGs must therefore have r-wise independence parameter $r = t$, $r = 2t$ and $r = 3t$ respectively. The main difference is that now there are only $n-1$ independent PRGs to generate the $r_{ij}^{(k)}$ (instead of $n(n-1)/2$ previously), because a given PRG generates those randoms for all indices j. The total number of input randoms in the finite field is therefore $n_r = (n-1) \cdot t + (n-1) \cdot 2t + (n-1) \cdot 3t = \mathcal{O}(t^2)$. Thus we have shown the following lemma. Asymptotically this is the most efficient technique (see Table 1 for a comparison), and also the most efficient in practice (see the next section for our implementation results on AES).

Construction 2: multiple PRGs with SecMultILR

1. Given a circuit C, generate a private circuit (I, C', O) with $n = t + 1$ shares as follows:
   - replace every AND gate by the "double-SNI" gadget with SecMultILR and the corresponding FullRefreshILR. Perform a locality refreshing LR after each SecMultILR and FullRefreshILR.
   - replace every XOR gate by the Xor gadget. Perform a locality refreshing LR after each Xor gadget.
2. Initialize $n-1$ PRG functions G_i for $1 \leq i < n$, each with r-wise independence parameter $r = t$.
3. Generate all randoms $r_{ij}^{(k)}$ in SecMultILR or FullRefreshILR gadget k with the PRG function G_i.
4. Initialize $n-1$ PRG functions G'_i for $1 \leq i < n$, each with r-wise independence parameter $r = 2t$.
5. Generate all randoms $s_{ij}^{(k)}$ in SecMultILR or FullRefreshILR gadget k using the PRG function G'_i.
6. Initialize $n-1$ PRG functions G''_i for $1 \leq i < n$, each with r-wise independence parameter $r = 3t$.
7. Generate all randoms $s_i^{(k)}$ in the LR algorithm using the PRG function G''_i.

Figure 7. Private circuit construction with multiple PRGs with SecMultILR.

Lemma 14 (multiple PRGs with SecMultILR). *Any function of circuit size s admits a t-private implementation (I, C, O) with the canonic encoder I and decoder O, where C uses $\mathcal{O}(t^2 \cdot \log(st))$ bits of randomness, and runs in time $\mathcal{O}(s \cdot t^3 \cdot \log^2(st))$.*

5 Application to AES

In this section we describe a concrete implementation of our techniques for AES; the goal is to minimize the total amount of randomness used to protect AES against t-th order attack. We provide the source code in C in [Cor19b].

5.1 The AES circuit and the Rivain-Prouff countermeasure

To implement the AES SBox, we need to perform 4 multiplications, and 2 mask refreshing per byte; see [RP10] for the sequence of operations. For the mask refreshing, we use the multiplication based refreshing FullRefresh recalled in the full version of our paper [CGZ19]. We refer to [BBD+16] for the proof that the x^{254} gadget is $(n-1)$-SNI; this implies that the gadget is PINI. Thus, this amounts to performing 6 multiplications per byte. Since there are 16 bytes to process per round, the number of required multiplications is $6 \times 16 = 96$ per round. Thus for the 10 rounds of the AES, one will perform $96 \times 10 = 960$ multiplications.

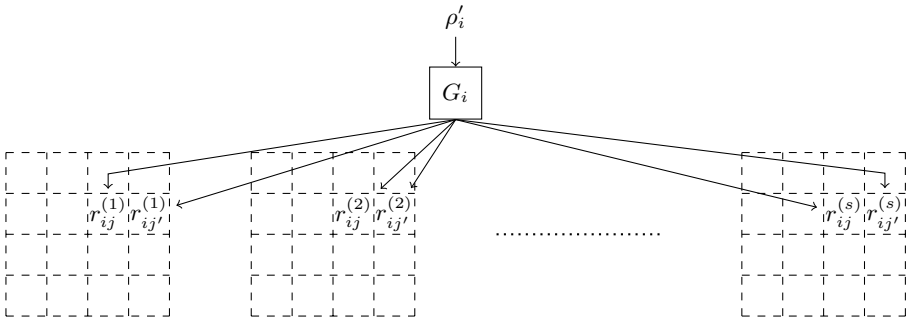

Figure 8. In Construction 2, a dedicated PRG generates the r_{ij}'s for a given row i in all gadgets, from a random seed ρ'_i.

5.2 Implementation with single robust PRG

We first consider an implementation with a single robust PRG as in Section 3, with 3 possible algorithms: the original [IKL+13] construction with a locality refresh after each multiplication gadget (SecMultFLR), and our new SecMultILR and SecMultILR2 algorithms. For those three algorithms, we provide in Table 4 the total number of pseudo-randoms to be generated for the AES circuit, the corresponding locality parameter ℓ, and the number of 8-bit randoms from the TRNG to generate the seed of the PRG, as a function of the number of shares n, for security against t probes with $n = t + 1$.

	SecMult [RP10]	SecMultFLR [IKL+13]	SecMultILR	SecMultILR2
Mult	$480n(n-1)$	$(480n+960)(n-1)$	$960n(n-1)$	$(480n+960)(n-1)$
Xor	–	$160(n-1)$		
Pseudo-rand	–	$(480n+1120)(n-1)$	$(960n+160)(n-1)$	$(480n+1120)(n-1)$
Locality ℓ	–	$\max(4(n-1),$ $n^2/4+5n/2-c)$	$4(n-1)$	$4(n-1)$
True-rand	$480n(n-1)$	$2n(n-1)\cdot\max(4(n-1),$ $n^2/4+5n/2-c)$	$8n(n-1)^2$	$8n(n-1)^2$

Table 4. For AES, total number of pseudo-randoms and number of 8-bit TRNG calls, for a single robust PRG, as a function of the number of shares n. We have $c = 3$ for even n, and $c = 11/4$ for odd n. We assume that $n \leq 12$.

We now explain the content of Table 4. For each of the 3 algorithms, the number of pseudo-randoms is the number of randoms from Table 3 in Section 3, multiplied by 960, since one must perform 960 multiplications. Furthermore, the MixColumns operation requires 48 xors. Normally we should perform a locality refresh after each xor, but in the particular case of the AES, we can do the locality refresh only after the 3 xors of the MixColumns for each byte. In that

case, the locality parameter with respect to MixColumns is then $4(n-1)$, instead of $2(n-1)$ for a single xor. The locality of the global circuit is then the max of locality parameter ℓ from Table 3 and $4(n-1)$. Equivalently, we can perform such locality refresh as input of the SubByte operation, which enables to keep the MixColumns unmodified. For the MixColumns, one therefore needs to perform 16 locality refresh per round, which gives a total of 160 locality refresh for the 10 rounds of the AES, which requires $160(n-1)$ pseudo-randoms. Finally, we assume that the round keys are already masked without PRG, and so we don't need to perform a locality refreshing after the AddRoundKey.

Let m the total number of pseudo-randoms over $\mathbb{F}_{2^8}$ that must be generated. To determine the finite field $\mathbb{F} = \mathbb{F}_{2^{8k}}$ used by the PRG, we must ensure $m \leq k \cdot |\mathbb{F}_{2^{8k}}| = k \cdot 2^{8k}$. Namely a single polynomial evaluation over $\mathbb{F}_{2^{8k}}$ generates k bytes of pseudo-random. One must then use a PRG with r-wise independence parameter $r = \ell \cdot (n-1)$. Using the trivial construction with the xor of $n = t+1$ polynomial evaluations (to provide resistance against t probes), the total number of fresh random values over $\mathbb{F}_{2^8}$ is then $n_r = k \cdot n \cdot r = k \cdot n(n-1) \cdot \ell$.

For the three algorithms one can work over $\mathbb{F}_{2^{16}}$ for $n \leq 12$; therefore for simplicity we take $k = 2$ in Table 4. For SecMultILR and SecMultILR2, the total number of TRNG calls over $\mathbb{F}_{2^8}$ is then $n_r = k \cdot n(n-1) \cdot 4(n-1) = 4k \cdot n(n-1)^2$ with $k = 2$ for $n \leq 12$, and $k = 3$ for $13 \leq n \leq 229$, instead of $480n(n-1)$ for the original Rivain-Prouff countermeasure; therefore one needs fewer TRNG calls than Rivain-Prouff for $n \leq 40$. We summarize in Table 6 below the number of input random bytes required for AES for small values of n, compared with the original Rivain-Prouff countermeasure.

5.3 Implementation with multiple PRGs

We now consider an implementation of AES with multiple PRGs, as in Section 4. We consider the SecMultFLR algorithm corresponding to Construction 1, and the SecMultILR algorithm corresponding to Construction 2. As previously, we provide in Table 5 the total number of pseudo-randoms to be generated for the AES circuit, and the number of 8-bit randoms from the TRNG.

As previously, we only perform a locality refresh after the 3 xors of the MixColumns (equivalently, before each SubByte). Moreover we don't perform the LR algorithm after SecMultILR as in Construction 2, since the output of SecMultILR is already locality refreshed. Therefore the number of pseudo-randoms is the same as in the previous section. We use two classes of independent PRGs. The first class of independent PRGs is used to generate the r_{ij}'s from SecMultFLR and SecMultILR algorithms, with locality $\ell_r = 1$; therefore the PRGs must be $\ell_r t$-wise independent. We need $n(n-1)/2$ such PRGs for SecMultFLR, and only $n-1$ for SecMultILR. Working over $\mathbb{F}_{2^{16}}$, each PRG requires $2\ell_r t = 2(n-1)$ random bytes. Similarly, the second class of PRGs is used to generate randoms s_i from the locality refresh, and also the randoms s_{ij} for the internal locality refresh in SecMultILR, with locality $\ell_s = 5$. Namely we only perform the locality refresh after the 3 xors of the MixColumns, and therefore the locality is $\ell_s = 5$ (instead of $\ell_s = 3$). Note that for SecMultILR we can use the same class of PRGs

	SecMult [RP10]	SecMultFLR	SecMultILR
Pseudo-rand	–	$(480n + 1120)(n-1)$	$(960n + 160)(n-1)$
Locality ℓ_r of r_{ij}	–	1	1
Number of PRGs (r_{ij})	–	$n(n-1)/2$	$n-1$
True-rand per PRG (r_{ij})	–	$2(n-1)$	$2(n-1)$
Locality ℓ_s of s_{ij} and s_i	–	5	5
Number of PRGs (s_i and s_{ij})	–	$n-1$	$n-1$
True-rand per PRG (s_{ij} and s_i)	–	$10(n-1)$	$10(n-1)$
Total True-Rand	$480n(n-1)$	$(n+10)(n-1)^2$	$12(n-1)^2$

Table 5. For AES, total number of Pseudo-random and True-random values to generate with the multiple PRGs approach, as a function of the number of shares n. Values for the Rivain-Prouff countermeasure are also recalled for comparison.

to generate the randoms s_{ij}'s from SecMultILR and the randoms s_i's from LR, instead of two classes in Construction 2 from Section 4; namely it is easy to see that the locality with respect to the corresponding randomness subsets is still equal to 5. Therefore the PRGs must be $\ell_s t$-wise independent; working over $\mathbb{F}_{2^{16}}$, each PRG requires $10(n-1)$ bytes of TRNG.

In summary, for SecMultFLR, the total number of 8-bit TRNG calls is therefore $n_r = n(n-1)/2 \cdot 2(n-1) + (n-1) \cdot 10(n-1) = (n+10)(n-1)^2$ and for SecMultILR, we get $n_r = (n-1) \cdot 2(n-1) + (n-1) \cdot 10(n-1) = 12(n-1)^2$ instead of $480n(n-1)$ in the original Rivain-Prouff countermeasure.

A simple 3-wise independent PRG. Finally, we consider the simple 3-wise independent PRG from Section 2.3:

$$G(x_1, \ldots, x_d, y_1, \ldots, y_d) = (x_i \oplus y_j)_{1 \le i,j \le d}$$

Since the PRG function G expands from $2d$ to d^2 bits (or bytes), the number of input randoms becomes $\mathcal{O}(\sqrt{s})$ instead of $\mathcal{O}(s)$, where s is the circuit size. Note that this is worse than the polynomial-based PRG used previously that requires only $\mathcal{O}(\log s)$ randoms, but the above function G is very fast since generating a pseudo-random only takes a single xor.

Since the above PRG only achieves 3-wise independence, we want to minimize the locality. Therefore, we perform a locality refresh of the 2 inputs of each gadget (with two distinct sets of independent PRGs), and we perform a locality refresh of the outputs of each gadget (SecMult, Xor and FullRefresh), using another distinct set of independent PRGs. As shown in the full version of our paper [CGZ19], the locality with respect to each subset of randoms is then always $\ell = 1$; therefore, we can use a PRG with r-wise independence $r = t = n-1$. This implies that this specific PRG only works for $n = 3$ and $n = 4$ shares. We argue in the full version of our paper [CGZ19] that the total number of input bytes for AES is 642 for $n = 3$ and 1056 for $n = 4$, instead of 2880 and 5760 respectively for the original Rivain-Prouff countermeasure.

		Single robust PRG			Multiple PRGs		
	[RP10]	SecMultFLR	SecMultILR	SecMultILR2	SecMultFLR	SecMultILR	3-wise SecMultFLR
$n = 3$	2880	96	96	96	52	48	642
$n = 4$	5760	288	288	288	126	108	1056
$n = 5$	9600	640	640	640	240	192	–
$n = 6$	14400	1260	1200	1200	400	300	–
$n = 7$	20160	2268	2016	2016	612	432	–
$n = 8$	26880	3696	3136	3136	882	588	–
$n = 9$	34560	5760	4608	4608	1216	768	–
$n = 10$	43200	8460	6480	6480	1620	972	–

Table 6. For AES, total number of TRNG bytes to generate for single and multiple PRGs methods, depending of the number of shares n. We also provide the number of TRNG bytes for the original Rivain-Prouff countermeasure.

Summary. We summarize in Table 6 the number of input random bytes required for AES for all previous methods, as a function of the number of shares n, in order to achieve t-th order security, with $t = n - 1$. We see that the most efficient method (in terms of minimizing the number of TRNG calls) is the SecMultILR algorithm with multiple PRGs. Namely for small values of t we obtain almost two orders of magnitude improvement compared to the original Rivain-Prouff countermeasure. We provide in Appendix A the results of an implementation of our countermeasure on an ARM-based embedded device. We provide the source code in [Cor19b].

References

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In *Advances in Cryptology - CRYPTO 2018 - Proceedings, Part III*, pages 427–455, 2018. Final version available at `https://eprint.iacr.org/2018/566.pdf`.

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In *Advances in Cryptology - EUROCRYPT 2015 - Proceedings, Part I*, pages 457–485, 2015.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129, 2016. Publicly available at `https://eprint.iacr.org/2015/506.pdf`.

[BBP+16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In *Advances in Cryptology - EUROCRYPT 2016 - Proceedings, Part II*, pages 616–648, 2016.

[CGZ19] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. Full version of this paper. Cryp-

tology ePrint Archive, Report 2019/1106, 2019. https://eprint.iacr.org/2019/1106.

[Cor18] Jean-Sébastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. In *Applied Cryptography and Network Security ACNS*, volume 10892, pages 65–82, 2018.

[Cor19a] Jean-Sébastien Coron. CheckMasks: formal verification of side-channel countermeasures, 2019. Publicly available at https://github.com/coron/checkmasks.

[Cor19b] Jean-Sébastien Coron. Implementation of higher-order countermeasures, 2019. Publicly available at https://github.com/coron/htable/.

[CS18] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. Cryptology ePrint Archive, Report 2018/438, 2018. https://eprint.iacr.org/2018/438.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *Advances in Cryptology - EUROCRYPT 2014 - Proceedings*, pages 423–440, 2014.

[FPS17] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In *Advances in Cryptology - ASIACRYPT 2017 - Proceedings, Part I*, pages 781–810, 2017.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009.

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust pseudorandom generators. In *ICALP 2013, Proceedings, Part I*, pages 576–588, 2013.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, Proceedings*, pages 463–481, 2003.

[OMHT06] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical second-order DPA attacks for masked smart card implementations of block ciphers. In *CT-RSA*, pages 192–207, 2006.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010, Proceedings*, pages 413–427, 2010.

A Concrete Implementation

We have implemented our constructions for AES in C, on a 44 MHz ARM-Cortex M3 processor. The processor is used in a wide variety of products such as passports, bank cards, SIM cards, secure elements, etc. The embedded TRNG module can run in parallel of the CPU, but it is relatively slow: according to our measurements on emulator, it outputs 32 bits of random in approximately 6000 cycles. Our results, obtained by running the code on emulator, are given in Table 7, and are compared with the classical Rivain-Prouff countermeasure.

We see that the most efficient countermeasure is the SecMultFLR algorithm with multiple PRGs, using the 3-wise independent PRG. For $n = 3$ and $n = 4$ we obtain a 52% and 61% speedup respectively, compared to Rivain-Prouff. We provide the source code in [Cor19b].

			Single robust PRG			Multiple PRGs		
		[RP10]	SecMultFLR	SecMultILR	SecMultILR2	SecMultFLR	SecMultILR	3-wise SecMultFLR
$n = 3$	Mcycles	20.6	65.6	76.8	65.4	12	14.1	9.8
	ratio	1	3.18	3.73	3.17	0.58	0.68	0.48
$n = 4$	Mcycles	40.2	235.1	425.1	324.9	24.6	34.7	15.5
	ratio	1	5.85	10.57	8.08	0.61	0.86	0.39
$n = 5$	Mcycles	65.8	1100	1541.5	1097.1	42.8	70	–
	ratio	1	16.72	23.43	16.67	0.65	1.06	–
$n = 6$	Mcycles	97.5	3042.1	4278.3	2898.5	67.2	124.1	–
	ratio	1	31.20	43.88	29.73	0.69	1.27	–

Table 7. Smart-card implementation results, on a 44 MHz ARM-Cortex M3 processor, with an embedded TRNG module. We provide the timings in millions of clock cycles, and the ratio with respect to the Rivain-Prouff countermeasure.

B The SecMult gadget

We recall in Algorithm 4 the SecMult gadget used in [RP10] for protecting AES against t-th order attacks. It is an extension to $\mathbb{F}_{2^k}$ of the original ISW countermeasure [ISW03] described in $\mathbb{F}_2$. The SecMult gadget was proven t-SNI in [BBD+16].

Algorithm 4 SecMult

Input: shares a_i satisfying $\bigoplus_{i=1}^{n} a_i = a$, shares b_i satisfying $\bigoplus_{i=1}^{n} b_i = b$
Output: shares c_i satisfying $\bigoplus_{i=1}^{n} c_i = a \cdot b$

for $i = 1$ **to** n **do**
  $c_i \leftarrow a_i \cdot b_i$
end for
for $i = 1$ **to** n **do**
  for $j = i + 1$ **to** n **do**
    $r \leftarrow \mathbb{F}_{2^k}$ # referred by $r_{i,j}$
    $c_i \leftarrow c_i \oplus r$ # referred by $c_{i,j}$
    $r \leftarrow (a_i \cdot b_j \oplus r) \oplus a_j \cdot b_i$ # referred by $r_{j,i}$
    $c_j \leftarrow c_j \oplus r$ # referred by $c_{j,i}$
  end for
end for
return $(c_1, \ldots, c_n)$