Improved High-Order Conversion From Boolean to Arithmetic Masking

Luk Bettale[1], Jean-Sébastien Coron[2], and Rina Zeitoun[1]

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

Abstract. Masking is a very common countermeasure against side channel attacks. When combining Boolean and arithmetic masking, one must be able to convert between the two types of masking, and the conversion algorithm itself must be secure against side-channel attacks. An efficient high-order Boolean to arithmetic conversion scheme was recently described at CHES 2017, with complexity independent of the register size. In this paper we describe a simplified variant with fewer mask refreshing, and still with a proof of security in the ISW probing model. In practical implementations, our variant is roughly 25% faster.

1 Introduction

The masking countermeasure. Masking with random values is a well known and effective countermeasure against side-channel attacks. Every variable x in the circuit can be masked into $x' = x \oplus r$, where r is a random value with the same bitsize as x. When the two shares x' and r are manipulated separately, any first-order attack is thwarted because every intermediate variable considered separately has the uniform distribution. However a second-order attack combining the two shares x' and r can still be feasible in practice, though it usually requires a much greater amount of side-channel acquisitions; see for example [OMHT06].

To prevent first order attacks, Boolean masking can be naturally extended to n shares, where every variable x in the circuit is written as:

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

In that case, a side-channel countermeasure should be resistant against any t-th order attack, in which the side-channel information from at most $t < n$ variables is combined. More precisely, it should have a proof of security in the t-probing model introduced by Ishai, Sahai and Wagner [ISW03]. In this model, the adversary can probe at most t wires in the circuit, and should not learn anything about the secret inputs. It was shown in [ISW03] that any circuit C can be transformed into a new circuit C' of size $\mathcal{O}(t^2 \cdot |C|)$ that is secure against t probes, using $n \geq 2t + 1$ shares.

Boolean vs arithmetic masking. Boolean masking is well adapted for algorithms that have Boolean operations only, such as AES. However for algorithms that combine Boolean and arithmetic operations (such as IDEA [LM90], RC6 [CRRY99], XTEA [NW97], SPECK [BSS+13] and SHA-1 [NIS95]), it can be advantageous to use arithmetic masking to protect the arithmetic operations, and switch between Boolean and arithmetic masking whenever necessary. Obviously the conversion

algorithm itself must be secure against side-channel attacks. More precisely, starting from a Boolean masking with n shares x_i such that:

$$x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$$

one must compute n arithmetic shares A_i such that:

$$x = A_1 + A_2 + \cdots + A_n \pmod{2^k}$$

without leaking information about x. In general all arithmetic operations are performed modulo 2^k for some fixed integer k; for example, $k = 32$ in SHA-1.

The first conversion algorithms between Boolean and arithmetic masking secure against first-order attacks were described by Goubin in [Gou01]. Goubin's first-order Boolean to arithmetic conversion is very efficient with complexity independent of the register size k. This is based on a surprising property of the function

$$\psi(x, y) = (x \oplus y) - y \pmod{2^k}$$

Although it contains an arithmetic operation, the function is affine in y with respect to the xor; see [Gou01] for a proof. The other direction (from arithmetic to Boolean) is less efficient with complexity $\mathcal{O}(k)$; this was later improved to $\mathcal{O}(\log k)$ in [CGTV15]; however the $\mathcal{O}(\log k)$ complexity hides a larger constant, and in practice for $k = 32$ the number of operations is similar.

For security against high-order attacks (*i.e.* with n shares), the first conversion algorithms were described in [CGV14], with complexity $\mathcal{O}(n^2 \cdot k)$ for n shares and k-bit addition in both directions, with a proof of security in the ISW model. As in Goubin's first-order case, the complexity can be improved to $\mathcal{O}(n^2 \cdot \log k)$ using the same technique as in [CGTV15]; however for $k = 32$ this does not really improve the practical efficiency.

Recently, an efficient high-order Boolean to arithmetic conversion algorithm was described in [Cor17c], with complexity independent of the register size k (as in Goubin's first-order algorithm), following an approach initiated by Hutter and Tunstall in [HT16]. The conversion algorithm has a security proof in the ISW model; in contrast, the original algorithm described in [HT16] had no such security proof, and quite unsurprisingly, a third-order attack was described in [Cor17c] for any number of shares n; the updated Hutter-Tunstall algorithm is also vulnerable to a similar third-order attack.[1]

Although the complexity of the new algorithm is $\mathcal{O}(2^n)$, in practice for small values of n it is at least one order of magnitude more efficient than [CGV14, CGTV15]. We summarize in Table 1 the complexities of Boolean to arithmetic conversions in both directions, for first-order attacks, and for high-order attacks.

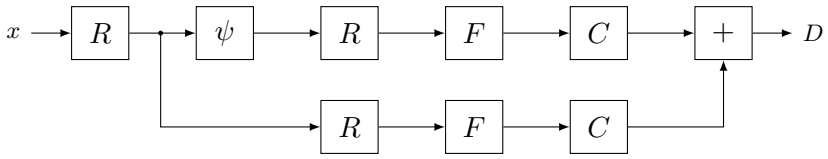
[1] The attack against [HT16], version posted on 02-Mar-2018, works as follows. We only look at Step 1, which computes the variable $\omega = x + (\alpha \oplus r_1 \oplus \cdots \oplus r_n) + (\mu_1 \oplus \cdots \oplus \mu_{n-1})$ with the intermediate variable $\zeta = (x + (\alpha \oplus r_1 \oplus \cdots \oplus r_n)) \oplus (\mu_1 \oplus \cdots \oplus \mu_{n-1})$ which is computed in (26). We also have the intermediate variable $f = (x' \oplus \gamma_1 \oplus \alpha) + \gamma_1$ which is computed in (24), where $x' = x \oplus r_1 \oplus \cdots \oplus r_n$.

We probe the 3 intermediate variables ω, ζ and f. Letting $R = \alpha \oplus r_1 \oplus \cdots \oplus r_n$ and $U = \mu_1 \oplus \cdots \oplus \mu_{n-1}$, we obtain $\omega = x + R + U$, $\zeta = (x + R) \oplus U$ and $f = (x \oplus R \oplus \gamma_1) + \gamma_1$. It is easy to check that for uniformly distributed γ_1, R and U, the triple (ω, ζ, f) leaks information about the secret x; intuitively this is because the pair (ω, ζ) leaks information about $x + R$, while f leaks information about $x \oplus R$, so the combination leaks information about x. This gives a 3rd order attack that works for any n. In particular, in the authors' explicit 3rd-order algorithm (Alg. 3), the variables ω, ζ and f correspond to the variables v_{63}, v_{52} and v_{29} respectively.

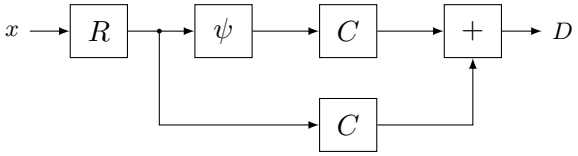
	Direction	First-order complexity	High-order complexity
Goubin's algorithm [Gou01]	B → A A → B	$\mathcal{O}(1)$ $\mathcal{O}(k)$	- -
[CGV14]	B → A A → B	-	$\mathcal{O}(n^2 \cdot k)$
[CGTV15]	B → A A → B	- $\mathcal{O}(\log k)$	$\mathcal{O}(n^2 \cdot \log k)$
[Cor17c]	B → A	-	$14 \cdot 2^n + \mathcal{O}(n)$
This paper	**B → A**	-	$10 \cdot 2^n + \mathcal{O}(n)$

Table 1. Complexities of Boolean to arithmetic conversions in both directions, for first-order attacks, and for high-order attacks.

Our contribution. In this paper we describe a simplified variant of the Boolean to arithmetic conversion algorithm from [Cor17c], with fewer mask refreshing, and still with a proof of security in the ISW probing model. As illustrated in Figure 1, our new conversion algorithm is simpler than [Cor17c]: it still makes two recursive calls to the same conversion algorithm C at order $n-1$, but performs only a single mask refreshing R instead of 3; moreover it does not use a compression function F. The asymptotical complexity of our variant is still $\mathcal{O}(2^n)$, that is exponential in the number of shares n and independent of the register size k. More precisely, the number of operations is $10 \cdot 2^n + \mathcal{O}(n)$, instead of $14 \cdot 2^n + \mathcal{O}(n)$ in [Cor17c]. In section 7, we describe a practical implementation of our new algorithm, showing that for small n it is still one order of magnitude faster than [CGV14], and roughly 25% more efficient than [Cor17c].



(a) The original countermeasure from [Cor17c].



(b) Our new countermeasure.

Fig. 1. The sequence of operations in the Boolean to arithmetic conversion algorithm.

Source code. A proof-of-concept implementation of our high-order conversion algorithm, using the C language, is available at:

`https://pastebin.com/WKnNyEU8`

2 Security Definitions

The ISW probing model. The theoretical study of securing a circuit against an adversary who can probe a fraction of its wires was initiated by Ishai, Sahai and Wagner [ISW03]. They showed how to transform any circuit of size $|C|$ into a circuit of size $\mathcal{O}(|C| \cdot t^2)$ secure against any adversary who can probe at most t wires. 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 learning any information about the initial variable x, for $n \geq 2t + 1$.

The ISW approach for proving security against an adversary who can probe at most t probes is based on simulation. The goal is to show that any set of t probes in the circuit can be perfectly simulated without the knowledge of any of the original (non-masked) input variables of the initial circuit; in particular, for a block-cipher, such simulation can be performed without knowing the secret key. This implies that probing those t wires is not going to help the adversary, since he could simulate those t probes by himself.

A proof in the ISW model usually proceeds as follows: given any set of t probes, one constructs iteratively a subset I of indices of the input shares x_i that are sufficient to simulate the t probes. If we can ensure that $|I| < n$, then only a proper subset of the input shares is required for the simulation. Then if the shares x_i are initially generated in such a way that any $n-1$ subset of shares are uniformly and independently distributed, the simulation can be performed without knowing the original variable x.

***t*-SNI Security.** Recently, a refined security definition under the ISW probing model was introduced in [BBD+16], called t-SNI security. The t-SNI security definition enables to prove that a gadget can be used in a full construction with $n \geq t + 1$ shares only, instead of $n \geq 2t + 1$ in the original ISW security proof. The main advantage of the t-SNI approach is that it enables modular security proofs, by first considering the t-SNI security of individual gadgets and then composing them in a more complex construction. In this paper, the security of our Boolean to arithmetic conversion algorithm will be proven under the t-SNI definition, as in [Cor17c].

We recall below the t-NI and t-SNI security notions introduced in [BBD+16]. We consider a gadget taking as input a single n-tuple $(x_i)_{1 \leq i \leq n}$ of shares, and outputting a single n-tuple $(y_i)_{1 \leq i \leq n}$. Given a subset $I \subset [1, n]$, we denote by $x_{|I}$ all elements x_i such that $i \in I$.

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 $t_c \leq t$ intermediate variables, there exists a subset I of input indices with $|I| \leq t_c$, such that the t_c intermediate variables can be perfectly simulated from $x_{|I}$.*

Definition 2 (t-SNI security). *Let G be a gadget which takes as input n shares $(x_i)_{1 \leq i \leq n}$ and outputs n shares $(y_i)_{1 \leq i \leq n}$. The gadget G is said to be t-SNI secure if for any set of t_c probed intermediate variables and any subset $O \subset [1, n]$ of output indices, such that $t_c + |O| \leq t$, there exists a subset $I \subset [1, n]$ of input indices which satisfies $|I| \leq t_c$, such that the t_c intermediate variables and the output variables $y_{|O}$ can be perfectly simulated from $x_{|I}$.*

The difference between the t-NI and t-SNI security notions is that in the t-SNI definition the upper-bound on the size of the input subset I does not depend on the number of output shares $|O|$ that must be simulated. As shown in [BBD+16], if several gadgets are t-SNI secure, then the composition of those gadgets remains t-SNI secure. Moreover the t-SNI security notion enables to prove the security of a full construction for $n \geqslant t+1$ shares, instead of $n \geqslant 2t+1$ in the original ISW security proof.

3 Existing Boolean to Arithmetic Conversion Algorithms

3.1 Goubin's First-order Conversion

We first recall Goubin's first-order algorithm for conversion from Boolean to arithmetic masking [Gou01]. The algorithm is based on the affine property of the function $\Psi(x, r) : \mathbb{F}_{2^k} \times \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$

$$\Psi(x, r) = (x \oplus r) - r \pmod{2^k}$$

More precisely, we have for any $x, r_1, r_2 \in \mathbb{F}_{2^k}$:

$$\Psi(x, r_1 \oplus r_2) = x \oplus \Psi(x, r_1) \oplus \Psi(x, r_2) \tag{1}$$

In the remaining of this paper, all additions and subtractions are performed modulo 2^k for some fixed parameter k, so we omit the mod 2^k. Moreover we grant higher precedence to xor than addition, so we simply write $\Psi(x, r) = x \oplus r - r$.

Goubin's Boolean to arithmetic conversion is based on the affine property of ψ, see (1). Given as input two Boolean shares x_1, x_2 such that

$$x = x_1 \oplus x_2$$

we can write from the definition of Ψ and from (1):

$$\begin{aligned} x &= (x_1 \oplus x_2 - x_2) + x_2 = \Psi(x_1, x_2) + x_2 \\ &= \big[\big(x_1 \oplus \Psi(x_1, r \oplus x_2)\big) \oplus \Psi(x_1, r)\big] + x_2 \end{aligned}$$

where $r \leftarrow \{0,1\}^k$ is a randomly generated value. One can therefore compute the arithmetic share:

$$A \leftarrow \big(x_1 \oplus \Psi(x_1, r \oplus x_2)\big) \oplus \Psi(x_1, r)$$

which eventually gives two arithmetic shares of x, as required:

$$x = A + x_2 \pmod{2^k}$$

We note that the above Boolean to arithmetic conversion algorithm is quite efficient as it requires only a constant number of operations, independent of k. Moreover it is easy to see that the above algorithm is secure against first-order attacks, because the left term $x_1 \oplus \Psi(x_1, r \oplus x_2)$ is independent of x_2 (thanks to the mask r), and the right term $\Psi(x_1, r)$ is also independent from x_2; eventually the arithmetic share A is uniformly distributed when x_2 is uniformly distributed.

3.2 The High-Order Boolean to Arithmetic Conversion Algorithm from [Cor17c]

A high-order Boolean to arithmetic conversion algorithm was recently described at CHES 2017 [Cor17c], with a security proof in the ISW model for $n \geq t+1$. The algorithm can be seen as a generalization of Goubin's algorithm to any order, still with a complexity independent of the register size k. 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

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

As illustrated in Fig. 2 the algorithm is recursive and makes two recursive calls to the same algorithm C with $n-1$ inputs. 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:

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

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

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

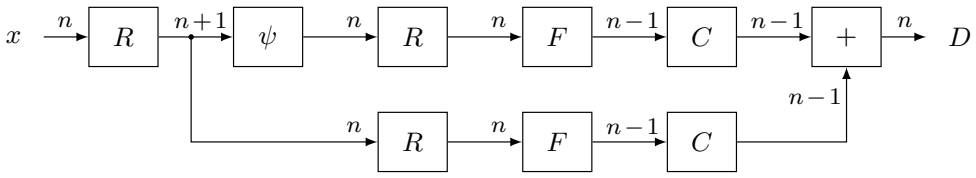


Fig. 2. The sequence of gadgets in the Boolean to arithmetic conversion algorithm from [Cor17c].

As illustrated in Fig. 2, the algorithm starts with n Boolean shares, then expands to $n+1$ shares; only n shares are used in the bottom branch, while the Ψ gadget on the upper branch decreases to

n shares. On both branches the compression function F decreases to $n-1$ shares in order to apply the algorithm recursively. Thus, on the whole, when calling $\mathcal{C}_n$ the number of shares follows the sequence:

$$\mathcal{C}_n : n \to n+1 \to n \to n-1$$

and $\mathcal{C}_{n-1}$ is recursively called with $n-1$ shares.

In this paper, our main contribution is to simplify the above Boolean to arithmetic conversion algorithm with the more compact sequence:

$$\mathcal{C}_n : n+1 \to n$$

As illustrated in Fig. 3, our new recursive algorithm $\mathcal{C}_n$ takes as input $n+1$ shares (instead of n) and recursively calls $\mathcal{C}_{n-1}$ with n shares on both branches, and eventually outputs n arithmetic shares after some arithmetic grouping. Actually we rely on the Ψ function to decrease the number of shares by one; therefore the previous compression function F is not necessary anymore, and we only need a single mask refreshing instead of 3. Finally, to obtain an algorithm that converts from n Boolean shares (instead of $n+1$ as input of $\mathcal{C}_n$) to n arithmetic shares, it suffices to initially fix $x_{n+1} = 0$ as input. We describe our new algorithm in more details in the next section, and we show that it achieves the same level of security as the original algorithm, namely t-SNI security with $n = t+1$ shares.

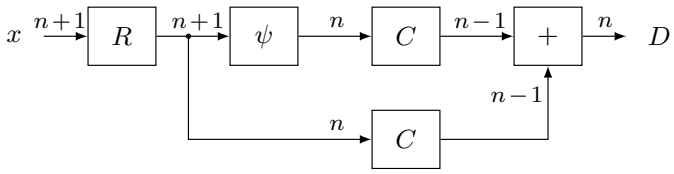


Fig. 3. The sequence of gadgets in our new Boolean to arithmetic conversion algorithm.

4 Improved High-order Conversion from Boolean to Arithmetic Masking

In this section, we describe our improved high-order Boolean to arithmetic conversion algorithm. We first describe a recursive algorithm $\mathcal{C}_n$ which, given $n+1$ Boolean shares as input, output n arithmetic shares. Then as mentioned previously, to convert from n Boolean shares instead of $n+1$, it suffices to initially let $x_{n+1} = 0$.

4.1 The new Algorithm

Our recursive algorithm $\mathcal{C}_n$ takes as input $n+1$ Boolean shares x_i such that:

$$x = x_1 \oplus \cdots \oplus x_{n+1}$$

and outputs n arithmetic shares D_i such that

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

For $n = 1$, the algorithm takes as input two shares x_1 and x_2 and simply outputs $D_1 = x_1 \oplus x_2$. We now assume that $n \geq 2$; see Fig. 3 for an illustration of the algorithm.

1. We first perform a mask refreshing of the $n+1$ shares x_i's, so that we obtain the following $n+1$ shares:

$$y_1, \ldots, y_{n+1} \leftarrow \mathsf{RefreshMasks}_{n+1}(x_1, \ldots, x_{n+1})$$

   See below for the definition of the RefreshMasks algorithm. We can write:

$$\begin{aligned} x &= y_1 \oplus y_2 \oplus \cdots \oplus y_{n+1} \\ &= y_2 \oplus \cdots \oplus y_{n+1} + (y_1 \oplus \cdots \oplus y_{n+1} - y_2 \oplus \cdots \oplus y_{n+1}) \\ &= y_2 \oplus \cdots \oplus y_{n+1} + \Psi(y_1, y_2 \oplus \cdots \oplus y_{n+1}) \end{aligned}$$

2. Thanks to the affine property of Ψ, this gives:

$$x = y_2 \oplus \cdots \oplus y_{n+1} + (\overline{n \wedge 1}) \cdot y_1 \oplus \Psi(y_1, y_2) \oplus \cdots \oplus \Psi(y_1, y_{n+1})$$

   Therefore, we let $z_1 \leftarrow (\overline{n \wedge 1}) \cdot y_1 \oplus \Psi(y_1, y_2)$ and $z_i \leftarrow \Psi(y_1, y_{i+1})$ for all $2 \leq i \leq n$. This gives:

$$x = y_2 \oplus \cdots \oplus y_{n+1} + z_1 \oplus \cdots \oplus z_n \tag{2}$$

3. We perform two recursive calls to the Boolean to arithmetic conversion algorithm $\mathcal{C}_{n-1}$:

$$\begin{aligned} A_1, \ldots, A_{n-1} &\leftarrow \mathcal{C}_{n-1}(y_2, \ldots, y_{n+1}) \\ B_1, \ldots, B_{n-1} &\leftarrow \mathcal{C}_{n-1}(z_1, \ldots, z_n) \end{aligned}$$

   This gives from (2):

$$x = A_1 + \cdots + A_{n-1} + B_1 + \cdots + B_{n-1}$$

4. We reduce the number of arithmetic shares from $2n-2$ to n by some additive grouping as in [Cor17c], letting $D_i \leftarrow A_i + B_i$ for $1 \leq i \leq n-2$, and $D_{n-1} \leftarrow A_{n-1}$ and $D_n \leftarrow B_{n-1}$. This gives as required the n arithmetic shares as output:

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

This completes the description of the recursive algorithm. Note that our recursive algorithm takes as input $n+1$ Boolean shares x_i and outputs n arithmetic shares D_i. To convert from n Boolean shares only, we simply let $x_{n+1} = 0$. Formally, we obtain the following algorithm.

Algorithm 1 High-order Boolean to Arithmetic Conversion

Input: $x_1, \ldots, x_n$
Output: $D_1, \ldots, D_n$ such that $x_1 \oplus \cdots \oplus x_n = D_1 + \cdots + D_n \pmod{2^k}$
1: $D_1, \ldots, D_n \leftarrow \mathcal{C}_n(x_1, \ldots, x_n, 0)$
2: **return** $D_1, \ldots, D_n$

Algorithm 2 $\mathcal{C}_n$: Recursive high-order Boolean to Arithmetic Conversion ($n+1 \to n$ shares)

Input: $x_1, \ldots, x_{n+1}$
Output: $D_1, \ldots, D_n$ such that $x_1 \oplus \cdots \oplus x_{n+1} = D_1 + \cdots + D_n \pmod{2^k}$
1: **if** $n = 1$ **then**
2: **return** $D_1 \leftarrow x_1 \oplus x_2$
3: **end if**
4: $y_1, \ldots, y_{n+1} \leftarrow \mathsf{RefreshMasks}(x_1, \ldots, x_{n+1})$
5: $z_1 \leftarrow (\overline{n \wedge 1}) \cdot y_1 \oplus \Psi(y_1, y_2)$
6: **for** $i = 2$ to n **do**
7: $z_i \leftarrow \Psi(y_1, y_{i+1})$
8: **end for**
9: $A_1, \ldots, A_{n-1} \leftarrow \mathcal{C}_{n-1}(y_2, \ldots, y_{n+1})$
10: $B_1, \ldots, B_{n-1} \leftarrow \mathcal{C}_{n-1}(z_1, \ldots, z_n)$
11: **for** $i = 1$ to $n-2$ **do**
12: $D_i \leftarrow A_i + B_i$
13: **end for**
14: $D_{n-1} \leftarrow A_{n-1}$
15: $D_n \leftarrow B_{n-1}$
16: **return** $D_1, \ldots, D_n$

Algorithm 3 RefreshMasks

Input: $x_1, \ldots, x_n$
Output: $y_1, \ldots, y_n$ such that $y_1 \oplus \cdots \oplus y_n = x_1 \oplus \cdots \oplus x_n$
1: $y_1 \leftarrow x_1$
2: **for** $i = 2$ to n **do**
3: $r_i \leftarrow \{0,1\}^k$
4: $y_i \leftarrow x_i \oplus r_i$
5: $y_1 \leftarrow y_1 \oplus r_i$ ▷ $y_{1,i} = x_1 \oplus \bigoplus_{j=2}^{i} r_j$
6: **end for**
7: **return** $y_1, \ldots, y_n$

Theorem 1 (Completeness). *Algorithm 1, when taking $x_1, \ldots, x_n$ as input, outputs $D_1, \ldots, D_n$ such that $x_1 \oplus \cdots \oplus x_n = D_1 + \cdots + D_n \pmod{2^k}$.*

Proof. The proof is straightforward from the above description. Since Algorithm 1 runs the recursive $\mathcal{C}_n$ algorithm with $x_{n+1} = 0$, it suffices to prove the completeness of $\mathcal{C}_n$. The completeness property clearly holds for $n = 1$. Assuming that completeness holds for $n-1$ shares, we obtain:

$$\sum_{i=1}^{n} D_i = \sum_{i=1}^{n-1} A_i + \sum_{i=1}^{n-1} B_i = \bigoplus_{i=2}^{n+1} y_i + \bigoplus_{i=1}^{n} z_i = \bigoplus_{i=2}^{n+1} y_i + \Psi\left(y_1, \bigoplus_{i=2}^{n+1} y_i\right) = \bigoplus_{i=1}^{n+1} y_i = \bigoplus_{i=1}^{n+1} x_i$$

and therefore completeness of $\mathcal{C}_n$ holds for n shares. □

4.2 Complexity Analysis

Time Complexity. We denote by T_n the number of operations of the $\mathcal{C}_n$ algorithm, with $n+1$ input shares and n output shares. Since $\mathcal{C}_1$ is a single xor, we have $T_1 = 1$. The complexity of

RefreshMasks with $n+1$ shares is $3(n+1)-3 = 3n$ operations. The computation of the Ψ function requires 2 operations, and as in [Cor17c], we assume that computing $(\overline{n \wedge 1}) \cdot y_1 \oplus \Psi(y_1, y_2)$ requires 5 operations, which gives a total of $2n+3$ operations. Finally, the additive grouping requires $n-2$ operations. We obtain:

$$\begin{aligned} T_n &= [3 \cdot n] + [2 \cdot n + 3] + 2 \cdot T_{n-1} + [n-2] \\ &= 2 \cdot T_{n-1} + 6 \cdot n + 1 \end{aligned}$$

which gives:

$$T_n = 10 \cdot 2^n - 6 \cdot n - 13 \quad ,$$

compared to $T_n = 14 \cdot 2^n - 12 \cdot n - 21$ in [Cor17c]. Therefore, the complexity of our algorithm is still exponential in n, but with an expected speed-up factor of 29% for large n, compared to [Cor17c].

Note that the complexity of our algorithm is $\mathcal{O}(2^n)$, instead of $\mathcal{O}(n^2 \cdot k)$ in [CGV14]. However for small values of n our conversion algorithm is still one order of magnitude more efficient than [CGV14]; see Table 2 for a comparison with existing algorithms; for [CGV14] we use the same operation count as estimated in [Cor17c], for $k = 32$ bits. Concrete implementation results will be further given in Section 7.

B → A conversion	Security order t								
	1	2	3	4	5	6	8	10	12
Goubin [Gou01]	7								
Hutter-Tunstall [HT16]		36							
CGV, 32 bits [CGV14]		2 098	3 664	7 752	10 226	14 698	28 044	39 518	56 344
Coron [Cor17c]		55	155	367	803	1 687	7 039	28 519	114 511
Our algorithm (Section 4.1)		49	123	277	591	1 225	5 053	20 401	81 829

Table 2. Operation count for Boolean to arithmetic conversion algorithms, up to security order $t = 12$, with $n = t + 1$ shares. For the Hutter-Tunstall algorithm we have written 36 operations instead of 31 to include the generation of 5 randoms.

Memory Complexity. Along with the $n+1$ input shares, the algorithm requires $2\,n+1$ other shares y_i and z_i's, $2\,(n-1)$ shares A_i and the B_i, and finally n output shares D_i for a total of $5\,n-1$ shares (excluding the input shares). As the two recursive calls are done sequentially, we add the memory consumption of a single call to $\mathcal{C}_{n-1}$. Denoting M_n the memory consumption of $\mathcal{C}_n$, we obtain:

$$M_n = 5\,n - 1 + M_{n-1}$$

with $M_1 = 1$. This gives:

$$M_n = 1 + \sum_{i=2}^{n} (5i - 1) = \frac{5}{2} n^2 + \frac{3}{2} n - 3$$

which is quadratic in the number of shares n.

Note that if the algorithm C_n computes its result in place, the input shares x_i can be used to store successively y_i, then A_i, then D_i. Only n additional shares z_i are required. They are also used

to store the B_i as the recursive call to C_{n-1} also performs in place. In this context, the memory complexity is reduced to

$$M_n = n + M_{n-1} = \frac{n\,(n+1)}{2}.$$

Random Generation. We denote by R_n the number of random generations performed within the $\mathcal{C}_n$ algorithm. Since the $\mathcal{C}_n$ algorithm executes a RefreshMasks with $n+1$ shares, which requires the generation of n random values, and then recursively calls $\mathcal{C}_{n-1}$ twice, we have:

$$R_n = n + 2 \cdot R_{n-1}$$

where $R_1 = 0$ since $\mathcal{C}_1$ only performs a xor. This gives the following complexity for R_n:

$$R_n = 3 \cdot 2^{n-1} - n - 2 \quad ,$$

which is also exponential in the number of shares, but about 50% smaller than in [Cor17c] for which $R_n = n + 2 \cdot (n-1) + 2 \cdot R_{n-1}$ with $R_2 = 2$, which gives $R_n = 3 \cdot 2^n - 3 \cdot n - 4$; see Table 3 for a comparison.

B → A conversion	Security order t								
	1	2	3	4	5	6	8	10	12
Previous algorithm [Cor17c]	2	11	32	77	170	359	1505	6107	24533
Our algorithm (Section 4.1)	2	7	18	41	88	183	757	3059	12273

Table 3. Number of random generations for Boolean to arithmetic conversion algorithms from [Cor17c] and this paper, up to security order $t = 12$, with $n = t + 1$ shares.

5 Security Proof of Algorithm 1

We prove that our new algorithm achieves the same level of security as the previous algorithm in [Cor17c], namely t-SNI security with $n = t+1$ shares, that is $(n-1)$-SNI security. This means that the algorithm with n shares is secure against an adversary with at most $n-1$ probes. Furthermore, the t-SNI property allows to compose the algorithm in a larger construction, without decreasing the security order.

Theorem 2 (t-SNI of Algorithm 1). *Let $(x_i)_{1\leq i\leq n}$ be the input and let $(D_i)_{1\leq i\leq n}$ be the output of the Boolean to arithmetic conversion Algorithm 1. For any set of t_c intermediate variables and any subset $O \subset [1, n]$ with $t_c + |O| < n$, there exists a subset I of input indices such that the t_c intermediate variables as well as $D_{|O}$ can be perfectly simulated from $x_{|I}$, with $|I| \leq t_c$.*

The rest of the section is devoted to the proof of Theorem 2.

5.1 Properties of RefreshMasks

In this section we first recall some security properties of the RefreshMasks algorithm for proving Theorem 2; we refer to [Cor17c] for the proof of lemmas 1, 2, 3, 4 and 5 below; we then prove an additional lemma. The following lemma shows that when RefreshMasks is not probed, any subset of $n-1$ output shares y_i is uniformly and independently distributed; see Fig. 4 for an illustration of RefreshMasks.

Lemma 1. *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. *Any subset of $n-1$ output shares y_i is uniformly and independently distributed.*

The following lemma shows that when RefreshMasks is not probed, the distribution of the n output shares y_i's can be perfectly simulated from the knowledge of $x_1 \oplus \cdots \oplus x_n$ only; that is, the knowledge of the individual shares x_i's is not required.

Lemma 2. *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 $x_1 \oplus \cdots \oplus x_n$.*

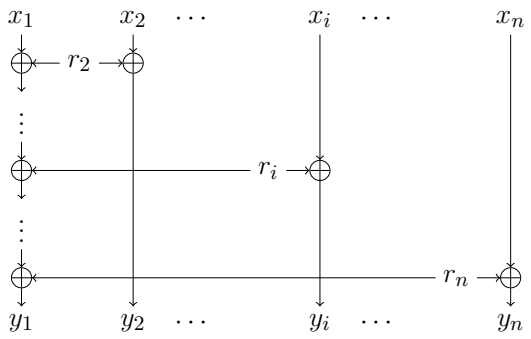


Fig. 4. The RefreshMasks algorithm, with the randoms r_i accumulated on the first column.

The following lemma shows that RefreshMasks achieves the $(n-1)$-NI property in a straightforward way. Namely it is easy to show that any t_c probes in RefreshMasks can be perfectly simulated from the knowledge of $x_{|I}$, with $|I| \leq t_c$.

Lemma 3 (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 $t < n$ intermediate variables, there exists a subset I of input indices such that the t_c intermediate variables can be perfectly simulated from $x_{|I}$, with $|I| \leq t_c$.*

The following lemma is a refinement of the basic t-NI property of RefreshMasks; namely it shows that a slightly better bound on $|I|$ can be obtained if we assume that y_1 is among the probed variables. Note that in this paper we consider a RefreshMasks algorithm where the randoms are accumulated on x_1, instead of x_n in [Cor17c], and therefore we must assume that y_1 is among the probed variables, instead of y_n. The improved bound will be required to prove the t-SNI property of Algorithm 1. Namely the adversary can probe for example t_c of the variables $z_i = \Psi(y_1, y_{i+1})$, whose simulation then requires the knowledge of $t_c + 1$ variables y_i; thanks to the stronger bound, since y_1 is always among the variables to be simulated, this requires the knowledge of only t_c input shares x_i (instead of $t_c + 1$), as required for the t-SNI bound.

Lemma 4 ([Cor17c]). *Let $x_1, \ldots, x_n$ be the input of a* RefreshMasks *where the randoms are accumulated on x_1, and let $y_1, \ldots, y_n$ be the output. Let t_c be the number of probed variables, with $t_c < n$. If y_1 is among the probed variables, then there exists a subset I such that all probed variables can be perfectly simulated from $x_{|I}$, with $|I| \leq t_c - 1$.*

The following lemma is similar, except that we consider an adversary with $t_c = n$ probes, instead of $t_c < n$. In that case the adversary could probe all y_i's and learn $x_1 \oplus \cdots \oplus x_n = y_1 \oplus \cdots \oplus y_n$; as shown in Lemma 2, if only the y_i's are probed, then all y_i's can be perfectly simulated from the knowledge of $x_1 \oplus \cdots \oplus x_n$ only, without the knowledge of the individual x_i's. The following lemma shows that this is essentially the best that the adversary can do, when we assume that y_1 is among the probed variables; in that case, either all n probes in the circuit can be simulated from $x_1 \oplus \cdots \oplus x_n$ only, or they can be simulated from $x_{|I}$ with the improved bound $|I| \leq n - 1$. As explained in [Cor17c], the restriction on probing y_1 is necessary, otherwise one could probe the n input shares x_i directly, and learn the value of the individual shares x_i (and not only their xor).

Lemma 5 ([Cor17c]). *Let $x_1, \ldots, x_n$ be the input of a* RefreshMasks *where the randoms are accumulated on x_1, and let $y_1, \ldots, y_n$ be the output. Let t_c be the number of probed variables, with $t_c = n$. If y_1 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 $x_{|I}$.*

Finally, we prove the following additional lemma for the proof of Theorem 2. We show that if the adversary probes exactly one intermediate variable z in the RefreshMasks algorithm except y_1, then this variable can be simulated with at most one input x_i, and moreover y_1 has the uniform distribution given z. In Section 6 we will describe a formal verification of the lemma.

Lemma 6. *Let $x_1, \ldots, x_n$ be the input of* RefreshMasks *and let $y_1, \ldots, y_n$ be the output, for $n \geq 3$. Consider a single probe z on any intermediate variable of the circuit except y_1. The distribution of z can be simulated from $x_{|I}$ with $|I| \leq 1$, and given z the distribution of y_1 is uniform.*

Proof. We consider the four possible cases for the probe z in RefreshMasks:

- if the probe is an input variable $z = x_i$, then we set $I = \{i\}$ and the variable x_i can be perfectly simulated from $x_{|I}$ with $|I| \leq 1$. Furthermore, from Lemma 1, the variable y_1 is uniformly distributed given z.
- if the probe is a random variable $z = r_i$, then we can perfectly simulate the variable r_i by a random value as in the real circuit, and we can take $I = \emptyset$. Furthermore, since we have $n \geq 3$, at least another random r_j has not been probed, and therefore the value y_1 has the uniform distribution given z.
- if the probe is an output variable $z = y_i$ (recall that y_1 is not probed), then one can apply Lemma 1 since $n \geq 3$ and therefore $n - 1 \geq 2$, which ensures that the two variables y_1 and y_i are uniformly and independently distributed. As a consequence, the variable $z = y_i$ can be perfectly simulated by a random value, with $I = \emptyset$, and the distribution of y_1 is uniform given z.
- if the probe is an intermediate variable $z = y_{1,i} = x_1 \oplus r_2 \oplus \cdots \oplus r_i$ with $2 \leq i < n$ (since by assumption the probe cannot be $y_{1,n} = y_1$), then because the random r_i has not been probed, the variable $y_{1,i}$ can be perfectly simulated by a random value, with $I = \emptyset$. Furthermore, since $i < n$ and the random r_n has not been probed, we deduce that y_1 has the uniform distribution given z.

Thus we have shown that if the probe z is not y_1, then z can be simulated from $x_{|I}$ with $|I| \leq 1$, and given z, the value y_1 has the uniform distribution. This concludes the proof of Lemma 6. □

5.2 Security of the Recursive Algorithm $\mathcal{C}_n$

In order to prove Theorem 2, it suffices to show that the recursive $\mathcal{C}_n$ algorithm (Algorithm 2) has the $(n-1)$-SNI property. As illustrated in Fig. 5, the algorithm is recursive, and therefore our security proof will be also recursive, *i.e.* the $(n-1)$-SNI property of $\mathcal{C}_n$ will be proven assuming the $(n-2)$-SNI property of $\mathcal{C}_{n-1}$.

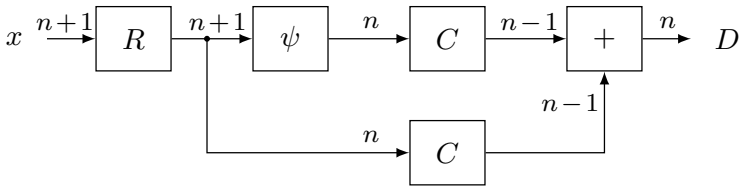


Fig. 5. The sequence of gadgets in $\mathcal{C}_n$.

Globally, our security proof is relatively similar to the security proof in [Cor17c]. However, to make our recursive proof work, we must prove a stronger property for $\mathcal{C}_n$. Namely when considering the $(n-1)$-SNI property of $\mathcal{C}_n$, the adversary has $n-1$ probes at his disposal, so he could use all those $n-1$ probes inside one of the two recursive calls to $\mathcal{C}_{n-1}$ (see Fig. 5); in that case the $(n-2)$-SNI condition of $\mathcal{C}_{n-1}$ would not be satisfied (since in principle $\mathcal{C}_{n-1}$ can accommodate at most $n-2$ probes), and nothing could be said about the $(n-1)$-SNI property of $\mathcal{C}_n$. Therefore we must prove recursively an additional property of $\mathcal{C}_n$, similar to the property of RefreshMasks in Lemma 5: when the adversary has n probes (which corresponds to $n-1$ probes when considering $\mathcal{C}_{n-1}$), those n probes can be simulated either from the xor of the input shares x_i, or from $x_{|I}$ with $|I| \leq n$ (remember that $\mathcal{C}_n$ has $n+1$ input shares). Formally, we prove the following lemma.

Lemma 7 (Security of $\mathcal{C}_n$). *Let $(x_i)_{1\leq i\leq n+1}$ be the input and let $(D_i)_{1\leq i\leq n}$ be the output of the Boolean to arithmetic conversion algorithm $\mathcal{C}_n$. For any set of t_c intermediate variables and any subset $O \subset [1,n]$ with $t_c + |O| < n$, there exists a subset I of input indices such that the t_c intermediate variables as well as $D_{|O}$ can be perfectly simulated from $x_{|I}$, with $|I| \leq t_c$. Moreover, if $t_c + |O| = n$, either there exists a subset I of input indices such that the t_c intermediate variables and $D_{|O}$ can be simulated from $x_{|I}$ with $|I| \leq n$, or those variables can be simulated from $x_1 \oplus \cdots \oplus x_{n+1}$ only.*

We prove Lemma 7 recursively, following the same process as in [BBD+16, Sect. 4.1], where the t-SNI security of a construction is deduced from the t-NI or t-SNI property of its component gadgets. The sequence of gadgets of our construction is illustrated in Figure 6. The gadgets are numbered from 1 to 5, and we denote by $\mathcal{I}^i$ the set of probed variables in Gadget i. Gadget 5 corresponds to the initial Refreshmasks at Step 1, which we denote by R. Gadget 4 is denoted by Ψ for the application of the Ψ function at Step 2. Gadgets 2 and 3 denote the recursive application of the conversion algorithm at Step 3. Finally, Gadget 1 denotes the additive grouping performed at Step 4.

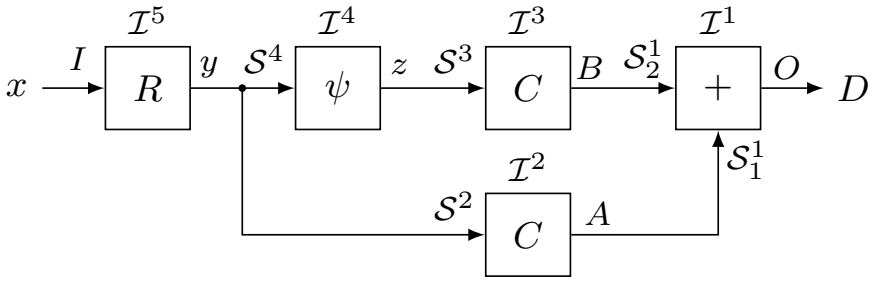


Fig. 6. The sequence of gadgets in the Boolean to arithmetic conversion algorithm.

It is easy to see that Lemma 7 is verified for $n = 1$, where only $D_1 = x_1 \oplus x_2$ is computed. Namely the first condition is $t_c + |O| < 1$, which implies $t_c = |O| = 0$ and we can take $I = \emptyset$. Secondly, when considering $n = 1$ probe, if x_i is probed then we can let $I = \{i\}$, and if $x_1 \oplus x_2$ is probed, it can be simulated from the knowledge of $x_1 \oplus x_2$ only.

Now we show the induction step. We denote by t_c the total number of probes in the circuit, and we let $O \subset [1, n]$ be any subset of output shares. We must prove the following two properties from Lemma 7:

- $\mathcal{P}_1(n)$: if $t_c + |O| < n$, then all t_c probed variables in the circuit and all variables $D_{|O}$ can be perfectly simulated from $x_{|I}$, for some subset I satisfying $|I| \leq t_c$;
- $\mathcal{P}_2(n)$: if $t_c + |O| = n$, then either there exists a subset I of input indices such that the n variables can be simulated from $x_{|I}$ with $|I| \leq n$, or the n variables can be simulated from $x_1 \oplus \cdots \oplus x_{n+1}$ only.

The two properties $\mathcal{P}_1(n)$ and $\mathcal{P}_2(n)$ will be proven by assuming that $\mathcal{P}_1(n-1)$ and $\mathcal{P}_2(n-1)$ are satisfied. As explained previously, $\mathcal{P}_1(1)$ and $\mathcal{P}_2(1)$ are trivially satisfied.

Property $\mathcal{P}_1(n)$. We first consider the property $\mathcal{P}_1(n)$ and therefore we assume:

$$t_c + |O| < n$$

As in [BBD+16], the technique consists in starting from the output variables $D_{|O}$, which corresponds to Gadget 1, and showing for each successive gadget that any subset of the output variables of the gadget can be simulated from a subset of its input variables; that is, we proceed in reverse order back to the input shares of the full algorithm.

Gadget 1. We have $D_i = A_i + B_i$ for $1 \leq i \leq n-2$, and $D_{n-1} = A_{n-1}$ and $D_n = B_{n-1}$. For simplicity, to avoid a change of index, we denote the last wire of the B_i's by B_n instead of B_{n-1}, so that we can write $D_n = B_n$. We denote by P_1 the set of probed indices in Gadget 1, with $|P_1| \leq |\mathcal{I}^1|$. To simulate D_i for $1 \leq i \leq n-2$, we must know both A_i and B_i; to simulate D_{n-1} we must know A_{n-1} and to simulate D_n we must know B_n. The simulation of Gadget 1 can therefore be performed from the shares $A_{|\mathcal{S}_1^1}$ and $B_{|\mathcal{S}_2^1}$, where the subsets $\mathcal{S}_1^1$ and $\mathcal{S}_2^1$ are defined as follows:

$$\mathcal{S}_1^1 = \big(O \cup P_1\big) \cap [1, n-1] \tag{3}$$
$$\mathcal{S}_2^1 = \big(O \cup P_1\big) \cap \big([1, n-2] \cup \{n\}\big) \tag{4}$$

We obtain the upper bound:

$$\begin{aligned}|\mathcal{S}_1^1| + |\mathcal{S}_2^1| = |\mathcal{S}_1^1 \cup \mathcal{S}_2^1| + |\mathcal{S}_1^1 \cap \mathcal{S}_2^1| = |O \cup P_1| + |(O \cup P_1) \cap [1, n-2]| \\ \leq |O| + |P_1| + n - 2 \leq |O| + |\mathcal{I}^1| + n - 2 \end{aligned} \tag{5}$$

Gadgets 2 and 3. The gadgets 2 and 3 are recursive applications of the Boolean to arithmetic conversion, with $n-1$ shares. The t-SNI conditions for gadgets 2 and 3 correspond to property $\mathcal{P}_1(n-1)$ and are therefore respectively:

$$|\mathcal{S}_1^1| + |\mathcal{I}^2| < n - 1, \quad |\mathcal{S}_2^1| + |\mathcal{I}^3| < n - 1 \tag{6}$$

We highlight that both conditions in (6) are not necessarily simultaneously satisfied. Indeed, the adversary could probe the $n-1$ shares A_i's which would give $|\mathcal{S}_1^1| = n-1$. However, we show that at least one of the two conditions in (6) must be satisfied. Indeed, if none of the two conditions is satisfied, we have $|\mathcal{S}_1^1| + |\mathcal{I}^2| \geq n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| \geq n-1$, which would give $|\mathcal{S}_1^1| + |\mathcal{S}_2^1| + |\mathcal{I}^2| + |\mathcal{I}^3| \geq 2n-2$. However, from (5) and $|\mathcal{I}^1| + |\mathcal{I}^2| + |\mathcal{I}^3| \leq t_c$, with $t_c + |O| < n$, we get:

$$\begin{aligned}|\mathcal{S}_1^1| + |\mathcal{S}_2^1| + |\mathcal{I}^2| + |\mathcal{I}^3| &\leq |O| + |\mathcal{I}^1| + n - 2 + |\mathcal{I}^2| + |\mathcal{I}^3| \\ &\leq t_c + |O| + n - 2 < 2n - 2\end{aligned}$$

which contradicts the previous inequality. Therefore, at least one of the t-SNI conditions for gadgets 2 and 3 must be satisfied. We now distinguish three cases depending on which of the two conditions in (6) is satisfied:

- Case $|\mathcal{S}_1^1| + |\mathcal{I}^2| < n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| \geq n-1$. Since we have $|\mathcal{S}_2^1| \leq |O| + |\mathcal{I}^1|$ from (4) and from the condition $t_c + |O| < n$, we obtain:

  $$n - 1 \leq |\mathcal{S}_2^1| + |\mathcal{I}^3| \leq |O| + |\mathcal{I}^1| + |\mathcal{I}^3| \leq |O| + t_c < n$$

  which gives $|\mathcal{S}_2^1| + |\mathcal{I}^3| = |O| + |\mathcal{I}^1| + |\mathcal{I}^3| = |O| + t_c = n-1$, and therefore $|\mathcal{I}^1| + |\mathcal{I}^3| = t_c$. This implies that there are no other probes in the circuit, that is, we must have $|\mathcal{I}^i| = 0$ for $i = 2, 4, 5$. In particular, since we have $|\mathcal{I}^2| = 0$, and the t-SNI condition for Gadget 2 is satisfied, we can apply the recursive property $\mathcal{P}_1(n-1)$ and simulate all outputs of Gadget 2 with $|\mathcal{S}^2| \leq |\mathcal{I}^2| = 0$, which means that we can take $\mathcal{S}^2 = \emptyset$.
  However, since the t-SNI condition for Gadget 3 is not satisfied, we cannot apply $\mathcal{P}_1(n-1)$, but we can still use $\mathcal{P}_2(n-1)$, because $|\mathcal{S}_2^1| + |\mathcal{I}^3| = n-1$. This ensures that the $n-1$ corresponding variables can either be simulated from $z_{|\mathcal{S}^3}$ with $|\mathcal{S}^3| \leq n-1$, or from $z_1 \oplus \cdots \oplus z_n$ only. In the following, we deal with both cases:
  - if the $n-1$ variables can be simulated from $z_{|\mathcal{S}^3}$ with $|\mathcal{S}^3| \leq n-1$, we obtain $|\mathcal{S}^4| \leq n$. Thanks to the initial RefreshMasks (Gadget 5) which is not probed in that case, one can apply Lemma 1 which ensures that the n (out of $n+1$) variables to be simulated are uniformly and independently distributed. Therefore they can be perfectly simulated by a random value. Moreover since $\mathcal{S}^2 = \emptyset$, those n variables are the only ones that must be simulated. Hence, we can take $I = \emptyset$.

- if the $n-1$ variables can be simulated from the knowledge of $z_1 \oplus \cdots \oplus z_n$ only, since by definition we have

$$z_1 \oplus \cdots \oplus z_n = \Psi(y_1, y_2 \oplus \ldots \oplus y_{n+1}) = y_1 \oplus \cdots \oplus y_{n+1} - y_2 \oplus \ldots \oplus y_{n+1}$$

and since we also have $x = x_1 \oplus \cdots \oplus x_{n+1} = y_1 \oplus \cdots \oplus y_{n+1}$, we get:

$$z_1 \oplus \cdots \oplus z_n = x - x \oplus y_1$$

Moreover since $\mathcal{S}^2 = \emptyset$ and $\mathcal{I}^4 = \emptyset$, this is the only variable that must be simulated. Thanks to the initial RefreshMasks (Gadget 5) which is not probed in that case, this has the uniform distribution (because y_1 has the uniform distribution from Lemma 1), and this can therefore be perfectly simulated by a random value. Therefore we can take $I = \emptyset$.

- Case $|\mathcal{S}_1^1| + |\mathcal{I}^2| \geq n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| < n-1$. The reasoning is similar to the previous case. As previously we have $|\mathcal{I}^1| + |\mathcal{I}^2| = t_c$ and there are no other probes in the circuit, that is, we have $|\mathcal{I}^i| = 0$ for $i = 3, 4, 5$. Since the t-SNI condition of Gadget 3 is satisfied and Gadget 3 is not probed, we can simulate all outputs of Gadget 3 with $\mathcal{S}^3 = \emptyset$, and therefore we can take $\mathcal{S}^4 = \emptyset$. Since the t-SNI condition for Gadget 2 is not satisfied but $|\mathcal{S}_1^1| + |\mathcal{I}^2| = n-1$, we can apply $\mathcal{P}_2(n-1)$ to Gadget 2. This ensures that the $n-1$ corresponding variables can either be simulated from $y_{|\mathcal{S}^2}$ with $|\mathcal{S}^2| \leq n-1$, or from $y_2 \oplus \cdots \oplus y_{n+1}$ only. We deal with both cases similarly as before:
  - if the $n-1$ variables can be simulated from $y_{|\mathcal{S}^2}$ with $|\mathcal{S}^2| \leq n-1$, thanks to the initial RefreshMasks (Gadget 5) which is not probed, Lemma 1 ensures that the $n-1$ variables are uniformly and independently distributed. Therefore they can be perfectly simulated by a random value, and we can take $I = \emptyset$ for the simulation of the entire circuit.
  - if the $n-1$ variables can be simulated from the knowledge of $y_2 \oplus \cdots \oplus y_{n+1} = x \oplus y_1$ only, thanks to the initial RefreshMasks (Gadget 5) which is not probed, this has the uniform distribution (because y_1 has the uniform distribution from Lemma 1), and this can therefore be perfectly simulated by a random value. Therefore we can take $I = \emptyset$ for the simulation of the entire circuit.
- Case $|\mathcal{S}_1^1| + |\mathcal{I}^2| < n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| < n-1$. In this case, the t-SNI condition is satisfied for both gadgets 2 and 3. Therefore, we obtain from the recursive hypothesis $\mathcal{P}_1(n-1)$ that the intermediate variables of gadgets 2 and 3 and the output variables $A_{|\mathcal{S}_1^1}$ and $B_{|\mathcal{S}_2^1}$ can be perfectly simulated from $y_{|\mathcal{S}^2}$ and $z_{|\mathcal{S}^3}$, where:

$$|\mathcal{S}^2| \leq |\mathcal{I}^2| \text{ and } |\mathcal{S}^3| \leq |\mathcal{I}^3| \tag{7}$$

In the sequel, we can assume that (7) is satisfied, since in the two other cases (when only one of the two t-SNI conditions is satisfied), as explained previously the simulation of all probed variables and $D_{|O}$ can be performed with $I = \emptyset$.

Gadget 4. By definition of the Ψ function, we have $z_1 \leftarrow (\overline{n \wedge 1}) \cdot y_1 \oplus \Psi(y_1, y_2)$ and $z_i \leftarrow \Psi(y_1, y_{i+1})$ for all $2 \leq i \leq n$. Therefore, the probed variables and the output variables $z_{|\mathcal{S}^3}$ can be simulated with the knowledge of $y_{|\mathcal{S}^4}$, where $|\mathcal{S}^4|$ is such that:

$$|\mathcal{S}^4| \leq |\mathcal{I}^4| + |\mathcal{S}^3| + 1 \tag{8}$$

Note that since the variable y_1 is involved during the computation of all the z_i's we must have $1 \in \mathcal{S}^4$.

Gadget 5. Let $t' = |\mathcal{S}^4| + |\mathcal{S}^2| + |\mathcal{I}^5|$ be the total number of variables within the initial RefreshMasks that must be simulated (see Fig. 6). By using (7) and (8), we get

$$\begin{aligned} t' = |\mathcal{S}^4| + |\mathcal{S}^2| + |\mathcal{I}^5| &\leq |\mathcal{I}^4| + |\mathcal{S}^3| + 1 + |\mathcal{I}^2| + |\mathcal{I}^5| \\ &\leq |\mathcal{I}^2| + |\mathcal{I}^3| + |\mathcal{I}^4| + |\mathcal{I}^5| + 1 \leq t_c + 1 < n + 1 \end{aligned}$$

Therefore we have $t' < n + 1$ and since $1 \in \mathcal{S}^4$, we can apply Lemma 4 where y_1 is among the t' probes variables (note that the initial RefreshMasks has $n + 1$ inputs and outputs, so we must apply Lemma 4 with $n + 1$ instead of n). We obtain that all variables can be perfectly simulated from $x_{|I}$, where:

$$|I| \leq t' - 1 \leq t_c + 1 - 1 \leq t_c$$

In summary all t_c probed variables in the circuit and all output variables $D_{|O}$ can be perfectly simulated from $x_{|I}$ with $|I| \leq t_c$. Hence the $(n-1)$-SNI property is satisfied for $\mathcal{C}_n$, which concludes the recursive proof of the first property $\mathcal{P}_1(n)$. Note that for the recursive proof of $\mathcal{P}_1(n)$ we needed both properties $\mathcal{P}_1(n-1)$ and $\mathcal{P}_2(n-1)$, that is we cannot prove the two properties $\mathcal{P}_1$ and $\mathcal{P}_2$ separately.

Property $\mathcal{P}_2(n)$. We now consider the second property $\mathcal{P}_2(n)$, assuming that there are $t_c + |O| = n$ variables to be simulated in the circuit. We must show that either there exists a subset I of input indices such that those n variables can be simulated from $x_{|I}$ with $|I| \leq n$ (remember that the $\mathcal{C}_n$ algorithm takes $n + 1$ shares as input), or the n variables can be simulated from $x_1 \oplus \cdots \oplus x_{n+1}$ only.

We first distinguish two cases depending on whether the RefreshMasks (Gadget 5) is probed or not. If RefreshMasks is not probed, then from Lemma 2 we can simulate all the variables y_i from the knowledge of $x_1 \oplus \cdots \oplus x_{n+1}$ only. Then the rest of the circuit can be simulated as in the real circuit, and property $\mathcal{P}_2(n)$ is satisfied. We now assume that RefreshMasks is probed, that is $|\mathcal{I}^5| \geq 1$.

Gadget 1. As in the $\mathcal{P}_1(n)$ property, we obtain:

$$\mathcal{S}_1^1 = \big(O \cup P_1\big) \cap [1, n-1] \tag{9}$$

$$\mathcal{S}_2^1 = \big(O \cup P_1\big) \cap \big([1, n-2] \cup \{n\}\big) \tag{10}$$

Gadgets 2 and 3. As for $\mathcal{P}_1(n)$, we start by showing that at least one of the two gadgets must satisfy the t-SNI condition, that is, we cannot have $|\mathcal{S}_1^1| + |\mathcal{I}^2| \geq n - 1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| \geq n - 1$. Namely from $|\mathcal{I}^5| \geq 1$ and $|\mathcal{I}^1| + |\mathcal{I}^2| + |\mathcal{I}^3| + |\mathcal{I}^5| \leq t_c$, we obtain:

$$|\mathcal{I}^1| + |\mathcal{I}^2| + |\mathcal{I}^3| \leq t_c - 1$$

From (5) and using the condition $t_c + |O| = n$, we get:

$$\begin{aligned} |\mathcal{S}_1^1| + |\mathcal{S}_2^1| + |\mathcal{I}^2| + |\mathcal{I}^3| &\leq |O| + |\mathcal{I}^1| + n - 2 + |\mathcal{I}^2| + |\mathcal{I}^3| \leq t_c - 1 + |O| + n - 2 \\ &\leq t_c + |O| + n - 3 \leq 2n - 3 \end{aligned}$$

and therefore as previously we cannot have both $|\mathcal{S}_1^1| + |\mathcal{I}^2| \geq n - 1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| \geq n - 1$. Therefore, at least one of the t-SNI conditions for gadgets 2 and 3 must be satisfied. As before, we deal with the three possible cases, depending on whether one or both conditions are satisfied:

- Case $|\mathcal{S}_1^1| + |\mathcal{I}^2| < n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| \geq n-1$. From (10), we know that $|\mathcal{S}_2^1| \leq |O| + |\mathcal{I}^1|$ and since we have $|\mathcal{I}^1| + |\mathcal{I}^3| \leq t_c - 1$ and $t_c + |O| = n$, we get:

$$n-1 \leq |\mathcal{S}_2^1| + |\mathcal{I}^3| \leq |O| + |\mathcal{I}^1| + |\mathcal{I}^3| \leq |O| + t_c - 1 \leq n-1$$

  which gives $|O| + |\mathcal{I}^1| + |\mathcal{I}^3| = |O| + t_c - 1$, and therefore $|\mathcal{I}^1| + |\mathcal{I}^3| = t_c - 1$. Since $|\mathcal{I}^5| \geq 1$ and $\sum_{i=1}^{5} |\mathcal{I}^i| = t_c$, we deduce that $|\mathcal{I}^5| = 1$, and that there are no other probes in the circuit, *i.e.* $|\mathcal{I}^2| = |\mathcal{I}^4| = 0$. As a consequence, since the t-SNI condition for Gadget 2 is satisfied, we can apply the recursive property $\mathcal{P}_1(n-1)$ and simulate all outputs of Gadget 2 with $|\mathcal{S}^2| \leq |\mathcal{I}^2| = 0$, hence we can take $\mathcal{S}^2 = \emptyset$.
  However, since the t-SNI condition for Gadget 3 is not satisfied, we cannot apply $\mathcal{P}_1(n-1)$, but we can still use $\mathcal{P}_2(n-1)$, because $|\mathcal{S}_2^1| + |\mathcal{I}^3| = n-1$. This ensures that the $n-1$ corresponding variables can either be simulated from $z_{|\mathcal{S}^3}$ with $|\mathcal{S}^3| \leq n-1$, or from $z_1 \oplus \cdots \oplus z_n$ only. In the following, we deal with both cases:

  - if the $n-1$ variables can be simulated from $z_{|\mathcal{S}^3}$ with $|\mathcal{S}^3| \leq n-1$, then one can apply the same reasoning as for Gadget 4 when considering the property $\mathcal{P}_1(n)$ and we obtain:

$$|\mathcal{S}^4| \leq |\mathcal{I}^4| + |\mathcal{S}^3| + 1 \leq 0 + (n-1) + 1 \leq n$$

    with $1 \in \mathcal{S}^4$. Therefore, by using that $|\mathcal{I}^5| = 1$ and $|\mathcal{S}^2| = 0$, the total number of variables t' within the initial RefreshMasks that must be simulated is such that

$$t' = |\mathcal{S}^4| + |\mathcal{S}^2| + |\mathcal{I}^5| \leq n+1$$

    As a consequence, one can apply Lemma 5 on the initial RefreshMasks with $n+1$ inputs and $n+1$ probed variables, one of whom being y_1. Then either all probed variables can be perfectly simulated from $x_1 \oplus \cdots \oplus x_{n+1}$, or there exists a subset I with $|I| \leq n$ such that they can be perfectly simulated from $x_{|I}$; this is exactly the property $\mathcal{P}_2(n)$.
  - if the $n-1$ variables can be simulated from the knowledge of $z_1 \oplus \cdots \oplus z_n = x - x \oplus y_1$ only, we proceed as follows. Recall that there is a single probed variable in RefreshMasks. For simplicity we assume that this probed variable is not y_1, because we can consider that the probing of y_1 is rather done in $\mathcal{I}^4$. We can therefore apply Lemma 6, which shows that z can be simulated from $x_{|I}$ with $|I| \leq 1$, and given z, the distribution of y_1 is uniform; thus the distribution of $x - x \oplus y_1$ is also uniform (since x is fixed). Therefore one can perfectly simulate $z_1 \oplus \cdots \oplus z_n = x - x \oplus y_1$ by a random value, and the remaining variable z can be perfectly simulated from $x_{|I}$ with $|I| \leq 1 \leq n$.

- Case $|\mathcal{S}_1^1| + |\mathcal{I}^2| \geq n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| < n-1$. The reasoning is similar to the previous case, with $|\mathcal{I}^1| + |\mathcal{I}^2| = t-1$, $|\mathcal{I}^5| = 1$ and $|\mathcal{I}^3| = |\mathcal{I}^4| = 0$. Since the t-SNI condition of Gadget 3 is satisfied and Gadget 3 is not probed, we can simulate all outputs of Gadget 3 with $\mathcal{S}^3 = \emptyset$, and therefore we can take $\mathcal{S}^4 = \emptyset$. Since the t-SNI condition for Gadget 2 is not satisfied but $|\mathcal{S}_1^1| + |\mathcal{I}^2| = n-1$, we can apply $\mathcal{P}_2(n-1)$. This ensures that the $n-1$ corresponding variables can either be simulated from $y_{|\mathcal{S}^2}$ with $|\mathcal{S}^2| \leq n-1$, or from $y_2 \oplus \cdots \oplus y_{n+1}$ only. We deal with both cases similarly as before:

  - if the $n-1$ variables can be simulated from $y_{|\mathcal{S}^2}$ with $|\mathcal{S}^2| \leq n-1$, then one can apply the basic t-NI property of RefreshMasks from Lemma 3, which gives $|I| \leq |\mathcal{S}^2| + |\mathcal{I}^5| \leq n-1+1 \leq n$ as required in $\mathcal{P}_2(n)$.

- if the $n-1$ variables can be simulated from the knowledge of $y_2 \oplus \cdots \oplus y_{n+1} = x \oplus y_1$ only, then we can apply Lemma 6 as in the previous case. We obtain that all probed variables can be perfectly simulated from $x_{|I}$ with $|I| \leq 1 \leq n$, hence $\mathcal{P}_2(n)$ is satisfied.
- Case $|\mathcal{S}_1^1| + |\mathcal{I}^2| < n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| < n-1$. In this case, the t-SNI condition is satisfied for both gadgets 2 and 3. Therefore, we obtain from the recursive hypothesis $\mathcal{P}_1(n-1)$ that:

$$|\mathcal{S}^2| \leq |\mathcal{I}^2| \text{ and } |\mathcal{S}^3| \leq |\mathcal{I}^3| \tag{11}$$

In the sequel, we can assume that (11) is satisfied, since in the two other cases (when only one of both t-SNI conditions is satisfied), the simulation of all probed variables and $D_{|O}$ is already performed.

Gadget 4. As in the proof of $\mathcal{P}_1(n)$, by definition of the Ψ function, we have that the probed variables and the output variables $z_{|\mathcal{S}^3}$ can be simulated with the knowledge of $y_{|\mathcal{S}^4}$, where $1 \in \mathcal{S}^4$ and:

$$|\mathcal{S}^4| \leq |\mathcal{I}^4| + |\mathcal{S}^3| + 1 \leq |\mathcal{I}^4| + |\mathcal{I}^3| + 1 \tag{12}$$

Gadget 5. Let $t' = |\mathcal{S}^4| + |\mathcal{S}^2| + |\mathcal{I}^5|$ be the total number of variables within the initial RefreshMasks that must be simulated. By using (11) and (12), we get

$$t' = |\mathcal{S}^4| + |\mathcal{S}^2| + |\mathcal{I}^5| \leq |\mathcal{I}^4| + |\mathcal{I}^3| + 1 + |\mathcal{I}^2| + |\mathcal{I}^5| \leq t_c + 1 \leq n + 1$$

Since $1 \in \mathcal{S}^4$, we can apply Lemma 5 with $n+1$ inputs and $n+1$ probed variables, one of whom being y_1. This allows to deduce that all t_c probed variables in the circuit and all output variables $D_{|O}$ can be perfectly simulated either from $x_1 \oplus \cdots \oplus x_{n+1}$, or from $x_{|I}$ with $|I| \leq n$, and the property $\mathcal{P}_2(n)$ is satisfied.

This concludes the recursive proof of the second property $\mathcal{P}_2(n)$. Thus we have shown that both properties $\mathcal{P}_1(n)$ and $\mathcal{P}_2(n)$ in Lemma 7 are satisfied, which terminates the proof of Lemma 7.

□

5.3 Proof of Theorem 2

The proof of Theorem 2 is straightforward from Lemma 7. Namely Algorithm 1 consists in applying the $\mathcal{C}_n$ algorithm with $x_{n+1} = 0$. By applying the property $\mathcal{P}_1(n)$ of Lemma 7, we obtain that for any set of t_c intermediate variables and any subset $O \subset [1, n]$ with $t_c + |O| < n$, there exists a subset I' of input indices such that the t_c intermediate variables as well as $D_{|O}$ can be perfectly simulated from $x_{|I'}$, with $|I'| \leq t$. Since $x_{n+1} = 0$ it suffices to take $I = I' \cap [1, n]$ for the input of Algorithm 1, and we still have $|I| \leq t_c$ as required for the $(n-1)$-SNI property. This terminates the proof of Theorem 2.

6 Formal verification of Algorithm 1

We have performed a formal verification of our new countermeasure, using the CheckMasks tool described in [Cor17b]; the source code is publicly available [Cor17a]. More precisely, we first describe a *partial* formal verification of the countermeasure, focusing on the Boolean part of the algorithm. We then describe a *full* formal verification in which we consider both Boolean and arithmetic operations.

6.1 Partial formal verification

In this section, we perform a partial formal verification of our new countermeasure, focusing on the properties of RefreshMasks. More precisely, the correctness of lemmas 4 and 5 for the RefreshMasks algorithm has already been formally verified in [Cor17b], and we have performed a formal verification of our Lemma 6, following the same approach as in [Cor17b]. In other words, all the non-trivial security properties of RefreshMasks considered in Section 5.1 and used in the proof of Theorem 2 are formally verified, at least for small values of n.

The generic verification of the masking countermeasure was initiated by Barthe *et al.* in [BBD+15] based on the EasyCrypt framework. The approach consists in considering all possible t-tuples of intermediate variables, and proving that for each particular t-tuple the adversary learns nothing from the secret key. The authors obtained 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 this paper we have used the CheckMasks tool described in [Cor17b]; the source code of the CheckMasks library is publicly available at [Cor17a], under the GPL v2.0 license. The circuit to be verified is represented as nested lists, which leads to a simple and concise implementation in Common Lisp, a programming language well suited to formal manipulations. In [Cor17b] a formal verification of the security of the Rivain-Prouff multiplication is described, with running times similar to those in [BBD+15]; additionally a formal verification of lemmas 4 and 5 for the RefreshMasks algorithm is described.

For the formal verification of Lemma 6, the approach consists in considering all possible intermediate variables z in the RefreshMasks circuit (including the output variables but not y_1), and checking that z can be simulated from at most one variable x_i, and moreover the variable y_1 is uniformly distributed given z. In principle, when applying a generic verification, the running time is exponential in n, because all possible t-tuples of intermediate variables must be considered, with $t = n - 1$; this implies that in general such generic verification can only be performed for small values of n. However in the particular case of Lemma 6, there is only a single intermediate variable z to consider (instead of a t-tuple), and the formal verification is therefore polynomial in n, and quite efficient in practice, so it can be performed for any n. For example, for $n = 3$ we obtain:

```
> (check-refreshmasks-oneprobe 3)
In: (X1 X2 X3)
Out: ((+ R2 (+ R1 X1)) (+ R2 X2) (+ R1 X3))
Number of intermediate variables: 8
T
```

which means that for all intermediate variables z of RefreshMasks (except y_1), the formal tool has verified that z can be simulated from some x_i, and moreover y_1 is uniformly distributed given z. We have run the verification algorithm up to $n = 10$. The source code is publicly available at [Cor17a].

Using the same tool, we can also quickly verify that Lemma 6 does not hold if the randoms in RefreshMasks are accumulated on x_n instead of x_1. More precisely, with the modified RefreshMasks for $n = 3$, we obtain:

```
> (check-refreshmasks-oneprobe 3 :rev 't)
In: (X1 X2 X3)
Out: ((+ R1 X1) (+ R2 X2) (+ R2 (+ R1 X3)))
Number of intermediate variables: 8
Failure: (R1 (+ R1 X1))
```

NIL

Namely, when the randoms are accumulated on x_3, we have $y_1 = x_1 \oplus r_1$, $y_2 = x_2 \oplus r_2$ and $y_3 = x_3 \oplus r_1 \oplus r_2$, and the formal tool has correctly identified that by probing $z = r_1$, the variable $y_1 = x_1 \oplus r_1$ is not uniformly distributed.

6.2 Full formal verification

In the previous section we have described a formal verification of the security properties of RefreshMasks that are required for the security proof of the Boolean to arithmetic conversion algorithm. 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.

In [Cor17b], it was shown how to extend [BBD+15] to handle a combination of arithmetic and Boolean operations, with an application to the full formal verification of the Boolean to arithmetic conversion algorithm from CHES 2017 [Cor17c]. In this section we first recall this approach, and we show that exactly the same verification tool can be used to formally verify our new Boolean to arithmetic conversion algorithm, moreover with better timings because our new conversion algorithm is simpler.

In order to verify the $(n-1)$-SNI property of the Boolean to arithmetic algorithm, we must check that for all possible $(n-1)$-tuples of intermediate variables (including the outputs D_i), the number of input variables x_i's that remain after the application of a sequence of simplification rules is always $\leq t_c$, where t_c is the number of non-output variables in the $(n-1)$-tuple.

For the verification of Boolean circuits in the previous section, we used only a single simplification rule, namely replacing $x \oplus r$ by r when the random r appears only once in the intermediate variables. For the verification of circuits combining Boolean and arithmetic operations, we use the same sequence of simplification rules as in [Cor17b]:

- Rule 1: when $\omega = x_1 + x_2 \bmod 2^k$ must be simulated, simulate both x_1 and x_2.
- Rule 2: from the affine property of the function Ψ, replace $\Psi(x, y) \oplus \Psi(x, z)$ by $x \oplus \Psi(x, y \oplus z)$.
- Rule 3: from the definition of Ψ, replace $\Psi(x, y)$ by $(x \oplus y) - y \bmod 2^k$.
- 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 $x - r \bmod 2^k$.
- Rule 5: when a random r is not used in two intermediate variables e_1 and e_2, replace the simulation of $(e_1 \oplus r, e_2 \oplus r)$ by the simulation of $(r, (e_1 \oplus r) \oplus e_2)$; this corresponds to the change of variable $r' = e_1 \oplus r$.
- Rule 6: when $\Psi(x_1, x_2)$ must be simulated, simulate both x_1 and x_2.

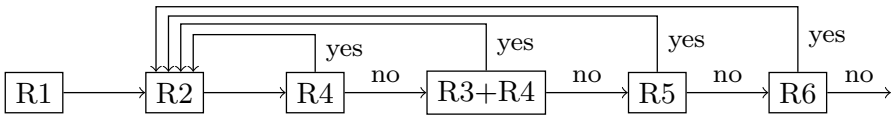


Fig. 7. The rule application strategy for the formal verification of Boolean to arithmetic conversion.

As explained in [Cor17b], the ordering of the application of the rules matters; we used the same ordering strategy as in [Cor17b], as illustrated in Fig. 7. We summarize in tables 4 and 5 the timings of formal verification for the algorithms in [Cor17c] and this paper. Note that the Boolean to arithmetic conversion algorithm has complexity $\mathcal{O}(2^n)$, and therefore the number of possible $(n-1)$-tuples of intermediate variables is $\mathcal{O}(2^{n^2})$; that is why we could only perform the formal verification up to $n = 5$.

n	#var.	#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 4. Formal verification of the t-SNI property of the Boolean to arithmetic conversion algorithm from [Cor17c], for $t = n - 1$.

n	#var.	#tuples	Security	Time
2	14	14	✓	ε
3	39	741	✓	0.06 s
4	94	134,044	✓	30 s
5	207	74,303,685	✓	12 h

Table 5. Formal verification of the t-SNI property of the Boolean to arithmetic conversion algorithm from Section 4.1, for $t = n - 1$.

7 Implementation Results

In Section 4.2, we showed that the number of operations T_n of our Boolean to arithmetic conversion algorithm $\mathcal{C}_n$ is $T_n = 10 \cdot 2^n - 6 \cdot n - 13$, compared to $T_n = 14 \cdot 2^n - 12 \cdot n - 21$ in [Cor17c]. Therefore, the complexity of our algorithm is still exponential in n, but with an expected speed-up factor of 28.6% for large n, compared to [Cor17c]. Furthermore, the complexity is independent of the register size k, while the conversion algorithm from [CGV14] has complexity $\mathcal{O}(k \cdot n^2)$.

We have implemented our new algorithm, in C on an iMac running a 3.2 GHz Intel processor, using the Clang compiler; see

`https://pastebin.com/WKnNyEU8`

for the source code. We summarize the execution times in Table 6, using the same timings as in [Cor17c] for the CGV algorithm and the algorithm from [Cor17c]. The results from Table 6 are consistent with the operation count from Table 2; for small values of n our algorithm is at least one order of magnitude faster than [CGV14]. We also see that the higher the order t, the greater the speed-up factor compared to [Cor17c]. For $t = 2$ the running times are similar, while for $t = 4$ we get a 24% speed-up, and for $t = 6$ we get a 26% speed-up.

B → A conversion	Security order t							
	2	3	4	5	6	8	10	12
CGV, 32 bits [CGV14]	1 593	2 697	4 297	5 523	7 301	10 919	15 819	21 406
Coron [Cor17c]	45	119	281	611	1 270	5 673	22 192	87 322
Our algorithm (Section 4.1)	42	96	214	448	921	3 774	13 899	54 572

Table 6. Running time in μs for Boolean to arithmetic conversion algorithms, up to security order $t = 12$, with $n = t + 1$ shares. The implementation was done in C on a iMac running a 3.2 GHz Intel processor.

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`.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.

[CGTV15] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to boolean masking with logarithmic complexity. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 130–149, 2015.

[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-Sebastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. Cryptology ePrint Archive, Report 2017/879, 2017. `http://eprint.iacr.org/2017/879`.

[Cor17c] 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. Full version available at `http://eprint.iacr.org/2017/252`.

[CRRY99] Scott Contini, Ronald L. Rivest, Matthew J. B. Robshaw, and Yiqun Lisa Yin. Improved analysis of some simplified variants of RC6. In *FSE*, 1999.

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

[HT16] Michael Hutter and Michael Tunstall. Constant-time higher-order boolean-to-arithmetic masking. Cryptology ePrint Archive, Report 2016/1023, 2016. `http://eprint.iacr.org/2016/1023`. Version posted on 22-Dec-2016.

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

[LM90] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In *EUROCRYPT*, pages 389–404, 1990.

[NIS95] NIST. Secure hash standard. In *Federal Information Processing Standard, FIPA-180-1*, 1995.

[NW97] Roger M. Needham and David J. Wheeler. Tea extentions. In *Technical report, Computer Laboratory, University of Cambridge*, 1997.

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

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