

# High-Order Conversion from Boolean to Arithmetic Masking

Jean-Sébastien Coron<sup>(✉)</sup>

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

**Abstract.** Masking with random values is an effective countermeasure against side-channel attacks. For cryptographic algorithms combining arithmetic and Boolean masking, it is necessary to switch from arithmetic to Boolean masking and vice versa. Following a recent approach by Hutter and Tunstall, we describe a high-order Boolean to arithmetic conversion algorithm whose complexity is independent of the register size  $k$ . Our new algorithm is proven secure in the Ishai, Sahai and Wagner (ISW) framework for private circuits. In practice, for small orders, our new countermeasure is one order of magnitude faster than previous work.

We also describe a 3rd-order attack against the 3rd-order Hutter-Tunstall algorithm, and a constant, 4th-order attack against the  $t$ -th order Hutter-Tunstall algorithms, for any  $t \geq 4$ .

## 1 Introduction

**The Masking Countermeasure.** Masking is a very common countermeasure against side channel attacks, first suggested in [CJRR99, GP99]. It consists in masking every variable  $x$  into  $x' = x \oplus r$ , where  $r$  is a randomly generated value. The two shares  $x'$  and  $r$  are then manipulated separately, so that a first-order attack that processes intermediate variables separately cannot succeed. However first-order masking is vulnerable to a second-order attack combining information on the two shares  $x'$  and  $r$ ; see [OMHT06] for a practical attack. Boolean masking can naturally be extended to  $n$  shares, with  $x = x_1 \oplus \dots \oplus x_n$ ; in that case an implementation should be resistant against  $t$ -th order attacks, in which the adversary combines leakage information from at most  $t < n$  variables. It was shown in [CJRR99, PR13, DDF14] that under a reasonable noisy model, the number of noisy samples required to recover a secret  $x$  from its shares  $x_i$  grows exponentially with the number of shares.

**Security Model.** The theoretical study of securing circuits against side-channel attacks was initiated by Ishai, Sahai and Wagner (ISW) [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 authors 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 such

an adversary. The construction is based on secret-sharing every variable  $x$  into  $n$  shares with  $x = x_1 \oplus \dots \oplus x_n$ , 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 approach for proving security is based on simulation: instead of considering all possible  $t$ -uples of probes, which would be unfeasible since this grows exponentially with  $t$ , the authors show how to simulate any set of  $t$  wires probed by the adversary, from a proper subset of the input shares of the transformed circuit  $C'$ . Since any proper subset of the input shares can be simulated without knowledge of the input variables of the original circuit (simply by generating random values), one can then obtain a perfect simulation of the  $t$  probes. This shows that the  $t$  probes do not bring any additional information to the attacker, since he could simulate those  $t$  probes by himself, without knowing the secret key.

In this paper, all our constructions are proven secure in the ISW model. More precisely, we use the refined  $t$ -SNI security notion introduced in [BBD+16]. This enables to show that a particular gadget can be used in a full construction with  $n \geq t + 1$  shares, instead of  $n \geq 2t + 1$  for the weaker definition of  $t$ -NI security (as used in the original ISW security proof). The  $t$ -SNI security notion is a very practical definition that enables modular proofs; this is done by first considering the  $t$ -SNI security of individual gadgets and then composing them inside a more complex construction.

**Boolean vs. Arithmetic Masking.** Boolean masking consists in splitting every variable  $x$  into  $n$  shares  $x_i$  such that  $x = x_1 \oplus x_2 \oplus \dots \oplus x_n$ , and the shares are then processed separately. However some algorithms use arithmetic operations, for example IDEA [LM90], RC6 [CRRY99], XTEA [NW97], SPECK [BSS+13] and SHA-1 [NIS95]. In that case it can be advantageous to use arithmetic masking. For example, if the variable  $z = x + y \bmod 2^k$  must be computed securely for some parameter  $k$ , a first-order countermeasure with arithmetic shares consists in writing  $x = A_1 + A_2$  and  $y = B_1 + B_2$  for arithmetic shares  $A_1, A_2, B_1, B_2$ . Then instead of computing  $z = x + y$  directly, which would leak information on  $x$  and  $y$ , one can add the shares separately, by letting  $C_1 \leftarrow A_1 + B_1$  and  $C_2 \leftarrow A_2 + B_2$ ; this gives the two arithmetic shares  $C_1$  and  $C_2$  using  $z = x + y = A_1 + A_2 + B_1 + B_2 = C_1 + C_2$ . Note that throughout the paper all additions and subtractions are performed modulo  $2^k$  for some  $k$ ; for example for SHA-1 we have  $k = 32$ .

When combining Boolean and arithmetic masking, one must be able to convert between the two types of masking; obviously the conversion algorithm itself must be secure against side-channel attacks. More precisely, a Boolean to arithmetic conversion algorithm takes as input  $n$  shares  $x_i$  such that:

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

and one must compute  $n$  arithmetic shares  $A_i$  such that:

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

without leaking information about  $x$ .

**Prior Work.** The first Boolean to arithmetic conversion algorithms were described by Goubin in [Gou01], with security against first-order attacks only. Goubin’s Boolean to arithmetic algorithm is quite elegant and has complexity  $\mathcal{O}(1)$ , that is independent of the register size  $k$ . The arithmetic to Boolean conversion is more complex and has complexity  $\mathcal{O}(k)$ ; this was later improved to  $\mathcal{O}(\log k)$  in [CGTV15]; however in practice for  $k = 32$  the number of operations is similar.

The first conversion algorithms secure against high-order attacks 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.<sup>1</sup> The authors of [CGV14] also describe an alternative approach that use Boolean masking only and employ secure algorithms to perform the arithmetic operations directly on the Boolean shares, with the same asymptotic complexity; they show that for HMAC-SHA-1 this leads to an efficient implementation.

Recently, Hutter and Tunstall have described in [HT16] a high-order Boolean to arithmetic conversion algorithm with complexity independent of the register size  $k$  (as in Goubin’s original algorithm). However no proof of security is provided, except for second-order and third-order attacks. The complexity of the algorithm for  $n$  shares is  $\mathcal{O}(2^{n/2})$ , but for small values of  $n$  the algorithm is much more efficient than [CGV14, CGTV15], at least by one order of magnitude<sup>2</sup>.

**Our Contributions.** In this paper our contributions are as follows:

- We describe a high-order Boolean to arithmetic conversion algorithm with complexity independent of the register size  $k$ , using a similar approach as in [HT16], but with a proof of security in the ISW model. Our algorithm achieves security against attacks of order  $n-1$  for  $n$  shares, for any  $n \geq 3$ . Our conversion algorithm has complexity  $\mathcal{O}(2^n)$ , instead of  $\mathcal{O}(n^2 \cdot k)$  in [CGV14], but for small values of  $n$  it is one order of magnitude more efficient. In Sect. 6 we report the execution times we achieved for both algorithms, using 32-bit registers.
- We describe a 4th order attack against the  $t$ -th order Hutter-Tunstall algorithm (with  $n = t + 1$  shares), for any  $t \geq 4$ . We also describe a 3rd order attack for  $t = 3$ . This implies that the conversion algorithm in [HT16] cannot offer more than second-order security<sup>3</sup>.

**Source Code.** A proof-of-concept implementation of our high-order conversion algorithm, using the C language, is available at: <http://pastebin.com/CSn67PxQ>

<sup>1</sup> This can also be improved to  $\mathcal{O}(n^2 \cdot \log k)$  using [CGTV15].

<sup>2</sup> In [HT16] the authors claim that the complexity of their algorithm is  $\mathcal{O}(n^2)$ , but it is actually  $\mathcal{O}(2^{n/2})$ , because it makes 2 recursive calls to the same algorithm with  $n - 2$  shares.

<sup>3</sup> Our attacks apply on the version posted on 22-Dec-2016 of [HT16]; it has been updated since then.

## 2 Security Definitions

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  $i \in I$ .

**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; based on the ISW multiplication gadget, it allows to prove the security of a transformed circuit with  $n \geq 2t + 1$  shares. The stronger  $t$ -SNI notion 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 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). For a complex construction involving many gadgets (as the one considered in this paper), this enables to easily prove that the full construction is  $t$ -SNI secure, based on the  $t$ -SNI security of its components.

## 3 Goubin's First-Order Conversion and Previous Works

### 3.1 Goubin's Algorithm

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  $\Psi(x_1, r) : \mathbb{F}_{2^k} \times \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$

$$\Psi(x_1, r) = (x_1 \oplus r) - r \pmod{2^k}$$

As mentioned previously, all additions and subtractions are performed modulo  $2^k$  for some parameter  $k$ , so in the following we omit the mod  $2^k$ . Moreover we grant higher precedence to xor than addition, so we simply write  $\Psi(x_1, r) = x_1 \oplus r - r$ .

**Theorem 1 (Goubin [Gou01]).** *The function  $\Psi(x_1, r)$  is affine with respect to  $r$  over  $\mathbb{F}_2$ .*

Thanks to the affine property of  $\Psi$ , the conversion from Boolean to arithmetic masking is relatively straightforward. Namely given as input the two Boolean shares  $x_1, x_2$  such that

$$x = x_1 \oplus x_2$$

we can write:

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

for random  $r \leftarrow \{0, 1\}^k$ . Therefore one can compute

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

and get the two arithmetic shares  $A$  and  $x_2$  of

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

The conversion algorithm is clearly secure against first-order attacks, because the left term  $\Psi(x_1, r \oplus x_2)$  is independent of  $x_2$  (thanks to the mask  $r$ ), and the right term  $\Psi(x_1, r)$  is also independent from  $x_2$ . The algorithm is quite efficient as it requires only a constant number of operations, independent of  $k$ .

### 3.2 $t$ -SNI Variant of Goubin's Algorithm

In this paper our goal is to describe a high-order conversion algorithm from Boolean to arithmetic masking, with complexity independent of the register size  $k$ , as in Goubin's first-order algorithm above. Moreover we will use Goubin's first-order algorithm as a subroutine, for which the stronger  $t$ -SNI property recalled in Sect. 2 is needed. However it is easy to see that Goubin's algorithm recalled above does not achieve the  $t$ -SNI security notion. This is because by definition the output share  $x_2$  in  $x = A + x_2$  is the same as the input share in  $x = x_1 \oplus x_2$ ; therefore if we take  $O = \{2\}$  in Definition 2, we need to set  $I = \{2\}$  to properly simulate  $x_2$ ; this contradicts the  $t$ -SNI bound  $|I| \leq t$ , since in that case for  $t = 0$  we should have  $I = \emptyset$ .

However, it is straightforward to modify Goubin's algorithm to make it  $t$ -SNI: it suffices to first refresh the 2 input shares  $x_1, x_2$  with a random  $s$ . We obtain the following first-order  $t$ -SNI Boolean to arithmetic algorithm (Algorithm 1).

**Lemma 1** (GoubinSNI). *Let  $x_1, x_2$  be the inputs of Goubin's algorithm (Algorithm 1) and let  $A_1$  and  $A_2$  be the outputs. Let  $t$  be the number of probed variables and let  $O \subset \{1, 2\}$ , with  $t + |O| < 2$ . There exists a subset  $I \subset \{1, 2\}$ , such that all probed variables and  $A_{|O}$  can be perfectly simulated from  $x_{|I}$ , with  $|I| \leq t$*

*Proof.* We distinguish two cases. If  $t = 0$ , then the variable  $s$  is not probed by the adversary, and therefore both  $A_2 = a_2 = x_2 \oplus s$  and  $A_1 = x - A_2 = x - x_2 \oplus s$  have

**Algorithm 1.** GoubinSNI: Boolean to arithmetic conversion,  $t$ -SNI variant**Input:**  $x_1, x_2$  such that  $x = x_1 \oplus x_2$ **Output:**  $A_1, A_2$  such that  $x = A_1 + A_2$ 

- 1:  $s \leftarrow \{0, 1\}^k$
- 2:  $a_1 \leftarrow x_1 \oplus s$
- 3:  $a_2 \leftarrow x_2 \oplus s$
- 4:  $r \leftarrow \{0, 1\}^k$
- 5:  $u \leftarrow a_1 \oplus \Psi(a_1, r \oplus a_2)$
- 6:  $A_1 \leftarrow u \oplus \Psi(a_1, r)$
- 7:  $A_2 \leftarrow a_2$
- 8: **return**  $A_1, A_2$

the uniform distribution separately; therefore any of these 2 output variables can be perfectly simulated with  $I = \emptyset$ .

If  $t = 1$ , then we must have  $O = \emptyset$ . It is easy to see that any single intermediate variable can be perfectly simulated from the knowledge of either  $x_1$  or  $x_2$ , as in Goubin's original conversion algorithm, which gives  $|I| \leq t$  as required.  $\square$

**Complexity Analysis.** We see that Algorithm 1 requires 2 random generations, 2 computations of  $\Psi$ , and 5 xors, for a total of 11 operations. In particular, the complexity of Goubin's algorithm is independent of the register size  $k$ .

### 3.3 High-Order Conversion Between Boolean and Arithmetic Masking

The first conversion algorithms secure against high-order attacks were described in [CGV14], with complexity  $\mathcal{O}(n^2 \cdot k)$  for  $n$  shares and  $k$ -bit addition in both directions. The algorithms in [CGV14] are proven secure in the ISW probing model [ISW03], with  $n \geq 2t + 1$  shares for security against  $t$  probes. The arithmetic to Boolean conversion proceeds by recursively applying a  $n/2$  arithmetic to Boolean conversion on both halves, and then performing a Boolean-protected arithmetic addition:

$$\begin{aligned}
 A &= A_1 + \cdots + A_{n/2} + A_{n/2+1} + \cdots + A_n \\
 &= x_1 \oplus \cdots \oplus x_{n/2} + y_1 \oplus \cdots \oplus y_{n/2} \\
 &= z_1 \oplus \cdots \oplus z_n
 \end{aligned}$$

The arithmetic addition can be based on Goubin's recursion formula [Gou01] with complexity  $\mathcal{O}(k)$  for  $k$ -bit register. This can be improved to  $\mathcal{O}(\log k)$  by using a recursion formula based on the Kogge-Stone carry look-ahead adder (see [CGTV15]); however for  $k = 32$  the number of operations is similar. In both cases the recursion formula only uses Boolean operation, so it can be protected with  $n$  shares with complexity  $\mathcal{O}(n^2 \cdot k)$  or  $\mathcal{O}(n^2 \cdot \log k)$ . For the other direction, *i.e.* Boolean to arithmetic, it is based on the above arithmetic to Boolean conversion, and it has also complexity  $\mathcal{O}(n^2 \cdot k)$  (and  $\mathcal{O}(n^2 \cdot \log k)$  with Kogge-Stone).

Recently, Hutter and Tunstall have described in [HT16] a different technique for high-order Boolean to arithmetic conversion, with complexity independent of the register size  $k$  (as in Goubin’s original algorithm). However no proof of security is provided, except for second-order and third-order attacks. The complexity of their algorithm for  $n$  shares is  $\mathcal{O}(2^{n/2})$ , but for small values of  $n$  the algorithm is much more efficient than [CGV14, CGTV15], at least by one order of magnitude. In [HT16] the authors claim that the complexity of their algorithm is  $\mathcal{O}(n^2)$ , but it is easy to see that it must be  $\mathcal{O}(2^{n/2})$ , because it makes 2 recursive calls to the same algorithm with  $n - 2$  shares.

However, in this paper we describe a 4th order attack against the  $t$ -th order Hutter-Tunstall algorithm (with  $n = t + 1$  shares), for any  $t \geq 4$ ; we also describe a 3rd order attack for  $t = 3$ ; see Sect. 5. This implies that the conversion algorithm in [HT16] cannot offer more than second-order security. In particular, we have not found any attack against the second-order Boolean to arithmetic conversion specified in [HT16, Algorithm 2].

## 4 High-order Conversion from Boolean to Arithmetic Masking

In this section, we describe our main contribution: a high-order conversion algorithm from Boolean to arithmetic masking, with complexity independent of the register size  $k$ , with a proof of security in the ISW model for  $n \geq t + 1$  shares against  $t$  probes ( $t$ -SNI security).

### 4.1 A Simple but Insecure Algorithm

To illustrate our approach, we first describe a simple but insecure algorithm; namely we explain why it fails to achieve the  $t$ -SNI security property. We start from the  $n$  shares  $x_i$  such that

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

and we must output  $n$  shares  $A_i$  such that

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

Our tentative conversion algorithm  $\mathcal{C}_n$  is defined recursively, using a similar approach as in [HT16], and works as follows:

1. We write

$$x = x_2 \oplus \cdots \oplus x_n + (x_1 \oplus x_2 \oplus \cdots \oplus x_n - x_2 \oplus \cdots \oplus x_n)$$

which gives using  $\Psi(x_1, u) = x_1 \oplus u - u$ :

$$x = x_2 \oplus \cdots \oplus x_n + \Psi(x_1, x_2 \oplus \cdots \oplus x_n)$$

From the affine property of the  $\Psi$  function, we obtain:

$$x = x_2 \oplus \cdots \oplus x_n + (n \wedge 1) \cdot x_1 \oplus \Psi(x_1, x_2) \oplus \cdots \oplus \Psi(x_1, x_n)$$

Therefore we let  $z_1 \leftarrow (n \wedge 1) \cdot x_1 \oplus \Psi(x_1, x_2)$  and  $z_i \leftarrow \Psi(x_1, x_{i+1})$  for all  $2 \leq i \leq n - 1$ . This gives:

$$x = x_2 \oplus \cdots \oplus x_n + z_1 \oplus \cdots \oplus z_{n-1}$$

2. We then perform two recursive calls to the Boolean to arithmetic conversion algorithm  $\mathcal{C}_{n-1}$ , with  $n - 1$  shares. This gives:

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

3. We reduce the number of arithmetic shares from  $2n - 2$  to  $n$  by some additive grouping, 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:

$$x = D_1 + \cdots + D_n$$

This terminates the description of our tentative algorithm. We explain why this simple algorithm is insecure. Namely if the adversary probes the  $n - 1$  variables  $z_i$ , since each  $z_i$  reveals information about both  $x_1$  and  $x_{i+1}$ , those  $n - 1$  variables reveal information about  $x$ . More precisely, from the probed  $z_i$ 's the adversary can compute:

$$z_1 \oplus \cdots \oplus z_{n-1} = \Psi(x_1, x_2 \oplus \cdots \oplus x_n)$$

Letting  $u = x_2 \oplus \cdots \oplus x_n$  and  $v = x_n$ , for  $n \geq 3$  we can assume that the two variables  $u$  and  $v$  are uniformly and independently distributed. Therefore the adversary obtains the two variables:

$$\Psi(x_1, u) = x_1 \oplus u - u, \quad \Psi(x_1, v) = x_1 \oplus v - v$$

and one can check that the distribution of  $(\Psi(x_1, u), \Psi(x_1, v))$  depends on  $x = x_1 \oplus u$ . Therefore, the  $n - 1$  probes leak information about  $x$ . Moreover, due to the recursive definition of the above algorithm, the number of required probes can be decreased by probing within the recursive calls, instead of the  $z_i$ 's. Namely if the adversary probes only  $n - 2$  variables within  $\mathcal{C}_{n-1}$ , this reveals information about the  $n - 1$  variables  $z_i$ 's, which in turn reveals information about  $x$ , as explained above.

The attack can be applied recursively down to a single probe. Namely one can check experimentally (for small  $k$  and  $n$ ) that for randomly distributed  $x_1, \dots, x_n$ , some intermediate variables in the recursion have a distribution that depends on  $x = x_1 \oplus \cdots \oplus x_n$ ; hence the algorithm is actually vulnerable to a first-order attack.



---

**Algorithm 2.** RefreshMasks

---

**Input:**  $x_1, \dots, x_n$

**Output:**  $y_1, \dots, y_n$  such that  $y_1 \oplus \dots \oplus y_n = x_1 \oplus \dots \oplus x_n$

1:  $y_n \leftarrow x_n$

2: **for**  $i = 1$  to  $n - 1$  **do**

3:      $r_i \leftarrow \{0, 1\}^k$

4:      $y_i \leftarrow x_i \oplus r_i$

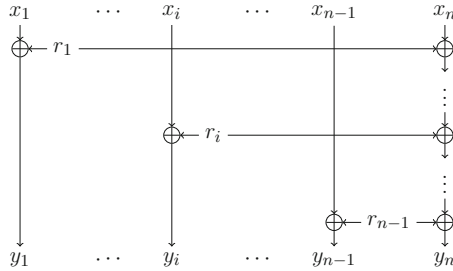
5:      $y_n \leftarrow y_n \oplus r_i$

$\triangleright y_{n,i} = x_n \oplus \bigoplus_{j=1}^i r_j$

6: **end for**

7: **return**  $y_1, \dots, y_n$

---



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

### 4.2 Mask Refreshing

To prevent the above attack (and any other attack), we must perform some mask refreshing on the intermediate shares. We use the same RefreshMasks procedure as in [RP10]; see Algorithm 2, and Fig. 1 for an illustration.

In the RefreshMasks algorithm above we denote by  $y_{n,i}$  the intermediate variables in the accumulated sum, namely for  $1 \leq i \leq n - 1$ :

$$y_{n,i} = x_n \oplus \bigoplus_{j=1}^i r_j$$

We add 3 applications of RefreshMasks in the previous conversion algorithm. The first application is to first expand the  $n$  input shares  $x_i$  into  $n + 1$  shares, so that we can now have  $n$  variables of the form  $z_i = \Psi(x_1, x_{i+1})$  instead of only  $n - 1$ ; this is to prevent the adversary from recovering all variables  $z_i$ 's. However, one must still compress to  $n - 1$  variables for the recursive application of the conversion algorithm with  $n - 1$  shares. This is done by using two RefreshMasks (one for each recursive application) followed by xoring the last two shares into one, to get  $n - 1$  shares. As will be seen in the next sections, we obtain a  $t$ -SNI conversion algorithm; this is essentially based on a careful analysis of the security properties of RefreshMasks.

### 4.3 Secure Conversion from Boolean to Arithmetic Masking

We are now ready to describe our new high-order conversion algorithm from Boolean to arithmetic masking; as previously, our algorithm  $\mathcal{C}_n$  is defined recursively. We start from the  $n$  shares:

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

If  $n = 2$ , we apply the  $t$ -SNI variant of Goubin's first order algorithm, as described in Algorithm 1. For  $n \geq 3$ , we proceed as follows.

1. We first perform a  $(n + 1)$ -RefreshMasks of the  $n$  shares  $x_i$ 's and  $x_{n+1} = 0$ , so that we obtain the following  $n + 1$  shares:

$$a_1, \dots, a_{n+1} \leftarrow \text{RefreshMasks}_{n+1}(x_1, \dots, x_n, 0)$$

Therefore we still have  $x = a_1 \oplus \cdots \oplus a_{n+1}$ . We can write as previously using  $\Psi(a_1, u) = a_1 \oplus u - u$ :

$$\begin{aligned} x &= a_2 \oplus \cdots \oplus a_{n+1} + (a_1 \oplus \cdots \oplus a_{n+1} - a_2 \oplus \cdots \oplus a_{n+1}) \\ &= a_2 \oplus \cdots \oplus a_{n+1} + \Psi(a_1, a_2 \oplus \cdots \oplus a_{n+1}) \end{aligned}$$

2. Thanks to the affine property of  $\Psi$ , this gives as previously:

$$x = a_2 \oplus \cdots \oplus a_{n+1} + (\overline{n \wedge 1}) \cdot a_1 \oplus \Psi(a_1, a_2) \oplus \cdots \oplus \Psi(a_1, a_{n+1})$$

Therefore, we let  $b_1 \leftarrow (\overline{n \wedge 1}) \cdot a_1 \oplus \Psi(a_1, a_2)$  and  $b_i \leftarrow \Psi(a_1, a_{i+1})$  for all  $2 \leq i \leq n$ . This gives:

$$x = a_2 \oplus \cdots \oplus a_{n+1} + b_1 \oplus \cdots \oplus b_n$$

3. We perform a RefreshMasks of the  $a_i$ 's and of the  $b_i$ 's, letting:

$$\begin{aligned} c_1, \dots, c_n &\leftarrow \text{RefreshMasks}(a_2, \dots, a_{n+1}) \\ d_1, \dots, d_n &\leftarrow \text{RefreshMasks}(b_1, \dots, b_n) \end{aligned}$$

Therefore we still have:

$$x = c_1 \oplus \cdots \oplus c_n + d_1 \oplus \cdots \oplus d_n$$

4. We compress from  $n$  shares to  $n - 1$  shares, by xoring the last two shares of the  $c_i$ 's and  $d_i$ 's. More precisely we let  $e_i \leftarrow c_i$  and  $f_i \leftarrow d_i$  for all  $1 \leq i \leq n - 2$ , and  $e_{n-1} \leftarrow c_{n-1} \oplus c_n$  and  $f_{n-1} \leftarrow d_{n-1} \oplus d_n$ . Therefore we still have:

$$x = e_1 \oplus \cdots \oplus e_{n-1} + f_1 \oplus \cdots \oplus f_{n-1}$$

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

$$\begin{aligned} A_1, \dots, A_{n-1} &\leftarrow \mathcal{C}_{n-1}(e_1, \dots, e_{n-1}) \\ B_1, \dots, B_{n-1} &\leftarrow \mathcal{C}_{n-1}(f_1, \dots, f_{n-1}) \end{aligned}$$

This gives:

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

6. We reduce the number of arithmetic shares from  $2n - 2$  to  $n$  by some additive grouping, 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:

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

This completes the description of the algorithm. For clarity we also provide a formal description in Appendix A.

**Theorem 2 (Completeness).** *The  $C_n$  Boolean to arithmetic conversion algorithm, when taking  $x_1, \dots, x_n$  as input, outputs  $D_1, \dots, D_n$  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. The completeness property holds for  $n = 2$  with Goubin’s conversion algorithm. Assuming that completeness holds for  $n - 1$  shares, we obtain:

$$\begin{aligned} \sum_{i=1}^n D_i &= \sum_{i=1}^{n-1} A_i + \sum_{i=1}^{n-1} B_i = \bigoplus_{i=1}^{n-1} e_i + \bigoplus_{i=1}^{n-1} f_i = \bigoplus_{i=1}^n c_i + \bigoplus_{i=1}^n d_i = \bigoplus_{i=2}^{n+1} a_i + \bigoplus_{i=1}^n b_i \\ &= \bigoplus_{i=2}^{n+1} a_i + \Psi \left( a_1, \bigoplus_{i=2}^{n+1} a_i \right) = \bigoplus_{i=1}^{n+1} a_i = \bigoplus_{i=1}^n x_i \end{aligned}$$

and therefore completeness holds for  $n$  shares.  $\square$

**Complexity Analysis.** We denote by  $T_n$  the number of operations for  $n$  shares. We assume that random generation takes unit time. We have  $T_2 = 11$  (see Sect. 3.2). The complexity of RefreshMasks with  $n$  shares is  $3n - 3$  operations. From the recursive definition of our algorithm, we obtain:

$$\begin{aligned} T_n &= [3 \cdot (n + 1) - 3] + [2 \cdot n + 3] + [2 \cdot (3n - 3)] + 2 + 2 \cdot T_{n-1} + [n - 2] \\ &= 2 \cdot T_{n-1} + 12 \cdot n - 3 \end{aligned}$$

This gives:

$$T_n = 14 \cdot 2^n - 12 \cdot n - 21$$

Therefore, the complexity of our algorithm is exponential in  $n$ , namely  $\mathcal{O}(2^n)$ , instead of  $\mathcal{O}(n^2 \cdot k)$  in [CGV14]; however for small values of  $n$  our conversion algorithm is one order of magnitude more efficient; see Sect. 6 for implementation results.

**Security.** The following theorem shows that our conversion algorithm achieves the  $t$ -SNI property. This means that our conversion algorithm is secure against any adversary with at most  $n - 1$  probes in the circuit. Moreover thanks to the  $t$ -SNI property, our conversion algorithm can be used within a larger construction (for example a block-cipher, or HMAC-SHA-1), so that the larger construction also achieves the  $t$ -SNI property.

**Theorem 3 ( $t$ -SNI of  $\mathcal{C}_n$ ).** *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  $\mathcal{C}_n$ . For any set of  $t$  intermediate variables and any subset  $O \subset [1, n]$ , there exists a subset  $I$  of input indices such that the  $t$  intermediate variables as well as  $D_{|O}$  can be perfectly simulated from  $x_{|I}$ , with  $|I| \leq t$ .*

The rest of the section is devoted to the proof of Theorem 3. The proof is based on a careful analysis of the properties of the RefreshMasks algorithm. In the next section, we start with three well known, basic properties of RefreshMasks.

#### 4.4 Basic Properties of RefreshMasks

The lemma below shows that RefreshMasks achieves the  $t$ -NI property in a straightforward way, for any  $t < n$ .

**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  $|I| \leq t$ .*

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

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

*Remark 1.* It is easy to see that RefreshMasks does not achieve the  $t$ -SNI property. Namely with  $t = 1$  we can probe  $y_{n,1} = x_n \oplus r_1$  and additionally require the simulation of the output variable  $y_1 = x_1 \oplus r_1$ . We have  $y_{n,1} \oplus y_1 = x_n \oplus x_1$ , hence the knowledge of both  $x_1$  and  $x_n$  is required for the simulation of the two variables, which contradicts the bound  $|I| \leq t$ .

The following lemma shows that any subset of  $n - 1$  output shares  $y_i$  of RefreshMasks is uniformly and independently distributed, when the algorithm is not probed.

**Lemma 3.** *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.*

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

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

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

The following lemma, whose proof is also straightforward, 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  $x_1 \oplus \dots \oplus x_n$  only; that is, the knowledge of the individual shares  $x_i$ 's is not required.

**Lemma 4.** *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  $x_1 \oplus \dots \oplus x_n$ .*

*Proof.* We have  $y_i = x_i \oplus r_i$  for all  $1 \leq i \leq n - 1$  and:

$$y_n = x_n \oplus \bigoplus_{i=1}^{n-1} r_i = \left( \bigoplus_{i=1}^n x_i \right) \oplus \left( \bigoplus_{i=1}^{n-1} y_i \right)$$

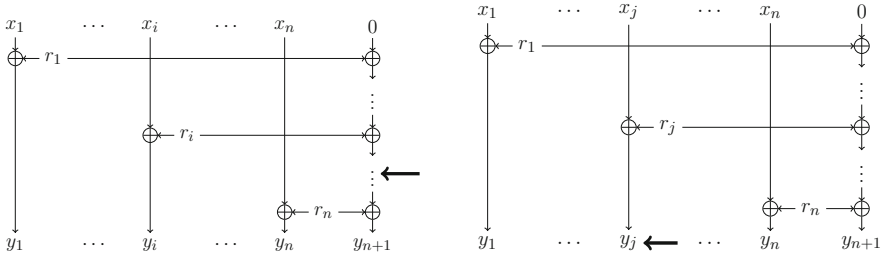
Therefore we can perfectly simulate the output  $(y_i)_{1 \leq i \leq n}$  by letting  $y_i \leftarrow \{0, 1\}^k$  for all  $1 \leq i \leq n - 1$  and  $y_n \leftarrow \left( \bigoplus_{i=1}^n x_i \right) \oplus \left( \bigoplus_{i=1}^{n-1} y_i \right)$ .  $\square$

## 4.5 Property of the Initial RefreshMasks

The lemma below gives the first non-trivial property of RefreshMasks. We consider the first RefreshMasks of our conversion algorithm  $\mathcal{C}_n$  described in Sect. 4.3, taking as input  $n + 1$  input shares  $x_i$  (instead of  $n$ ), but with  $x_{n+1} = 0$ . The lemma below is a refinement of the basic  $t$ -NI lemma (Lemma 2); namely we show that if at least one of the output variables  $y_j$  is probed, then it can be simulated “for free”, that is without increasing the size of the input index  $I$ . More precisely, we get the bound  $|I| \leq t - 1$  under that condition, instead of  $|I| \leq t$  in Lemma 2. This stronger bound will be used for the security proof of our conversion algorithm; namely at Step 2 in Sect. 4.3 the adversary can probe  $t$  of the variables  $b_i = \Psi(a_1, a_{i+1})$ , whose simulation then requires the knowledge of  $t + 1$  variables  $a_i$ . Thanks to the stronger bound, this requires the knowledge of only  $t$  input shares  $x_i$  (instead of  $t + 1$ ), as required for the  $t$ -SNI bound.

Below we actually prove a slightly stronger lemma: if we fix  $x_{n+1} = 0$ , then we can always simulate the  $t$  probes from  $x_{|I|}$  with  $|I| \leq t - 1$ , except in the trivial case of the adversary probing the input  $x_i$ 's only.

**Lemma 5.** *Let  $x_1, \dots, x_n$  be  $n$  inputs shares, and let  $x_{n+1} = 0$ . Consider the circuit  $y_1, \dots, y_{n+1} \leftarrow \text{RefreshMasks}_{n+1}(x_1, \dots, x_n, x_{n+1})$ , 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.*



**Fig. 2.** Illustration of Lemma 5. Case 1 (left): the adversary has spent at least one probe on the last column for which  $x_{n+1} = 0$ , therefore we can have  $|I| \leq t - 1$ . Case 2 (right): no intermediate variable is probed on the last column; therefore  $r_j$  can play the role of a one-time pad for the simulation of the probed  $y_j$ , hence  $x_j$  is not required and again  $|I| \leq t - 1$ .

*Proof.* As illustrated in Fig. 2, we distinguish two cases. If  $x_{n+1}$  or  $y_{n+1}$  or any intermediate variable  $y_{n+1,j}$  has been probed, we construct the set  $I$  as follows. If for some  $1 \leq i \leq n$ , any of the variables  $x_i$ ,  $y_i$  or  $r_i$  is probed, we add  $i$  to  $I$ . In the construction of  $I$  we have omitted at least one probed variable (on the column of index  $n + 1$ ), and therefore we must have  $|I| \leq t - 1$  as required. The simulation is then straightforward and proceeds as in Lemma 2. Namely all the randoms  $r_i$  are simulated as in the actual algorithm, and all probed variables  $x_i$  and  $y_i$  can be perfectly simulated from  $x_i$ , since in that case  $i \in I$ . The only difference is that  $n + 1$  need not be in  $I$  since  $x_{n+1} = 0$  by definition.

We now consider the second case. If neither  $x_{n+1}$  nor  $y_{n+1}$  nor any intermediate variable  $y_{n+1,i}$  has been probed, we construct the set  $I$  as follows. By assumption, there exists an index  $j$  such that  $r_j$  or  $y_j$  or both have been probed, with  $1 \leq j \leq n$ ; namely we have excluded the case of the adversary probing only the input  $x_i$ 's. For all  $1 \leq i \leq n$  and  $i \neq j$ , if  $x_i$  or  $r_i$  or  $y_i$  has been probed, we add  $i$  to  $I$ . Moreover if  $x_j$  has been probed, or if both  $r_j$  and  $y_j$  have been probed, we add  $j$  to  $I$ . From the construction of  $I$ , we must have  $|I| \leq t - 1$  as required. Namely either a single variable among  $r_j$  and  $y_j$  has been probed, and this probe does not contribute to  $I$ , or both  $r_j$  and  $y_j$  have been probed, and these two probes contribute to only one index in  $I$ .

The simulation of probed  $x_i$ ,  $r_i$  or  $y_i$  is straightforward for  $i \neq j$ , from the knowledge of  $x_i$ . If  $j \in I$ , the simulation of  $x_j$ ,  $r_j$  and  $y_j$  is also straightforward. If  $j \notin I$ , then either  $r_j$  or  $y_j$  has been probed (but not both). If  $r_j$  has been probed, it can be simulated by generating a random value. If  $y_j$  has been probed,

since we have  $y_j = x_j \oplus r_j$  and moreover  $r_j$  does not appear in the computation of any other probed variable (since in that case  $r_j$  has not been probed, nor any intermediate variable  $y_{n+1,i}$ ), we can simulate  $y_j$  as a random value in  $\{0,1\}^k$ . Therefore all probed variables can be perfectly simulated from  $x_{|I}$ .  $\square$

*Remark 2.* The lemma does not necessarily hold if we don't assume that  $x_{n+1} = 0$ , or if we only assume that  $x_i = 0$  for some  $i \neq n + 1$ . For example, assuming that  $x_2 = 0$ , the adversary can probe both  $y_1 = x_1 \oplus r_1$  and  $y_{n+1,1} = x_{n+1} \oplus r_1$ , which gives  $y_1 \oplus y_{n+1} = x_1 \oplus x_{n+1}$ . Hence the knowledge of 2 input shares is required to simulate the 2 probes (including the output variable  $y_1$ ), which contradicts the bound  $|I| \leq t - 1$ .

#### 4.6 More Results on RefreshMasks

In this section we consider the properties of RefreshMasks required for the compression from  $n$  shares to  $n - 1$  shares as performed at steps 3 and 4 of our conversion algorithm in Sect. 4.3. Namely if the adversary probes  $t$  of the variables  $f_i$ 's, because of the last variable  $f_{n-1} = d_{n-1} \oplus d_n$ , this can require the knowledge of  $t + 1$  of the variables  $d_i$ 's. Without RefreshMasks the knowledge of  $t + 1$  of the variables  $b_i$ 's would be required, and eventually  $t + 1$  of the input shares  $x_i$ 's, which would contradict the  $t$ -SNI bound. In this section, we show that thanks to RefreshMasks we can still get the bound  $|I| \leq t$  instead of  $|I| \leq t + 1$ .

We first prove two preliminary lemmas. The first lemma below is analogous to Lemma 5 and shows that when the randoms in RefreshMasks are accumulated on  $x_n$ , the corresponding output variable  $y_n$  can always be simulated “for free”, that is, without increasing the size of the input index  $I$ ; more precisely, if we require that  $y_n$  is among the  $t$  probes, then we can have  $|I| \leq t - 1$  instead of  $|I| \leq t$  in Lemma 2. This will enable to show that when a subsequent compression step to  $n - 1$  shares is performed with  $z_{n-1} \leftarrow y_{n-1} \oplus y_n$ , we can still keep the bound  $|I| \leq t$  instead of  $|I| \leq t + 1$ . Namely either the adversary does not probe  $z_{n-1} = y_{n-1} \oplus y_n$  and he does not benefit from getting information on two variables with a single probe, or  $z_{n-1}$  is probed and we can apply Lemma 6 below with probed  $y_n$ ; in both cases we get  $|I| \leq t$  instead of  $|I| \leq t + 1$ . We provide the proof of Lemma 6 in the full version of this paper [Cor17].

**Lemma 6.** *Let  $x_1, \dots, x_n$  be the input of a RefreshMasks where the randoms are accumulated on  $x_n$ , and let  $y_1, \dots, y_n$  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 3.* 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  $|I| \leq t - 1$ .

In the previous lemma we have restricted ourselves to  $t < n$  probes (including the probe on  $y_n$ ). Namely if  $t = n$ , the adversary can probe all  $y_i$ 's and learn  $x_1 \oplus \cdots \oplus x_n = y_1 \oplus \cdots \oplus y_n$ ; therefore the simulation cannot be performed using a proper subset  $I$  of  $[1, n]$ . In Lemma 4 we have showed that when no intermediate variables of RefreshMasks are probed, the  $n$  output shares  $y_i$  can be simulated from the knowledge of  $x_1 \oplus \cdots \oplus x_n$  only. The lemma below shows that this is essentially the best that the adversary can do: when the adversary has  $n$  probes, and if one of which must be  $y_n$ , then either all 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  $|I| \leq n - 1$ . As previously this only holds if  $y_n$  must be among the  $n$  probes; namely without this restriction the attacker could probe the  $n$  input shares  $x_i$  directly and learn the value of the individual shares  $x_i$  (and not only the xor of the  $x_i$ 's); see also Remark 4 below.

As previously, this will enable to show that when a subsequent compression step is performed with  $z_{n-1} \leftarrow y_{n-1} \oplus y_n$ , if the adversary has a total of  $n - 1$  probes, then the simulation can be performed from  $x_1 \oplus \cdots \oplus x_n$  only, or from  $x_{|I}$  with  $|I| \leq n - 1$ . Namely either the adversary does not probe  $z_{n-1} = y_{n-1} \oplus y_n$  and we can simulate from  $x_{|I}$  with  $|I| \leq n - 1$ , or  $z_{n-1}$  is probed and we can apply Lemma 7 below with probed  $y_n$ . The proof of Lemma 7 can be found in the full version of this paper [Cor17].

**Lemma 7.** *Let  $x_1, \dots, x_n$  be the input of a RefreshMasks where the randomness are accumulated on  $x_n$ , and let  $y_1, \dots, y_n$  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  $x_{|I}$ .*

*Remark 4.* As previously, the lemma does not hold if any other output variable  $y_i$  is required to be probed instead of  $y_n$ . Namely the adversary can probe the  $n$  variables  $y_1 = x_1 \oplus r_1$ ,  $x_2, \dots, x_{n-1}$  and  $y_{n,1} = x_n \oplus r_1$ . The xor of these  $n$  variables gives  $x_1 \oplus \cdots \oplus x_n$ , but the adversary also learns the individual shares  $x_2, \dots, x_{n-1}$ . Whereas in Lemma 7, the adversary either learns  $x_1 \oplus \cdots \oplus x_n$  and nothing else, or at most  $n - 1$  of the shares  $x_i$ .

The lemma below is the main result of the section. As mentioned previously, it enables to show that when we perform the compression from  $n$  shares to  $n - 1$  shares at steps 3 and 4 of our conversion algorithm from Sect. 4.3, we can still have the bound  $|I| \leq t$  instead of  $|I| \leq t + 1$  when  $t < n$ ; and for  $t = n - 1$ , the simulation can be performed either from  $x_1 \oplus \cdots \oplus x_n$ , or from  $x_{|I}$  with  $|I| \leq n - 1$ . We refer to the full version of this paper [Cor17] for the proof of Lemma 8.

**Lemma 8.** *Consider the circuit with  $y_1, \dots, y_n \leftarrow \text{RefreshMasks}(x_1, \dots, x_n)$ ,  $z_i \leftarrow y_i$  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  $x_{|I}$ .*



*Remark 5.* The lemma does not hold if the two xored output variables of RefreshMasks do not include  $y_n$  (when the randoms of RefreshMasks are accumulated on  $x_n$ ). For example, if we let  $z_1 \leftarrow y_1 \oplus y_2$  instead, the adversary could probe both  $z_1 = y_1 \oplus y_2 = (x_1 \oplus r_1) \oplus (x_2 \oplus r_2)$  and  $y_{n,2} = x_n \oplus r_1 \oplus r_2$ , which gives  $z_1 \oplus y_{n,2} = x_1 \oplus x_2 \oplus x_n$ . Hence to simulate those 2 probes the knowledge of 3 shares is required, which contradicts the bound  $|I| \leq t$ .

Note that the value  $x_1 \oplus \dots \oplus x_n$  in the above lemma corresponds to either  $a_2 \oplus \dots \oplus a_{n+1}$  or  $b_1 \oplus \dots \oplus b_n$  at Step 3 of our conversion algorithm from Sect. 4.3. In that case, the adversary has already spent  $n - 1$  probes, and no other variable is probed. As shown in the next section, this enables to prove that these values can be simulated without knowing the input shares. Namely when the initial RefreshMasks is not probed, the distribution of  $a_2 \oplus \dots \oplus a_{n+1}$  is uniform because of Lemma 3, and can therefore be simulated by a random value. Similarly we have:

$$b_1 \oplus \dots \oplus b_n = \Psi(a_1, a_2 \oplus \dots \oplus a_{n+1}) = x - x \oplus a_1$$

where  $x = x_1 \oplus \dots \oplus x_n = a_1 \oplus \dots \oplus a_{n+1}$ . Since in that case the initial RefreshMasks is not probed, the variable  $a_1$  has the uniform distribution, hence the value  $b_1 \oplus \dots \oplus b_n$  can also be simulated by a random value.

#### 4.7 Proof of Theorem 3

We refer to the full version of this paper [Cor17] for the proof of Theorem 3.

## 5 Cryptanalysis of the Hutter-Tunstall Boolean to Arithmetic Conversion Algorithm

In this section, we describe two attacks against the high-order Hutter-Tunstall Boolean to arithmetic conversion algorithm in [HT16], breaking all the conversion algorithms except the second-order algorithm. For clarity we will use the same notations as in [HT16] and denote by  $n$  the maximum number of probes in the circuit; therefore the conversion algorithm takes as input  $n + 1$  shares and outputs  $n + 1$  shares, instead of  $n$  in the previous sections. Following [HT16], we say that a countermeasure is of order  $n$  when it should be resistant against  $n$  probes (hence with  $n + 1$  shares as input and output).

Our two attacks are as follows:

- An attack of order 4 against the  $n$ -th order countermeasure, for  $n \geq 4$ .
- An attack of order  $n$  against the  $n$ -th order countermeasure, for  $n \geq 3$ .

Therefore the second attack is only useful for  $n = 3$ , as for  $n \geq 4$  the first attack is of constant order 4. In particular, we show that our second attack can be applied against the third-order algorithm explicitly described in [HT16, Algorithm 3]. Our two attacks imply that the conversion algorithm in [HT16] cannot offer more than second-order security.

In the following we do not provide a full description of the conversion algorithm from [HT16]; for simplicity we only provide the relevant part leading to the attack; we refer to [HT16] for the full description.

### 5.1 Attack of Order 4 Against $n$ -th Order Countermeasure

We have as input the  $n + 1$  shares  $x', r_1, \dots, r_n$ , where:

$$x = x' \oplus r_1 \oplus \dots \oplus r_n$$

We copy Eq. (24) from [HT16]:

$$\left( \bigoplus_{i=1}^{n-1} \kappa_i \right) - \left( \alpha \oplus \bigoplus_{i=1}^n r_i \right) = ((-n \wedge 1)\beta) \oplus \bigoplus_{i=1}^{n-1} \Psi(\beta, \delta_i) \oplus \Psi \left( \beta, \alpha \oplus r_n \oplus \bigoplus_{i=1}^{n-1} \delta_i \oplus r_i \right)$$

The above equation means that the variable

$$X = ((-n \wedge 1)\beta) \oplus \bigoplus_{i=1}^{n-1} \Psi(\beta, \delta_i) \oplus \Psi \left( \beta, \alpha \oplus r_n \oplus \bigoplus_{i=1}^{n-1} \delta_i \oplus r_i \right) \quad (1)$$

is explicitly computed, using a certain sequence of operations following from the right-hand side of the equation. From the affine property of the  $\Psi$  function, we have:

$$X = \Psi \left( \beta, \alpha \oplus \bigoplus_{i=1}^n r_i \right)$$

Letting  $u := \alpha \oplus \bigoplus_{i=1}^n r_i$ , we can write:

$$\begin{aligned} X &= \Psi(\beta, u) = \beta \oplus u - u \\ x &= x' \oplus \alpha \oplus u \end{aligned}$$

Therefore, if the variable  $\beta$  is explicitly computed when evaluating (1), our attack works by probing the 4 variables  $\beta$ ,  $X$ ,  $\alpha$  and  $x'$ . From  $\beta$  and  $X = \beta \oplus u - u$ , we obtain information about  $u$ . From  $\alpha$  and  $x'$ , this reveals information about  $x = x' \oplus \alpha \oplus u$ .

Alternatively, if the variable  $\beta$  is not explicitly computed<sup>4</sup>, the variable  $Y = \Psi(\beta, \delta_1)$  must still be explicitly computed when evaluating  $X$ . Therefore our attack works by probing the 4 variables  $Y$ ,  $X$ ,  $\alpha$  and  $x'$ . We obtain the two variables:

$$X = \Psi(\beta, u), \quad Y = \Psi(\beta, \delta_1)$$

<sup>4</sup> In the concrete description of the third-order conversion algorithm in [HT16, Algorithm 3], the variable  $\beta$  is not explicitly computed when computing  $\Psi(\beta, \delta_i) = \beta \oplus \delta_i - \delta_i$ , by only computing  $\beta \oplus \delta_i$  instead.

and one can check that for randomly distributed  $\beta, \delta_1$ , the distribution of  $(X, Y)$  leaks information about  $u$ . Namely, the variable  $Y = \Psi(\beta, \delta_1) = \beta \oplus \delta_1 - \delta_1$  leaks information about  $\beta$ , which combined with  $X = \Psi(\beta, u) = \beta \oplus u - u$  leaks information about  $u$ . From  $\alpha$  and  $x'$ , this reveals information about  $x = x' \oplus \alpha \oplus u$ . Hence in both cases we obtain an attack of constant order 4 against the  $n$ -th order countermeasures for any  $n \geq 4$ .

## 5.2 Attack of Order $n$ Against the $n$ -th Order Countermeasure

We refer to the full version of this paper [Cor17] for the description of our second attack.

## 6 Operation Count and Implementation

As shown in Sect. 4.3, the number of operations of our Boolean to arithmetic conversion algorithm is given by  $T_n = 14 \cdot 2^n - 12 \cdot n - 21$ , so it has complexity  $\mathcal{O}(2^n)$  independent of the register size  $k$ , while the conversion algorithm from [CGV14] has complexity  $\mathcal{O}(k \cdot n^2)$ . We summarize in Table 1 the operation count for [CGV14] (for  $k = 32$ ) and for our new algorithm from Sect. 4.3. We see that for small orders  $t$ , our new countermeasure is at least one order of magnitude faster than previous work.

**Table 1.** Operation count for Boolean to arithmetic conversion algorithms, up to security order  $t = 12$ , with  $n = t + 1$  shares.

B $\rightarrow$ A conversion	Security order $t$								
	1	2	3	4	5	6	8	10	12
Goubin [Gou01]	7								
Hutter-Tunstall [HT16]		31							
CGV, 32 bits [CGV14]	2 098	3 664	7 752	10 226	14 698	28 044	39 518	56 344	
Our algorithm (Sect. 4.3)		55	155	367	803	1 687	7 039	28 519	114 511

We have also implemented the algorithm in [CGV14] and our new algorithm, in C on an iMac running a 3.2 GHz Intel processor, using the Clang compiler. We summarize the execution times in Table 2, which are consistent with the operation count from Table 1. This confirms that in practice for small orders, our new countermeasure is at least one order of magnitude faster than previous work.

**Table 2.** Running time in  $\mu\text{s}$  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.

B $\rightarrow$ 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
Our algorithm (Sect. 4.3)	45	119	281	611	1 270	5 673	22 192	87 322

## References

- [BBD+16] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October, pp. 116–129 (2016)
- [BSS+13] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptology ePrint Archive **2013**, 404 (2013)
- [CGTV15] Coron, J.-S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to boolean masking with logarithmic complexity. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 130–149. Springer, Heidelberg (2015). doi:[10.1007/978-3-662-48116-5\\_7](https://doi.org/10.1007/978-3-662-48116-5_7)
- [CGV14] Coron, J.-S., Großschädl, J., Vadnala, P.K.: Secure conversion between boolean and arithmetic masking of any order. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 188–205. Springer, Heidelberg (2014). doi:[10.1007/978-3-662-44709-3\\_11](https://doi.org/10.1007/978-3-662-44709-3_11)
- [CJRR99] Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). doi:[10.1007/3-540-48405-1\\_26](https://doi.org/10.1007/3-540-48405-1_26)
- [Cor17] Coron, J.-S.: High-order conversion from boolean to arithmetic masking. Cryptology ePrint Archive, Report 2017/252 (2017). <http://eprint.iacr.org/2017/252>
- [CRRY99] Contini, S., Rivest, R.L., Robshaw, M.J.B., Yin, Y.L.: Improved analysis of some simplified variants of RC6. In: Knudsen, L. (ed.) FSE 1999. LNCS, vol. 1636, pp. 1–15. Springer, Heidelberg (1999). doi:[10.1007/3-540-48519-8\\_1](https://doi.org/10.1007/3-540-48519-8_1)
- [DDF14] Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). doi:[10.1007/978-3-642-55220-5\\_24](https://doi.org/10.1007/978-3-642-55220-5_24)
- [Gou01] Goubin, L.: A sound method for switching between boolean and arithmetic masking. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 3–15. Springer, Heidelberg (2001). doi:[10.1007/3-540-44709-1\\_2](https://doi.org/10.1007/3-540-44709-1_2)

- [GP99] Goubin, L., Patarin, J.: DES and differential power analysis the “Duplication” method. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999). doi:[10.1007/3-540-48059-5\\_15](https://doi.org/10.1007/3-540-48059-5_15)
- [HT16] Hutter, M., Tunstall, M.: 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] Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). doi:[10.1007/978-3-540-45146-4\\_27](https://doi.org/10.1007/978-3-540-45146-4_27)
- [LM90] Lai, X., Massey, J.L.: A proposal for a new block encryption standard. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 389–404. Springer, Heidelberg (1991). doi:[10.1007/3-540-46877-3\\_35](https://doi.org/10.1007/3-540-46877-3_35)
- [NIS95] NIST. Secure hash standard. In: Federal Information Processing Standard, FIPA-180-1 (1995)
- [NW97] Needham, R.M., Wheeler, D.J.: Tea extentions. Technical report, Computer Laboratory, University of Cambridge (1997)
- [OMHT06] Oswald, E., Mangard, S., Herbst, C., Tillich, S.: Practical second-order DPA attacks for masked smart card implementations of block ciphers. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 192–207. Springer, Heidelberg (2006). doi:[10.1007/11605805\\_13](https://doi.org/10.1007/11605805_13)
- [PR13] Prouff, E., Rivain, M.: Higher-order side channel security and mask refreshing. In: Advances in Cryptology - EUROCRYPT 2013 - 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26–30, 2013. Proceedings, pp. 142–159 (2013)
- [RP10] Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-15031-9\\_28](https://doi.org/10.1007/978-3-642-15031-9_28)

## A Formal Description of the High-order Boolean to Arithmetic Conversion

---

**Algorithm 3.**  $\mathcal{C}_n$ : high-order Boolean to Arithmetic Conversion

---

**Input:**  $x_1, \dots, x_n$

**Output:**  $D_1, \dots, D_n$  such that  $x_1 \oplus \dots \oplus x_n = D_1 + \dots + D_n \pmod{2^k}$

```

1: if  $n = 2$  then
2:    $D_1, D_2 \leftarrow \text{GoubinSNI}(x_1, x_2)$ 
3:   return  $D_1, D_2$ 
4: end if
5:  $a_1, \dots, a_{n+1} \leftarrow \text{RefreshMasks}_{n+1}(x_1, \dots, x_n, 0)$ 
6:  $b_1 \leftarrow (\overline{n \wedge 1}) \cdot a_1 \oplus \Psi(a_1, a_2)$ 
7: for  $i = 2$  to  $n$  do
8:    $b_i \leftarrow \Psi(a_1, a_{i+1})$ 
9: end for
10:  $c_1, \dots, c_n \leftarrow \text{RefreshMasks}_n(a_2, \dots, a_{n+1})$ 
11:  $d_1, \dots, d_n \leftarrow \text{RefreshMasks}_n(b_1, \dots, b_n)$ 
12:  $e_1, \dots, e_{n-2} \leftarrow c_1, \dots, c_{n-2}$  and  $e_{n-1} \leftarrow c_{n-1} \oplus c_n$ 
13:  $f_1, \dots, f_{n-2} \leftarrow d_1, \dots, d_{n-2}$  and  $f_{n-1} \leftarrow d_{n-1} \oplus d_n$ 
14:  $A_1, \dots, A_{n-1} \leftarrow \mathcal{C}_{n-1}(e_1, \dots, e_{n-1})$ 
15:  $B_1, \dots, B_{n-1} \leftarrow \mathcal{C}_{n-1}(f_1, \dots, f_{n-1})$ 
16: for  $i = 1$  to  $n - 2$  do
17:    $D_i \leftarrow A_i + B_i$ 
18: end for
19:  $D_{n-1} \leftarrow A_{n-1}$ 
20:  $D_n \leftarrow B_{n-1}$ 
21: return  $D_1, \dots, D_n$ 

```

---