Formal Verification of Side-channel Countermeasures via Elementary Circuit Transformations

Jean-Sébastien Coron

University of Luxembourg
jean-sebastien.coron@uni.lu

September 12, 2017

Abstract. We describe a technique to formally verify the security of masked implementations against side-channel attacks, based on elementary circuit transformations. We describe two complementary approaches: a generic approach for the formal verification of any circuit, but for small attack orders only, and a specialized approach for the verification of specific circuits, but at any order. We describe the implementation of CheckMasks, a formal verification tool for side-channel countermeasures, using the Common Lisp programming language. Using this tool, we show how to formally verify the security of the Rivain-Prouff countermeasure for AES.

1 Introduction

The masking countermeasure. Masking is the most widely used countermeasure against side-channel attacks for block-ciphers and symmetric-key algorithms. In a first-order countermeasure, all intermediate variables x are masked into $x' = x \oplus r$ where r is a randomly generated value. For such countermeasure, it is usually straightforward to verify its security against first-order attacks; namely it suffices to check that all intermediate variables have the uniform distribution, or at least that their distribution is independent from the key; therefore an attacker processing the side-channel leakage of intermediate variables separately (as in a first-order attack) does not get useful information.

However second-order attacks combining the leakage on x' and r can be mounted in practice (see for example [OMHT06]), so it makes sense to design masking algorithms resisting higher-order attacks. This is done by extending Boolean masking to n shares with $x = x_1 \oplus \cdots \oplus x_n$; in that case an implementation should be resistant against t-th order attacks, in which the adversary combines leakage information from at most $t < n$ intermediate variables.

Security proofs. In principle any countermeasure against high-order attacks should have a security proof, but such proof can be either missing, incomplete, or incorrect. In this paper we describe the construction of a tool, called CheckMasks, to formally verify the security of high-order masking schemes.

The first step is to specify what it means for a masking countermeasure to have a security proof, *i.e.* what is the security model. Such formalization was initiated by Ishai, Sahai and Wagner in [ISW03]. In this model, the adversary can probe at most t wires in the circuit, but he should not learn anything about the secret key. The approach for proving security is based on simulation: one must show that any set of t wires probed by the adversary can be perfectly simulated without the knowledge of the secret-key. This shows that these t probes do not bring any useful information to the attacker, since he could run this simulation by himself.

More precisely, the technique consists in showing that any set of t probes can be perfectly simulated by the knowledge of only a proper subset of the input shares x_i. At the beginning of the algorithm one uses a pre-sharing of x into n shares x_i that cannot be probed by the adversary, when x is part of the secret-key; therefore any subset of at most $n-1$ of the input shares x_i are uniformly and independently distributed. This implies that the simulation of the probed variables can be performed without knowing the secret-key.

The main result in [ISW03] is to show that any circuit C can be transformed into a new circuit C' of size $\mathcal{O}(t^2 \cdot |C|)$ that is resistant against an adversary probing at most t wires in the circuit. The construction is based on secret-sharing every variable x into n shares with $x = x_1 \oplus \cdots \oplus x_n$, and processing the shares in a way that prevents a t-limited adversary from leaning any information about the initial variable x, using $n \geq 2t+1$ shares.

Formal verification of masking. The formal verification of the masking countermeasure was initiated by Barthe *et al.* in [BBD$^+$15]. The authors describe an automated method to prove the security of masked implementation against t-th order attacks, for small values of t (in practice, $t < 5$). The method only works for small values of t because the number of possible t-uples of intermediate variables to consider grows exponentially with t. To formally prove the security of a masking algorithm, the authors describe an algorithm to construct a bijection between the observations of the adversary (corresponding to a t-uple of intermediate variables) and a distribution that is syntactically independent from the secret inputs; this implies that the adversary learns nothing from this particular t-uple of intermediate variables. All possible t-uples of intermediates variables are then examined by exhaustive search. The authors also describe a divide-and-conquer approach to reduce the complexity of the formal verification (but this remains exponential in the attack order t).

The approach initiated by the authors enables to obtain a formal verification of various masked implementations, up to second order masked implementation of AES, and up to 5-th order for the masked Rivain-Prouff multiplication [RP10]. In particular, the authors were able to rediscover some known attacks and discover new ways of attacking already broken schemes. Their approach is implemented in the framework of EasyCrypt [BDG$^+$14], and relies on its internal representations of programs and expressions.

The main drawback of the previous approach is that it can only work for small orders t, since the running time is exponential in t. To overcome this problem, in a follow-up work [BBD$^+$16], Barthe *et al.* studied the composition property of masked algorithms. In particular, the authors introduce the notion of *strong simulatability*, a stronger property which requires that the number of input shares necessary to simulate the observations of the adversary in a given gadget is independent from the number of observations made on output wires. This ensures some separation between the input and the output wires: no matter how many output wires must be simulated (to ensure the composition of gadgets), the number of input wires that must be known to perform the simulation only depends on the number of internal probes within the gadget.

The paper [BBD$^+$16] has a number of important contributions that we summarize below. Firstly, the authors introduce the t-NI and t-SNI definitions. The t-NI security notion corresponds to the original security definition in the ISW probing model [ISW03]; it requires that any t probes of the gadget circuit can be simulated from at most t of its input shares. The stronger t-SNI notion corresponds to the strong simulatability property mentioned above, in which the number of input shares required for the simulation is upper bounded by the number of probes t in the circuit, and is independent from the number of output variables $|\mathcal{O}|$ that must be simulated (as long as the condition $t + |\mathcal{O}| < n$ is satisfied). We recall these definitions in Section 2, as they are fundamental in our paper.

The authors show that the t-SNI definition allows for securely composing masked algorithms; *i.e.* for a construction involving many gadgets, one can prove that the full construction is t-SNI secure, based on the t-SNI security of its components. The advantages are twofold: firstly the proof becomes modular and much easier to describe. Secondly as opposed to

[ISW03] the masking order does not need to be doubled throughout the circuit, as one can work with $n \geq t+1$ shares, instead of $n \geq 2t+1$ shares. Since most gadgets have complexity $\mathcal{O}(n^2)$, this usually gives a factor 4 improvement in efficiency. In [BBD$^+$16], the authors prove the t-SNI property of several useful gadgets: the multiplication of Rivain-Prouff [RP10], the mask refreshing based on the same multiplication algorithm, and the multiplication between linearly dependent inputs from [CPRR13].

Moreover, in [BBD$^+$16] the authors also machine-checked the multiplication of Rivain-Prouff and the multiplication-based mask refreshing in the EasyCrypt framework [BDG$^+$14]. The main point is that their machine verification works for any order, whereas in [BBD$^+$15] the formal verification could only be performed at small orders t, since the number of t-uples to consider grows exponentially with t. However, the approach seems difficult to understand (at least for a non-expert in formal methods), and when reading [BBD$^+$16] it is far from obvious how the automated verification of the countermeasure can be implemented concretely; this seems to require a deep knowledge of the EasyCrypt framework.

Finally, the authors built an automated approach for verifying that an algorithm constructed by composing provably secure gadgets is itself secure. They also implemented an algorithm for transforming an input program P into a program P' secure at order t; their algorithm automatically inserts mask refreshing gadgets whenever required.

Our contributions. Our main goal in this paper is to achieve essentially the same formal verification results as in [BBD$^+$15] and [BBD$^+$16], but in a much simpler way. We describe two complementary approaches: a generic approach for the formal verification of any circuit, but for small attack orders only (as in [BBD$^+$15]), and a specialized approach for the verification of specific circuits, but at any order (as in [BBD$^+$16]). We describe the implementation of CheckMasks, a formal verification tool for side-channel countermeasures, using the Common Lisp programming language. Using this tool, we show how to formally verify the security of the Rivain-Prouff countermeasure for AES.

For the generic verification at small orders, our approach is essentially the same as in [BBD$^+$15], except that we use the Common Lisp programming language. In principle, this enables to get a much shorter implementation, because Common Lisp is very well suited to formal manipulations. In particular, we are able to formally verify the security of the Rivain-Prouff multiplication [RP10] with very few lines of code. Our running times for formal verification are similar to those in [BBD$^+$15].

For the verification of specific gadgets at any order, our technique is quite different from [BBD$^+$16] and consists in applying elementary transformations to the circuit, until the t-NI or t-SNI properties become straightforward to verify. We show that for a set of well-chosen elementary transformations, the formal verification time becomes polynomial in t (instead of exponential with the generic approach). This implies that there is no gap between the running time of the formal verification and the practical running time of the countermeasure. In particular, we provide a formally verified proof of t-SNI property of the multiplication algorithm in the Rivain-Prouff countermeasure, and of the mask refreshing based on the same multiplication algorithm; in both cases the running time of the formal verification is polynomial in the number of shares n.

Source Code. The source code of our CheckMasks verification tool is publicly available at [Cor17b], under the GPL v2.0 license.

2 Security Properties

In this section we recall the t-NI and t-SNI security definitions from [BBD$^+$16]. For simplicity we only provide the definitions for a simple gadget taking as input a single variable x (given by n shares x_i) and outputting a single variable y (given by n shares y_i). Given a vector of n shares $(x_i)_{1\le i\le n}$, we denote by $x_{|I} := (x_i)_{i\in I}$ the sub-vector of shares x_i with $i \in I$.

Definition 1 (t-NI security). *Let G be a gadget taking as input $(x_i)_{1\le i\le n}$ and outputting the vector $(y_i)_{1\le i\le n}$. The gadget G is said t-NI secure if for any set of t intermediate variables, there exists a subset I of input indices with $|I| \le t$, such that the t intermediate variables can be perfectly simulated from $x_{|I}$.*

Definition 2 (t-SNI security). *Let G be a gadget taking as input $(x_i)_{1\le i\le n}$ and outputting $(y_i)_{1\le i\le n}$. The gadget G is said t-SNI secure if for any set of t intermediate variables and any subset $\mathcal{O}$ of output indices such that $t + |\mathcal{O}| < n$, there exists a subset I of input indices with $|I| \le t$, such that the t intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

The t-NI security notion corresponds to the original security definition in the ISW probing model; based on the ISW multiplication gadget, it allows to prove the security of a transformed circuit with $n \ge 2t + 1$ shares. The stronger t-SNI notion allows to prove the security with $n \ge t + 1$ shares only [BBD$^+$16]. The difference between the two notions is as follows: in the stronger t-SNI notion, the size of the input shares subset I can only depend on the number of internal probes t and is independent of the number of output variables $|\mathcal{O}|$ that must be simulated (as long as the condition $t + |\mathcal{O}| < n$ is satisfied). The t-SNI security notion is very convenient for proving the security of complex constructions, as one can prove that the t-SNI security of a full construction based on the t-SNI security of its components.

In this paper, for simplicity we always work in a finite field of characteristic 2, and we use the $\oplus$ and $+$ operators indistinctly; our techniques could be easily adapted to any finite field.

3 Formal Verification of Generic Circuits for Small Order

In this section, we show that the t-NI and t-SNI properties can be easily verified formally for any circuit, using a generic approach. As in [BBD$^+$15] the complexity of the formal verification is exponential in the number of shares n, so this can only work for small n. In Section 4 we will show how to formally verify the above properties in time polynomial in n, but for specific circuits.

3.1 The RefreshMasks Algorithm

To illustrate our approach we consider the RefreshMasks algorithm below from [RP10]; see Figure 1 for an illustration. The RefreshMasks algorithm was also used is the randomized table countermeasure from [Cor14].

We first recall a straightforward property of the RefreshMasks algorithm: when the intermediate variables of the algorithm are not probed, any subset of $n-1$ output shares y_i of RefreshMasks is uniformly and independently distributed. We first provide a pen-and-paper proof; we then explain in the next section how this property can be formally verified in our tool.

Lemma 1. *Let $(y_i)_{1\le i\le n}$ be the output of* RefreshMasks*. Any subset of $n-1$ output shares y_i is uniformly and independently distributed.*

Algorithm 1 RefreshMasks

Input: $x_1, \ldots, x_n$, where $x_i \in \{0,1\}^k$
Output: $y_1, \ldots, y_n$ such that $y_1 \oplus \cdots \oplus y_n = x_1 \oplus \cdots \oplus x_n$

$y_n \leftarrow x_n$
for $i = 1$ to $n-1$ **do**
  $r_i \leftarrow \{0,1\}^k$
  $y_i \leftarrow x_i \oplus r_i$
  $y_n \leftarrow y_n \oplus r_i$ ▷ $y_{n,i} = x_n \oplus \bigoplus_{j=1}^{i} r_j$
end for
return $y_1, \ldots, y_n$

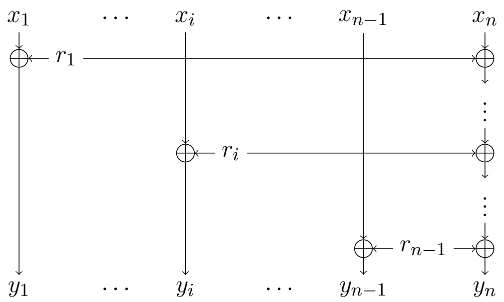

Fig. 1. The RefreshMasks algorithm, with the randoms r_i accumulated on the last column.

Proof. Let $S \subsetneq [1, n]$ be the corresponding subset. We distinguish two cases. If $n \notin S$, we have $y_i = x_i \oplus r_i$ for all $i \in S$, and therefore those y_i's are uniformly and independently distributed. If $n \in S$, let $i^\star \notin S$. We have $y_i = x_i \oplus r_i$ for all $i \in S \setminus \{n\}$. Moreover by definition:

$$y_n = \left(x_n \oplus \bigoplus_{i=1, i \neq i^\star}^{n-1} r_i \right) \oplus r_{i^\star}$$

where $r_{i^\star}$ is not used in another y_i for $i \in S$. Therefore the $n-1$ output y_i's are uniformly and independently distributed. ☐

3.2 Formal Verification of Circuits

We represent a circuit with nested lists, using the prefix notation. Consider the circuit taking as input x and y and outputting $x + y$; we represent it as (+ X Y). Similarly the circuit computing $x \cdot y$ is represented as ($*$ X Y). To represent more complex circuits the lists are recursively nested. For example, to represent the circuit $x + y \cdot z$, we write (+ X ($*$ Y Z)). If a circuit has many outputs, we represent the list of outputs without any prefix operator; for example, the circuit outputting $(x + y, x \cdot y)$ can be represented as ((+ X Y) ($*$ X Y)).

It is easy to write a program in Common Lisp that generates the circuit corresponding to RefreshMasks; we refer to [Cor17b] for the source code. For example, we obtain for $n = 3$ input shares:

```
> (RefreshMasks '(x1 x2 x3))
((+ R1 X1) (+ R2 X2) (+ R2 (+ R1 X3)))
```

Note that the above RefreshMasks function in Common Lisp takes as input a list of n shares (here $n = 3$) and outputs a list of n shares; therefore it can be easily composed with other such Common Lisp functions to create a more complex circuit.

We now show how Lemma 1 can be formally verified. Consider for example the two output variables (+ R1 X1)) and (+ R2 (+ R1 X3)) from above. We would like to show that these two variables are uniformly and independently distributed. Since the random R2 is used only once in those two outputs, it can play the role of a one-time pad, and we can perform the following substitution in the second output:

$$(+\ \mathrm{R2}\ (+\ \mathrm{R1}\ \mathrm{X3})) \longrightarrow \mathrm{R2}$$

Namely, since R2 is used only once, the distribution of (+ R2 (+ R1 X3)) is the same as the distribution of R2; therefore the knowledge of X3 is not needed to perform the simulation. Starting with the above list of two output variables, we can perform the following sequence of elementary substitutions:

$$((+\ \mathrm{R1}\ \mathrm{X1})\ (+\ \mathrm{R2}\ (+\ \mathrm{R1}\ \mathrm{X3}))) \longrightarrow ((+\ \mathrm{R1}\ \mathrm{X1})\ \mathrm{R2}) \longrightarrow (\mathrm{R1}\ \mathrm{R2})$$

which shows that neither X1 nor X3 is required for the simulation of the two output variables; moreover since we have obtained two distinct randoms (R1 R2) at the end, the two probes are uniformly and independently distributed.

These transformations on lists are easy to implement in Common Lisp. Namely it suffices to perform a tree search to count the number of times a given random R is used. If a random R is used only once, we can then perform the substitution:

$$(+\ \mathrm{R}\ \mathrm{X}) \longrightarrow \mathrm{R} \tag{1}$$

Such substitution can be recursively applied until the rule cannot be applied anymore.

To formally verify Lemma 1, it suffices to consider all possible subsets of $n-1$ output shares y_i among n, and check that for every subset, we obtain after a series of such elementary substitutions a list of $n-1$ distinct randoms. Since there are n such subsets, in this particular case the formal verification can be done in polynomial time. We obtain for example for $n = 3$:

```
> (Check-RefreshMasks-Uni 3)
Input: (X0 X1 X2)
Output: ((+ R1 X0) (+ R2 X1) (+ R2 (+ R1 X2)))
Case 0: ((+ R2 X1) (+ R2 (+ R1 X2))) => ((+ R2 X1) (+ R2 R1))
        => ((+ R2 X1) R1) => (R2 R1)
Case 1: ((+ R1 X0) (+ R2 (+ R1 X2))) => ((+ R1 X0) R2) => (R1 R2)
Case 2: ((+ R1 X0) (+ R2 X1)) => ((+ R1 X0) R2) => (R1 R2)
```

The above transcript shows that Lemma 1 is formally verified for $n = 3$, as in all possible cases we obtain a list of distinct randoms after a sequence of elementary substitutions; see [Cor17b] for the source code.

3.3 Security properties of RefreshMasks

We first recall another straightforward property of RefreshMasks, namely that it achieves the t-NI property.

Lemma 2 (t-NI of RefreshMasks). *Let $(x_i)_{1\le i\le n}$ be the input of* RefreshMasks *and let $(y_i)_{1\le i\le n}$ be the output. For any set of t intermediate variables, there exists a subset I of input indices such that the t intermediate variables can be perfectly simulated from $x_{|I}$, with $|I| \le t$.*

Proof. The set I is constructed as follows. If for some $1 \le i \le n-1$, any of the variables x_i, r_i or y_i is probed, we add i to I. If x_n or y_n or any intermediate variable $y_{n,j}$ is probed, we add n to I. Since we add at most one index to I per probe, we must have $|I| \le t$.

The simulation of the probed variable is straightforward. All the randoms r_i for $1 \leq i \leq n-1$ can be simulated as in the real algorithm, by generating a random element from $\{0,1\}^k$. If y_i is probed, then we must have $i \in I$, so it can be perfectly simulated from $y_i = x_i \oplus r_i$, from the knowledge of x_i. Similarly, if any intermediate variable $y_{n,j}$ is probed, then we must have $n \in I$, so it can be perfectly simulated from x_n. Therefore all probes can be perfectly simulated from $x_{|I}$. $\square$

While the RefreshMasks algorithm achieves the t-NI property, it is easy to see that it does not achieve the stronger t-SNI property, as already observed in [BBD$^+$16]. Namely one can probe the output $y_1 = r_1 \oplus x_1$ and the internal variable $y_{n,1} = r_1 \oplus x_n$; see Fig. 1. This gives $y_1 \oplus y_{n,1} = x_1 \oplus x_n$ and therefore the knowledge of both inputs x_1 and x_n is required for the simulation, while only $t = 1$ internal variables are probed (and therefore, at most one input x_i can be used for the simulation to achieve the t-SNI property). More generally, using our CheckMasks formal tool, we can obtain the list of all $(n-1)$-uples of probes that contradict the t-SNI property of RefreshMasks with n input shares. For example, we obtain for $n = 4$:

```
> (Check-Refreshmasks-Non-SNI 4 :all 't)
Input: (X1 X2 X3 X4)
Output: ((+ R1 X1) (+ R2 X2) (+ R3 X3) (+ R3 (+ R2 (+ R1 X4))))
((+ R1 X1) X2 (+ R1 X4))
((+ R1 X1) (+ R2 X2) (+ R1 X4))
((+ R1 X1) (+ R2 X2) (+ R2 (+ R1 X4)))
((+ R1 X1) X3 (+ R1 X4))
((+ R1 X1) (+ R3 X3) (+ R1 X4))
((+ R1 X1) (+ R1 X4) (+ R3 (+ R2 (+ R1 X4))))
```

Consider for example the first 3-uple of probes ((+ R1 X1) X2 (+ R1 X4)). We see that the substitution rule (1) does not apply as the random R1 occurs twice. Therefore the simulation of this 3-uple requires the knowledge of the 3 inputs x_1, x_2 and x_4; this contradicts the t-SNI property, as only $t = 2$ intermediate variables x_2 and $y_{4,1} = x_4 \oplus r_1$ are probed, since $y_1 = x_1 \oplus r_1$ is an output variable.

Consider also the second 3-uple ((+ R1 X1) (+ R2 X2) (+ R1 X4)). Since R2 occurs only once, we can perform the substitution:

$$((+\text{ R1 X1})\ (+\text{ R2 X2})\ (+\text{ R1 X4})) \Rightarrow ((+\text{ R1 X1})\ \text{R2}\ (+\text{ R1 X4}))$$

However, since R1 appears twice we cannot perform any further substitution, and the knowledge of both x_1 and x_4 is required to simulate the 3-uple; this again contradicts the t-SNI property, as only $t = 1$ intermediate variable $y_{4,1} = x_4 \oplus r_1$ has been probed and the other 2 variables are output variables.

3.4 The FullRefresh Algorithm

We recall below an improved mask refreshing algorithm that satisfies the t-SNI property, as opposed to the previous RefreshMasks, as shown in [BBD$^+$16]. The algorithm is based on the masked multiplication from [ISW03] and was already used in [ISW03] and [DDF14]. Note that the algorithm has complexity $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n)$. For completeness we recall the pen-and-paper security proof in Appendix A.

Lemma 3 (t-SNI of FullRefresh [BBD$^+$16]). *Let $(x_i)_{1 \leq i \leq n}$ be the input shares of the FullRefresh operation, and let $(y_i)_{1 \leq i \leq n}$ be the output shares. For any set of t intermediate variables and any subset $\mathcal{O}$ of output shares such that $t + |\mathcal{O}| < n$, there exists a subset I of indices with $|I| \leq t$, such that the t intermediate variables as well as the output shares $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

Algorithm 2 FullRefresh

Input: $x_1, \ldots, x_n$
Output: $y_1, \ldots, y_n$ such that $\bigoplus_{i=1}^{n} y_i = \bigoplus_{i=1}^{n} x_i$
for $i = 1$ **to** n **do** $y_i \leftarrow x_i$
for $i = 1$ to n **do**
  for $j = i + 1$ to n **do**
    $r \leftarrow \{0, 1\}^k$ ▷ Referred by $r_{i,j}$
    $y_i \leftarrow y_i \oplus r$ ▷ Referred by $y_{i,j}$
    $y_j \leftarrow y_j \oplus r$ ▷ Referred by $y_{j,i}$
  end for
end for
return $y_1, \ldots, y_n$

Formal Verification of FullRefresh. In the following, we describe the formal verification of Lemma 3 using our CheckMasks tool. As previously we first implement the FullRefresh algorithm in Common Lisp; for $n = 3$ shares, we get the following output:

```
> (FullRefresh '(x1 x2 x3))
((+ R2 (+ R1 X1)) (+ R3 (+ R1 X2)) (+ R3 (+ R2 X3)))
```

Using our CheckMasks tool, Lemma 3 can be easily verified for small values of n. Namely it suffices to compute the list of all $(n-1)$-uples of intermediate variables (including the outputs y_i) and check that every such $(n-1)$-uple can be perfectly simulated from the knowledge of at most t inputs x_i, where t is the number of non-output variables in the $(n-1)$-uple. For example, considering the two variables (+ R2 (+ R1 X1)) and (+ R1 X2) in the circuit above for $n = 3$, since (+ R2 (+ R1 X1)) is an output variable, the simulation must be performed using at most a single input x_i. We obtain using elementary substitutions:

((+ R2 (+ R1 X1)) (+ R1 X2)) => (R2 (+ R1 X2)) => (R2 R1)

and therefore no input x_i is actually needed to simulate those two variables. Note that the running time to consider all possible $(n-1)$-uples of intermediate variables is exponential in n. We summarize in Table 1 the running time of the formal verification of FullRefresh, up to $n = 6$. Although we are only able to verify Lemma 3 for small values of n, this still provides some confidence in the correctness of Lemma 3 for any n. In Section 4.3 we will show how to formally verify Lemma 3 in time polynomial in n, so that the formal verification can be performed for any number of shares n used in practice.

n	#variables	#tuples	Security	Time
3	12	66	✓	ε
4	22	1,540	✓	0.02 s
5	35	52,360	✓	0.6 s
6	51	2,349,060	✓	46 s

Table 1. Formal verification of the t-SNI property of FullRefresh, for small values of n.

3.5 Another property of RefreshMasks

We now return to the RefreshMasks algorithm (Alg. 1) and consider a non-trivial property of RefreshMasks that was used in the Boolean to arithmetic conversion algorithm from [Cor17a].

The property is the following: if the output y_n is among the t probed variables, then we can simulate those t probed variables with $t-1$ input shares only, instead of t as in Lemma 2. This property was crucial for obtaining a provably secure Boolean to arithmetic conversion algorithm in [Cor17a]. For completeness we recall the pen-and-paper proof from [Cor17a] in Appendix B.

Lemma 4 (RefreshMasks **[Cor17a]**)**.** *Let $x_1, \ldots, x_n$ be the input of a* RefreshMasks *where the randoms are accumulated on x_n, and let $y_1, \ldots, y_n$ be the output. Let t be the number of probed variables, with $t < n$. If y_n is among the probed variables, then there exists a subset I such that all probed variables can be perfectly simulated from $x_{|I}$, with $|I| \leq t-1$.*

Remark 1. The lemma does not hold for other output variables. For example the adversary can probe both $y_1 = x_1 \oplus r_1$ and $y_{n,1} = x_n \oplus r_1$. Since $y_1 \oplus y_{n,1} = x_1 \oplus x_n$, both x_1 and x_n are required for the simulation, which contradicts the bound $|I| \leq t-1$.

Using our CheckMasks formal tool, Lemma 4 can be easily verified for small values of n. Namely we can check that all t-uples of probes containing y_n require at most $t-1$ inputs x_i to be simulated. We first claim that it is sufficient to check this property for $t = n-1$ only, instead of all $1 \leq t \leq n-1$. Namely, assume that the property is not satisfied for some $t < n-1$; then there exists a set of t probes which can only be simulated by a subset I of inputs with $|I| \geq t$. If $|I| = n$, then this also holds for some superset of $n-1$ probes. If $|I| \leq n-1$, then we can complement the set of t probes with $n-1-t$ additional probes, among which $n-1-|I|$ are directly on some input shares x_i for $i \notin I$. We obtain a set of $t' = n-1$ probes which can only be simulated by a subset I' of the inputs, with $|I'| = n-1$. In both cases this would contradict Lemma 4 for $t = n-1$.

```
> (Check-RefreshMasks-L2 4)
Input: (X0 X1 X2 X3)
Output: ((+ R1 X0) (+ R2 X1) (+ R3 X2) (+ R3 (+ R2 (+ R1 X3))))
(X0 X1 X2)
(X0 X1 X3)
(X0 X2 X3)
((+ R1 X0) X1 (+ R1 X3))
((+ R1 X0) (+ R2 X1) (+ R2 (+ R1 X3)))
((+ R1 X0) X2 (+ R1 X3))
(X1 X2 X3)
```

Fig. 2. Formal verification of Lemma 4 for $n = 4$. We compute the list of 3-uples of probes whose simulation require the knowledge of at least 3 inputs; none of these 3-uples contains the last output (+ R3 (+ R2 (+ R1 X3))) of the circuit.

To formally verify Lemma 4, our CheckMasks tool computes in Figure 2 the list of t-uples that require at least t inputs x_i to be simulated, for $n = 4$ shares, with $t = 3$. We see that as required none of these t-uples include the output y_n; therefore Lemma 4 is formally verified for $n = 4$. Note that since the number of intermediate variables in RefreshMasks is $4n-3$ and we must consider all possible subsets of $n-1$ variables, the formal verification of Lemma 4 takes $\binom{4n-3}{n-1} \simeq 2^{3.2\,n}$ time and is therefore exponential in n. We summarize the observed running times in Table 2, up to $n = 8$. In Section 4.4 we will show how to formally verify the correctness of Lemma 4 for any value of n.

In Appendix C, we also describe a formal verification of some other properties of RefreshMasks, namely lemmas 5, 7 and 8 from [Cor17a], again for small values of n.

n	#variables	#tuples	Security	Time
3	9	36	✓	ε
4	13	286	✓	ε
5	17	2,380	✓	ε
6	21	20,349	✓	0.1 s
7	25	177,100	✓	0.9 s
8	29	1,560,780	✓	10 s

Table 2. Formal verification of Lemma 4, for small values of n.

3.6 The Rivain-Prouff Countermeasure

The Rivain-Prouff countermeasure for the AES block-cipher is based on the SecMult algorithm below; it is an extension over $\mathbb{F}_{2^k}$ of the masked AND gate from [ISW03]. It enables to securely compute a n-sharing of the product $c = a \cdot b$ over $\mathbb{F}_{2^k}$, from an n-sharing of a and b.

Algorithm 3 SecMult

Require: shares a_i satisfying $\bigoplus_{i=1}^n a_i = a$, shares b_i satisfying $\bigoplus_{i=1}^n b_i = b$
Ensure: 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 $c_{i,j}$
        $r \leftarrow (a_i \cdot b_j + r) + a_j \cdot b_i$ ▷ referred by $r_{j,i}$
        $c_j \leftarrow c_j \oplus r$ ▷ referred by $c_{j,i}$
    end for
end for
return $(c_1, \ldots, c_n)$

It was shown in [BBD$^+$16] that the SecMult algorithm is t-SNI secure for any $t < n$; see also [CGPZ16] for a slightly more detailed security proof.

Lemma 5 (t-SNI of SecMult [BBD$^+$16]). *Let $(a_i)_{1 \le i \le n}$ and $(b_i)_{1 \le i \le n}$ be the input shares of the SecMult operation, and let $(c_i)_{1 \le i < n}$ be the output shares. For any set of t intermediate variables and any subset $\mathcal{O}$ of output shares such that $t + \mathcal{O} < n$, there exist two subsets I and J of indices with $|I| \le t$ and $|J| \le t$, such that those t intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $a_{|I}$ and $b_{|J}$.*

Formal verification of SecMult. As previously, the first step is to implement the SecMult algorithm in Common Lisp; this requires only 12 lines of Common Lisp (see [Cor17b] for the source code). For $n = 3$, we obtain:

```
> (SecMult '(a1 a2 a3) '(b1 b2 b3))
((+ R2 (+ R1 (* A1 B1)))
 (+ R3 (+ (* A2 B2) (+ (+ (* A1 B2) R1) (* A2 B1))))
 (+ (* A3 B3) (+ (+ (+ (* A2 B3) R3) (* A3 B2))
                 (+ (+ (* A1 B3) R2) (* A3 B1)))))
```

As previously, to formally verify the t-SNI property of SecMult as stated in Lemma 5, it suffices to compute the list of all $(n-1)$-uples of intermediate variables (including the output c_i's) and check that every such $(n-1)$-uple can be perfectly simulated from the knowledge of at most t inputs a_i and at most t inputs b_j, where t is the number of non-output variables in the $(n-1)$-uple. For example, if we probe the non-output variables (+ R1 (∗ A1 B1)) and (+ (+ (∗ A1 B2) R1) (∗ A2 B1)) from above, we cannot perform any substitution because the random R1 is used twice, so we must know the inputs (A1 A2) and (B1 B2). On the other hand, if we consider the first two outputs, we have the substitutions:

```
((+ R2 (+ R1 (* A1 B1)))
 (+ R3 (+ (* A2 B2) (+ (+ (* A1 B2) R1) (* A2 B1)))))
=> ((+ R2 (+ R1 (* A1 B1))) R3) => (R2 R3)
```

and therefore no inputs a_i or b_i is needed, as required for the t-SNI property (since we have considered output variables only). We obtain the following timings for the formal verification of SecMult using our CheckMasks tool:

n	#variables	#tuples	Security	Time
3	30	435	✓	ε
4	54	24,804	✓	0.5 s
5	85	2,024,785	✓	80 s

Table 3. Formal verification of the t-SNI property of SecMult, for small values of n.

As previously, using this generic approach we can only verify Lemma 5 for small values of n. In Section 4.5 we describe a specialized approach to formally verify the security of SecMult for any value of n.

3.7 Discussion

Our approach is essentially the same as in [BBD+15], except that we use the Common Lisp language instead of the C language, in order to get a simpler implementation.

Another difference is that we verify the t-NI or t-SNI properties directly on the input shares x_i, while in [BBD+15] the input shares x_i come from a pre-sharing of the original secret variable x, where the pre-sharing cannot be probed by the adversary. More precisely one would use:

$$(x_1, \ldots, x_n) \leftarrow (x \oplus r_1, r_2, \ldots, r_{n-1}, r_1 \oplus \cdots \oplus r_{n-1})$$

and in [BBD+15] the authors do not consider the input shares x_i but rather the variables $(x \oplus r_1, r_2, \ldots, r_{n-1}, r_1 \oplus \cdots \oplus r_{n-1})$. Their approach consists in checking that in the masking algorithm the distribution of any set of $n-1$ probes can be made syntactically independent from the original secret variable x, while in our approach we show that the distribution of any set of $n-1$ probes only depends on at most $n-1$ of the input shares x_i; this in turn also implies that the distribution is independent from the original x, since any subset of $n-1$ input shares x_i is uniformly and independently distributed. In principle the two approaches are equivalent when verifying the t-NI property; our approach also enables to easily verify the stronger t-SNI property.

4 Formal Verification in Polynomial Time

The main drawback of the previous approach for formal verification is that it has exponential complexity in the number of shares n, because the number of t-uples to consider grows exponentially with n. In this section we describe a new approach for proving the security of a side-channel countermeasure. Instead of performing a simulation of the probed variables as in [ISW03], our new approach is based on a sequence of elementary circuit transformations, until the transformed circuit becomes so simple that the security property becomes straightforward. The main advantage is that in the context of formal verification, our new approach seems much easier to verify formally than the classical simulation-based approach from [ISW03]. In particular, we show that the security of the gadgets considered in the previous section can always be verified in polynomial time in n. Our technique is based on the following two elementary transforms:

- The Random-zero transform: we set to 0 a subset of the randoms r_i used in the circuit.
- The One-time-pad transform: if a random r appears only once in a circuit, and moreover r is not probed, we can replace any variable $x \oplus r$ by r.

4.1 The Random-zero Transform

Our first circuit transformation consists in setting to 0 a subset of the randoms r_i used in the circuit. This enables to significantly simplify the circuit; namely the variable $x \oplus r$ can be replaced by x, and we can then remove duplicate variables in the circuit. In the following we show that it is sufficient to verify the security of a masking algorithm when a subset of its randoms are set to 0, if the circuit is *additively masked*.

Definition 3 (Additive masking). *Let C be a circuit taking as input $x_1, \ldots, x_n$. We say that C is additively masked if every intermediate variable y in the circuit can be written as $y = f(x_1, \ldots, x_n) + g(r_1, \ldots, r_n)$, where g is a linear function.*

For example, the circuit computing $y = x_1 \cdot x_2 + r_1 + r_2$ is additively masked, while the circuit computing $y = x \cdot r$ is not. Most side-channel countermeasures for block-ciphers are additively masked. In particular, this is true for the RefreshMasks, FullRefresh and SecMult algorithms considered in the previous sections. In this paper, we only consider additively masked circuits. The following Lemma shows that it is sufficient to consider the security of such circuit when a subset of the randoms are fixed to 0.

Lemma 6 (Random-zero transform). *Let C be an additively masked circuit and let C_0 be the same circuit as C but with a subset of the randoms fixed to 0. Anything an adversary can compute from a set of probes in C, he can compute from the same set of probes in the circuit C_0.*

Proof. Let $\boldsymbol{y}$ be a vector of probed intermediate variables. Let $\boldsymbol{x}$ be the vector of inputs of the circuit and let $\boldsymbol{r}$ be the vector of randoms used in the circuit. Since the circuit is additively masked, we can write:

$$\boldsymbol{y} = h(\boldsymbol{x}, \boldsymbol{r}) = f(\boldsymbol{x}) + g(\boldsymbol{r})$$

for some functions h, f and g, where g is linear.

We write $\boldsymbol{r} = \boldsymbol{r'} + \boldsymbol{r''}$ where the randoms corresponding to $\boldsymbol{r'}$ are distributed as in the real circuit, while the randoms corresponding to $\boldsymbol{r''}$ are distributed as in the real circuit in C and set to 0 in C_0. Since g is a linear function, we have:

$$h(\boldsymbol{x}, \boldsymbol{r}) = h(\boldsymbol{x}, \boldsymbol{r'} + \boldsymbol{r''}) = f(\boldsymbol{x}) + g(\boldsymbol{r'} + \boldsymbol{r''}) = f(\boldsymbol{x}) + g(\boldsymbol{r'}) + g(\boldsymbol{r''})$$

which gives:

$$h(\boldsymbol{x}, \boldsymbol{r}) = h(\boldsymbol{x}, \boldsymbol{r'}) + g(\boldsymbol{r''}) \tag{2}$$

In the circuit C, the adversary obtain the probes $\boldsymbol{y} = h(\boldsymbol{x}, \boldsymbol{r})$, while in the circuit C_0 the adversary obtains the probes $\boldsymbol{y}_0 = h(\boldsymbol{x}, \boldsymbol{r'})$. From (2), we have that anything the adversary can compute from $\boldsymbol{y} = h(\boldsymbol{x}, \boldsymbol{r})$, he can compute from $\boldsymbol{y}_0 = h(\boldsymbol{x}, \boldsymbol{r'})$, simply by first computing:

$$\boldsymbol{y} \leftarrow \boldsymbol{y}_0 + g(\boldsymbol{r''})$$

using for $\boldsymbol{r''}$ the same distribution as in the real circuit. This proves Lemma 6. $\square$

Remark 2. This is not true for general circuits. Consider a circuit taking as input sk and outputting $(sk \cdot r, r)$. When considering the output only, the circuit would be secure when r is fixed to 0, but the output leaks the secret sk whenever $r \neq 0$.

Application: t-NI of RefreshMasks. The t-NI property of RefreshMasks, as stated in Lemma 2, is easily verified formally using the Random-zero transform. Namely, if we fix all randoms of RefreshMasks to 0, we obtain the identity function, which is trivially t-NI. For example, we obtain for $n = 4$:

```
> (check-refreshmasks-tni-poly 4)
Input: (X1 X2 X3 X4)
Output: ((+ R1 X1) (+ R2 X2) (+ R3 X3) (+ R3 (+ R2 (+ R1 X4))))
Random zero => (X1 X2 X3 X4)
Identity function: T
```

Note that the verification is performed in polynomial time in n, while in the generic approach, considering all possible t-uples the complexity would be exponential in n.

4.2 The One-time Pad Transform

The One-time Pad transform is defined as follows: if a random r is used only once in a circuit, and moreover r is not probed, then we can replace the variable $x \oplus r$ by r. Note that in principle the variable x can still be probed, so it must not be removed from the circuit.

Consider for example a circuit taking as input x_1, x_2 and outputting $(x_1 \oplus r) \oplus x_2$; it can be represented as (+ (+ X1 R) X2). If we assume that R is not probed, we can replace (+ (+ X1 R) X2) by (+ R X2); however if the original (+ X1 R) can still be probed, we cannot assume that R will not be probed in the new circuit (+ R X2), so we cannot apply the same transformation again on (+ R X2). Moreover, even if the variable X1 does not appear anymore in the circuit (+ R X2), in principle it can still be probed, so we must keep it separately on a list of variables that can be probed.

In general we cannot assume that a certain random r is not probed by the adversary. We can only make this assumption when we have an upper bound on the number of probes in the circuit, as it is the case for the t-NI and t-SNI properties. For example, if a circuit contains n randoms r_i but the adversary has only access to $t = n - 1$ probes, then we are guaranteed that at least one of the random r_i has not been probed, and we can apply the previous One-time Pad transform on this random (if r_i is not used twice in the circuit). The proof technique then consists in considering all possible n cases separately (corresponding to the non-probed r_i, for $1 \leq i \leq n$), and then applying the admissible One-time Pad transform in each case.

In the next sections, we illustrate this approach by providing a formal verification of the same security properties of the RefreshMasks, FullRefresh and SecMult algorithms as considered in Section 3, but this time with complexity polynomial in n, instead of exponential. This implies that the security of these algorithms can be formally verified for any value of n for which the countermeasure would be used in practice.

4.3 Formal Verification of Lemma 3 for FullRefresh

In this section we provide a formal proof of Lemma 3 for the t-SNI property of FullRefresh (see Alg. 2 in Section 3.4); as opposed to Section 3.4 the formal verification time is polynomial in n. The proof strategy is to perform a sequence of elementary circuit transformations until we obtain a simple circuit C for which the t-SNI property is straightforward to verify. The proof can then be formally verified by computing those circuit transformations in Common Lisp and checking that we indeed obtain the simple circuit C.

Lemma 7. *Let C be the circuit taking as input as input $x_1, \ldots, x_n$ and outputting $y_i = x_i \oplus r_i$ for all $1 \leq i \leq n$, where the randoms r_i are uniformly and independently distributed. The circuit C is t-SNI for any $t \leq n$.*

Proof. The proof is straightforward. If x_i or r_i or $y_i = x_i \oplus r_i$ is probed, we put i in I. We obtain $|I| \leq t$. From the knowledge of $x_{|I}$ we can simulate any probed variable x_i, r_i and $y_i = x_i \oplus r_i$ since in that case $i \in I$. Consider now any $i \in O \setminus I$; in that case $y_i = x_i \oplus r_i$ can be simulated by a random value since r_i is not probed, because $i \in O \setminus I$. □

We recall the t-SNI property of FullRefresh below from Section 3.4. In Appendix A we recall the pen-and-paper proof from [BBD$^+$16]. Below we provide an alternative proof of Lemma 3, based on elementary circuit transformations, so that it can be formally verified.

Lemma 3 (t-SNI of FullRefresh). *Let $(x_i)_{1 \leq i \leq n}$ be the input shares of the FullRefresh operation, and let $(y_i)_{1 \leq i \leq n}$ be the output shares. For any set of t intermediate variables and any subset $\mathcal{O}$ of output shares such that $t + |\mathcal{O}| < n$, there exists a subset I of indices with $|I| \leq t$, such that the t intermediate variables as well as the output shares $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

$$
\begin{array}{c} x_1 \\ \vdots \\ x_{i^\star} \\ \vdots \\ x_n \end{array}
\begin{pmatrix} 0 & \cdots & r_{1,i^\star} & \cdots & r_{1,n} \\ \vdots & \ddots & & & \vdots \\ r_{1,i^\star} & \cdots & 0 & \cdots & r_{i^\star,n} \\ \vdots & & & \ddots & \vdots \\ r_{1,n} & \cdots & r_{i^\star,n} & \cdots & 0 \end{pmatrix}
\begin{array}{c} y_1 \\ \vdots \\ y_{i^\star} \\ \vdots \\ y_n \end{array}
\longrightarrow
\begin{array}{c} x_1 \\ \vdots \\ x_{i^\star-1} \\ x_{i^\star+1} \\ \vdots \\ x_n \end{array}
\begin{pmatrix} 0 & r_{1,i^\star} & 0 \\ \vdots & \vdots & \\ 0 & r_{i^\star-1,i^\star} & 0 \\ 0 & r_{i^\star,i^\star+1} & 0 \\ \vdots & & \vdots \\ 0 & r_{i^\star,n} & 0 \end{pmatrix}
\begin{array}{c} y_1 \\ \vdots \\ y_{i^\star-1} \\ y_{i^\star+1} \\ \vdots \\ y_n \end{array}
$$

Fig. 3. Proof of Lemma 3: after removing the row $i^\star$ and setting all randoms to 0 except on the column $i^\star$, there remains only a one-time pad of the $n-1$ inputs x_i for $i \neq i^\star$, corresponding to the circuit C from Lemma 7.

Proof. We first construct a subset I of indices as follows. If x_i or any intermediate variable $y_{i,j}$ is probed (including y_i), we add the i to I. Since we have considered at most t probes, we obtain $|I| \leq t$. Moreover we have $|I \cup \mathcal{O}| \leq |I| + |\mathcal{O}| \leq t + |\mathcal{O}| < n$, therefore there exists some $1 \leq i^\star \leq n$ such that $i^\star \notin I \cup \mathcal{O}$. Since neither $x_{i^\star}$ nor any intermediate variable $y_{i^\star,j}$ has been probed on the row $i^\star$, and moreover $y_{i^\star}$ must not be simulated (since $i^\star \notin O$), we can remove the row $i^\star$ from the circuit.

We obtain a circuit with $n-1$ inputs x_i for $1 \leq i \leq n$ and $i \neq i^\star$. We now apply the Random-zero transform and set to 0 all randoms r_{ij} in the circuit, except the randoms on the column $i^\star$, namely $r_{i,i^\star}$ for $i \neq i^\star$. We obtain a circuit taking as input x_i and outputting $x_i \oplus r_{i,i^\star}$ for all $i \neq i^\star$; see Fig 3 for an illustration. Since from Lemma 7 this circuit is t-SNI for all $t \leq n-1$, the FullRefresh circuit is t-SNI for all $t < n$, which proves the lemma. □

Note that the main difference with the original proof of Lemma 3 is that we have not performed an explicit simulation of the probed variables; instead we have performed a sequence of elementary circuit transformations (conditioned on some of the intermediate variables being probed or not) until we have obtained a trivial circuit.

The above proof can be formally verified by performing a loop over all possible $1 \leq i^\star \leq n$. For each $i^\star$ we first remove the row $i^\star$ from the circuit, and then we set to 0 all randoms in the circuit, except the randoms $r_{i,i^\star}$ for $i \neq i^\star$. For any given n, we can check formally that this leads to a circuit equivalent to taking $a_1, \ldots, a_{n-1}$ as input and outputting $a_1 \oplus r_1, \ldots, a_{n-1} \oplus r_{n-1}$. Since such circuit is t-SNI from Lemma 7, the original circuit is t-SNI. We provide in Fig. 4 the formal verification for $n = 3$. Note that the formal verification has a running time polynomial in n (as opposed to exponential in Section 3.4); therefore it can be performed for any n for which the countermeasure is used in practice.

```
> (check-fullrefresh-tsni-poly 3)
Input: (X1 X2 X3)
Output: ((+ R2 (+ R1 X1)) (+ R3 (+ R1 X2)) (+ R3 (+ R2 X3)))
Case 0: no output, no probe in (+ R2 (+ R1 X1))
  Subcircuit: ((+ R3 (+ R1 X2)) (+ R3 (+ R2 X3)))
  Setting all randoms to 0 except (R1 R2) => ((+ R1 X2) (+ R2 X3))
Case 1: no output, no probe in (+ R3 (+ R1 X2))
  Subcircuit: ((+ R2 (+ R1 X1)) (+ R3 (+ R2 X3)))
  Setting all randoms to 0 except (R1 R3) => ((+ R1 X1) (+ R3 X3))
Case 2: no output, no probe in (+ R3 (+ R2 X3))
  Subcircuit: ((+ R2 (+ R1 X1)) (+ R3 (+ R1 X2)))
  Setting all randoms to 0 except (R2 R3) => ((+ R2 X1) (+ R3 X2))
```

Fig. 4. Formal verification of the FullRefresh circuit for $n = 3$.

4.4 Formal Verification of Lemma 4 for RefreshMasks

We now consider the RefreshMasks algorithm (see Fig. 1), and we recall the security property of RefreshMasks considered in Section 3.5: if the output y_n is among the t probed variables, then we can simulate any t probed variables with $t-1$ input shares only, instead of t in the basic t-NI property in Lemma 2. We recall the corresponding Lemma 4 below. The pen-and-paper proof from [Cor17a] is recalled in Appendix B. Below we provide an alternative proof that can be formally verified in time polynomial in n, using our CheckMasks tool.

Lemma 4. *Let $x_1, \ldots, x_n$ be the input of a* RefreshMasks *where the randoms are accumulated on x_n, and let $y_1, \ldots, y_n$ be the output. Let t be the number of probed variables, with $t < n$. If y_n is among the probed variables, then there exists a subset I such that all probed variables can be perfectly simulated from $x_{|I}$, with $|I| \leq t-1$.*

Proof. Without loss of generality, we can consider $t = n-1$ probes (see Section 3.5). We first construct a subset I of indices as follows. For any $1 \leq i \leq n-1$, if x_i or r_i or $y_i = x_i \oplus r_i$ is probed, then we put i in I. Since by assumption y_n has been probed, we have considered at most $n-2$ probes in the construction of I, and therefore we have $|I| \leq n-2$. Therefore there must be some $1 \leq i^\star \leq n-1$ such that there was no probe in the subcircuit $y_{i^\star} = x_{i^\star} \oplus r_{i^\star}$, that is neither $y_{i^\star}$ nor $x_{i^\star}$ nor $r_{i^\star}$ have been probed.

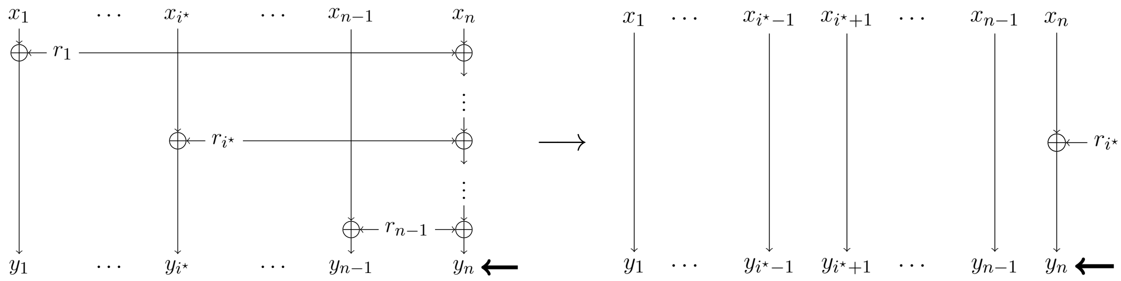

Fig. 5. Proof of Lemma 4: after removing the sub-circuit corresponding to $i^\star$ and setting to zero all randoms except $i^\star$, the remaining circuit is the identity circuit except $y_n = x_n \oplus r_{i^\star}$.

For a given $i^\star$, we can remove the subcircuit $y_{i^\star} = x_{i^\star} \oplus r_{i^\star}$ from the original circuit, since there are no probes in it. Note that $r_{i^\star}$ is still used in the computation of y_n. We then apply the Random-zero transform to all randoms except $r_{i^\star}$. As illustrated in Figure 5, we obtain a circuit taking as input the x_i's for $1 \le i \le n$ and $i \neq i^\star$, and outputting $y_i = x_i$ for $1 \le i \le n-1$ and $i \neq i^\star$, and $y_n = x_n \oplus r_{i^\star}$.

It is easy to see that the transformed circuit satisfies the required property from Lemma 4. This could be proved using the classical simulation-based approach, but we can also continue with elementary transformations, as follows. Since by assumption $r_{i^\star}$ has not been probed, we can apply the One-time-pad transform to $r_{i^\star}$, and we obtain $y_n = r_{i^\star}$ (and we also keep x_n in the circuit). Finally, we apply the Random-zero transform to $r_{i^\star}$, and we obtain $y_n = 0$. Therefore we have obtained a final circuit taking as input $(x_1, \ldots, x_n)$ and outputting $(x_1, \ldots, x_{n-1}, 0)$. Moreover we have a set of $n-1$ probes, one of which is 0 (corresponding to y_n), and the remaining $n-2$ probes are on the inputs x_i and can therefore be simulated from the knowledge of at most $n-2$ inputs. This proves Lemma 4. $\square$

It is easy to verify the above proof with a formal tool, since it consists in elementary circuit transformations conditioned on the value of $1 \le i^\star \le n-1$; we provide the transcript of the formal proof for $n = 4$ in Fig. 6; see [Cor17b] for the source code.

```
> (check-refreshmasks-last-poly 4)
Input: (X1 X2 X3 X4)
Output: ((+ R1 X1) (+ R2 X2) (+ R3 X3) (+ R3 (+ R2 (+ R1 X4))))
First probe: ((+ R3 (+ R2 (+ R1 X4))))
Case 0: no probe in (+ R1 X1)
    Subcircuit: ((+ R2 X2) (+ R3 X3) (+ R3 (+ R2 (+ R1 X4))))
    Set all randoms to 0 except R1 => (X2 X3 (+ R1 X4))
    One-time pad: (X2 X3 R1 X4). Random zero: (X2 X3 0 X4)
    First probe: 0. Other 2 probes in (X2 X3 X4)
Case 1: no probe in (+ R2 X2)
    Subcircuit: ((+ R1 X1) (+ R3 X3) (+ R3 (+ R2 (+ R1 X4))))
    Set all randoms to 0 except R2 => (X1 X3 (+ R2 X4))
    One-time pad: (X1 X3 R2 X4). Random zero: (X1 X3 0 X4)
    First probe: 0. Other 2 probes in (X1 X3 X4)
Case 2: no probe in (+ R3 X3)
    Subcircuit: ((+ R1 X1) (+ R2 X2) (+ R3 (+ R2 (+ R1 X4))))
    Set all randoms to 0 except R3 => (X1 X2 (+ R3 X4))
    One-time pad: (X1 X2 R3 X4). Random zero: (X1 X2 0 X4)
    First probe: 0. Other 2 probes in (X1 X2 X4)
```

Fig. 6. Formal verification of Lemma 4 for $n = 4$, using our CheckMasks tool for performing the sequence of elementary transformations.

Remark 3. The above formal verification of Lemma 4 has time complexity polynomial in n, so we can perform the verification for any n. For example, generating the transcript of the formal proof for $n = 50$ takes only a few seconds (since there are only $n-1$ cases to consider), while this would be completely unfeasible with the generic technique of Section 3.5, which has complexity $2^{3.2\,n}$ (see Table 2 for the corresponding timings).

4.5 Formal Verification of Lemma 5 for SecMult

In this section our goal is to provide a proof of the t-SNI property of SecMult, corresponding to Lemma 5 from Section 3.6, that can be formally verified in polynomial time. However the original pen-and-paper proof in [BBD+16] is relatively complex (and also its variant in [CGPZ16]), so as a warm-up we provide a formally verifiable proof of the weaker t-NI security property, for which a simpler pen-and-paper proof was already given in [ISW03].

Lemma 8 (t-NI of SecMult). *Let $(a_i)_{1\leq i\leq n}$ and $(b_i)_{1\leq i\leq n}$ be the input shares of the SecMult circuit, and let $(c_i)_{1\leq i<n}$ be the output shares. For any set of t intermediate variables and any subset $\mathcal{O}$ of output shares, there exists a subset I of indices such that $I = J \cup \mathcal{O}$ where $|J| \leq 2t$, such that those t intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $a_{|I}$ and $b_{|I}$.*

Proof. We prove the result recursively on n. The property holds for $n = 1$. We now assume that it holds for $n-1$, and we prove that it must hold for n. We construct a set of indices U as follows, starting from $U = \mathcal{O}$. If one the variables $\{a_i, b_i, a_i \cdot b_i, c_{i,j}\}$ is probed, we add i to U. If one of the variables $\{a_i \cdot b_j, r_{i,j}, a_i b_j + r_{i,j}\}$ is probed (for any $i \neq j$), we add both i and j to U. We obtain $|U| \leq 2t + |\mathcal{O}|$. We distinguish two cases. If $|U| = n$, we can perfectly simulate all variables in the circuit by letting $I = U = [1, n]$, and we have $|I| \leq 2t + |\mathcal{O}|$ as required.

$$\left(\begin{array}{ccccc} 0 & \cdots & r_{1,i^\star} & \cdots & r_{1,n}\\ \vdots & \ddots & & & \vdots \\ r_{i^\star,1} & \cdots & 0 & \cdots & r_{i^\star,n} \\ \vdots & & & \ddots & \vdots \\ r_{n,1} & \cdots & r_{n,i^\star} & \cdots & 0 \end{array}\right) \begin{array}{c} c_1\\ \vdots \\ c_{i^\star} \\ \vdots \\ c_n\end{array} \longrightarrow \left(\begin{array}{ccccc} 0 & \cdots & r_{1,i^\star} & \cdots & r_{1,n}\\ \vdots & & \vdots & & \vdots \\ r_{i^\star-1,1} & \cdots & r_{i^\star-1,i^\star} & \cdots & r_{i^\star-1,n} \\ r_{i^\star+1,1} & \cdots & r_{i^\star+1,i^\star} & \cdots & r_{i^\star+1,n} \\ \vdots & & \vdots & & \vdots \\ r_{n,1} & \cdots & r_{n,i^\star} & \cdots & 0 \end{array}\right) \begin{array}{c} c_1\\ \vdots \\ c_{i^\star-1} \\ c_{i^\star+1} \\ \vdots \\ c_n\end{array}$$

$$\longrightarrow \left(\begin{array}{ccccc} 0 & \cdots & 0 & \cdots & r_{1,n}\\ \vdots & & \vdots & & \vdots \\ r_{i^\star-1,1} & \cdots & 0 & \cdots & r_{i^\star-1,n} \\ r_{i^\star+1,1} & \cdots & 0 & \cdots & r_{i^\star+1,n} \\ \vdots & & \vdots & & \vdots \\ r_{n,1} & \cdots & 0 & \cdots & 0 \end{array}\right) \begin{array}{c} c_1\\ \vdots \\ c_{i^\star-1} \\ c_{i^\star+1} \\ \vdots \\ c_n\end{array}$$

Fig. 7. After removing the $i^\star$-th row and applying the one-time pad transform, we obtain a column $i^\star$ in which all variables $r_{j,i^\star}$ are independent randoms. One can then apply the random-zero transform, and eventually remove the column $i^\star$.

We now consider the case $|U| < n$, so we can let $1 \leq i^\star \leq n$ such that $i^\star \notin U$. Since none of the variables $c_{i^\star,j}$ has been probed, we can remove them from the circuit. We now consider the $r_{i^\star,j}$ variables; none of these variables has been probed. On the row $i^\star$ and before the diagonal ($j < i^\star$), the $r_{i^\star,j} = (a_j b_{i^\star} + r_{j,i^\star}) + a_{i^\star} b_j$ variables are only used in the $c_{i^\star,j}$ variables on the same row (see Fig. 7). Since we have already removed the $c_{i^\star,j}$ variables, we can also remove those $r_{i^\star,j}$ variables for $j < i^\star$ from the circuit. Moreover, since $a_j b_{i^\star} + r_{j,i^\star}$ has not been probed, we can also remove the corresponding variables from the circuit. Therefore we can remove the row $i^\star$ from the circuit.

As illustrated in Figure 7, there remains a circuit in which the original randoms $r_{i^\star,j}$ for $j > i^\star$ (after the diagonal) are used only once, namely in the variable $r_{j,i^\star} = (a_{i^\star} b_j + r_{i^\star,j}) + a_j b_{i^\star}$. Since $r_{i^\star,j}$ is not probed, and moreover $a_{i^\star} b_j + r_{i^\star,j}$ is not probed, we can apply the One-time-pad transform twice and replace the variables $r_{j,i^\star}$ below the diagonal by an independently generated random value, which we still denote by $r_{j,i^\star}$. We obtain a circuit in which on the column $i^\star$, all $r_{j,i^\star}$ for $j \neq i^\star$ are independently generated random values, which are used only once in the circuit. We can therefore apply the Random-zero transform to these randoms, *i.e.* we set to 0 all the randoms $r_{j,i^\star}$ on the $i^\star$ column; see Figure 7 for an illustration.

Since all elements on the $i^\star$ column are now zero, we can remove the $i^\star$ column and eventually obtain a circuit with $n-1$ inputs a_i and b_i that is equivalent to the original SecMult circuit, but with $n-1$ inputs instead of n, and still the same value of t. We can therefore apply the recursive hypothesis: there exists a subset I of indices such that $I = J \cup \mathcal{O}$ where $|J| \leq 2t$, such that those t intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $a_{|I}$ and $b_{|I}$. This implies that the same property holds for the original circuit with n inputs; this proves the lemma. □

To verify the above proof formally, as previously it suffices to do a loop on all possible values of $1 \leq i^\star \leq n$. We provide below the transcript of the formal verification for $n = 3$; we refer to [Cor17b] for the source code. We see in the transcript below that in each case, one obtains after a sequence of elementary transformations a circuit that is equivalent to the original circuit but with $n-1$ input shares; therefore one can apply the recursive hypothesis.

Finally, we provide in Appendix D a formally verifiable proof of the t-SNI property of SecMult (instead of t-NI only as above), corresponding to Lemma 3 from Section 3.6.

5 Conclusion

We have described a simple technique to formally verify the security of masked implementations against side-channel attacks, using two complementary approaches. The generic approach is essentially the same as in [BBD+15], but using the Common Lisp language to get a very short source code; it enables to verify the security of any circuit, but for small orders only. For the specialized approach, we have introduced an alternative proof technique based on elementary circuit transformations; we can then formally verify the security of specific circuits for any order n, in time polynomial in n. Our new approach is potentially simpler than the technique in [BBD+16] based on the EasyCrypt framework; the difference is that we do not perform an explicit simulation of the probed variables; instead we perform a sequence of elementary circuit transformations, until the simulation of the probes in the final circuit becomes trivial.

We have described the implementation of CheckMasks, our formal verification tool for side-channel countermeasures, based on the Common Lisp programming language. Using this tool, we have shown how to formally verify the security of the Rivain-Prouff countermeasure for AES.

References

[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 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 457–485, 2015.

```
> (check-secmult-ni-poly 3)
Input: (X1 X2 X3) (Y1 Y2 Y3)
Output: (M 1 1)       R1           R2
        (M 1 2 R1)    (M 2 2)      R3
        (M 1 3 R2)    (M 2 3 R3)   (M 3 3)
Case 0: no probe in (M 1 1) R1 R2
  New circuit: (M 1 2 R1)   (M 2 2)      R3
               (M 1 3 R2)   (M 2 3 R3)   (M 3 3)
  Simplify:    R1           (M 2 2)      R3
               R2           (M 2 3 R3)   (M 3 3)
  Random zero: (M 2 2)      R3
               (M 2 3 R3)   (M 3 3)
Case 1: no probe in (M 1 2 R1) (M 2 2) R3
  New circuit: (M 1 1)      R1           R2
               (M 1 3 R2)   (M 2 3 R3)   (M 3 3)
  Simplify:    (M 1 1)      R1           R2
               (M 1 3 R2)   R3           (M 3 3)
  Random zero: (M 1 1)      R2
               (M 1 3 R2)   (M 3 3)
Case 2: no probe in (M 1 3 R2) (M 2 3 R3) (M 3 3)
  New circuit: (M 1 1)      R1           R2
               (M 1 2 R1)   (M 2 2)      R3
  Simplify:    (M 1 1)      R1           R2
               (M 1 2 R1)   (M 2 2)      R3
  Random zero: (M 1 1)      R1
               (M 1 2 R1)   (M 2 2)
```

Fig. 8. Formal verification of the t-NI property of the SecMult circuit for $n = 3$. For simplicity we use a different notation to represent the $a_i b_j$ and r_{ij} variables, namely we write (M i j) for $a_i b_j$ and (M i j R) for $r_{ij} = (a_j b_i + r_{ji}) + a_i b_j$ where R corresponds to r_{ji}, for $j < i$.

[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.

[BDG+14] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. *EasyCrypt: A Tutorial*, pages 146–166. Springer International Publishing, Cham, 2014.

[CGPZ16] Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. Faster evaluation of sboxes via common shares. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 498–514, 2016.

[Cor14] Jean-Sébastien Coron. Higher order masking of look-up tables. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 441–458, 2014.

[Cor17a] Jean-Sébastien Coron. High-order conversion from boolean to arithmetic masking. In *CHES*, 2017.

[Cor17b] Jean-Sébastien Coron. CheckMasks: formal verification of side-channel countermeasures, 2017. Publicly available at https://github.com/coron/checkmasks.

[CPRR13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 410–424, 2013.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 423–440, 2014.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 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 *CHES*, pages 413–427, 2010.

A Proof of Lemma 3

We first ignore the probes of the randoms r_{ij}. If x_i or any intermediate variable $y_{i,j}$ is probed (including y_i), we add i in I. We obtain $|I| \leq t$. Since have $|I| + |O| \leq t + |\mathcal{O}| < n$, there exists $j^\star \notin I \cup \mathcal{O}$. We now process the probes on the randoms r_{ij}'s as follow. If $r_{i,j^\star}$ has been probed for $i < j^\star$, or if $r_{j^\star,i}$ has been probed for $j^\star < i$, we add i to I. This terminates the construction of the set I. We still have $|I| \leq t$.

We first show that all probed variables can simulated. We simulate all randoms r_{ij} as in the real circuit, except the randoms $r_{i,j^\star}$ for $i < j^\star$ and $i \notin I$, and $r_{j^\star,i}$ for $j^\star < i$ and $i \notin I$ which are not probed. Therefore any probed $r_{i,j}$ is perfectly simulated. Moreover, for any $i \in I$, all randoms $r_{j,i}$ for $j < i$ and all randoms $r_{i,j}$ for $j > i$ are perfectly simulated. This implies that all probed x_i or intermediate variable $y_{i,j}$ can be perfectly simulated, since in that case $i \in I$. In particular, we can perfectly simulate all y_i for $i \in I$.

It remains to show how to simulate y_i for $i \in \mathcal{O} \setminus I$. For simplicity we let $r_{ji} := r_{ij}$ for all $1 \leq i < j \leq n$. We can then write:

$$y_i = x_i \oplus r_{i,j^\star} \oplus \bigoplus_{j \neq i, j^\star} r_{ij}$$

Since $i \notin I$, the random $r_{i,j^\star}$ (if $i < j^\star$) or the random $r_{j^\star,i}$ (if $i > j^\star$) has not been probed, and moreover does not enter into the computation of any other probed variable. Therefore, y_i can be perfectly simulated by generating a random value.

B Proof of Lemma 4

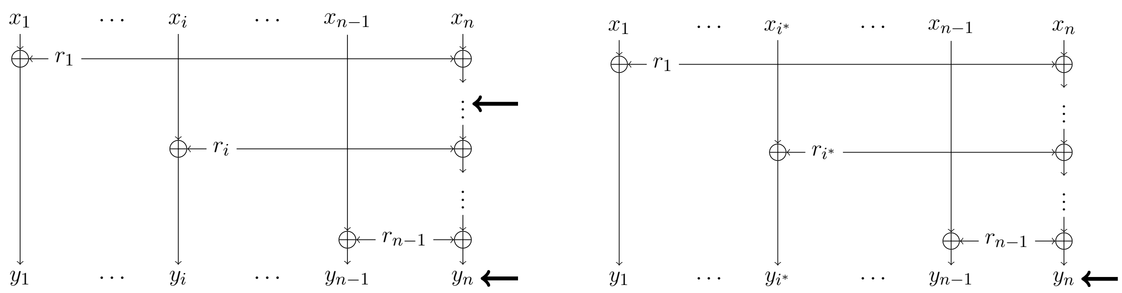

Fig. 9. Illustration of Lemma 4. Case 1 (left): the adversary has spent two probes on the column index n, and therefore $|I| \leq t-1$. Case 2 (right): no intermediate variable is probed on the last column except y_n; then r_{i*} can play the role of a one-time pad for the simulation of y_n, hence x_n is not required and again $|I| \leq t-1$.

We construct the subset I as follows. If r_i or x_i or y_i is probed for any $1 \leq i \leq n-1$, we add i to I. If x_n or any intermediate variable $y_{n,j}$ (excluding y_n) is probed, we add n to I. Since by assumption y_n has been probed, we only consider at most $t-1$ probes in the construction of I, and therefore $|I| \leq t-1$.

For the simulation we distinguish two cases. If $n \in I$, the simulation is straightforward and proceeds as in the proof of Lemma 2. Namely all the randoms r_i are simulated as in the actual algorithm, and all probed variables x_i and y_i can be perfectly simulated from x_i, since $i \in I$. This is also the case for all intermediate variables $y_{n,j}$ and y_n, which can be simulated from x_n since $n \in I$; see Figure 9 (left) for an illustration.

If $n \notin I$, then by the construction of I neither x_n nor any intermediate variable $y_{n,j}$ has been probed, except y_n. Since $|I| \leq t-1 \leq n-2$, there exists $1 \leq i^* \leq n-1$ such that $i^* \notin I$. The simulation then proceed as follows. For $i \in I$, we let $r_i \leftarrow \{0,1\}^k$ and one can perfectly simulate the probed variables r_i, x_i and y_i. It remains to simulate y_n. We can write:

$$y_n = x_n \oplus \bigoplus_{i=1}^{n-1} r_i = \left(x_n \oplus \bigoplus_{i=1,\, i \neq i^*}^{n-1} r_i \right) \oplus r_{i^*}$$

From the definition of i^*, the random r_{i^*} does not appear in the definition of any other probed variable. Therefore it can play the role of a one-time pad in the above equation, and we can simulate y_n with a random value in $\{0,1\}^k$, without knowing x_n; see Figure 9 (right) for an illustration.

C Other Properties of RefreshMasks

We recall Lemma 7 from [Cor17a].

Lemma 9. *Let $x_1, \ldots, x_n$ be the input of a* RefreshMasks *where the randoms are accumulated on x_n, and let $y_1, \ldots, y_n$ be the output. Let t be the number of probed variables, with $t = n$. If y_n is among the probed variables, then either all probed variables can be perfectly simulated from $x_1 \oplus \cdots \oplus x_n$, or there exists a subset I with $|I| \leq n-1$ such that they can be perfectly simulated from $x_{|I}$.*

Using our CheckMasks formal tool, Lemma 9 can be easily verified for small values of n. In the proof of the lemma in [Cor17a], it appears that the knowledge of $x_1 \oplus \cdots \oplus x_n$ is only necessary when the n probes are the n outputs $y_1, \ldots, y_n$ of RefreshMasks. This case is already covered by the following straightforward lemma recalled in [Cor17a].

Lemma 10. *Let $(x_i)_{1 \leq i \leq n}$ be the input and let $(y_i)_{1 \leq i \leq n}$ be the output of* RefreshMasks*. The distribution of $(y_i)_{1 \leq i \leq n}$ can be perfectly simulated from $x_1 \oplus \cdots \oplus x_n$.*

Therefore, to formally verify Lemma 9, we can exclude the previous case; one must then verify that the n probes can always be perfectly simulated from the knowledge of at most $n-1$ variables. We obtain the following timings:

n	#variables	#tuples	Security	Time
3	9	84	✓	ε
4	13	715	✓	ε
5	17	6,188	✓	ε
6	21	54,264	✓	0.4 s
7	25	480,700	✓	4.3 s

Table 4. Formal verification of Lemma 9, for small values of n.

We also formally verify Lemma 8 from [Cor17a], showing that if we xor the last two output variables y_{n-1} and y_n of RefreshMasks, then the circuit is t-NI for all $t \leq n-1$; as previously, for $t = n-1$ we must exclude the case of all $n-1$ output variables being probed. The proof is a straightforward application of Lemma 4 and Lemma 9.

Lemma 11. *Consider the circuit with $y_1, \ldots, y_n \leftarrow \mathsf{RefreshMasks}(x_1, \ldots, x_n)$, $z_i \leftarrow y_i$ for all $1 \le i \le n-2$ and $z_{n-1} \leftarrow y_{n-1} \oplus y_n$. Let t be the number of probed variables. If $t < n-1$, there exists a subset I with $|I| \le t$ such that all probed variables can be perfectly simulated from $x_{|I}$. If $t = n-1$, then either all probed variables can be perfectly simulated from $x_1 \oplus \cdots \oplus x_n$, or there exists a subset I with $|I| \le n-1$ such that they can be perfectly simulated from $x_{|I}$.*

Using our CheckMasks tool, we obtain the following timings. As explained previously, it suffices to check the t-NI property for $t = n-1$.

n	#variables	#tuples	Security	Time
3	10	45	✓	ε
4	14	364	✓	ε
5	18	3,060	✓	ε
6	22	26,334	✓	0.5 s
7	26	230,230	✓	5.7 s

Table 5. Formal verification of Lemma 11, for small values of n.

Finally, we also verify Lemma 5 from [Cor17a]. We consider the RefreshMasks algorithm taking as input $n+1$ shares (instead of n), but we fix $x_{n+1} = 0$. In that case, we show that any t probes in the circuit can be simulated from $t-1$ input shares (instead of t), except in the trivial case of the adversary probing the input x_i's only.

Lemma 12. *Let $x_1, \ldots, x_n$ be n inputs shares, and let $x_{n+1} = 0$. Consider the circuit $y_1, \ldots, y_{n+1} \leftarrow \mathsf{RefreshMasks}_{n+1}(x_1, \ldots, x_n, x_{n+1})$, where the randoms are accumulated on x_{n+1}. Let t be the number of probed variables. There exists a subset I such that all probed variables can be perfectly simulated from $x_{|I}$, with $|I| \le t-1$, except if only the input x_i's are probed.*

From the reasoning of Section 3.5, we only have to verify Lemma 12 for $t = n$. We obtain the following timings.

n	#variables	#tuples	Security	Time
3	12	220	✓	ε
4	16	1,820	✓	ε
5	20	15,504	✓	0.3 s
6	24	134,596	✓	2.9 s
7	28	1,184,040	✓	34 s

Table 6. Formal verification of Lemma 12, for small values of n.

D Proof of Lemma 3 via Circuit Transformations

The proof of the t-SNI property of SecMult proceeds in two steps. In the first step, we define an index $i^\star$ and we show that we can remove the row $i^\star$ from the circuit; as in the t-NI proof from Section 4.5, we obtain a transformed circuit C in which all the variables $r_{j,i^\star}$ on the column $i^\star$ are independent randoms (see Fig. 10). We then show recursively that the resulting circuit C is t-SNI. For this, in the second step, we define another index $k^\star \neq i^\star$, and we show that we can remove the row and column corresponding to $k^\star$. We then obtain a circuit similar to C but with $n-1$ inputs instead of n; one can then apply the recursive hypothesis.

$$
\begin{array}{c}
\left(\begin{array}{ccccc}
0 & \cdots & r_{1,i^\star} & \cdots & r_{1,n} \\
\vdots & \ddots & & & \vdots \\
r_{i^\star,1} & \cdots & 0 & \cdots & r_{i^\star,n} \\
\vdots & & & \ddots & \vdots \\
r_{n,1} & \cdots & r_{n,i^\star} & \cdots & 0
\end{array}\right)
\begin{array}{c} c_1 \\ \vdots \\ c_{i^\star} \\ \vdots \\ c_n \end{array}
\end{array}
\longrightarrow
\begin{array}{c}
\left(\begin{array}{ccccc}
0 & \cdots & r_{1,i^\star} & \cdots & r_{1,n} \\
\vdots & & \vdots & & \vdots \\
r_{i^\star-1,1} & \cdots & r_{i^\star-1,i^\star} & \cdots & r_{i^\star-1,n} \\
r_{i^\star+1,1} & \cdots & r_{i^\star+1,i^\star} & \cdots & r_{i^\star+1,n} \\
\vdots & & \vdots & & \vdots \\
r_{n,1} & \cdots & r_{n,i^\star} & \cdots & 0
\end{array}\right)
\begin{array}{c} c_1 \\ \vdots \\ c_{i^\star-1} \\ c_{i^\star+1} \\ \vdots \\ c_n \end{array}
\end{array}
$$

Fig. 10. In the first step, we remove the $i^\star$-th row, and we obtain a transformed circuit in which the variables $r_{j,i^\star}$ on the column $i^\star$ are all independent randoms.

First step. We let U be the set of of indices i such that $r_{i,j}$ or $c_{i,j}$ has been probed (for any j). We also construct a set V using the following rule:

$$\text{If } a_i b_j + r_{ij} \text{ has been probed: put } j \text{ in } V \text{ if } i \in O \text{ or } i \in U\text{, otherwise put } i \text{ in } V. \tag{3}$$

Since we have considered at most t probes in the definition of U and V, we must have $|U| + |V| \leq t$, which gives $|U| + |V| + |O| \leq t + |O| < n$. Therefore we can let $1 \leq i^\star \leq n$ such that $i^\star \notin U \cup V \cup O$.

By definition of $i^\star$, none of the $r_{i^\star,j}$ or $c_{i^\star,j}$ variables has been probed. In particular, on the row $i^\star$ and before the diagonal ($j < i^\star$), the variable $r_{i^\star,j} = (a_j b_{i^\star} + r_{j,i^\star}) + a_{i^\star,j}$ has not been probed. Therefore we can remove these variables from the circuit. This implies that we can remove the row corresponding to $i^\star$ from the circuit; however the variables $a_{i^\star} b_j$ or $a_j b_{i^\star}$ can still be probed, so we must keep them in a separate list L of variables that can be probed. On the row $i^\star$ and after the diagonal ($j > i^\star$) the variable $r_{i^\star,j}$ is not probed; it is used only in the variable $a_{i^\star} b_j + r_{i^\star,j}$, which is used in $r_{j,i^\star} = (a_{i^\star} b_j + r_{i^\star,j}) + a_j b_{i^\star}$. We claim that the $a_{i^\star} b_j + r_{i^\star,j}$ variable is also not probed; namely, if it had been probed, since $i^\star \notin O$ and $i^\star \notin U$, from Rule (3) we would have $i^\star \in V$, a contradiction. We can therefore apply the one-time pad transform twice on $r_{i^\star,j}$, and consider a modified circuit in which $r_{j,i^\star}$ for $j > i^\star$ (below the diagonal) is an independent random. In summary, we obtain a transformed circuit in which on the column $i^\star$, the variables $r_{j,i^\star}$ are independent randoms for all $j \neq i^\star$; see Figure 10 for an illustration. Moreover, above the diagonal ($j < i^\star$), the variables $a_j b_{i^\star} + r_{j,i^\star}$ can still be probed; we note that for such j, we must have $j \notin U \cup O$ (otherwise, from Rule (3) we would have $i^\star \in V$, a contradiction).

Second step. We consider the transformed circuit from the first step and taking as input n shares. We show recursively that the circuit is t-SNI. We still define the sets U and V as previously. We must have $|U| \leq n-1$. We distinguish two cases. If $|U| = n-1$, we must have $t \geq n-1$. We again distinguish two cases. If none of the variables $a_j b_{i^\star} + r_{j,i^\star}$ has been probed, then neither $a_{i^\star}$ nor $b_{i^\star}$ is required for the simulation; we can therefore let $I = [1,n] \setminus \{i^\star\}$ for the simulation of the full circuit. If at least one of the variables $a_j b_{i^\star} + r_{j,i^\star}$

has been probed, we must have $t \geq n$ and therefore we can let $I = [1, n]$ for the simulation of the full circuit. In both cases we have $|I| \leq t$ as required.

We now consider the second case, namely $|U| < n-1$. In that case we can let $k^\star \notin U \cup \{i^\star\}$. Recall that on the $i^\star$ column, all variables $r_{j,i^\star}$ are independent randoms (see Fig. 11), and moreover above the diagonal ($j < i^\star$), the variables $a_j b_{i^\star} + r_{j,i^\star}$ can be probed. We distinguish two cases. If the variable $a_{k^\star} b_{i^\star} + r_{k^\star,i^\star}$ has been probed, from Rule (3) we must have $k^\star \in V$ and $k^\star \notin O$. Since $k^\star \notin U \cup O$, the random $r_{k^\star,i^\star}$ has not been probed and is used only once, in the computation of the previous variable $a_{k^\star} b_{i^\star} + r_{k^\star,i^\star}$. Therefore we can perfectly simulate the previous variable, without knowing $a_{k^\star}$ and $b_{i^\star}$.

We now consider the second case, in which the variable $a_{k^\star} b_{i^\star} + r_{k^\star,i^\star}$ has not been probed. Since in that case the random $r_{k^\star,i^\star}$ is used only once and in the computation of $c_{k^\star}$, the $c_{k^\star}$ output variable can be perfectly simulated if $k^\star \in O$, without knowing $a_{k^\star}$ and $b_{k^\star}$.

$$
\left(\begin{array}{ccccccc|c}
0 & \cdots & r_{1,k^\star} & \cdots & r_{1,i^\star} & \cdots & r_{1,n} & c_1 \\
\vdots & \ddots & \vdots & & \vdots & & \vdots & \vdots \\
r_{k^\star,1} & \cdots & 0 & \cdots & r_{k^\star,i^\star} & \cdots & r_{k^\star,n} & c_{k^\star} \\
\vdots & & \vdots & & \vdots & & \vdots & \vdots \\
r_{i^\star-1,1} & \cdots & r_{i^\star-1,k^\star} & \cdots & r_{i^\star-1,i^\star} & \cdots & r_{i^\star-1,n} & c_{i^\star-1} \\
r_{i^\star+1,1} & \cdots & r_{i^\star+1,k^\star} & \cdots & r_{i^\star+1,i^\star} & \cdots & r_{i^\star+1,n} & c_{i^\star+1} \\
\vdots & & \vdots & & \vdots & & \vdots & \vdots \\
r_{n,1} & \cdots & r_{n,k^\star} & \cdots & r_{n,i^\star} & \cdots & 0 & c_n
\end{array}\right)
$$

Fig. 11. In the second step, we define a second index $k^\star \neq i^\star$. Thanks to the random $r_{k^\star,i^\star}$, we can perfectly simulate the output $c_{k^\star}$, and then remove the row and column corresponding to $k^\star$.

In both cases, on the row $k^\star$, none of the variables $c_{k^\star,j}$ has been probed, so they can be removed from the circuit. Moreover, on the row $k^\star$, before the diagonal ($j < k^\star$), the variables $r_{k^\star,j} = (a_j b_{k^\star} + r_{j,k^\star}) + a_{k^\star} b_j$ are also not probed, so they can also be removed from the circuit. After the diagonal ($j > k^\star$), the randoms $r_{k^\star,j}$ are not probed and are used only in the variable $a_{k^\star} b_j + r_{k^\star,j}$, which are used in the variables $r_{j,k^\star} = (a_{k^\star} b_j + r_{k^\star,j}) + a_j b_{k^\star}$. Therefore, we can replace the variable $r_{j,k^\star}$ by an independent random, and replace the variable $a_{k^\star} b_j + r_{k^\star,j}$ by the identical variable $a_j b_{k^\star} + r_{j,k^\star}$. Therefore, on the column $k^\star$, all the variables $r_{i,k^\star}$ are independent randoms; moreover the variables $a_i b_{k^\star} + r_{i,k^\star}$ can possibly be probed. We apply the Random-zero transform to all these randoms on the $k^\star$ column; the variables $a_i b_{k^\star}$ are put in a separate list of variables that can be probed. We can then remove the column and row corresponding to $k^\star$. Therefore, for the simulation of the resulting circuit, the knowledge of $a_{k^\star}$ and $b_{k^\star}$ is not necessary anymore. After removing the $k^\star$ row and column, we obtain a circuit with $n-1$ inputs, with the same structure as in the beginning of the second step. We can therefore apply the recursive hypothesis: all t probes and all output variables c_i for $i \in O$ can be perfectly simulated from $a_{|I}$ and $b_{|J}$, where $|I| \leq t$ and $|J| \leq t$. This implies that the same property holds for the original circuit; this proves the lemma. □

Formal verification. As previously, the above proof can be formally verified by performing the elementary circuit transformations for all possible indices $i^\star \neq k^\star$. Note that the number of cases to consider is now quadratic in n, instead of linear in the t-NI proof from Section

4.5; therefore, as opposed to the previous transcripts, it is too long to be included in this paper, even for small values of n. We refer to [Cor17b] for the source code to generate such transcript, for any value of n.