Formal Verification of Side-channel Countermeasures via Elementary Circuit Transformations

Jean-Sébastien Coron

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

January 23, 2018

Abstract. We describe a technique to formally verify the security of masked implementations against side-channel attacks, based on elementary circuit transforms. 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 also show how to generate security proofs automatically, for simple circuits. We describe the implementation of CheckMasks, a formal verification tool for side-channel countermeasures. Using this tool, we formally verify the security of the Rivain-Prouff countermeasure for AES, and also the recent Boolean to arithmetic conversion algorithm from CHES 2017.

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, 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 automatically verify the security of high-order masking schemes.

The first step is to specify what it means for a masking countermeasure to be secure, *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 the t probes do not bring any useful information to the attacker, since he could run this simulation by himself.

More precisely, the simulation 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 an original variable x is shared into n shares x_i. When x is part of the secret-key, this pre-sharing cannot be probed by the adversary. Since any subset of at most

$n-1$ input shares x_i are uniformly and independently distributed, 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 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 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. 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 simplify and extend the formal verification results from [BBD+15] and [BBD+16]. 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]).

For the generic verification of countermeasures at small orders, we use a different formal language from [BBD+15]. In particular we represent the underlying circuit as nested lists, which leads to a simple and concise implementation in Common Lisp, a programming language well suited to formal manipulations. We are then able to formally verify the security of the Rivain-Prouff countermeasure with very few lines of code. Our running times for formal verification are similar to those in [BBD+15]. Thanks to this simpler approach, we could also extend [BBD+15] to handle a combination of arithmetic and Boolean operations, and we have formally verified the security of the recent Boolean to arithmetic conversion algorithm from [Cor17b]. To perform these formal verifications we describe the implementation of CheckMasks, our formal verification tool for side-channel countermeasures.

For the verification of specific gadgets at any order, our technique is quite different from [BBD+16] and consists in applying elementary transforms 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 transforms, the formal verification time becomes polynomial in t (instead of exponential with the generic approach); this implies that the formal verification can be performed at any order. Using our CheckMasks tool, we provide a formally verified proof of the 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.

Finally, we show how to get the best of both worlds, at least for simple circuits: we show how to automatically apply the circuit transforms that lead to a polynomial time verification, based on a limited set of generic rules. Namely we identify a set of three simple rules that enable to automatically prove the t-SNI property of the multiplication based mask refreshing, and also two security properties of mask refreshing considered in [Cor17b].

Source Code. The source code of our CheckMasks verification tool is publicly available at [Cor17a], 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 $(x_i)_{1\leq i\leq n}$, we denote by $x_{|I} := (x_i)_{i\in I}$ the sub-vector of shares x_i with $i \in I$. In general we wish to simulate any subset of intermediate variables of a gadget from the knowledge of as few x_i's as possible.

Definition 1 (t-NI security). *Let G be a gadget taking as input $(x_i)_{1\leq i\leq n}$ and outputting the vector $(y_i)_{1\leq i\leq n}$. 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 $|I| \leq t$, 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 $(x_i)_{1\leq i\leq n}$ and outputting $(y_i)_{1\leq i\leq n}$. 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 $|I| \leq t$, 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, in which $n \geq 2t + 1$ shares are required. The stronger t-SNI notion allows for securely composing masked algorithms, and allows to prove the security with $n \geq t + 1$ 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.

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 Boolean 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.

3.1 The RefreshMasks Algorithm

To illustrate our approach we first consider the RefreshMasks algorithm below from [RP10]; see Figure 1 for an illustration.

Algorithm 1 RefreshMasks

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

1: $y_n \leftarrow x_n$
2: **for** $i = 1$ to $n - 1$ **do**
3: $\quad r_i \leftarrow \{0,1\}^k$
4: $\quad y_i \leftarrow x_i \oplus r_i$
5: $\quad y_n \leftarrow y_n \oplus r_i$ $\quad \triangleright\ y_{n,i} = x_n \oplus \bigoplus_{j=1}^{i} r_j$
6: **end for**
7: **return** $y_1, \ldots, y_n$

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. In the next section, we show how to formally verify this property.

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

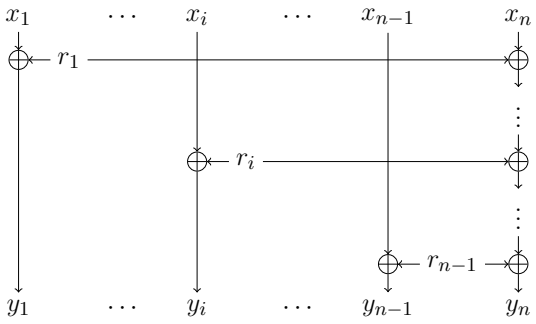

Fig. 1. The RefreshMasks algorithm, with the randoms r_i accumulated on the last column.

3.2 Formal Verification of Circuits

Circuit representation. We represent a circuit with nested lists, using the prefix notation. Consider for example the circuit taking as input x and y and outputting $x \oplus y$; 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 $x \cdot (y \oplus z)$, 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 $(x \oplus y, x \cdot y)$ 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 [Cor17a] for the source code. For example, we obtain for $n = 3$ input shares:

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

which corresponds to $y_1 = r_1 \oplus x_1$, $y_2 = r_2 \oplus x_2$ and $y_3 = r_2 \oplus (r_1 \oplus x_3)$. 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 more complex circuits.

List substitutions. We now explain how to formally verify Lemma 1. 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. Starting with the above list of two output variables, we can perform the following sequence of elementary substitutions:

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

The first substitution is possible because R2 is used only once, and the second substitution is possible because R1 is used only once after the first substitution. Since we have obtained two distinct randoms (R1 R2) at the end, the two output variables are uniformly and independently distributed, as required.

Formal verification. 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 elementary substitutions a list of $n-1$ distinct randoms. These substitutions 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, and if a random R is used only once, we can then perform the substitution:

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

In the particular case of Lemma 1, there are only n subsets to consider, so the formal verification is performed 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 $n = 3$; namely in all 3 possible cases, after a sequence of elementary substitutions, we obtain a list of 2 distinct randoms, showing that the two output variables are uniformly and independently distributed; see [Cor17a] for the source code.

3.3 Security properties of RefreshMasks

In this section we show how to formally verify some existing properties of RefreshMasks. We first consider the straightforward t-NI property.

Lemma 2 (t-NI of RefreshMasks). *Let $(x_i)_{1\leq i\leq n}$ be the input of* RefreshMasks *and let $(y_i)_{1\leq i\leq n}$ 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 $|I| \leq t$.*

Formal verification of the t-NI property of RefreshMasks. The t-NI property of RefreshMasks is straightforward because in the definition of RefreshMasks, any intermediate variable depends on at most one input x_i; therefore any subset of t probes can be perfectly simulated from the knowledge of at most t inputs x_i. Consider for example RefreshMasks with $n = 3$ as previously:

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

If we probe the two intermediate variables (+ R1 X1) and (+ R1 X3), then the knowledge of the two inputs X1 and X2 is sufficient for the simulation; moreover we cannot perform any substitution because the random R1 is used twice. On the other hand if we probe the two variables (+ R2 X2) and (+ R1 X3), we can perform the substitution:

$$((+\ \mathrm{R2}\ \mathrm{X2})\ (+\ \mathrm{R1}\ \mathrm{X3})) \rightarrow (\mathrm{R2}\ (+\ \mathrm{R1}\ \mathrm{X3})) \rightarrow (\mathrm{R2}\ \mathrm{R1})$$

showing that the knowledge of the input variables X2 and X3 is not required for that simulation.

More generally, to verify the t-NI property of any circuit, it suffices to exhaustively consider all possible t-uples of intermediate variables, and verify that after a set of elementary substitutions the knowledge of at most t input variables is needed for the simulation of the t-uple, for any $1 \leq t \leq n-1$.

Other Security Properties of RefreshMasks. In Appendix A, we perform a formal verification of several non-trivial properties of RefreshMasks that were used to prove the security of the Boolean to arithmetic conversion algorithm from [Cor17b]. The first 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 x_i only, instead of t as in Lemma 2. This property was crucial for obtaining a provably secure Boolean to arithmetic conversion algorithm in [Cor17b].

Lemma 3 (RefreshMasks [Cor17b]). *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 $|I| \leq t-1$.*

As previously, to perform a formal verification of Lemma 3, it suffices to consider all possible t-uples of intermediate variables (where y_n is part of the t-uple) and show that after a sequence of elementary substitutions, there remains at most $t-1$ input variables. In Appendix A, we argue that it is actually sufficient to perform such verification for $t = n-1$ only, instead of all $1 \leq t \leq n-1$. The timings of formal verification are summarized in Table 1. 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. We refer to Appendix A for some other properties of RefreshMasks and their formal verification 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	✓	0.2 s
7	25	177,100	✓	1.5 s
8	29	1,560,780	✓	17 s
9	33	13,884,156	✓	195 s

Table 1. Formal verification of Lemma 3, for small values of n.

3.4 Formal Verification of t-SNI properties: the FullRefresh and SecMult Algorithms

It is easy to see that that the RefreshMasks algorithm from the previous section 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 $y_{n,1} = r_1 \oplus x_n$. 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, whereas only $t = 1$ internal variables has been probed.

The FullRefresh algorithm. We recall below an improved mask refreshing algorithm that does satisfy the t-SNI property, as shown in [BBD+16]. The algorithm FullRefresh 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 RefreshMasks.

Lemma 4 (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 4 using our CheckMasks tool. As previously we first implement the FullRefresh algorithm in Common Lisp; for example, we get the following output for $n = 3$ shares:

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

Using our CheckMasks tool, the t-SNI property in Lemma 4 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. Consider for example 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:

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

and therefore no input x_i is actually needed to simulate those two variables. However if we probe the two variables (+ R2 (+ R1 X1)) and X2, we can perform the substitutions:

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

and therefore the knowledge of X2 is required for the simulation.[1] Note that the running time to consider all possible $(n-1)$-uples of intermediate variables is exponential in n. We summarize in Table 2 the running time of the formal verification of FullRefresh, up to $n = 6$. In Section 5 we will show how to formally verify Lemma 4 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 2. Formal verification of the t-SNI property of FullRefresh, for small values of n.

[1] This is still according to the t-SNI property, because (+ R2 (+ R1 X1)) is an output variable and therefore $t = 1$.

The Rivain-Prouff Countermeasure. The Rivain-Prouff countermeasure for AES is based on 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. The algorithm was proven t-SNI in [BBD+16]. In Appendix B we recall the corresponding SecMult algorithm, and we show how to formally verify its t-SNI property for small values of n.

4 Formal Verification of Boolean to Arithmetic Conversion

In this section we show how to extend [BBD+15] to handle a combination of arithmetic and Boolean operations. This enables to formally verify the security of the high-order Boolean to arithmetic conversion algorithm recently described at CHES 2017 [Cor17b], with a t-SNI security proof for $n \geq t+1$. The algorithm can be seen as a generalization of Goubin's algorithm [Gou01] to any order, still with a complexity independent of the register size k. Although the algorithm has complexity $\mathcal{O}(2^n)$, instead of $\mathcal{O}(n^2 \cdot k)$ in [CGV14], for small values of n it is an order of magnitude more efficient. The algorithm takes as input n Boolean shares x_i such that

$$x = x_1 \oplus \cdots \oplus x_n$$

and using a recursive algorithm computes n arithmetic shares D_i such that

$$x = D_1 + \cdots + D_n \pmod{2^k}$$

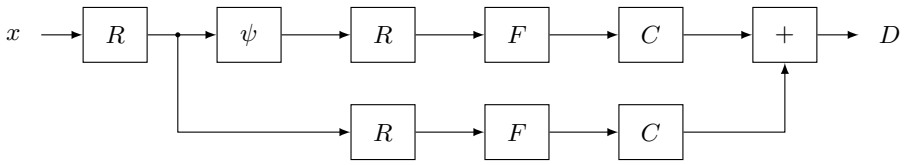

Fig. 2. Sequence of operations in the Boolean to arithmetic conversion algorithm from [Cor17b].

Boolean to arithmetic conversion. The algorithm is based on the affine property of the function $\Psi(x, r) := (x \oplus r) - r \pmod{2^k}$. As illustrated in Fig. 2 the algorithm is recursive and makes two recursive calls to the same algorithm C with $n-1$ inputs. For $n = 2$ one uses a t-SNI variant of Goubin's algorithm:

$$D_1 = \big((x_1 \oplus r_1) \oplus \Psi(x_1 \oplus r_1, r_2 \oplus (x_2 \oplus r_1))\big) \oplus \Psi(x_1 \oplus r_1, r_2) \tag{2}$$

$$D_2 = x_2 \oplus r_1 \tag{3}$$

For $n \geq 3$ the algorithm works as follows. One first performs a mask refreshing R, while expanding the x_i's to $n+1$ shares. One obtains, from the definition of the Ψ function:

$$\begin{aligned} x &= x_1 \oplus x_2 \oplus \cdots \oplus x_{n+1} \\ &= (x_1 \oplus \cdots \oplus x_{n+1} - x_2 \oplus \cdots \oplus x_{n+1}) + x_2 \oplus \cdots \oplus x_{n+1} \\ &= \Psi(x_1, x_2 \oplus \cdots \oplus x_{n+1}) + x_2 \oplus \cdots \oplus x_{n+1} \end{aligned}$$

From the affine property of the Ψ function, the left term can be decomposed into the xor of n shares $\Psi(x_1, x_i)$ for $2 \leq i \leq n+1$, where the first share is $(\overline{n \wedge 1}) \cdot x_1 \oplus \Psi(x_1, x_2)$:

$$x = (\overline{n \wedge 1}) \cdot x_1 \oplus \Psi(x_1, x_2) \oplus \Psi(x_1, x_3) \oplus \cdots \oplus \Psi(x_1, x_{n+1}) + x_2 \oplus \cdots \oplus x_{n+1}$$

We obtain that x is the arithmetic sum of two terms, each with n Boolean shares; this corresponds to the two branches in Fig. 2. One then performs a mask refreshing R on both branches, and then a compression function F that simply xors the last two shares, so there remains only $n-1$ shares on both branches. One can then apply the Boolean to arithmetic conversion C recursively on both branches, taking as input $n-1$ Boolean shares (instead of n), and outputting $n-1$ arithmetic shares; we obtain:

$$x = \big(A_1 + \cdots + A_{n-1}\big) + \big(B_1 + \cdots + B_{n-1}\big) \pmod{2^k}$$

Eventually it suffices to do some additive grouping to obtain n arithmetic shares as output, as required:

$$x = D_1 + \cdots + D_n \pmod{2^k}$$

We refer to [Cor17b] for the details of the algorithm. The algorithm in proven t-SNI secure with $n \geq t+1$ shares in [Cor17b].

Algorithm representation. In Section 3.3 and Appendix A we have described a formal verification of the security properties of RefreshMasks that are required for the security proof of the above Boolean to arithmetic conversion algorithm in [Cor17b]. However this provides only a *partial* verification of the algorithm, since in that case the adversary is restricted to only probing the Boolean operations performed within the RefreshMasks. To obtain a *full* verification, we must consider an adversary who can probe any variable in the Boolean to arithmetic algorithm. In that case the formal verification becomes more complex as we must handle both Boolean and arithmetic operations.

Since in our nested list representation we have already using the + operator for the xor, we use the ADD keyword to denote the arithmetic sum. For example, the final additive grouping can be represented as:

```
> (additive-grouping '(A1 A2) '(B1 B2))
((ADD A1 B1) A2 B2)
```

which corresponds to the three arithmetic shares $D_1 = A_1 + B_1 \pmod{2^k}$, $D_2 = A_2$ and $D_3 = B_2$. We also use the PSI operator to denote the application of the Ψ function. For example, the Boolean to arithmetic conversion algorithm for $n = 2$ gives from (2) and (3):

```
> (convba '(X1 X2))
((+ (+ (+ X1 R1) (PSI (+ X1 R1) (+ R2 (+ X2 R1))))
    (PSI (+ X1 R1) R2))
 (+ X2 R1))
```

Simplification rules. Given a list of intermediate variables that must be simulated, as previously we must use a set of simplification rules to determine how many inputs x_i are required for the simulation. For the verification of Boolean circuits in the previous section, this was relatively straightforward as we had essentially a single simplification rule, namely replacing $x \oplus r$ by r when the random r appears only once in the intermediate variables. However when combining arithmetic and Boolean operations the formal verification becomes more complex and we used the following simplification rules. We illustrate every rule by an example that can be run from the source code [Cor17a].

- Rule 1: when $x_1 + x_2 \bmod 2^k$ must be simulated, simulate both x_1 and x_2.

```
> (prop-add '((ADD X1 X2)))
(X1 X2)
```

- Rule 2: from the affine property of the function Ψ, replace $\Psi(x,y) \oplus \Psi(x,z)$ by $x \oplus \Psi(x, y \oplus z)$.

```
> (replace-psi '(+ (PSI A B) (PSI A C)))
(+ A (PSI A (+ B C)))
```

- Rule 3: from the definition of Ψ, replace $\Psi(x, y)$ by $(x \oplus y) - y \bmod 2^k$; we denote by SUB the arithmetic subtraction.

```
> (replace-psi-sub '(PSI A B)
(SUB (+ A B) B)
```

- Rule 4: when a random r is used only once, replace $x \oplus r$ by r, and similarly for $x + r \bmod 2^k$ and $x - r \bmod 2^k$. This is an extension of the rule given by (1).

```
> (iter-simplify '((+ X1 R1) (ADD X2 R2) (SUB X3 R3)))
(R1 R2 R3)
```

- Rule 5: when a random r is not used in two intermediate variables e_1 and e_2, replace the simulation of $(e_1 \oplus r, e_1 \oplus r)$ by the simulation of $(r, (e_1 \oplus r) \oplus e_2)$; this corresponds to the change of variable $r' = e_1 \oplus r$.

```
> (simplify-x '((+ R1 X1) (+ R1 X2)))
(R1 (+ (+ R1 X1) X2))
```

- Rule 6: when $\Psi(x_1, x_2)$ must be simulated, simulate both x_1 and x_2.

```
> (prop-psi '((PSI A B)))
(A B)
```

We note that the order in which the rules are applied matters. For example, once Rule 3 has been applied, Rule 2 cannot be applied to the same expression, because the PSI operator has been replaced by SUB. One must therefore use the right strategy for the application of the rules; an overview is provided in Figure 3. In particular, we only apply Rule 3 if subsequently applying Rule 4 enables to eliminate the SUB operator, and Rule 6 is only applied as a last resort, when other rules have failed.

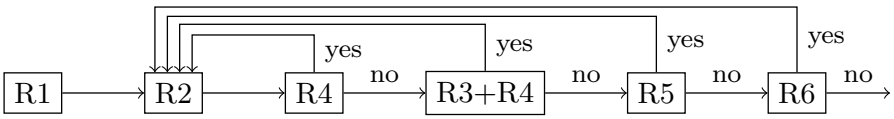

Fig. 3. The rule application strategy for the formal verification of Boolean to arithmetic conversion.

Formal verification. In order to verify the t-SNI property of the Boolean to arithmetic algorithm, as previously we must check that for all possible $(n-1)$-uples of intermediate variables (including the outputs D_i), the number of input variables x_i's that remain after the application of the above rules is always $\leq t$, where t is the number of non-output variables in the $(n-1)$-uple.

We summarize in Table 3 the timings of formal verification. Note that the Boolean to arithmetic conversion algorithm has complexity $\mathcal{O}(2^n)$, and therefore the number of possible $(n-1)$-uples of intermediate variables is $\mathcal{O}(2^{n^2})$; that is why we could only perform the formal verification up to $n = 5$.

5 Formal Verification in Polynomial Time

The main drawback of the previous approach 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

n	#variables	#tuples	Security	Time
2	11	11	✓	ε
3	48	1,128	✓	0.08 s
4	133	383,306	✓	85 s
5	312	387,278,970	✓	88 h

Table 3. Formal verification of the t-SNI property of the Boolean to arithmetic conversion algorithm from [Cor17b].

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 approach consists in applying a sequence of elementary circuit transforms, until the transformed circuit becomes so simple that the security property becomes straightforward to verify. 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]. For Boolean circuits 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.

The Random-zero Transform. Our first circuit transformation consists in setting to 0 a subset of the randoms r_i used in the circuit. The transform only applies to *additively masked* circuits.

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 $y = x_1 \cdot r_1$ is not. Most side-channel countermeasures for block-ciphers are additively masked. In particular, this holds for the RefreshMasks, FullRefresh and SecMult algorithms considered in the previous sections. The following lemma shows that it is sufficient to consider the security of a simpler circuit C_0 where a subset of the randoms are fixed to 0. Namely if there is an attack against the original circuit C, then the same attack applies against C_0; see Appendix C for the proof.

Lemma 5 (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.*

Remark 1. Lemma 5 does not hold for general circuits; consider for example the 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 the complexity would be exponential in n when examining all possible t-uples.

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.

We can assume that a certain random r has not been probed 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 One-time Pad transform on this random. 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.

Formal verification in polynomial-time. More generally, the proof strategy is to perform a sequence of elementary circuit transforms until we obtain a simple circuit C for which the t-NI or t-SNI properties is straightforward to verify. In appendices D, E and F 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. We refer to [Cor17a] for the source code of the formal verification.

6 Towards Automatic Generation of Security Proofs

The drawback of the previous approach is that for the security verification to happen in polynomial time, we must select ourselves the right sequence of circuit transforms. Instead we would like to have the circuit transforms being selected automatically by our verification tool, based on a limited set of elementary rules, and still in polynomial-time.

In the following, we show that this can be achieved for simple circuits based on the three following rules. We denote by P the property that must be checked; for example, for t-NI security, the property P would require that any t-uple of intermediate variables is simulatable from a subset of the inputs $x_{|I}$, with $|I| \leq t$. Below we denote by C_{otp} the circuit $y_i = x_i \oplus r_i$ for $1 \leq i \leq n$ (see Appendix D). We assume that the property P is already verified by C_{otp}, so that P does not need to be verified explicitly for C_{otp}.

(R1) Perform a loop to select and remove the subset of the circuit that is unprobed.

(R2) Apply the random-zero transform, except on randoms used only once in the circuit.

(R3) Check whether the resulting circuit is equal to C_{otp}. Otherwise check the property P for all possible t-uple of probes.

We show in Table 4 that from the three above rules, we can formally verify in polynomial time the main properties of RefreshMasks and FullRefresh considered in this paper; we refer to Appendix G for the details, and to [Cor17a] for the source code of the formal verification.

Algorithm	Property	Lemma	Rules	Final circuit
RefreshMasks	t-NI	Lemma 2	R2, R3	$(x_1, \ldots, x_n)$
FullRefresh	t-SNI	Lemma 4	R1, R2, R3	C_{otp}
RefreshMasks	$\lvert I\rvert \leq t-1$ with probed y_n	Lemma 3	R1, R2, R3	$(x_1, \ldots, x_{n-1}, \quad x_n \oplus r_i)$
RefreshMasks	$\lvert I\rvert \leq t-1$ with $x_{n+1} = 0$	Lemma 6	R1, R2, R3	$(x_1, \ldots, x_{n-1}, \quad x_n \oplus r_i)$ or C_{otp}

Table 4. Rules and final circuit to verify a security property in polynomial-time in n.

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. Publicly available at https://eprint.iacr.org/2015/060.

[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. See also a preliminary version, under the title "Compositional Verification of Higher-Order Masking: Application to a Verifying Masking Compiler", publicly available at https://eprint.iacr.org/2015/506/20150527:192221.

[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.

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between boolean and arithmetic masking of any order. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 188–205, 2014.

[Cor17a] Jean-Sébastien Coron. CheckMasks: formal verification of side-channel countermeasures, 2017. Publicly available at https://github.com/coron/checkmasks.

[Cor17b] Jean-Sébastien Coron. High-order conversion from boolean to arithmetic masking. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 93–114, 2017.

[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.

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In *CHES*, pages 3–15, 2001.

[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.

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

A Other Security Properties of RefreshMasks

A.1 Formal Verification of Lemma 3

We provide a formal verification of a non-trivial property of RefreshMasks from [Cor17b, Lemma 6] mentioned in Section 3.3: 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.

Lemma 3 (RefreshMasks [**Cor17b**]). *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 2. 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 3 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 $t < n - 1$; we show that it will not be satisfied for $t = n - 1$. More precisely, assume that there exists a set of t probes which can only be simulated by a subset I of inputs with $|I| \geq t$, for some $t < n-1$. If $|I| \geq n-1$, then we can take any superset of $t' = n-1$ probes and we get $|I| \geq t'$. If $|I| < n-1$, 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 3 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. 4. Formal verification of Lemma 3 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.

We provide in Figure 4 the transcript of the formal verification for $n = 4$ input shares, using our CheckMasks tool. More precisely, we compute the list of 3-uples that require at least 3 inputs x_i to be simulated. We see that as required none of these 3-uples include the output y_n; therefore Lemma 3 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 3 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 1 in Section 3.3, up to $n = 9$. We refer to [Cor17a] for the source code.

A.2 Formal Verification of [Cor17b, Lemma 5]

We consider the RefreshMasks algorithm taking as input $n + 1$ shares (instead of n), but we fix $x_{n+1} = 0$. In that case, 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 6 (RefreshMasks [**Cor17b**]). *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 $|I| \leq t-1$, except if only the input x_i's are probed.

From the previous reasoning, we only have to verify Lemma 6 for $t = n$; we summarize the timings in Table 5.

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 5. Formal verification of Lemma 6, for small values of n.

A.3 Formal Verification of [Cor17b, Lemma 7]

Lemma 7 (RefreshMasks **[Cor17b]**). *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 7 can be easily verified for small values of n. In the proof of the lemma in [Cor17b], 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 [Cor17b].

Lemma 8. *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 7, 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 6. Formal verification of Lemma 7, for small values of n.

A.4 Formal Verification of [Cor17b, Lemma 8]

We also formally verify [Cor17b, Lemma 8], 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 3 and Lemma 7. Using our CheckMasks tool, we obtain the timings from Table 7. As explained previously, it suffices to check the t-NI property for $t = n - 1$.

Lemma 9 (RefreshMasks [**Cor17b**]). *Consider the circuit* $y_1, \ldots, y_n \leftarrow \mathsf{RefreshMasks}(x_1, \ldots, x_n)$, $z_i \leftarrow y_i$ *for all* $1 \leq i \leq n-2$ *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* $|I| \leq t$ *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* $|I| \leq n-1$ *such that they can be perfectly simulated from* $x_{|I}$.

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 7. Formal verification of Lemma 9, for small values of n.

B The Rivain-Prouff Countermeasure

The Rivain-Prouff countermeasure for the AES block-cipher is based on the SecMult algorithm below [RP10]; 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 10 (t-**SNI of** SecMult [**BBD+16**]). *Let* $(a_i)_{1 \leq i \leq n}$ *and* $(b_i)_{1 \leq i \leq n}$ *be the input shares of the* SecMult *operation, 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 such that* $t + \mathcal{O} < n$, *there exist two subsets* I *and* J *of indices with* $|I| \leq t$ *and* $|J| \leq t$, *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 [Cor17a] 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 10, 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), which gives $|I| = |J| = 2$. 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 8. Formal verification of the t-SNI property of SecMult, for small values of n.

C The Random-zero Transform: Proof of Lemma 5

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:

$$h(\boldsymbol{x}, \boldsymbol{r}) = h(\boldsymbol{x}, \boldsymbol{r'}) + g(\boldsymbol{r''}) \tag{4}$$

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 (4), 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 5. □

D Formal Verification of Lemma 4 for FullRefresh

In this section we provide a formal proof of Lemma 4 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 now polynomial in n. The proof strategy is to perform a sequence of elementary circuit transforms 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 transforms in Common Lisp and checking that we indeed obtain this simple circuit C. In the case of FullRefresh we obtain the following simple circuit C_{otp}, which is t-SNI.

Lemma 11 (t-SNI of C_{otp}). *Let C_{otp} 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 $i \in \mathcal{O} \setminus I$; in that case $y_i = x_i \oplus r_i$ can be simulated by a random value since r_i is not probed, because $i \in \mathcal{O} \setminus I$. □

We recall the t-SNI property of FullRefresh below from Section 3.4; this t-SNI property was already proven in [BBD+16]. Below we provide an alternative proof of Lemma 4, based on elementary circuit transforms, so that the proof can be formally verified in our CheckMasks tool.

Lemma 4 (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 $x_{|I}$.*

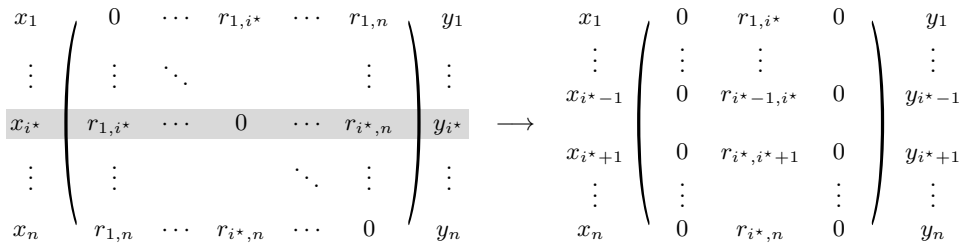

Fig. 5. Proof of Lemma 4: after removing the row $i^\star$ and setting all randoms to 0 except on the column $i^\star$, there remains only a one-time pad of the $n-1$ inputs x_i for $i \neq i^\star$, corresponding to the circuit C_{otp} from Lemma 11.

Proof. We first construct a subset I of indices as follows. We refer to the definition of Alg. 2 for the notations. 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; see Fig 5 for an illustration.

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 5 for an illustration. This is exactly the circuit C_{otp} from Lemma 11 with $n-1$ inputs. Since from Lemma 11 this circuit is t-SNI for all $t \leq n-1$, using Lemma 5 the FullRefresh circuit is also t-SNI for all $t < n$, which proves the lemma. □

Note that the main difference with the original proof of Lemma 4 in [BBD+16] is that we have not performed an explicit simulation of the probed variables; instead we have performed a sequence of elementary circuit transforms (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 $a_1 \oplus r_1, \ldots, a_{n-1} \oplus r_{n-1}$, namely the C_{otp} circuit. Since such circuit is t-SNI from Lemma 11, the original circuit is t-SNI. We illustrate in Fig. 6 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)))
  Set 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)))
  Set 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)))
  Set all randoms to 0 except (R2 R3) => ((+ R2 X1) (+ R3 X2))
```

Fig. 6. Formal verification of the FullRefresh circuit for $n = 3$.

E Formal Verification of lemmas 3 and 6 for RefreshMasks in Polynomial Time

E.1 Formal Verification of Lemma 3 for RefreshMasks

We now consider the RefreshMasks algorithm (see Fig. 1), and we recall the security property of RefreshMasks considered in Section 3.3: 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. The lemma below was already proven in [Cor17b]. In this section we provide an alternative proof based on elementary circuit transforms that can be formally verified in time polynomial in n, using our CheckMasks tool.

Lemma 3. *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 3.3). 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.

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

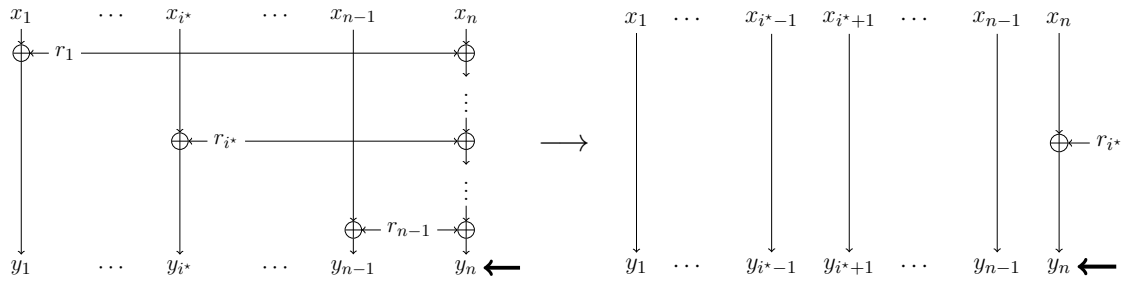

Fig. 7. Proof of Lemma 3: after removing the sub-circuit corresponding to $i^\star$ and setting to zero all randoms except $i^\star$, the remaining circuit is the identity circuit except $y_n = x_n \oplus r_{i^\star}$.

the Random-zero transform to all randoms except $r_{i^\star}$. As illustrated in Figure 7, we obtain a circuit taking as input the x_i's for $1 \leq i \leq n$ and $i \neq i^\star$, and outputting $y_i = x_i$ for $1 \leq i \leq n-1$ 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 3. This could be proved using the classical simulation-based approach, but we can also continue with elementary transforms, 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 $(x_1, \ldots, x_n)$ except $x_{i^\star}$ 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 3. □

It is easy to verify the above proof with a formal tool, since it consists in elementary circuit transforms conditioned on the value of $1 \leq i^\star \leq n-1$; we provide the transcript of the formal proof for $n = 4$ in Fig. 8; see [Cor17a] 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. 8. Formal verification of Lemma 3 for $n = 4$, using our CheckMasks tool for performing the sequence of elementary transforms.

Remark 3. The above formal verification of Lemma 3 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 3.3, which has complexity $2^{3.2\,n}$ (see Table 1 for the corresponding timings).

E.2 Formal Verification of Lemma 6 for RefreshMasks

We consider the RefreshMasks algorithm, and we recall the other security property of RefreshMasks considered in Appendix A.2; see [Cor17b, Lemma 5] for the pen-and-paper proof.

Lemma 6 (RefreshMasks) **[Cor17b].** *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 $|I| \leq t-1$, except if only the input x_i's are probed.*

Below we provide an alternative proof that can be formally verified in time polynomial in n, using our CheckMasks tool. We first prove the following simple lemma, on the same C_{otp} circuit as considered in Appendix D.

Lemma 12. *Let C_{otp} 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. For any set of t intermediate variables, there exists a subset I of indices with $|I| \leq t-1$, such that the t intermediate variables can be perfectly simulated from $x_{|I}$, except if only the input shares x_i are probed.*

Proof. By assumption, there exists an index $i^\star$ such that $r_{i^\star}$ or $y_{i^\star}$ or both have been probed, with $1 \leq i^\star \leq n$. We construct the set I as follows. For any $i \neq i^\star$, if x_i or r_i or y_i has been probed, we add i to I; moreover if $x_{i^\star}$ has been probed, or if both $r_{i^\star}$ and $y_{i^\star}$ have been probed, we add $i^\star$ to I. We first show that we must have $|I| \leq t-1$ as required. Namely either a single variable among $r_{i^\star}$ and $y_{i^\star}$ has been probed, and this probe does not contribute to I, or both $r_{i^\star}$ and $y_{i^\star}$ have been probed, and these two probes contribute to only one index in I.

One can then simulate any probed variable x_i, y_i and r_i for $i \neq i^\star$ from $i \in I$. If $i^\star \in I$, then $x_{i^\star}$, $y_{i^\star}$ and $r_{i^\star}$ can also be simulated. Finally, if $i^\star \notin I$, then either $r_{i^\star}$ or $y_{i^\star}$ has been probed (but not both); in both cases such variable can be perfectly simulated. □

We now proceed with the proof of Lemma 6. As previously, the proof strategy is to perform a sequence of elementary circuit transforms until we obtain the above circuit C_{otp} on which the property is proven by Lemma 12.

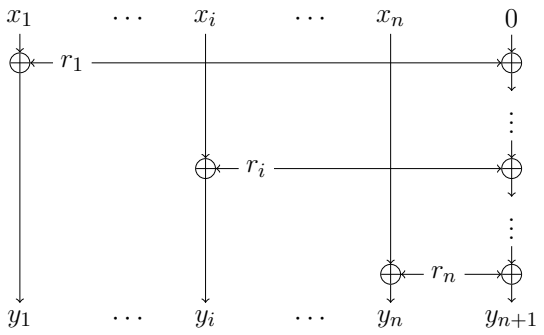

Fig. 9. The RefreshMasks circuit with $n+1$ inputs, with $x_{n+1} = 0$.

Proof. From the reasoning of Section 3.3, we only have to prove the lemma for $t = n$. We distinguish two cases; see Figure 9 for an illustration. If none of the intermediate variables $y_{n+1,j}$ has been probed nor y_{n+1}, we remove the corresponding subcircuit, and there remains the circuit C_{otp} for which the property is proven by Lemma 12. In the second case, if one of the intermediate variables $y_{n+1,j}$ or y_{n+1} has been probed, we apply the random-zero transform to all r_i's. There remains a circuit outputting $(x_1, \ldots, x_n, 0)$, where one of the probe is now 0. Therefore the remaining $n-1$ probes can be simulated by $x_{|I}$ with $|I| \leq n-1$. □

We provide in Figure 10 the transcript of the formal proof for $n = 4$; see [Cor17a] for the source code.

```
> (check-refreshmasks-zero-poly 4)
Input: (X1 X2 X3 X4 0)
Output: ((+ R1 X1) (+ R2 X2) (+ R3 X3) (+ R4 X4)
         (+ R4 (+ R3 (+ R2 R1))))
Excluded: (X1 X2 X3 X4)
Case 1: one probe in ((+ R4 (+ R3 (+ R2 R1))))
  Random zero: (X1 X2 X3 X4 0)
  First probe: 0
  Other 3 probes in: (X1 X2 X3 X4 0)
Case 2: no probe in ((+ R4 (+ R3 (+ R2 R1))))
  Subcircuit: ((+ R1 X1) (+ R2 X2) (+ R3 X3) (+ R4 X4))
```

Fig. 10. Formal verification of Lemma 6 for $n = 4$, using our CheckMasks tool for performing the sequence of elementary transforms.

F Formal Verification of SecMult in Polynomial-Time

In this section our goal is to provide a proof of the t-SNI property of SecMult from the Rivain-Prouff countermeasure [RP10], that can be formally verified in polynomial time; this corresponds to Lemma 10 from Appendix B.

F.1 Formal Verification of the t-NI Property in Polynomial Time

As a warm-up we consider the weaker t-NI security property, for which a pen-and-paper proof was already given in [ISW03]; below we provide a proof that can be formally verified in our CheckMasks tool, based on circuit transforms.

Lemma 13 (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.

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

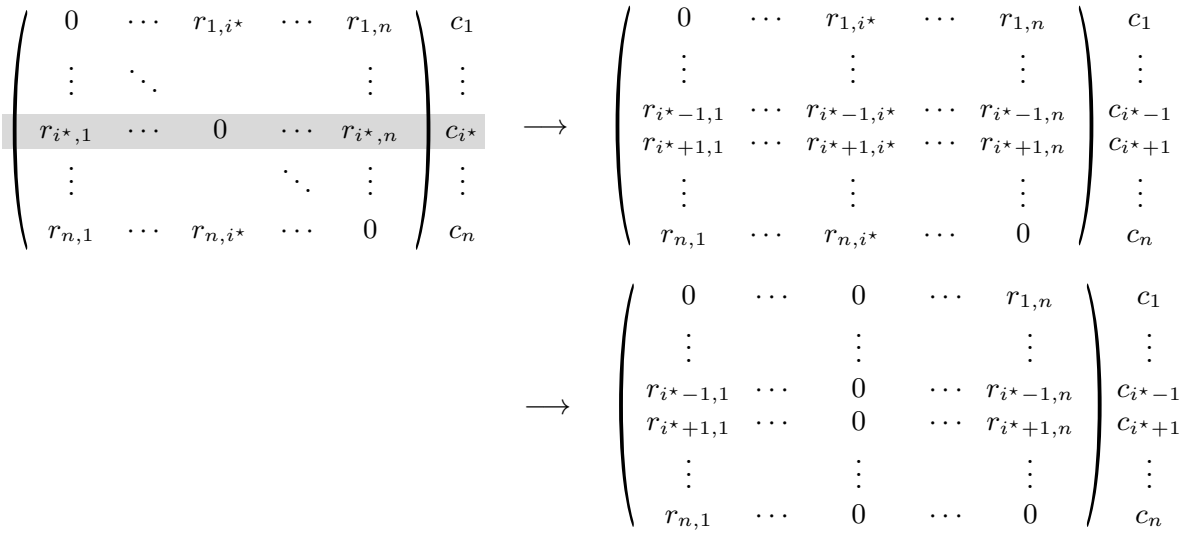

Fig. 11. After removing the $i^\star$-th row and applying the one-time pad transform, we obtain a column $i^\star$ in which all variables $r_{j,i^\star}$ are independent randoms. One can then apply the random-zero transform, and eventually remove the column $i^\star$.

$r_{i^\star,j}$ 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. 11). 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 11, 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 11 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 in Figure 14 in Appendix H the transcript of the formal verification for $n = 3$; we refer to [Cor17a] for the source code. We see that in each case, one obtains after a sequence of elementary transforms a circuit that is equivalent to the original circuit but with $n-1$ input shares; therefore one can apply the recursive hypothesis.

F.2 Proof of Lemma 10 via Circuit Transforms

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; 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.

12). 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.

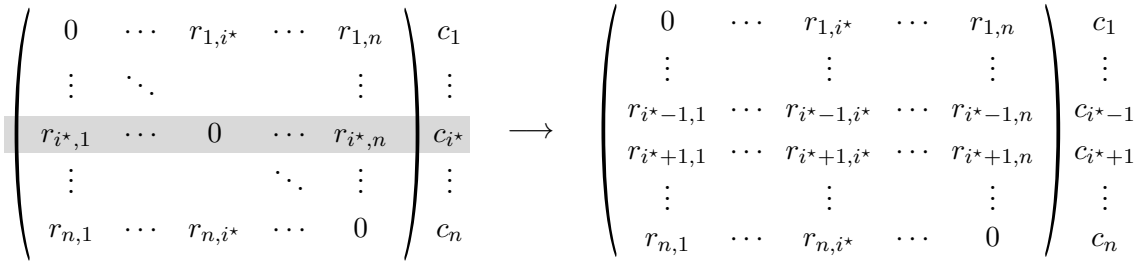

Fig. 12. In the first step, we remove the $i^\star$-th row, and we obtain a transformed circuit in which the variables $r_{j,i^\star}$ on the column $i^\star$ are all independent randoms.

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:

$$\text{If } a_i b_j + r_{ij} \text{ has been probed: put } j \text{ in } V \text{ if } i \in O \text{ or } i \in U, \text{ otherwise put } i \text{ in } V. \tag{5}$$

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 (5) 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 12 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 (5) 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. 13), and

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

Fig. 13. 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 $k^\star$.

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 (5) 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 $b_{k^\star}$.

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. The above proof can be formally verified by performing the elementary circuit transforms for all possible indices $i^\star \neq k^\star$. We refer to [Cor17a] for the source code to generate a proof transcript, for any value of n.

G Automatic Verification for Simple Circuits

In this section, we show that by applying the rules R1, R2 and R3 from Section 6, the security properties of RefreshMasks from lemma 2, 3 and 6 can be formally verified in polynomial time, as well as Lemma 4 for FullRefresh; see Table 4.

Namely, for verifying the t-NI property of RefreshMasks, as explained in Section 5, it suffices to apply the Random-zero transform to all randoms r_i of the circuit, and one obtains the identity

circuit which is trivially t-NI. More precisely, we can apply rule R2 which sets all randoms to 0, since all randoms are used twice in the circuit. Then the t-NI property for $t = n - 1$ is immediately verified, since there are $\binom{n}{n-1} = n$ uples of $n-1$ probes; for this we use the generic technique from Section 3, and in this case it works in polynomial time. Alternatively one can first apply Rule R1 and then rules R2 and R3 on each subcircuit, and the complexity of verification is still polynomial-time.

For the t-SNI property of FullRefresh, the successive application of rules R1, R2 and R3 is exactly what is done in the proof of Lemma 4 in Section D. Namely for each subcircuit obtained by removing the subcircuit corresponding to the output y_i, after applying the random-zero transform to all r_{ij} except those used only once, we obtain the C_{otp} circuit whose t-SNI property is proven in Lemma 11.

For the property of RefreshMasks proven in Lemma 3, the successive application of rules R1, R2 and R3 also leads to a polynomial-time verification. Namely, since y_n is probed, rule R1 is applied by performing a loop on $1 \leq i \leq n-1$ and removing the subcircuit corresponding to y_i. In the resulting subcircuit, the only random that is used only once is r_i. Therefore by applying rule R2, one obtains the final circuit $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_{n-1}, x_n \oplus r_i)$ where the last output $y_n = x_n \oplus r_i$ is probed. The final circuit above has now $n+2$ intermediate variables, and since y_n is already probed, the number of $(n-1)$-uples to consider is $\binom{n+1}{n-2} \leq n^3$. Therefore the property can be verified using the generic technique from Section 3, and in this case it works in polynomial time.

Finally, for the property of RefreshMasks from Lemma 6, the successive application of rules R1, R2 and R3 also leads to a polynomial-time verification. Recall that in Lemma 6 the RefreshMasks circuit takes as input $n+1$ shares with $x_{n+1} = 0$, and outputs $n+1$ shares y_i. Firstly, rule R1 is applied by performing a loop on $1 \leq i \leq n+1$ and removing the subcircuit corresponding to y_i. When the subcircuit corresponding to y_{n+1} has been removed, the resulting circuit is exactly the C_{otp} circuit; since all randoms are used once, the rule R2 does nothing, and eventually the rule R3 applies, and the required property is proved in Lemma 12. Moreover, when the subcircuit corresponding to y_i has been removed for $1 \leq i \leq n$, all randoms are used twice except r_i. By applying Rule R2, one obtains the final circuit $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n, r_i)$. As previously, the required property can therefore be verified by rule R3 using the generic technique from Section 3, and in this case it also works in polynomial time.

For the automatic verification of the four above properties, we refer to [Cor17a] for the source code, and to Appendix I for the transcript of the proof for $n = 3$.

H Transcript of Lemma 13 Formal Verification

```
> (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. 14. 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 $j < i$.

I Transcript of Automatic Verification

```
> (check-circuits 3)
Refreshmasks: t-NI property:
Input: (X1 X2 X3)
Circuit: ((+ R1 X1) (+ R2 X2) (+ R2 (+ R1 X3)))
  R1: ((+ R2 X2) (+ R2 (+ R1 X3)))  R2: (X2 (+ R1 X3))  R3: is OTP: NIL
    R3: check: T
  R1: ((+ R1 X1) (+ R2 (+ R1 X3)))  R2: (X1 (+ R2 X3))  R3: is OTP: NIL
    R3: check: T
  R1: ((+ R1 X1) (+ R2 X2))  R2: ((+ R1 X1) (+ R2 X2))  R3: is OTP: T
  Verif: T

FullRefresh: t-SNI property:
Input: (X1 X2 X3)
Circuit: ((+ R2 (+ R1 X1)) (+ R3 (+ R1 X2)) (+ R3 (+ R2 X3)))
  R1: ((+ R3 (+ R1 X2)) (+ R3 (+ R2 X3)))  R2: ((+ R1 X2) (+ R2 X3))
R3: is OTP: T
  R1: ((+ R2 (+ R1 X1)) (+ R3 (+ R2 X3)))  R2: ((+ R1 X1) (+ R3 X3))
R3: is OTP: T
  R1: ((+ R2 (+ R1 X1)) (+ R3 (+ R1 X2)))  R2: ((+ R2 X1) (+ R3 X2))
R3: is OTP: T
  Verif: T

Refreshmasks: with probed yn:
Input: (X1 X2 X3)
Circuit: ((+ R1 X1) (+ R2 X2) (+ R2 (+ R1 X3)))
  R1: ((+ R2 X2) (+ R2 (+ R1 X3)))  R2: (X2 (+ R1 X3))  R3: is OTP: NIL
    R3: check: T
  R1: ((+ R1 X1) (+ R2 (+ R1 X3)))  R2: (X1 (+ R2 X3))  R3: is OTP: NIL
    R3: check: T
  R1: ((+ R1 X1) (+ R2 X2))  R2: ((+ R1 X1) (+ R2 X2))  R3: is OTP: T
  Verif: T

Refreshmasks: with probed x_{n+1}=0:
Input: (X1 X2 X3 0)
Circuit: ((+ R1 X1) (+ R2 X2) (+ R3 X3) (+ R3 (+ R2 (+ R1 0))))
  R1: ((+ R2 X2) (+ R3 X3) (+ R3 (+ R2 (+ R1 0))))  R2: (X2 X3 R1)
    R3: is OTP: NIL  R3: check: T
  R1: ((+ R1 X1) (+ R3 X3) (+ R3 (+ R2 (+ R1 0))))  R2: (X1 X3 R2)
    R3: is OTP: NIL  R3: check: T
  R1: ((+ R1 X1) (+ R2 X2) (+ R3 (+ R2 (+ R1 0))))  R2: (X1 X2 R3)
    R3: is OTP: NIL  R3: check: T
  R1: ((+ R1 X1) (+ R2 X2) (+ R3 X3))  R2: ((+ R1 X1) (+ R2 X2)
    (+ R3 X3))  R3: is OTP: T
  Verif: T
```

Fig. 15. Automatic verification of lemmas 2, 4, 3 and 6 for $n = 3$, based on rules R1, R2 and R3.