High-Order Conversion From Boolean to Arithmetic Masking

Jean-Sébastien Coron

University of 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 $x = x_1 \oplus \cdots \oplus x_n$; in that case an implementation should be resistant against t-th order attacks, in which the adversary combines leakage information from at most $t < n$ 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 $x = x_1 \oplus \cdots \oplus x_n$, and processing the shares in a way that prevents a t-limited adversary from leaning any information about the initial variable x, 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 $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$, 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:

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

and 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.

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.[1] 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.[2]

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

[2] 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.

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 Section 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 [3].

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

http://pastebin.com/CSn67PxQ

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

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

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

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 Ψ, 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:

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

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

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

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 Section 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).

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

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 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. □

Complexity analysis. We see that Algorithm 1 requires 2 random generations, 2 computations of Ψ, 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 Section 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 Ψ 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 $1 \le i \le n-2$, 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 $n \ge 3$ 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 $x_1, \ldots, x_n$, 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.

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 Figure 1 for an illustration.

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

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

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

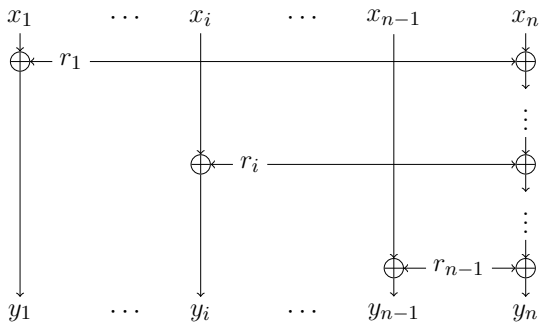

Fig. 1. The RefreshMasks algorithm, with the randoms r_i accumulated on the last column.

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:

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

   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 Ψ, 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 $\mathsf{RefreshMasks}$ of the a_i's and of the b_i's, letting:

$$c_1, \ldots, c_n \leftarrow \mathsf{RefreshMasks}(a_2, \ldots, a_{n+1})$$
$$d_1, \ldots, d_n \leftarrow \mathsf{RefreshMasks}(b_1, \ldots, b_n)$$

   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 $\mathcal{C}_{n-1}$:

$$A_1, \ldots, A_{n-1} \leftarrow \mathcal{C}_{n-1}(e_1, \ldots, e_{n-1})$$
$$B_1, \ldots, B_{n-1} \leftarrow \mathcal{C}_{n-1}(f_1, \ldots, f_{n-1})$$

   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 $\mathcal{C}_n$ Boolean to arithmetic conversion algorithm, 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. The completeness property holds for $n = 2$ with Goubin's conversion algorithm. Assuming that completeness holds for $n-1$ shares, we obtain:

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

and therefore completeness holds for n shares. □

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 Section 3.2). The complexity of $\mathsf{RefreshMasks}$ with n shares is $3n-3$ operations. From the recursive definition of our algorithm, we obtain:

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

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 Section 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 $x_{|I}$. □

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.

□

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

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 $y_n \leftarrow \left(\bigoplus_{i=1}^{n} x_i \right) \oplus \left(\bigoplus_{i=1}^{n-1} y_i \right)$. □

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

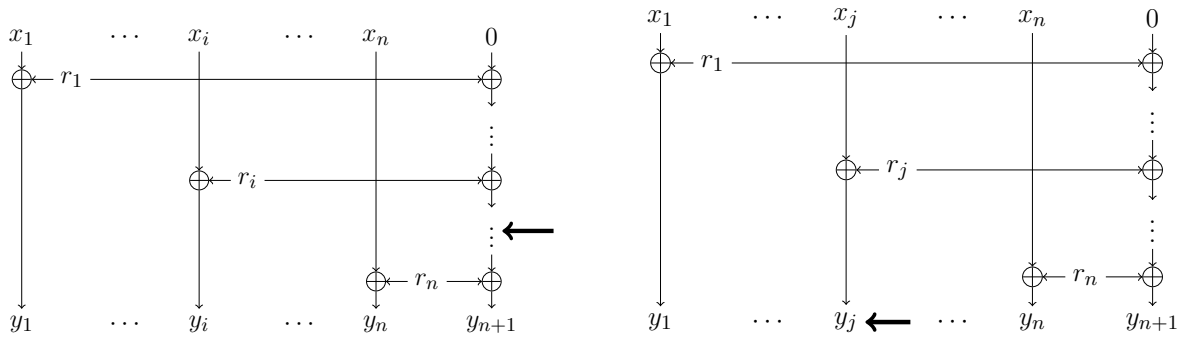

Fig. 2. Illustration of Lemma 5. Case 1 (left): the adversary has spent at least one probe on the last column for which $x_{n+1} = 0$, therefore we can have $|I| \leq t-1$. Case 2 (right): no intermediate variable is probed on the last column; therefore r_j can play the role of a one-time pad for the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

Proof. As illustrated in Figure 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 $x_{|I}$. □

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 Section 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 $|I| \leq t+1$.

Lemma 6. *Let $x_1, \ldots, x_n$ be the input of a* RefreshMasks *where the randoms are accumulated on x_n, and let $y_1, \ldots, y_n$ be the output. Let t be the number of probed variables, with $t < n$. If y_n is among the probed variables, then there exists a subset I such that all probed variables can be perfectly simulated from $x_{|I}$, with $|I| \leq t-1$.*

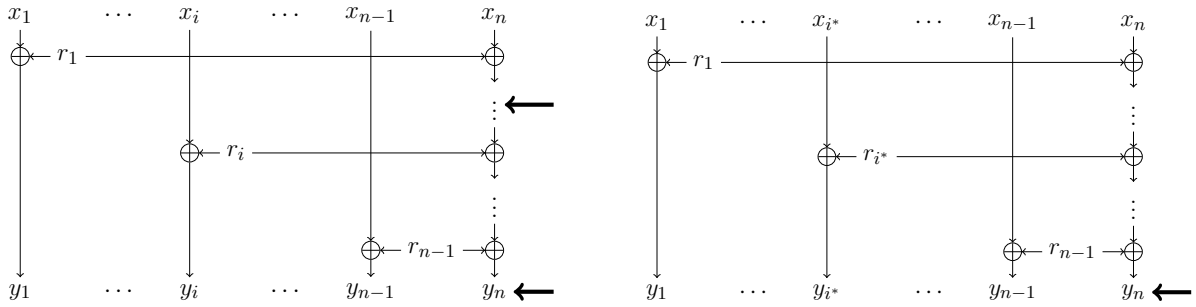

Fig. 3. Illustration of Lemma 6. Case 1 (left): the adversary has spent two probes on the column index n, and therefore $|I| \leq t-1$. Case 2 (right): no intermediate variable is probed on the last column except y_n; then r_{i*} can play the role of a one-time pad for the simulation of y_n, hence x_n is not required and again $|I| \leq t-1$.

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

For the simulation we distinguish two cases. If $n \in I$, the simulation is straightforward and proceeds as in the proof of Lemma 2. Namely all the randoms r_i are simulated as in the actual algorithm, and all probed variables x_i and y_i can be perfectly simulated from x_i, since $i \in I$. This is also the case for all intermediate variables $y_{n,j}$ and y_n, which can be simulated from x_n since $n \in I$; see Figure 3 (left) for an illustration.

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

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

From the definition of i^*, the random r_{i^*} does not appear in the definition of any other probed variable. Therefore it can play the role of a one-time pad in the above equation, and we can simulate y_n with a random value in $\{0,1\}^k$, without knowing x_n; see Figure 3 (right) for an illustration. □

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.

Lemma 7. *Let $x_1, \ldots, x_n$ be the input of a* RefreshMasks *where the randoms are accumulated on x_n, and let $y_1, \ldots, y_n$ be the output. Let t be the number of probed variables, with $t = n$. If y_n is among the probed variables, then either all probed variables can be perfectly simulated from $x_1 \oplus \cdots \oplus x_n$, or there exists a subset I with $|I| \leq n-1$ such that they can be perfectly simulated from $x_{|I}$.*

Proof. We first construct a subset J as in the proof of Lemma 6. If r_i or x_i or y_i is probed for any $1 \leq i \leq n-1$, we add i to J. If x_n or any intermediate variable $y_{n,j}$ (excluding y_n) is probed, we add n to J. Since by assumption y_n has been probed, we only consider at most $n-1$ probes in the construction of J; therefore we must have $|J| \leq n-1$.

We now distinguish two cases.

- If $n \in J$ or $|J| < n-1$, we can perform the simulation of probed variables from $x_{|I}$ with $I = J \cup \{n\}$, which gives $|I| \leq n-1$, as in the proof of Lemma 2. Namely all probed variables r_i, x_i or y_i can be simulated from x_i for $1 \leq i \leq n-1$, and any probed variable x_n or $y_{n,j}$ or y_n can be simulated from x_n.

- If $n \notin J$ and $|J| = n-1$, we must have $J = [1, n-1]$. Recall that in the construction of J, at most $n-1$ probes are considered. This implies that for every $1 \leq i \leq n-1$, exactly one of the 3 variables x_i, y_i and r_i is probed. We again distinguish three cases:
  - If only the variables y_i's are probed (including y_n), then as showed in Lemma 4 the simulation of the probed variables can be performed from the knowledge of $x_1 \oplus \cdots \oplus x_n$ only.
  - If r_i has been probed for some i, then neither x_i nor y_i has been probed, and x_i is not needed for the simulation. Therefore the simulation of all probed variable can be performed from $x_{|I}$ with $I = (J \cup \{n\}) \setminus \{i\}$, and $|I| \leq n-1$ as required.
  - If x_i has been probed for some i, then neither r_i nor y_i has been probed. Moreover no intermediate variable $y_{n,j}$ has been probed except y_n (since $n \notin J$). Therefore, as in the proof of Lemma 6, the random r_i can play the role of a one-time pad for the simulation of y_n, and x_n is not required. Therefore all probed variables can be simulated from $x_{|I}$, with $I = J = [1, n-1]$ and $|I| \leq n-1$ as required.

□

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 $y_1 = x_1 \oplus r_1$, $x_2, \ldots, x_{n-1}$ 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 $x_2, \ldots, x_{n-1}$. 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 Section 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 $|I| \leq n-1$.

Lemma 8. *Consider the circuit with $y_1, \ldots, y_n \leftarrow \mathsf{RefreshMasks}(x_1, \ldots, x_n)$, $z_i \leftarrow y_i$ for all $1 \leq i \leq n-2$ 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 $x_{|I}$.*

Proof. The proof is a straightforward application of Lemma 6 and Lemma 7. We distinguish two cases. If z_{n-1} is not probed, the simulation is straightforward since we can apply Lemma 2 directly; we obtain that all probed variables can be simulated from $x_{|I}$ with $|I| \leq t$. If z_{n-1} is probed, then we must simulate both y_{n-1} and y_n, which corresponds to a total of $t' = t+1$ probes in the circuit $y_1, \ldots, y_n \leftarrow \mathsf{RefreshMasks}(x_1, \ldots, x_n)$; see Figure 4 for an illustration. Since y_n is included in this set of t' probes, we can apply Lemma 6 and Lemma 7 directly. More precisely, if $t < n-1$, we can apply Lemma 6 with $t' = t+1 < n$; we obtain that all probed variables can be simulated from $x_{|I}$, with $|I| \leq t' - 1 \leq t$ as required. Similarly, if $t = n-1$, we can apply Lemma 7 with $t' = t+1 = n$. As required, we obtain that 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 $|I| \leq t$.

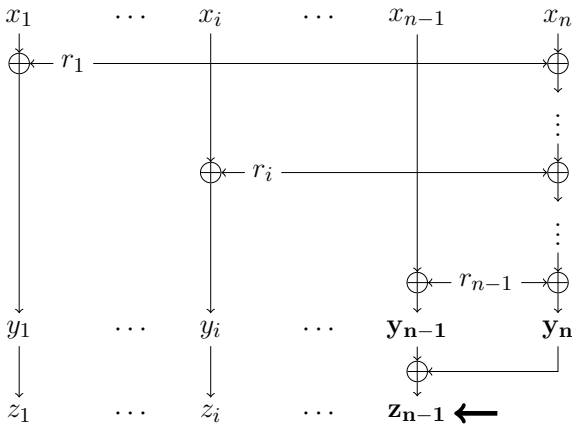

Fig. 4. Illustration of Lemma 8. If z_{n-1} is probed, then both y_{n-1} and y_n must be simulated, so we can apply Lemma 6 or Lemma 7 with probed y_n. If z_{n-1} is not probed, the simulation is straightforward, as in Lemma 2.

Note that the value $x_1 \oplus \cdots \oplus x_n$ in the above lemma corresponds to either $a_2 \oplus \cdots \oplus a_{n+1}$ or $b_1 \oplus \cdots \oplus b_n$ at Step 3 of our conversion algorithm from Section 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 $a_2 \oplus \cdots \oplus a_{n+1}$ is uniform because of Lemma 3, and can therefore be simulated by a random value. Similarly we have:

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

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

4.7 Proof of Theorem 3

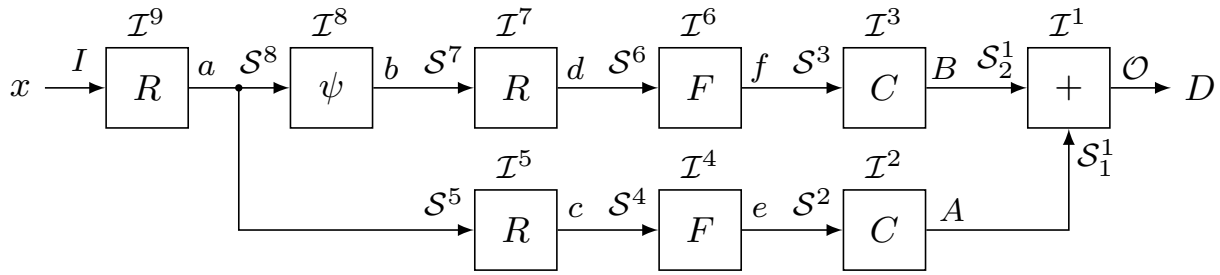

Fig. 5. The sequence of gadgets in the Boolean to arithmetic conversion algorithm.

In this section we show recursively that our conversion algorithm $\mathcal{C}_n$ described in Section 4.3 is t-SNI, based on the previous lemmas on RefreshMasks. We follow the same process as in [BBD+15, 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 5.

The gadgets are numbered from 1 to 9, and we denote by $\mathcal{I}^i$ the set of probed variables in Gadget i. Gadget 9 corresponds to the initial Refreshmasks at Step 1, which we denote by R. Gadget 8 is denoted by Ψ for the application of the Ψ function at Step 2. Gadgets 5 and 7 denote the two RefreshMasks performed at Step 3. Gadgets 4 and 6 denote the compression to $n-1$ shares performed at Step 4. Gadgets 2 and 3 denote the recursive application of the conversion algorithm at Step 5. Finally, Gadget 1 denotes the additive grouping performed at Step 6.

From Lemma 1 the t-SNI condition is satisfied for $n = 2$. We now assume that $\mathcal{C}_{n-1}$ is t-SNI, and we must show that $\mathcal{C}_n$ is t-SNI. We denote by t_c the total number of probes in the circuit. Let $O \subset [1, n]$ be any subset of output shares. We must show that under the condition:

$$t_c + |\mathcal{O}| < n$$

all t_c probed variables in the circuit and all variables $D_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$, for some subset I satisfying $|I| \leq t_c$.

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:

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

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 are therefore respectively:

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

We stress that the two conditions in (3) are not necessarily satisfied; for example the adversary could probe the $n-1$ shares A_i's directly which would give $|\mathcal{S}_1^1| = n-1$. However, we show that at least one of the two conditions in (3) must be satisfied. Namely we obtain from (1) and (2), using $|P_1| \leq |\mathcal{I}^1|$:

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

This gives using $|\mathcal{I}^1| + |\mathcal{I}^2| + |\mathcal{I}^3| \leq t_c$ and $t_c + |\mathcal{O}| < n$:

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

Now if none of the two conditions in (3) is satisfied, we get $|\mathcal{S}_1^1| + |\mathcal{I}^2| \geq n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| \geq n-1$, hence $|\mathcal{S}_1^1| + |\mathcal{S}_2^1| + |\mathcal{I}^2| + |\mathcal{I}^3| \geq 2n-2$, which contradicts the previous inequality. Therefore at least one of the t-SNI conditions for gadgets 2 and 3 must be satisfied. We can therefore distinguish 3 cases, depending on which of the two conditions in (3) are satisfied:

- $|\mathcal{S}_1^1| + |\mathcal{I}^2| < n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| \geq n-1$. We obtain using $|\mathcal{S}_2^1| \leq |\mathcal{O}| + |\mathcal{I}^1|$ from (2):

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

and therefore we must have $|\mathcal{O}| + |\mathcal{I}^1| + |\mathcal{I}^3| = |\mathcal{O}| + t_c = n-1$, which gives $|\mathcal{I}^1| + |\mathcal{I}^3| = t_c$, which implies that there are no other probes in the circuit. In particular, we must have $|\mathcal{I}^2| = 0$, and since the t-SNI condition for Gadget 2 is satisfied, we can simulate all outputs of Gadget 2 with $|\mathcal{S}^2| \leq |\mathcal{I}^2| = 0$, hence we can take $\mathcal{S}^2 = \emptyset$. This implies that for gadgets 4 and 5, we can take $\mathcal{S}^4 = \emptyset$ and $\mathcal{S}^5 = \emptyset$ (see Fig. 5).
However, since the t-SNI condition for Gadget 3 is not satisfied, we must take $\mathcal{S}^3 = [1, n-1]$ and therefore $\mathcal{S}^6 = [1, n]$. Since in that case the RefreshMasks at Gadget 7 is not probed, we can apply Lemma 4 and all the d_i's can be perfectly simulated from the knowledge of

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

only. Moreover since $\mathcal{S}^5 = \emptyset$ and $\mathcal{I}^8 = \emptyset$, this is the only variable that must be simulated. Letting $x = x_1 \oplus \cdots \oplus x_n = a_1 \oplus \cdots \oplus a_{n+1}$, we get:

$$b_1 \oplus \cdots \oplus b_n = x - x \oplus a_1$$

Thanks to the initial RefreshMasks (Gadget 9) which is not probed in that case, this has the uniform distribution (because a_1 has the uniform distribution from from Lemma 3), and this can therefore be perfectly simulated by a random value. Therefore we can take $I = \emptyset$ for the simulation of the entire circuit.

- $|\mathcal{S}_1^1| + |\mathcal{I}^2| \geq n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| < n-1$. The reasoning is similar to the previous case, with $|\mathcal{I}^1| + |\mathcal{I}^2| = t_c$ and the rest of the circuit is not probed. 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 $\mathcal{S}^6 = \mathcal{S}^7 = \mathcal{S}^8 = \emptyset$. Since the t-SNI condition for Gadget 2 is not satisfied we must take $\mathcal{S}^2 = [1, n-1]$ and therefore $\mathcal{S}^4 = [1, n]$. Since the RefreshMasks at Gadget 5 is not probed in that case, we can apply Lemma 4 and all the variables c_i can be simulated from the knowledge of $a_2 \oplus \cdots \oplus a_{n+1}$ only. Since the initial RefreshMasks is not probed in that case, applying Lemma 3 such simulation can be performed without the knowledge of any of the input x_i's, that is with $I = \emptyset$.
- $|\mathcal{S}_1^1| + |\mathcal{I}^2| < n-1$ and $|\mathcal{S}_2^1| + |\mathcal{I}^3| < n-1$. This means that t-SNI condition for gadgets 2 and 3 is satisfied. Therefore, from the recursive hypothesis we have that the probed variables and output variables $A_{|\mathcal{S}_1^1}$ and $B_{|\mathcal{S}_2^1}$ of gadgets 2 and 3 can be perfectly simulated from the input variables $e_{|\mathcal{S}^2}$ and $f_{|\mathcal{S}^3}$ respectively, where:

$$|\mathcal{S}^2| \leq |\mathcal{I}^2| \text{ and } |\mathcal{S}^3| \leq |\mathcal{I}^3| \tag{4}$$

For the rest of the proof, we can therefore assume that (4) is satisfied, since in the two other cases above the simulation of all probed variables and $D_{|\mathcal{O}}$ can be performed with $I = \emptyset$.

Gadgets 4 and 5. From $|\mathcal{S}^2| \leq |\mathcal{I}^2|$ we must have $|\mathcal{S}^2| + |\mathcal{I}^4| + |\mathcal{I}^5| \leq |\mathcal{I}^2| + |\mathcal{I}^4| + |\mathcal{I}^5| \leq n-1$. We apply Lemma 8 to gadgets 4 and 5, where the number of probes in Lemma 8 is $t = |\mathcal{S}^2| + |\mathcal{I}^4| + |\mathcal{I}^5| \leq n-1$. As in Lemma 8, we must therefore distinguish two cases:

- If $|\mathcal{S}^2| + |\mathcal{I}^4| + |\mathcal{I}^5| < n-1$, then all probed intermediate variables and output variables in $e_{|\mathcal{S}^2}$ can be perfectly simulated from $a_{|\mathcal{S}^5}$, where:

$$|\mathcal{S}^5| \leq |\mathcal{S}^2| + |\mathcal{I}^4| + |\mathcal{I}^5| \leq |\mathcal{I}^2| + |\mathcal{I}^4| + |\mathcal{I}^5|$$

- If $|\mathcal{S}^2|+|\mathcal{I}^4|+|\mathcal{I}^5| = n-1$, from $|\mathcal{S}^2| \leq |\mathcal{I}^2|$ we must have $|\mathcal{I}^2|+|\mathcal{I}^4|+|\mathcal{I}^5| = n-1$, which implies that the rest of the circuit is not probed. This implies as previously that $\mathcal{S}^3 = \mathcal{S}^6 = \mathcal{S}^7 = \mathcal{S}^8 = \emptyset$. Applying Lemma 8, either all probed variables in gadgets 4 and 5 and output variables $e_{|\mathcal{S}^2}$ can be perfectly simulated from $a_2 \oplus \cdots \oplus a_{n+1}$, or there exists a subset $\mathcal{S}^5$ with $|\mathcal{S}^5| \leq n-1$ such that they can be perfectly simulated from $a_{|\mathcal{S}^5}$.
  In the first case, as previously we can apply Lemma 3 to the initial RefreshMasks which is not probed (moreover $\mathcal{S}^8 = \emptyset$), and $a_2 \oplus \cdots \oplus a_{n+1}$ can be simulated by a random value. Hence the full circuit can be simulated with $I = \emptyset$. In the second case, we still have as previously $|\mathcal{S}^5| \leq |\mathcal{I}^2| + |\mathcal{I}^4| + |\mathcal{I}^5|$.

For the rest of the proof, we can therefore assume that all probed variables in gadgets 4 and 5 and output variables $e_{|\mathcal{S}^2}$ can be perfectly simulated from $a_{|\mathcal{S}^5}$, with:

$$|\mathcal{S}^5| \leq |\mathcal{I}^2| + |\mathcal{I}^4| + |\mathcal{I}^5| \tag{5}$$

since in the previous first case, all probed variables and output variables $D_{|\mathcal{O}}$ can be perfectly simulated with $I = \emptyset$.

Gadgets 6 and 7. The reasoning is essentially the same as for gadgets 4 and 5. Namely from $|\mathcal{S}^3| \leq |\mathcal{I}^3|$ we must have $|\mathcal{S}^3| + |\mathcal{I}^6| + |\mathcal{I}^7| \leq |\mathcal{I}^3| + |\mathcal{I}^6| + |\mathcal{I}^7| \leq n-1$. We apply Lemma 8 to gadgets 6 and 7, where the number of probes in Lemma 8 is $t = |\mathcal{S}^3| + |\mathcal{I}^6| + |\mathcal{I}^7| \leq n-1$. As previously, we distinguish two cases:

- If $|\mathcal{S}^3| + |\mathcal{I}^6| + |\mathcal{I}^7| < n-1$, then all probed intermediate variables and output variables in $f_{|\mathcal{S}^3}$ can be perfectly simulated from $b_{|\mathcal{S}^7}$, where:

$$|\mathcal{S}^7| \leq |\mathcal{S}^3| + |\mathcal{I}^6| + |\mathcal{I}^7| \leq |\mathcal{I}^3| + |\mathcal{I}^6| + |\mathcal{I}^7|$$

- If $|\mathcal{S}^3| + |\mathcal{I}^6| + |\mathcal{I}^7| = n-1$, we must have $|\mathcal{I}^3| + |\mathcal{I}^6| + |\mathcal{I}^7| = n-1$ and the rest of the circuit is not probed. Applying Lemma 8, either all probed variables in gadgets 6 and 7 and output variables $f_{|\mathcal{S}^3}$ can be perfectly simulated from $b_1 \oplus \cdots \oplus b_n$, or there exists a subset $\mathcal{S}^7$ with $|\mathcal{S}^7| \leq n-1$ such that they can be perfectly simulated from $b_{|\mathcal{S}^7}$.
  In the first case, as previously the simulation can be performed from the knowledge of $b_1 \oplus \cdots \oplus b_n = x - a_2 \oplus \cdots \oplus a_{n+1}$, which has the uniform distribution thanks to the initial RefreshMasks which is not probed, with $\mathcal{S}^5 = \emptyset$. Hence the full circuit can be simulated with $I = \emptyset$. In the second case, we still have as previously $|\mathcal{S}^7| \leq |\mathcal{I}^3| + |\mathcal{I}^6| + |\mathcal{I}^7|$.

  For the rest of the proof, we can therefore assume that:

$$|\mathcal{S}^7| \leq |\mathcal{I}^3| + |\mathcal{I}^6| + |\mathcal{I}^7| \tag{6}$$

Gadget 8. Since by definition we have $b_1 \leftarrow (\overline{n \wedge 1}) \cdot a_1 \oplus \Psi(a_1, a_2)$ and $b_i \leftarrow \Psi(a_1, a_{i+1})$ for all $2 \leq i \leq n$, the probed variables and output variables $b_{|\mathcal{S}^7}$ can be simulated with the knowledge of $a_{|\mathcal{S}^8}$, where:

$$|\mathcal{S}^8| \leq |\mathcal{I}^8| + |\mathcal{S}^7| + 1 \tag{7}$$

and we have $1 \in \mathcal{S}^8$, since the a_1 variable appears in the computation of all the b_i's.

Gadget 9. We can apply Lemma 5, where the total number of probes is $t' = |\mathcal{S}^8| + |\mathcal{S}^5| + |\mathcal{I}^9|$, and moreover the output variable a_1 is probed within Lemma 5, since $1 \in \mathcal{S}^8$. We obtain that all variables can be perfectly simulated from $x_{|I}$, where from (7):

$$\begin{aligned}|I| &\leq t' - 1 \leq |\mathcal{S}^8| + |\mathcal{S}^5| + |\mathcal{I}^9| - 1 \\ &\leq |\mathcal{I}^8| + |\mathcal{S}^7| + 1 + |\mathcal{S}^5| + |\mathcal{I}^9| - 1 \leq |\mathcal{I}^8| + |\mathcal{S}^7| + |\mathcal{S}^5| + |\mathcal{I}^9|\end{aligned}$$

Eventually, using (5) and (6), we obtain:

$$|I| \leq |\mathcal{I}^2| + |\mathcal{I}^3| + |\mathcal{I}^4| + |\mathcal{I}^5| + |\mathcal{I}^6| + |\mathcal{I}^7| + |\mathcal{I}^8| + |\mathcal{I}^9|$$

which gives $|I| \leq t_c$ as required. In summary all t_c probed variables in the circuit and all output variables $D_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$ with $|I| \leq t_c$. Hence the t-SNI condition is satisfied for $\mathcal{C}_n$. This terminates 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 countermeasures 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', $r_1, \ldots r_n$, where:

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

We copy Equation (24) from [HT16]:

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

The above equation means that the variable

$$X = ((\neg n \wedge 1)\beta) \oplus \bigoplus_{i=1}^{n-1} \Psi(\beta, \delta_i) \oplus \Psi\left(\beta, \alpha \oplus r_n \oplus \bigoplus_{i=1}^{n-1} \delta_i \oplus r_i\right) \tag{8}$$

is explicitly computed, using a certain sequence of operations following from the right-hand side of the equation. From the affine property of the Ψ 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:

$$X = \Psi(\beta, u) = \beta \oplus u - u$$
$$x = x' \oplus \alpha \oplus u$$

Therefore, if the variable β is explicitly computed when evaluating (8), our attack works by probing the 4 variables β, X, α and x'. From β and $X = \beta \oplus u - u$, we obtain information about u. From α and x', this reveals information about $x = x' \oplus \alpha \oplus u$.

Alternatively, if the variable β is not explicitly computed[4], 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, α and x'. We obtain the two variables:

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

and one can check that for randomly distributed β, δ_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 β, which combined with $X = \Psi(\beta, u) = \beta \oplus u - u$ leaks information about u. From α 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

As previously we have as input the $n + 1$ shares:

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

We copy Equation (22) from [HT16]:

$$\left(x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right)\right) \oplus \bigoplus_{i=1}^{n-2} \mu_i = ((n \wedge 1)(x' \oplus \alpha)) \oplus \left(\bigoplus_{i=1}^{n-2} \Psi(x' \oplus \alpha, \gamma_i) \oplus \mu_i\right)$$
$$\oplus \Psi(x' \oplus \alpha, \gamma_{n-1}) \oplus \Psi(x' \oplus \alpha, \gamma_n) \oplus \Psi\left(x' \oplus \alpha, \alpha \oplus \bigoplus_{i=1}^{n} \gamma_i \oplus r_i\right)$$

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

where $\alpha, \mu_1, \ldots, \mu_{n-2}, \gamma_1, \ldots, \gamma_n$ are randomly generated values[5]. The previous equation means that the intermediate variable:

$$X = \left(x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i \right) \right) \oplus \bigoplus_{i=1}^{n-2} \mu_i$$

is explicitly computed, where the computation is performed according to the right-hand side of the equation. Letting as previously $u := \alpha \oplus \bigoplus_{i=1}^{n} r_i$, we obtain:

$$X = (x + u) \oplus \bigoplus_{i=1}^{n-2} \mu_i$$
$$x = (x' \oplus \alpha) \oplus u$$

If the variable $x' \oplus \alpha$ is explicitly computed when computing the right-hand side of [HT16, Equation (22)] recalled above, we can probe the following n variables: X, $x' \oplus \alpha$ and $\mu_1, \ldots, \mu_{n-2}$. We can then compute:

$$Y = X \oplus \bigoplus_{i=1}^{n-2} \mu_i = x + u$$

From $x' \oplus \alpha = x \oplus u$ we also know $x \oplus u$. It is easy to see that for randomly distributed u, the joint variable $(x' \oplus \alpha, Y) = (x \oplus u, x + u)$ leaks information about x.

If the variable $x' \oplus \alpha$ is not explicitly computed when computing the right-hand side of [HT16, Equation (22)], the variable $Z = \Psi(x' \oplus \alpha, \gamma_n)$ must still be computed, which still leaks information on $x' \oplus \alpha$. More precisely, our attack consists in probing the n variables X, $\mu_1, \ldots, \mu_{n-2}$ and Z. As previously, we recover $Y = x + u$ and from $x' \oplus \alpha = x \oplus u$ we can therefore obtain the two variables:

$$x + u, \quad \Psi(x \oplus u, \gamma_n)$$

One can check that for uniformly distributed u and γ_n, this still leaks information about x. Therefore in both cases we have an attack of order n against the n-th order algorithm.

Application to the third-order algorithm. We show that our attack can be applied in particular against the third-order algorithm explicitly described in [HT16, Algorithm 3]. The algorithm takes as input the 4 shares x', r_1, r_2 and r_3, with:

$$x = x' \oplus r_1 \oplus r_2 \oplus r_3$$

We probe the following 3 intermediate variables of [HT16, Algorithm 3]; we refer to [HT16, Appendix B] for the list of intermediate variables that appear during the computation.

$$V_3 = r_2, \quad V_{29} = (x' \oplus \gamma_2 \oplus \alpha) + \gamma_2, \quad V_{43} = [x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)] \oplus r_2$$

We have:

$$V_{29} = (x \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha \oplus \gamma_2) + \gamma_2$$
$$V_3 \oplus V_{43} = x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)$$

[5] Moreover in this equation one uses $\Psi(x, u) = x \oplus u + u$ instead of $\Psi(x, u) = x \oplus u - u$, but this does not change anything in the attack, so for simplicity we keep the same notation.

One can let as previously $u := r_1 \oplus r_2 \oplus r_3 \oplus \alpha$, which gives:

$$V_{29} = (x \oplus u \oplus \gamma_2) + \gamma_2$$
$$V_3 \oplus V_{43} = x + u$$

and as mentioned previously one can check that for random u, γ_2 the two variables leak information about x. Hence we have a 3-rd order attack against the 3-rd order algorithm.

6 Operation Count and Implementation

As shown in Section 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 $\mathcal{O}(k \cdot n^2)$. In Appendix B we estimate the operation count of the conversion algorithm from [CGV14]; we obtain similar results as the estimate provided in [HT16]. We summarize in Table 1 the operation count for [CGV14] (for $k = 32$) and for our new algorithm from Section 4.3. We see that for small orders t, our new countermeasure is at least one order of magnitude faster than previous work.

B → 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 (Section 4.3)		55	155	367	803	1 687	7 039	28 519	114 511

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

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.

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
Our algorithm (Section 4.3)	45	119	281	611	1 270	5 673	22 192	87 322

Table 2. 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, and Benjamin Grégoire. Compositional verification of higher-order masking: Application to a verifying masking compiler. Cryptology ePrint Archive, Report 2015/506, 2015. http://eprint.iacr.org/.

[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.

[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.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, 1999.

[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.

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

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

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In *CHES*, pages 158–172, 1999.

[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.

[PR13] Emmanuel Prouff and Matthieu Rivain. 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*, pages 142–159, 2013.

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

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

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

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

if $n = 2$ **then**
  $D_1, D_2 \leftarrow \mathsf{GoubinSNI}(x_1, x_2)$
  return D_1, D_2
end if
$a_1, \ldots, a_{n+1} \leftarrow \mathsf{RefreshMasks}_{n+1}(x_1, \ldots, x_n, 0)$
$b_1 \leftarrow (\overline{n \wedge 1}) \cdot a_1 \oplus \Psi(a_1, a_2)$
for $i = 2$ to n **do**
  $b_i \leftarrow \Psi(a_1, a_{i+1})$
end for
$c_1, \ldots, c_n \leftarrow \mathsf{RefreshMasks}_n(a_2, \ldots, a_{n+1})$
$d_1, \ldots, d_n \leftarrow \mathsf{RefreshMasks}_n(b_1, \ldots, b_n)$
$e_1, \ldots, e_{n-2} \leftarrow c_1, \ldots, c_{n-2}$ and $e_{n-1} \leftarrow c_{n-1} \oplus c_n$
$f_1, \ldots, f_{n-2} \leftarrow d_1, \ldots, d_{n-2}$ and $f_{n-1} \leftarrow d_{n-1} \oplus d_n$
$A_1, \ldots, A_{n-1} \leftarrow \mathcal{C}_{n-1}(e_1, \ldots, e_{n-1})$
$B_1, \ldots, B_{n-1} \leftarrow \mathcal{C}_{n-1}(f_1, \ldots, f_{n-1})$
for $i = 1$ to $n - 2$ **do**
  $D_i \leftarrow A_i + B_i$
end for
$D_{n-1} \leftarrow A_{n-1}$
$D_n \leftarrow B_{n-1}$
return $D_1, \ldots, D_n$

B Operation Count of [CGV14]

In this section we estimate the number of operations for the Boolean to arithmetic conversion algorithm from [CGV14].

The complexity of SecAnd algorithm in [CGV14, Algorithm 1] is the same as for the AND gadget in [ISW03], and is given by:

$$A_n = 5 \cdot \frac{n(n-1)}{2} + n^2 = \frac{7n^2}{2} - \frac{5n}{2}$$

For the SecAddGoubin algorithm in [CGV14, Algorithm 3], this gives:

$$B_n = A_n + n + (k-1) \cdot (A_n + 2n) + 2n = k \cdot (A_n + 2n) + n$$

For the Arithmetic to Boolean conversion from [CGV14, Algorithm 4], we have the recursion:

$$T_n = 2 \cdot \left(T_{n/2} + \frac{3n}{2} \right) + B_n$$

and $T_1 = 1$. For simplicity we assume that n is a power of 2; if this is not the case, the equation becomes more complex with two recursive calls of $T_{\lfloor n/2 \rfloor}$ and $T_{\lceil n/2 \rceil}$ operations respectively. Finally, for the Boolean to arithmetic conversion, we get:

$$T'_n = 2n + T_n + A_n + 3 \cdot n^2 + n$$

As shown in [CGV14], the complexity of both conversion algorithms is $\mathcal{O}(k \cdot n^2)$. Based on the above equations, we provide the operation count for small values of n and $k = 32$ in Section 6.