High Order Masking of Look-up Tables with Common Shares

Jean-Sébastien Coron[1], Franck Rondepierre[2], and Rina Zeitoun[2]

[1] University of Luxembourg
jean-sebastien.coron@uni.lu
[2] Oberthur Technologies, Cryptography and Security Group, France
{f.rondepierre,r.zeitoun}@oberthur.com

Abstract. Masking is an effective countermeasure against side-channel attacks. In this paper, we improve the efficiency of the high-order masking of look-up tables countermeasure introduced at Eurocrypt 2014, based on a combination of three techniques, and still with a proof of security in the Ishai-Sahai-Wagner (ISW) probing model. The first technique consists in proving security under the stronger t-SNI definition, which enables to use $n = t + 1$ shares instead of $n = 2t + 1$ against t-th order attacks. The second technique consists in progressively incrementing the number of shares within the countermeasure, from a single share to n, thereby reducing the complexity of the countermeasure. The third technique consists in adapting the common shares approach introduced by Coron *et al.* at CHES 2016, so that half of a randomized look-up table can be pre-computed for multiple SBoxes. When combined, our three techniques lead to a factor 10.7 improvement in efficiency, asymptotically for a large number of shares n. For a practical implementation with a reasonable number of shares, we get a 4.8 speed-up factor compared to the initial countermeasure for AES.

1 Introduction

The masking countermeasure. Masking is an effective countermeasure against side-channel attacks. As first suggested in [CJRR99, GP99], it consists in xoring every internal variable x with a random r, so that r and the masked variable $x' = x \oplus r$ are processed separately. This implies that a first-order attack will reveal no information to the attacker, because the power consumption at a single point has the same distribution as the power consumption of a random value. However, a second-order attack can still break such first-order masking, by combining information from two leakage points; see [OMHT06] for a practical attack.

More generally, Boolean masking can be extended to n shares by letting $x = x_1 \oplus \cdots \oplus x_n$. The goal is then to process the shares x_i separately, so that the implementation should be resistant against t-th order attacks, in which the adversary combines leakage information from at most $t < n$ variables. High order masking is a sound approach because as shown in [CJRR99, PR13, DDF14], the number of noisy samples required to recover a secret x from its shares x_i grows exponentially with the number of shares.

The ISW probing model and t-SNI security. Ishai, Sahai and Wagner [ISW03] initiated the theoretical study of securing circuits against an adversary who can probe a fraction of its wires. They showed how to transform any circuit of size $|C|$ into a circuit of size $\mathcal{O}(|C| \cdot t^2)$ secure against any adversary who can probe at most t wires. The construction is based on secret-sharing every variable x into n shares with $x = x_1 \oplus \cdots \oplus x_n$, and processing the shares in a way that prevents a t-limited adversary from learning any information about the initial variable x, for $n \geq 2t + 1$.

In the ISW model, the approach for proving security is based on simulation, by showing that any set of t probes can be perfectly simulated without the knowledge of any input variable of the original

circuit (in particular, the secret-key). This demonstrates that the t probes cannot help the attacker, since he could simulate those t probes by himself, without knowing the secret key. More precisely, the simulation is usually done by iteratively constructing a subset I of indices of the input shares x_i that are sufficient to simulate the t probes; if we can ensure that $|I| < n$, then only a proper subset of the input shares is required for the simulation, and such input shares can be generated without knowing the original input variable, simply by generating independently and uniformly distributed bits. In the ISW security proof of the n-shared AND gadget, every probe adds at most two indices in I, so for t probes we get $|I| \leq 2t$ and therefore $n \geq 2t + 1$ is sufficient to achieve perfect secrecy against a t-limited adversary. As shown in [ISW03], the simulation performed at the gadget level easily extends to the full circuit, by maintaining a global subset of indices I as in a single gate.

Recently, a refined security definition under the ISW probing model was introduced in [BBD+15], called t-SNI security. This stronger definition enables to prove that a gadget can be used in a full construction with $n \geq t + 1$ shares only, instead of $n \geq 2t + 1$ for the weaker definition of t-NI security (corresponding to the original ISW security proof). The new definition is very practical as it enables modular security proofs, by first considering the t-SNI security of individual gadgets and then composing them in a more complex construction. Since in this paper we are interested in improving the efficiency of side-channel countermeasures, we always prove the security of our algorithms under the stronger t-SNI definition.

The Rivain-Prouff countermeasure. The first provably secure higher-order masking scheme for the AES block-cipher was described by Rivain and Prouff in [RP10], by adapting the ISW multiplication gadget to the AES finite field $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$. More precisely, since the non-linear part of the AES SBox can be written as $S(x) = x^{254}$ over $\mathbb{F}_{2^8}$, it can be evaluated as a sequence of non-linear multiplications and linear squarings, moreover with only 4 non-linear multiplications. In order to achieve resistance against an attack of order t, the Rivain-Prouff algorithm requires $n \geq 2t + 1$ shares, as in the original ISW construction. This was later improved to $n \geq t + 1$ by showing that the ISW multiplication gadget achieves the t-SNI property [BBD+15]. This enables to use the Rivain-Prouff countermeasure for the full AES with $n = t + 1$ shares only (with some additional mask refreshing; see [BBD+15]).

The Rivain-Prouff countermeasure was later extended to any look-up table by Carlet, Goubin, Prouff, Quisquater and Rivain (CGPQR) in [CGP+12]. Any given k-bit SBox can be represented by a polynomial $\sum_{i=0}^{2^k-1} a_i\, x^i$ over $\mathbb{F}_{2^k}$ using Lagrange's interpolation theorem, and can therefore be securely evaluated with a sequence of n-shared additions, squarings and multiplications. Asymptotically, the running time of the countermeasure is dominated by the number of non-linear multiplications, which have complexity $\mathcal{O}(n^2)$.

To minimize the number of such non-linear multiplications, the authors described a technique called Parity-Split, with a proven complexity of $\mathcal{O}(2^{k/2})$ non-linear multiplications for k-bit SBoxes. This generic technique was later improved by Roy and Vivek [RV13] to further reduce the number of non-linear multiplications for various concrete SBoxes, for example DES with 7 multiplications (from 10). This was further improved in [CRV14], with a generic technique for fast polynomial evaluation in $\mathbb{F}_{2^k}$, with heuristic complexity $\mathcal{O}(2^{k/2}/\sqrt{k})$; using this method, the DES SBoxes require only 4 multiplications. In summary, the asymptotic running time of the Rivain-Prouff countermeasure for AES is $\mathcal{O}(n^2)$, and for k-bit generic SBoxes the running time is $\mathcal{O}(2^{k/2} \cdot n^2)$.[1]

[1] or $\mathcal{O}(2^{k/2}/\sqrt{k} \cdot n^2)$ using the heuristic technique from [CRV14].

The randomized table countermeasure. A completely different high-order countermeasure for SBox evaluation was described in [Cor14], based on table randomization. The countermeasure is an extension of the classical first-order randomized table countermeasure, first described in [CJRR99]. The first-order countermeasure consists in re-computing in RAM the original SBox S with inputs shifted by some random r and with masked outputs. More precisely, one computes the randomized table

$$T(u) = S(u \oplus r) \oplus s$$

for all $u \in \{0,1\}^k$, where $r \in \{0,1\}^k$ is the input mask and $s \in \{0,1\}^k$ is the output mask. To evaluate $S(x)$ from the masked value $x' = x \oplus r$, it suffices to compute $y' = T(x')$, which gives $y' = T(x') = S(x' \oplus r) \oplus s = S(x) \oplus s$, and therefore y' is a masked value for $S(x)$.

The first-order countermeasure was generalized to any order in [Cor14] as follows. Every row of the randomized table T now consists of n shares. One starts with an n-encoding of the original SBox S, with:

$$T(u) = (S(u), 0, \ldots, 0) \tag{1}$$

for all rows $u \in \{0,1\}^k$, and one progressively shifts the table T by the successive input shares $x_1, \ldots, x_{n-1}$. Between every shift one refreshes the n-shared encodings on each row of the table. After the last shift by x_{n-1} the table T satisfies for all $u \in \{0,1\}^k$:

$$\bigoplus_{j=1}^{n} T(u)[j] = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1}) \tag{2}$$

Therefore it suffices to evaluate the table at the last input share x_n to get the n output shares y_i. More precisely, one lets $(y_1, \ldots, y_n) \leftarrow T(x_n)$, which from (2) gives as required:

$$y_1 \oplus \cdots \oplus y_n = S(x_n \oplus x_1 \oplus \cdots \oplus x_{n-1}) = S(x)$$

As explained in [Cor14], the intermediate mask refreshing are necessary; for example if only a single mask refreshing is performed after the initialization of $T(u) = (S(u), 0, \ldots, 0)$, the adversary could probe a table output share at the beginning and after the table shifts, which would leak information about the accumulated shift $x_1 \oplus \cdots \oplus x_{n-1}$ and therefore break the countermeasure.

The above countermeasure is proven secure against any attack of order t in the ISW model, with at least $n = 2t + 1$ shares [Cor14]. The proof works thanks to the following observation: when a given shift by x_i and subsequent mask refreshing has not been probed, the knowledge of x_i is not required for the simulation of the output shares, because when a mask refreshing is not probed, one can always simulate any subset of at most $n - 1$ output shares with randomly generated values. Hence it is possible to perform the simulation of all probed intermediate variables with only a subset of the input shares x_i, which proves the security of the countermeasure in the ISW model.

The asymptotic complexity of the randomized countermeasure for k-bit SBoxes is $\mathcal{O}(2^k \cdot n^2)$, while the CGPQR countermeasure has complexity $\mathcal{O}(2^{k/2} \cdot n^2)$ only. In [Cor14], a variant countermeasure for processors with large register size is described, with the same time complexity $\mathcal{O}(2^{k/2} \cdot n^2)$ as the CGPQR countermeasure, using a similar approach as in [RDP08]. The variant consists in packing multiple SBox rows into a single register, so that the table shifts can be performed more efficiently at the register level first; for example for AES, 4 rows of 8-bit SBox outputs can be stored on the same 32-bit register.

The common shares technique. Recently it was shown in [CGPZ16] that two vectors of n shares used as input of two gadgets can have $n/2$ of their shares in common, without decreasing the security level. This enables to mutualise some part of the computation between the two gadgets, thereby decreasing the running time of the countermeasure. The authors show how to apply this technique to the ISW multiplication gadget, and as shown in [CGPZ16] this saves the equivalent of 1/2 multiplication over 2 ISW multiplications. The technique can be generalized to multiple gadgets processed in parallel, for example the evaluation of the 16 AES SBoxes in the Rivain-Prouff countermeasure. As shown in [CGPZ16], the technique can also be applied to other variants of the Rivain-Prouff countermeasure, such as the quadratic evaluation method described in [CPRR15], and also to the Threshold Implementations approach to resist glitch attacks. In practice, the authors obtained a 20% speed-up compared to existing algorithms.

Our Contributions. The goal of our paper is to improve the practical efficiency of the randomized table countermeasure introduced in [Cor14]. We have the following three contributions:

- We prove the security of the high-order randomized table countermeasure under the stronger t-SNI security definition; this enables to use $n = t + 1$ shares instead of $n = 2t + 1$ for resistance against t-th order attacks, when the countermeasure is integrated inside a larger construction (such as a full block-cipher). This is actually relatively straightforward, because the proof is essentially the same as in [Cor14]. Since the original countermeasure has complexity $\mathcal{O}(2^k \cdot n^2)$, this enables to gain a factor 4 in running time asymptotically.
- We describe a variant of the randomized table countermeasure, in which we progressively increase the number of output shares in the randomized table T, from 1 to n, instead of always n output shares. Since on average we are now using $n/2$ output shares within the countermeasure instead of n, this saves a factor 2 in running time. We prove that our variant countermeasure achieves the same level of security, that is t-SNI security.
- We adapt the common shares approach introduced in [CGPZ16], so that half of the randomized look-up table evaluation can be pre-computed for multiple SBoxes. Namely the randomized table algorithm works by progressively shifting a randomized table T by the successive input shares, so if two n-encodings have half of their first input shares $r_1, \ldots, r_{n/2}$ in common, we can mutualise the first half of the table shifts by $r_1, \ldots, r_{n/2}$. For multiple SBoxes this again saves a factor 2 in complexity. As previously, we provide a security proof under the t-SNI security definition.

Finally, we have performed a practical implementation of our new countermeasures for both AES and DES, using a 32-bit architecture. In theory, the combination of the 3 above techniques should lead to a factor 10.7 improvement in efficiency, asymptotically for a large number of shares n. We report the results of a practical implementation in Section 7, for various number of shares. Our results show that the techniques perform relatively well in practice, as for AES we obtain a speed-up factor of roughly 4.8, and 2.5 for DES.

2 Definitions

In this section we recall the t-NI and t-SNI security notions introduced in [BBD+15]. For simplicity and more concrete definitions, as in [CGPZ16], we consider a gadget taking as input a single n-uple

$(x_i)_{1\leq i\leq n}$ of shares, and outputs a single n-uple $(y_i)_{1\leq i\leq n}$. Given a subset $I \subset [1, n]$, we denote by $x_{|I}$ all elements x_i such that $i \in I$.

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

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

The difference between the t-NI and t-SNI security notions is that the size of the input subset I from which the probed intermediate variables and output shares can be perfectly simulated does not depend on the number of output shares $|\mathcal{O}|$ that must be simulated. As shown in [BBD+15], if several gadgets are t-SNI secure, then the composition of those gadgets remain t-SNI secure. Moreover the t-SNI security notion enables to prove the security of a full construction for $n \geqslant t + 1$ shares, instead of $n \geqslant 2t + 1$ for the weaker t-NI security notion.

3 The Original High Order Look-up Table Algorithm

We recall the algorithm in [Cor14] for securely computing $y = S(x)$, where

$$S : \{0, 1\}^k \to \{0, 1\}^{k'}$$

is a look-up table with k-bit input and k'-bit output. The algorithm takes as input $x_1, \ldots, x_n$ such that $x = x_1 \oplus \cdots \oplus x_n$ and must output $y_1, \ldots, y_n$ such that $y = S(x) = y_1 \oplus \cdots \oplus y_n$, without leaking information about x. The algorithm consists in progressively shifting a randomized table T, using the input shares $x_1, \ldots, x_{n-1}$ for the successive shifts. Each row of the randomized table T is actually a vector of n shares, which encodes the original table $S(x)$ but progressively shifted by $x_1, \ldots, x_i$. Eventually the randomized table is read at index x_n, which gives an n-sharing of $y = S(x)$ as required. Between every shift, the n shares of every row are refreshed using the same RefreshMasks procedure below as in [RP10].

The procedure is described in Algorithm 1. The algorithm uses two temporary tables T and T' in RAM; both have k-bit input and a vector of n elements of k'-bit as output, namely

$$T, T' : \{0, 1\}^k \to (\{0, 1\}^{k'})^n$$

We denote by $T(u)[j]$ and $T'(u)[j]$ the j-th component of the vectors $T(u)$ and $T'(u)$ respectively, for $1 \leq j \leq n$. At Line 2 of Algorithm 1 the table T is initialized with:

$$T(u) \leftarrow \big(S(u), 0, \ldots, 0\big) \in (\{0, 1\}^{k'})^n$$

Given an encoding $\boldsymbol{v} = (v_1, \ldots, v_n)$ with n shares, we denote by

$$\oplus(\boldsymbol{v}) = v_1 \oplus \cdots \oplus v_n$$

Algorithm 1 Masked computation of $y = S(x)$

Input: $x_1, \ldots, x_n$ such that $x = x_1 \oplus \cdots \oplus x_n$
Output: $y_1, \ldots, y_n$ such that $y = S(x) = y_1 \oplus \cdots \oplus y_n$
1: **for all** $u \in \{0,1\}^k$ **do**
2: $\quad T(u) \leftarrow \big(S(u), 0, \ldots, 0\big) \in (\{0,1\}^{k'})^n$ ▷ $\oplus\big(T(u)\big) = S(u)$
3: **end for**
4: **for** $i = 1$ to $n-1$ **do**
5: $\quad$ **for all** $u \in \{0,1\}^k$ **do**
6: $\quad\quad$ **for** $j = 1$ **to** n **do** $T'(u)[j] \leftarrow T(u \oplus x_i)[j]$ ▷ $T'(u) \leftarrow T(u \oplus x_i)$
7: $\quad$ **end for**
8: $\quad$ **for all** $u \in \{0,1\}^k$ **do**
9: $\quad\quad T(u) \leftarrow \mathsf{RefreshMasks}\big(T'(u)\big)$ ▷ $\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_i)$
10: $\quad$ **end for**
11: **end for** ▷ $\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1})$
12: $(y_1, \ldots, y_n) \leftarrow \mathsf{RefreshMasks}\big(T(x_n)\big)$ ▷ $\oplus\big(T(x_n)\big) = S(x)$
13: **return** $y_1, \ldots, y_n$

Algorithm 2 RefreshMasks

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

the encoded element. Therefore initially we have $\oplus\big(T(u)\big) = S(u)$. At Line 6 the table is then shifted by x_1 into T', which gives $\oplus\big(T'(u)\big) = S(u \oplus x_1)$ for all rows u. The rows are then refreshed at Line 9, and we still have $\oplus\big(T(u)\big) = S(u \oplus x_1)$ at Line 10. More generally, one can show recursively that at step i of the loop we have at Line 10:

$$\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_i) \tag{3}$$

for all $u \in \{0,1\}^k$. Namely after the new shift performed at Line 6 with x_{i+1} we obtain:

$$\oplus\big(T'(u)\big) = \oplus\big(T(u \oplus x_{i+1})\big) = S\big((u \oplus x_{i+1}) \oplus x_1 \oplus \cdots \oplus x_i\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{i+1})$$

and therefore the equation still holds at step $i+1$. After all the input shares $x_1, \ldots, x_{n-1}$ have been processed we have:

$$\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1})$$

Therefore from the final look-up table $(y_1, \ldots, y_n) \leftarrow \mathsf{RefreshMasks}\big(T(x_n)\big)$ we obtain $\oplus\big(\boldsymbol{y}\big) = \oplus\big(T(x_n)\big) = S(x_n \oplus x_1 \oplus \cdots \oplus x_{n-1}) = S(x)$ which gives as required:

$$S(x) = y_1 \oplus \cdots \oplus y_n$$

Complexity. We assume that randomness generation takes unit time, as well as table read and write. For n shares the number of operations of RefreshMasks is $3n - 2$. The time complexity of the

countermeasure is therefore:

$$T_n = 2^k \cdot \big(n + (n-1) \cdot (1 + 2n + 3n - 2)\big) + 3n - 2$$

which gives $T_n \simeq 5 \cdot 2^k \cdot n^2$ for large n and 2^k.

Security. We recall the main theorem from [Cor14], proving the security of the countermeasure against t-th order attacks, for any t such that $n \geq 2t + 1$.

Theorem 1 (t-NI of [Cor14]). *Let $(x_i)_{1 \leq i \leq n}$ be the input shares of Algorithm 1 and let t be such that $2t < n$. For any set of t intermediate variables, there exists a subset $I \subset [1, n]$ of indices such that $|I| \leq 2t < n$ and the distribution of those t variables can be perfectly simulated from the shares $x_{|I}$. The output shares $y_{|I}$ can also be perfectly simulated from $x_{|I}$.*

The theorem shows that from any given set of t probed intermediate variables, one can always define a set $I \subset [1, n]$ with $|I| < n$ such that only the knowledge of the input indices $x_{|I} := (x_i)_{i \in I}$ is required to perfectly simulate those t intermediate variables. Then since $|I| < n$, those input shares can be perfectly simulated without knowing the original input variable x, simply by generating independently and uniformly distributed variables. Moreover as shown in [Cor14] the countermeasure can be integrated in a larger construction (for example a full block-cipher), and one still obtains security against t-th order attacks with $n = 2t + 1$ shares.

Variant with large registers. As shown in [Cor14], the efficiency of the randomized table countermeasure can be improved by packing multiple SBox rows into a single register, so that the table shifts can be performed more efficiently at the register level first; for example for AES, 4 rows of the 8-bit SBox output can be stored on the same 32-bit register. We recall this variant in Appendix A.

4 Improved Security Proof for the High-order Look-up Table Countermeasure

Our first contribution in this paper is to prove the security of the high-order look-up table countermeasure recalled under the stronger t-SNI security definition (Definition 2), instead of the weaker t-NI notion used in [Cor14]. As shown in [BBD+15], this enables to use $n = t + 1$ shares instead of $n = 2t + 1$ for resistance against t-th order attacks, when the countermeasure is integrated inside a larger construction, such as a full block-cipher. This is actually relatively straightforward, because the proof is essentially the same as in [Cor14]. Since the original countermeasure has complexity $\mathcal{O}(2^k \cdot n^2)$, this enables to gain a factor 4 in running time asymptotically.

Theorem 2 (t-SNI of [Cor14]). *Let $(x_i)_{1 \leq i \leq n}$ be the input and let $(y_i)_{1 \leq i \leq n}$ be the output of Algorithm 1. For any set of t intermediate variables and any subset O of output indices such that $t + |O| < n$, there exists a subset I of input indices with $|I| \leq t$, such that the t intermediate variables and the output variables $y_{|O}$ can be perfectly simulated from $x_{|I}$.*

We first prove a simple Lemma on the RefreshMasks procedure; see Figure 1 for an illustration.

Lemma 1 (t-NI of RefreshMasks). *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*. For any set of t intermediate variables and any set O of output variables with $t + |O| < n$, there exists a subset I of input indices such that the t intermediate variables and $y_{|O}$ can be perfectly simulated from $x_{|I}$, where $I = O \cup J$ with $|J| \leq t$.*

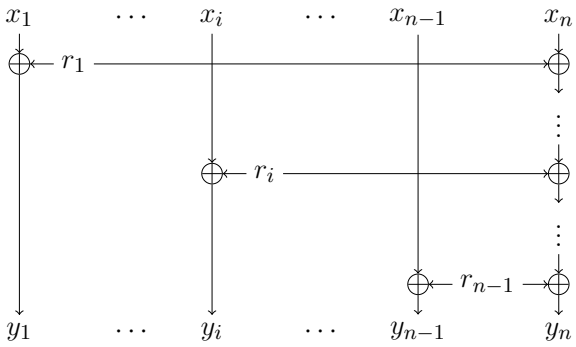

Fig. 1. The RefreshMasks algorithm.

Proof. The set J 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 J. If x_n or y_n or any intermediate variable $y_{n,j}$ is probed, we also add n to J. Since we add at most one index to J per probe, we must have $|J| \leq t$.

The simulation of the probed variables is straightforward. We let $I = O \cup J$. 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 or if $i \in O$, then we must have $i \in I$, so it can be perfectly simulated from $y_i = x_i \oplus r_i$. 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 and all variables $y_{|O}$ can be perfectly simulated from $x_{|I}$. □

As mentioned previously the proof of Theorem 2 is essentially the same as in [Cor14] (with the stronger t-SNI bound $|I| \leq t$ instead of $|I| \leq 2t$). However in this paper we use a recursive approach. Such approach is not strictly necessary for the proof of Theorem 2, but it will be quite useful for proving the security of our improved countermeasure in the next section.

We will actually prove the security of a larger circuit than the original circuit corresponding to Algorithm 1, in order to have a recursive security proof. Namely we assume that instead of performing a final table-look up at Line 12 of Algorithm 1, one performs again a final shift of the full table by x_n, followed by a RefreshMasks of all the rows, and eventually the full table T is returned, instead of only a single row. The output $(y_1, \ldots, y_n)$ in Algorithm 1 then corresponds to the row found at index 0 when the table T has been shifted by x_n. Since we are considering a larger circuit, the t-SNI security of the larger circuit implies the t-SNI security of the original circuit, because when considering a larger circuit we are only giving more power to the adversary.

More precisely, we consider an algorithm HT_i taking as input $x_1, \ldots, x_i$ and outputting a table T with n output shares such that for all $u \in \{0,1\}^k$:

$$\bigoplus_{j=1}^{n} T(u)[j] = S(u \oplus x_1 \oplus \cdots \oplus x_i)$$

The algorithm HT_i can be defined recursively as follows. We define HT_0 as outputting the table T with n output shares, with $T(u) = \big(S(u), 0, \ldots, 0\big)$ for all $u \in \{0,1\}^k$. As illustrated in Figure 2, the algorithm HT_{i+1} takes as input $x_1, \ldots, x_{i+1}$, and the algorithm HT_i is then recursively applied on $x_1, \ldots, x_i$. A shift by x_{i+1} is then applied on the table T returned by HT_i, followed by a RefreshMasks of all the rows of the table T; we denote by SR the combination of the shift by x_{i+1}

and the subsequent RefreshMasks. We denote by $\mathcal{I}^i$ the set of observations made in the gadget HT_i, with $t_i = |\mathcal{I}^i|$ and $\mathcal{I}'$ the set of observations made in the gadget SR.

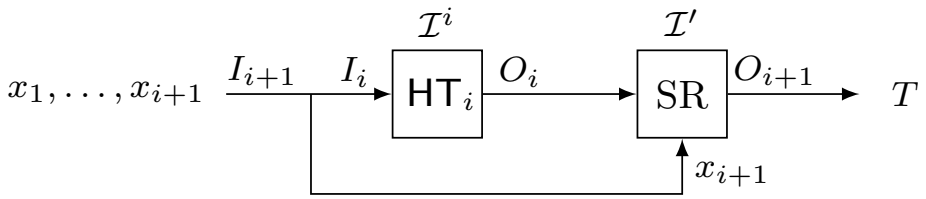

Fig. 2. Illustration of the recursive HT_{i+1} algorithm, with observations $\mathcal{I}^{i+1} = \mathcal{I}^i \cup \mathcal{I}'$.

We prove recursively that the HT_i algorithm satisfies the t-SNI property. This means that for any set of $t_i = |\mathcal{I}^i|$ intermediate variables and any subset O_i of output indices such that $t_i + |O_i| < n$, there exists a subset I_i of input indices with $|I_i| \leq t_i$, such that the t_i intermediate variables and the output variables $T_{|O_i}$ can be perfectly simulated from $x_{|I_i}$. We denote by $T_{|O_i}$ the set of variables $T(u)[j]$ for all $j \in O_i$ and all rows $u \in \{0,1\}^k$.

It is easy to check that HT_0 is t-SNI, since it does not take any share as input, and it processes only public values. Assuming now that HT_i is t-SNI, we must prove that HT_{i+1} is t-SNI. Therefore letting $t_{i+1} = |\mathcal{I}^{i+1}|$ be the number of probes in the circuit, we have the condition:

$$t_{i+1} + |O_{i+1}| < n$$

and we must show that the t_{i+1} probed variables and the output variables $T_{|O_{i+1}}$ can be perfectly simulated from $x_{|I_{i+1}}$, for some I_{i+1} with $|I_{i+1}| \leq t_{i+1}$.

We first consider the SR gadget corresponding to the shift of the table T by x_{i+1}, followed by a RefreshMasks of all the rows of the table T; see Figure 2. We apply Lemma 1 to the RefreshMasks performed on the rows of the table T. We obtain that all probed intermediate variables and all output variables corresponding to O_{i+1} can be perfectly simulated from the input variables corresponding to O_i, where $O_i = O_{i+1} \cup J$ with $|J| \leq |\mathcal{I}'|$.

We now consider the HT_i gadget. We have from $O_i = O_{i+1} \cup J$ and $t_{i+1} = |\mathcal{I}^{i+1}| = |\mathcal{I}^i| + |\mathcal{I}'|$:

$$|\mathcal{I}^i| + |O_i| \leq |\mathcal{I}^i| + |O_{i+1}| + |J| \leq |\mathcal{I}^i| + |O_{i+1}| + |\mathcal{I}'| \leq t_{i+1} + |O_{i+1}| < n$$

Therefore the t-SNI condition is satisfied for Gadget HT_i, and all probed intermediate variables and all output variables corresponding to O_i can be perfectly simulated from $x_{|I_i}$, with $|I_i| \leq |\mathcal{I}^i|$. We now distinguish two cases.

- If $|\mathcal{I}'| = 0$, this means that the SR gadget has not been probed. Since $|O_{i+1}| < n$, thanks to the RefreshMasks of every row we can simulate all output variables corresponding to O_{i+1} without the knowledge of any input variables. In particular, we don't need to know x_{i+1} to perform the simulation, and we can let $I_{i+1} = I_i$.
- If $|\mathcal{I}'| \geq 1$, the knowledge of x_{i+1} is required to perform the simulation of the SR gagdet, so we let $I_{i+1} = I_i \cup \{i+1\}$.

In both cases we obtain $|I_{i+1}| \leq |I_i| + |\mathcal{I}'|$. From $|I_i| \leq |\mathcal{I}^i|$, we obtain:

$$|I_{i+1}| \leq |I_i| + |\mathcal{I}'| \leq |\mathcal{I}^i| + |\mathcal{I}'| \leq |\mathcal{I}^{i+1}| = t_{i+1}$$

which shows that HT_{i+1} is t-SNI. This terminates the proof of Theorem 2.

5 High-order Look-up Table with Increasing Number of Shares

Our second contribution in this paper is to describe a variant countermeasure for high order masking of look-up tables, in which we progressively increase the number of output shares, from 1 to n. Namely considering Algorithm 1, we see that at Line 2 we immediately start with n output shares for the table T. This means that at the beginning of the algorithm we are already using n shares as output while only a few x_i's have been processed, which sounds like a waste of ressources. A natural idea is therefore to start with only a single output share for the table T, and then progressively increase the number of output shares when more x_i's are processed. We obtain the algorithm below.

Algorithm 3 Masked computation of $y = S(x)$, increasing number of shares

Input: $x_1, \ldots, x_n$ such that $x = x_1 \oplus \cdots \oplus x_n$
Output: $y_1, \ldots, y_n$ such that $y = S(x) = y_1 \oplus \cdots \oplus y_n$
1: **for all** $u \in \{0,1\}^k$ **do**
2: $\quad T(u) \leftarrow \big(S(u)\big) \in (\{0,1\}^{k'})^1$ $\quad \triangleright \oplus\big(T(u)\big) = S(u)$
3: **end for**
4: **for** $i = 1$ to $n-1$ **do**
5: $\quad$ **for all** $u \in \{0,1\}^k$ **do**
6: $\quad\quad$ **for** $j = 1$ **to** i **do** $T'(u)[j] \leftarrow T(u \oplus x_i)[j]$ $\quad \triangleright\ T'(u) \leftarrow T(u \oplus x_i)$
7: $\quad$ **end for**
8: $\quad$ **for all** $u \in \{0,1\}^k$ **do**
9: $\quad\quad T(u) \leftarrow (T'(u)[1], \ldots, T'(u)[i], 0)$
10: $\quad\quad T(u) \leftarrow \mathsf{RefreshMasks}_{i+1}\big(T(u)\big)$ $\quad \triangleright \oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_i)$
11: $\quad$ **end for**
12: **end for** $\quad \triangleright \oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1})$ for all $u \in \{0,1\}^k$.
13: $(y_1, \ldots, y_n) \leftarrow \mathsf{RefreshMasks}_n\big(T(x_n)\big)$ $\quad \triangleright \oplus\big(T(x_n)\big) = S(x)$
14: **return** $y_1, \ldots, y_n$

Our new algorithm is therefore similar to Algorithm 1, except that at Line 2 we start with a single share instead of n. Every-time a new input share x_i is processed, we add one more share as output of T at Line 9; therefore at the end of the processing of x_i, the table T has $i+1$ shares as output, and the RefreshMasks are now performed on every row of $i+1$ shares of the table T (instead of n in the original countermeasure). After the processing of x_{n-1} the table T has therefore n shares as output (Line 12). The last input share x_n is then processed by a look-up table at Line 13 as previously. The soundness of the countermeasure is straightforward; formally it can be proven recursively as in Section 3, the only difference being that in the main loop over i the encodings of the rows of T are over $i+1$ shares instead of n.

Complexity. The time complexity of the countermeasure is:

$$T_n = 2^k \cdot \left(1 + \sum_{i=1}^{n-1} (1 + 2i + 3i - 2)\right) + 3n - 2$$

which gives $T_n \simeq \frac{5}{2} \cdot 2^k \cdot n^2$ for large n and 2^k. Therefore asymptotically the new countermeasure is twice as efficient as the original countermeasure from Section 3.

Security. The following theorem shows that the improved algorithm achieves the same level of security as the original countermeasure.

Theorem 3 (t-SNI of Algorithm 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 Algorithm 3. For any set of t intermediate variables and any subset O of output indices such that $t+|O|<n$, there exists a subset I of input indices with $|I|\leq t$, such that the t intermediate variables and the output variables $y_{|O}$ can be perfectly simulated from $x_{|I}$.*

The rest of the section is devoted to the proof of Theorem 3. The proof is more complex than the t-SNI proof of the original algorithm provided in Section 4. Namely, since at step i of the loop the randomized table has only $i+1$ shares as output (instead of n), the adversary could probe all $i+1$ shares of a given row, whose simulation would then require the knowledge of all input shares $x_1,\ldots,x_i$. This is actually not a problem, because the size of the input subset of shares I would still be bounded as $|I|\leq i$, which is according to the SNI bound for $i+1$ probes. However this makes the proof more complex, because we cannot always assume that the t-SNI condition will be recursively satisfied for a subset of the circuit, as it was the case in the proof of Section 4.

Another challenge is to handle the simulation of the output variables $T(u)[j]$ for $j\in O$. Since we have a decreasing number of shares (when starting from the end of the algorithm), we cannot simply apply Lemma 1 for the RefreshMasks gadget, because otherwise at some point we could have $|O|\geq i+1$, and the simulation of the $i+1$ output shares of the randomized table would require the knowledge of all inputs $x_1,\ldots,x_i$. This time this would contradict the t-SNI bound $|I|\leq t$ which can only depend on the number of probes t in the circuit, and not on the size of O. Therefore in the next section we prove a more subtle property of RefreshMasks, showing that this in fact does not happen, because the number of input shares required for the simulation of RefreshMasks can be decreased by one compared to the original lemma.

5.1 Property of RefreshMasks

The proof of Theorem 3 is based on the following lemma concerning the security of RefreshMasks, which is an improvement over Lemma 1. Namely we show that when the last input share x_n of RefreshMasks is such that $x_n=0$ (as it is the case in Algorithm 3 at Step 9), then we can get the improved condition $I=(O\cap[1,n-1])\cup J$ with $|J|\leq t-1$, instead of $I=O\cup J$ with $|J|\leq t$ in Lemma 1. As mentioned previously, this enables to prove the t-SNI security of Algorithm 3 with the increasing number of shares.

Lemma 2 (RefreshMasks with $x_n=0$). *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. Assume that $x_n=0$. For any set of t intermediate variables and any set O of output variables with $t+|O|<n$, there exists a subset J of input indices such that the t intermediate variables and $y_{|O}$ can be perfectly simulated from $x_{|I}$, where $I=(O\cap[1,n-1])\cup J$ with $|J|\leq t-1$, or $I=J$ with $|J|\leq t$.*

Proof. The construction of the set J is performed as follows: for every probed variable x_j or r_j or y_j, we add j to J for any $1\leq j\leq n-1$. Note that we never add n into J since x_n is known, with $x_n=0$. As illustrated in Figure 3, we distinguish two cases.

If x_n or y_n or any intermediate variable $y_{n,j}$ is probed, then we must have $|J|\leq t-1$ since in that case we have considered at most $t-1$ probes in the construction of J. Moreover the simulation

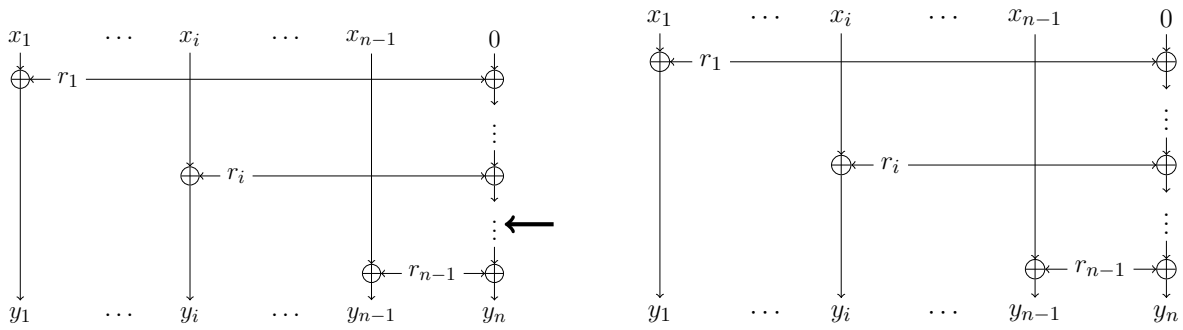

Fig. 3. Illustration of Lemma 2. Case 1 (left): the adversary has spent at least one probe on the last column for which $x_n = 0$, therefore we can have $|J| \leq t - 1$. Case 2 (right): no intermediate variable is probed on the last column, except possibly y_n; therefore r_i can play the role of a one-time pad for the simulation of y_i, hence x_i is not required and $|I| \leq t$.

of the t probed variables and $y_{|O}$ is straightforward and proceeds as in the proof of Lemma 1, by letting $I = (O \cap [1, n-1]) \cup J$. Namely all intermediate variables r_i are simulated by generating a uniform independent value as in the original algorithm. For any probed variable x_i or y_i with $1 \leq i \leq n-1$, we must have $i \in J \subset I$ and therefore y_i can be perfectly simulated from x_i with $y_i = x_i \oplus r_i$. This is also the case for the output variables y_i with $i \in O$. From $x_n = 0$, we can also perfectly simulate all intermediate variables $y_{n,j}$ and the output y_n.

We now assume that no variable x_n, y_n or $y_{n,j}$ is probed. From the construction of J we still have $|J| \leq t$. We show that the t probed variables and $y_{|O}$ can be perfectly simulated from $x_{|I}$, where $I = J$. Note that although y_n is not among the t probes we can still have $n \in O$ and in that case y_n must still be simulated. The simulation of the probed variables x_i, r_i and y_i for $1 \leq i \leq n-1$ is straightforward. Namely in that case we have $i \in J$ and we can generate r_i as in the real circuit; the variables x_i and $y_i = x_i \oplus r_i$ are also simulated as in the real circuit, from the knowledge of x_i.

It remains to simulate the variables y_i for $i \in O \setminus J$. We first exclude the case of y_n. Since $i \notin J$, neither x_i nor r_i has been probed, and therefore r_i does not occur in the computation of any probed variable or any other output variable in $y_{|O}$ (except possibly y_n, which we consider thereafter), and one can simulate $y_i = x_i \oplus r_i$ by generating a uniform independent variable, without the knowledge of x_i.

Finally we consider the simulation of y_n when $n \in O$. From $|J \cup O| \leq |J| + |O| \leq t + |O| < n$ and $n \in J \cup O$, there exists $i^\star \notin J \cup O$ with $1 \leq i^* \leq n-1$. Since $i^\star \notin J$ and $i^\star \notin O$, we have that $r_{i^\star}$ does not occur in the computation of any probed variable or any output variable $y_{|O}$ except y_n. Therefore we can use $r_{i^\star}$ as a one-time pad for the simulation of y_n. Namely we can write:

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

and y_n can be simulated by generating a uniform independent variable. This terminates the proof of Lemma 2. □

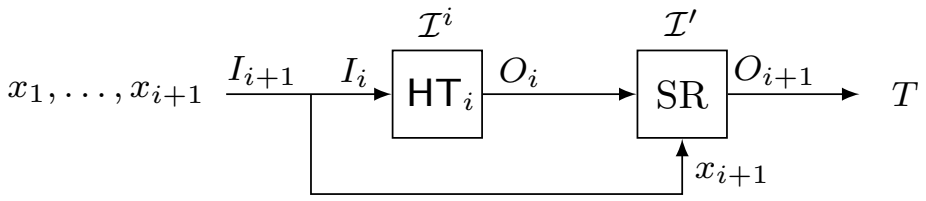

Fig. 4. Illustration of the recursive HT_{i+1} algorithm, with observations $\mathcal{I}^{i+1} = \mathcal{I}^i \cup \mathcal{I}'$.

5.2 Proof of Theorem 3

We proceed with the proof of Theorem 3. As in the proof of Theorem 2, we consider an algorithm HT_i taking as input $x_1, \ldots, x_i$ and outputting a table T with rows of $i+1$ shares such that for all $u \in \{0,1\}^k$:

$$\bigoplus_{j=1}^{i+1} T(u)[j] = S(u \oplus x_1 \oplus \cdots \oplus x_i)$$

As illustrated in Figure 4, the algorithm HT_{i+1} takes as input $x_1, \ldots, x_{i+1}$, and first recursively applies HT_i on $x_1, \ldots, x_i$, followed by a shift of the table T by x_{i+1}, and then appends 0 to get $i+2$ shares as output; eventually it applies a $(i+2)$-$\mathsf{RefreshMasks}$ on all the table rows. We note that the 0 is appended in the index corresponding to the accumulated sum in the $\mathsf{RefreshMasks}$, that is the last index $i+2$. As illustrated in Fig. 4, we denote by SR the gadget corresponding to the shift by x_{i+1}, the appending of 0, and the $(i+2)$-$\mathsf{RefreshMasks}$.

We prove by induction that HT_i is t-SNI. This is trivially satisfied for R_0 which does not take input shares, and whose output $T(u) = \big(S(u)\big)$ can be computed directly. We now assume that the recursion hypothesis is satisfied for HT_i, and we must show that it is satisfied for HT_{i+1}. Since HT_{i+1} has $i+2$ output shares, we must show that if

$$t_{i+1} + |O_{i+1}| < i + 2 \tag{4}$$

then the t_{i+1} probed intermediate variables and the output variables $T(u)[j]_{j \in O_{i+1}}$ can be perfectly simulated from $x_{|I_{i+1}}$ with $|I_{i+1}| \leq t_{i+1}$.

We first consider the gadget SR. We can apply Lemma 2 because by definition the last input share of $\mathsf{RefreshMasks}_{i+2}$ is equal to 0. We denote by t' the number of probes in the SR gadget, that is $t' = |\mathcal{I}'|$. We hence have $t_{i+1} = t_i + t'$, where t_i is the number of probes in HT_i and t_{i+1} the number of probes in HT_{i+1}. From Lemma 2 we obtain that there exists a subset J' of input indices such that the t' intermediate variables and the output $T(u)[j]_{j \in O_{i+1}}$ can be perfectly simulated from O_i, where $O_i = (O_{i+1} \cap [1, i+1]) \cup J'$ with $|J'| \leq t' - 1$, or $O_i = J'$ with $|J'| \leq t'$. Therefore, we distinguish two cases:

- If $O_i = (O_{i+1} \cap [1, i+1]) \cup J'$ with $|J'| \leq t' - 1$, we obtain from (4):

  $$|O_i| + t_i \leq |O_{i+1}| + t' - 1 + t_i \leq t_{i+1} + |O_{i+1}| - 1 < i + 1$$

  This implies that the t-SNI condition is satisfied for HT_i, and we can therefore apply the recursion hypothesis. We obtain that the t_i probed intermediate variables as well as the output variables with indices in O_i can be perfectly simulated from $x_{|I_i}$ with $|I_i| \leq t_i$.

- If $O_i = J'$ with $|J'| \leq t'$, we obtain from (4):

$$|O_i| + t_i \leq t' + t_i \leq t_{i+1} < i + 2$$

  We can therefore distinguish again two cases:
  - If $|O_i| + t_i = i + 1$, then from the above inequality we must have $t_{i+1} = i + 1$, and we can take $I_{i+1} = [1, i + 1]$ for the simulation of all variables in HT_{i+1}. This gives $|I_{i+1}| \leq t_{i+1}$ as required.
  - If $|O_i| + t_i < i + 1$, then as previously the t-SNI condition is satisfied for Gadget HT_i, and we can apply the recursion hypothesis, which gives $|I_i| \leq t_i$.

In the analysis above we have either obtained $|I_{i+1}| \leq t_{i+1}$ or $|I_i| \leq t_i$. When we have $|I_i| \leq t_i$, we can distinguish two cases:

- If $t' = 0$, then none of the RefreshMasks performed on the table rows at round $i + 1$ has been probed. Since there are $i + 2$ shares as output and $|O_{i+1}| < i + 2$ from (4), we can can perfectly simulate all output variables corresponding to O_{i+1} by generating uniformly distributed random values . In particular, the last share x_{i+1} is not required for the simulation, and we obtain $I_{i+1} = I_i$.
- If $t' \geq 1$, the knowledge of x_i is required for the simulation and we let $I_{i+1} = I_i \cup \{i + 1\}$.

Therefore in both cases we obtain:

$$|I_{i+1}| \leq |I_i| + t' \leq t_i + t' \leq t_{i+1}$$

which implies that HT_{i+1} is t-SNI. This proves that HT_i is t-SNI for all i.

It remains to consider the table look-up with the last share x_n and subsequent RefreshMasks at Line 13 of Algorithm 3. Applying Lemma 1 and from the t-SNI property of HT_{n-1}, we obtain as in the proof of Theorem 2 that Algorithm 3 is t-SNI. This terminates the proof of Theorem 3.

6 Improved Evaluation of SBoxes with Common Input Shares

Our third contribution in this paper consists in adapting the common shares approach introduced in [CGPZ16], so that half of a randomized look-up table can be pre-computed for multiple SBoxes. The technique works as follows. Assume that two SBox computations must be performed, on inputs a and b:

$$c = S(a), \quad d = S(b)$$

and let $(a_i)_{1 \leq i \leq n}$ and $(b_i)_{1 \leq i \leq n}$ be the n shares of a and b respectively. The technique in [CGPZ16] consists in first ensuring that a and b have $n/2$ of their shares in common. Assuming for simplicity that n is even, we obtain:

$$a = r_1 \oplus \cdots \oplus r_{n/2} \oplus a'_1 \oplus \cdots \oplus a'_{n/2}$$
$$b = r_1 \oplus \cdots \oplus r_{n/2} \oplus b'_1 \oplus \cdots \oplus b'_{n/2}$$

Since the high-order look-up table algorithms work by progressively shifting a randomized table T by the successive input shares, if a and b have half of their first shares in common as above,

then we can mutualise the first half of the table shifts by $r_1, \ldots, r_{n/2}$. More precisely, we start as in Algorithm 1 with a table with n shares:

$$T(u) \leftarrow \big(S(u), 0, \ldots, 0\big)$$

for all $u \in \{0,1\}^k$, and we progressively shift the table by $r_1, \ldots, r_{n/2}$, to obtain for all $u \in \{0,1\}^k$:

$$\oplus\big(T(u)\big) = S(u \oplus r_1 \oplus \cdots \oplus r_{n/2})$$

In a second phase, we first copy the table T into $T^{(1)}$ and $T^{(2)}$, and as in Algorithm 1 we progressively shift the tables $T^{(1)}$ and $T^{(2)}$ with the remaining shares of a and b respectively. Eventually we obtain two tables $T^{(1)}$ and $T^{(2)}$ satisfying for all $u \in \{0,1\}^k$:

$$\oplus\big(T^{(1)}(u)\big) = S(u \oplus r_1 \oplus \cdots \oplus r_{n/2} \oplus a'_1 \oplus \cdots \oplus a'_{n/2-1})$$
$$\oplus\big(T^{(2)}(u)\big) = S(u \oplus r_1 \oplus \cdots \oplus r_{n/2} \oplus b'_1 \oplus \cdots \oplus b'_{n/2-1})$$

and therefore as in Alg. 1 it suffices to perform two table look-up of $T^{(1)}$ and $T^{(2)}$ with the last shares $a'_{n/2}$ and $b'_{n/2}$ respectively.

With the above approach, for the mutualised computation of two SBoxes, only 3 progressive shifts on half of the input shares are performed, instead of 4 in the original algorithm. This gives a speed-up factor of 3/4. More generally, if ℓ SBoxes must be evaluated in parallel (for example, $\ell = 16$ for AES), the speed-up factor becomes:

$$\frac{1+\ell}{2 \cdot \ell} \simeq \frac{1}{2}$$

for large ℓ. In the following, we provide a detailed description of the resulting algorithms, and as previously a t-SNI security proof. For simplicity we start with the parallel computation of two SBoxes, and we later generalize to ℓ SBoxes.

Remark 1. As explained in [CGPZ16], one cannot have more than $n/2$ shares in common between the inputs a and b, because otherwise there would be a straightforward attack with fewer than n probes. Namely assume that $k > n/2$ of the shares r_i are in common between a and b. Then we can probe the $2(n-k) < n$ remaining shares a'_i and b'_i, whose xor gives the secret variable $a \oplus b$. Hence having half of the shares in common is optimal.

6.1 The CommonShares Algorithm

We start by recalling the CommonShares algorithm from [CGPZ16]. We use a slightly different definition with 3 outputs of $n/2$ shares instead of 2 outputs of n shares. For simplicity we assume that n is even. As explained previously, the algorithm takes as input the n-sharing $(a_i)_{1 \le i \le n}$ and $(b_i)_{1 \le i \le n}$ of a and b and outputs three vectors of $n/2$ shares $(r_i)_{1 \le i \le n/2}$, $(a'_i)_{1 \le i \le n/2}$ and $(b'_i)_{1 \le i \le n/2}$ such that:

$$a = r_1 \oplus \cdots \oplus r_{n/2} \oplus a'_1 \oplus \cdots \oplus a'_{n/2}$$
$$b = r_1 \oplus \cdots \oplus r_{n/2} \oplus b'_1 \oplus \cdots \oplus b'_{n/2}$$

so that $(r_i)_{1 \le i \le n/2}$ is commonly used between a and b.

Algorithm 4 CommonShares

Require: shares $(a_i)_{1\le i\le n}$ and $(b_i)_{1\le i\le n}$, with $\bigoplus_{i=1}^{n} a_i = a$ and $\bigoplus_{i=1}^{n} b_i = b$
Ensure: shares $(r_i)_{1\le i\le n/2}$, $(a'_i)_{1\le i\le n/2}$ and $(b'_i)_{1\le i\le n/2}$, such that $\bigoplus_{i=1}^{n/2} r_i \oplus \bigoplus_{i=1}^{n/2} a'_i = a$ and $\bigoplus_{i=1}^{n/2} r_i \oplus \bigoplus_{i=1}^{n/2} b'_i = b$
1: **for** $i = 1$ **to** $n/2$ **do**
2: $\quad r_i \leftarrow^{\$} \mathbb{F}_{2^k}$
3: $\quad a'_i \leftarrow (a_{n/2+i} \oplus r_i) \oplus a_i$
4: $\quad b'_i \leftarrow (b_{n/2+i} \oplus r_i) \oplus b_i$
5: **end for**
6: **return** $(r_i)_{1\le i\le n/2}$, $(a'_i)_{1\le i\le n/2}$ and $(b'_i)_{1\le i\le n/2}$

The following lemma shows that the above CommonShares algorithm achieves the t-NI property; the proof is essentially the same as in [CGPZ16] and is provided in Appendix B.

Lemma 3 (t-NI of CommonShares). *Let $(a_i)_{1\le i\le n}$ and $(b_i)_{1\le i\le n}$ be the input shares of the algorithm* CommonShares*, and let $(r_i)_{1\le i\le n/2}$, $(a'_i)_{1\le i\le n/2}$ and $(b'_i)_{1\le i\le n/2}$ be the output shares. For any set of t intermediate variables, there exists a subset $\mathcal{S} \subset [1, n]$ such that those t variables can be perfectly simulated from $a_{|\mathcal{S}}$ and $b_{|\mathcal{S}}$, with $|\mathcal{S}| \le t$.*

Remark 2. The CommonShares algorithm above works assuming that n is even. For odd n, as in [CGPZ16] we can adapt the algorithm by having $\lfloor n/2 \rfloor$ shares in common instead of $n/2$.

6.2 Partial Evaluation of Randomized Table

Since the high-order computation of the randomized table with the common shares $r_1, \ldots, r_{n/2}$ will be mutualised, and the resulting table will be processed separately with the remaining shares, we must define an algorithm that takes as input a table T and progressively shifts with input shares $x_1, \ldots, x_\lambda$.

Algorithm 5 HTable

Input: $x_1, \ldots, x_\lambda$ and T with n shares.
Output: $\bigoplus T^{out}(u) = \bigoplus T(u \oplus x_1 \oplus \cdots \oplus x_\lambda)$
1: **for** $i = 1$ to λ **do**
2: $\quad$ **for all** $u \in \{0,1\}^k$ **do**
3: $\quad\quad$ **for** $j = 1$ **to** n **do** $T'(u)[j] \leftarrow T(u \oplus x_i)[j]$
4: $\quad$ **end for**
5: $\quad$ **for all** $u \in \{0,1\}^k$ **do**
6: $\quad\quad$ $T(u) \leftarrow \mathsf{RefreshMasks}(T'(u))$
7: $\quad$ **end for**
8: **end for**
9: **return** T

The following lemma shows that the above algorithm achieves the t-SNI property with respect to the input shares $(x_i)_{1\le i\le\lambda}$, and the t-NI property with respect to the input table T. We denote by $T_{|I}$ the set of variables $T(u)[i]$ for all $u \in \{0,1\}^k$ and all $i \in I$. The proof is relatively similar to the proof of Theorem 2 and is given in Appendix C.

Lemma 4 (t-SNI of HTable). *Let $(x_i)_{1\le i\le\lambda}$ and T be the input of* HTable *and let T^{out} be the output table. For any set of t intermediate variables and any subset of indices O, if $t + |O| < n$,*

there exist subsets $I \subset [1, \lambda]$ and $J \subset [1, n]$ with $|I| \leq t$ and $|J| \leq t$, such that those t variables as well as the output shares $T^{out}_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$ and $T_{|J \cup \mathcal{O}}$.

6.3 Evaluation of SBoxes with Common Input Shares

We are now ready to describe our evaluation of SBoxes with common input shares. The common high-order evaluation of two SBoxes is described in the algorithm below.

Algorithm 6 Common Table: high-order evaluation of $y^{(1)} = S(x^{(1)})$ and $y^{(2)} = S(x^{(2)})$

Input: $x_1^{(\ell)}, \ldots, x_n^{(\ell)}$ for $\ell \in \{1, 2\}$
Output: $y_1^{(\ell)}, \ldots, y_n^{(\ell)}$ such that $y_1^{(\ell)} \oplus \cdots \oplus y_n^{(\ell)} = S(x_1^{(\ell)} \oplus \cdots \oplus x_n^{(\ell)})$ for $\ell \in \{1, 2\}$.
1: $(r_i)_{1 \leq i \leq n/2}, (a_i^{(1)})_{1 \leq i \leq n/2}, (a_i^{(2)})_{1 \leq i \leq n/2} \leftarrow \mathsf{CommonShares}(x_i^{(1)}, x_i^{(2)})$
2: **for all** $u \in \{0,1\}^k$ **do** $T(u) \leftarrow \big(S(u), 0, \ldots, 0\big) \in (\{0,1\}^{k'})^n$ ▷ $\oplus\big(T(u)\big) = S(u)$
3: $T^{(1)} \leftarrow \mathsf{Htable}(T, r_1, \ldots, r_{n/2})$
4: $T^{(2)} \leftarrow T^{(1)}$
5: $T^{(1)} \leftarrow \mathsf{Htable}(T^{(1)}, a_1^{(1)}, \ldots, a_{n/2-1}^{(1)})$
6: $T^{(2)} \leftarrow \mathsf{Htable}(T^{(2)}, a_1^{(2)}, \ldots, a_{n/2-1}^{(2)})$
7: $(y_1^{(1)}, \ldots, y_n^{(1)}) \leftarrow \mathsf{RefreshMasks}\big(T^{(1)}(a_{n/2}^{(1)})\big)$
8: $(y_1^{(2)}, \ldots, y_n^{(2)}) \leftarrow \mathsf{RefreshMasks}\big(T^{(2)}(a_{n/2}^{(2)})\big)$
9: **return** $y_1^{(\ell)}, \ldots, y_n^{(\ell)}$ for $\ell \in \{1, 2\}$

The theorem below shows that the shared evaluation achieves the t-SNI security as previous algorithms.

Theorem 4 (t-SNI of Common Table). *Let $(x_i^{(\ell)})_{1 \leq i \leq n}$ for $\ell \in \{1, 2\}$ be the input of* Common Table *and $(y^{(\ell)})_{1 \leq i \leq n}$ be the output. For any set of t intermediate variables and any subset of indices $\mathcal{O}$ with $t + |\mathcal{O}| < n$, there exists a subset $I \subset [1, n]$ with $|I| \leq t$, such that those t variables as well as the output shares $(y^{(1)})_{|\mathcal{O}}$ and $(y^{(2)})_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$ and $y_{|I}$.*

Proof. We use Lemmas 3 and 4 to prove that the composition of the CommonShares gadget with the three HTable gadgets allows the entire circuit to be t-SNI. We label the gadgets from 1 to 4 starting from right to left (see Figure 5).

Let $\mathcal{I} = \mathcal{I}^1 \cup \mathcal{I}^2 \cup \mathcal{I}^3 \cup \mathcal{I}^4$ be a set of indices such that $|\mathcal{I}| \leq t$, corresponding to observations of intermediate variables done by the attacker in the four gadgets, and let $\mathcal{O}$ be a set of indices such that $t + |\mathcal{O}| < n$, corresponding to observations on the outputs made by the attacker.

Gadget 1 By assumption, we know that $\big|\mathcal{I}^1\big| + |\mathcal{O}| \leq |\mathcal{I}| + |\mathcal{O}| \leq t + |\mathcal{O}| < n$. Since from Lemma 4, the HTable gadget is t-SNI, this means that there exist two sets of indices $\mathcal{S}_1^1, \mathcal{S}_2^1$ such that $\big|\mathcal{S}_1^1\big| \leq \big|\mathcal{I}^1\big|$, $\big|\mathcal{S}_2^1\big| \leq \big|\mathcal{I}^1\big|$ and the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}_1^1$ and $\mathcal{S}^1 = \mathcal{S}_2^1 \cup \mathcal{O}$.

Gadget 2 Similarly, since $\big|\mathcal{I}^2\big| + |\mathcal{O}| \leq |\mathcal{I}| + |\mathcal{O}| \leq t + |\mathcal{O}| < n$, from Lemma 4, there exist two sets of indices $\mathcal{S}_1^2, \mathcal{S}_2^2$ such that $\big|\mathcal{S}_1^2\big| \leq \big|\mathcal{I}^2\big|$ and $\big|\mathcal{S}_2^2\big| \leq \big|\mathcal{I}^2\big|$, and the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}_2^2$ and $\mathcal{S}^2 = \mathcal{S}_1^2 \cup \mathcal{O}$.

Gadget 3 From gadgets 1 and 2, we have $\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| = \left|(\mathcal{S}_2^1 \cup \mathcal{O}) \cup (\mathcal{S}_1^2 \cup \mathcal{O})\right| + \left|\mathcal{I}^3\right| = \left|\mathcal{S}_2^1 \cup \mathcal{S}_1^2 \cup \mathcal{O}\right| + \left|\mathcal{I}^3\right| \leq \left|\mathcal{S}_2^1\right| + \left|\mathcal{S}_1^2\right| + \left|\mathcal{I}^3\right| + |\mathcal{O}| \leq \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| + |\mathcal{O}| \leq t + |\mathcal{O}| < n$. Therefore, one can apply Lemma 4 which ensures that there exist two sets of indices $\mathcal{S}_1^3, \mathcal{S}_2^3$ such that $\left|\mathcal{S}_1^3\right| \leq \left|\mathcal{I}^3\right|$, $\left|\mathcal{S}_2^3\right| \leq \left|\mathcal{I}^3\right|$ and the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}_1^3$ and $\mathcal{S}^3 = \mathcal{S}_2^3 \cup (\mathcal{S}_2^1 \cup \mathcal{S}_1^2 \cup \mathcal{O})$. Since Gadget 3 takes as input a known table T, namely $T(u) = \big(S(u), 0, \ldots, 0\big)$ for all $u \in \{0,1\}^k$, the input shares of Gadget 3 corresponding to $\mathcal{S}^3$ can be perfectly simulated; therefore we omit $\mathcal{S}^3$ in Figure 5 and keep only $\mathcal{S}_1^3$.

Gadget 4 From Lemma 3 there exists a set of indices $\mathcal{S}^4$ such that $\left|\mathcal{S}^4\right| \leq \left|\mathcal{I}^4\right| + \left|\mathcal{S}_1^3\right| + \left|\mathcal{S}_1^1\right| + \left|\mathcal{S}_2^2\right|$ and Gadget 4 can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^4$. From gadgets 1, 2 and 3, it follows that $\left|\mathcal{S}^4\right| \leq \left|\mathcal{I}^4\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right|$.

Each of the previous steps ensures the existence of a simulator for each gadget. Let $I = \mathcal{S}^4$. We can then compose these simulators to perfectly simulate the computation of Algorithm 6 from $x_{|I}^{(1)}$ and $x_{|I}^{(2)}$. Furthermore, from Gadget 4 we have $|I| = \left|\mathcal{S}^4\right| \leq \left|\mathcal{I}^4\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| \leq t$. This concludes the proof.

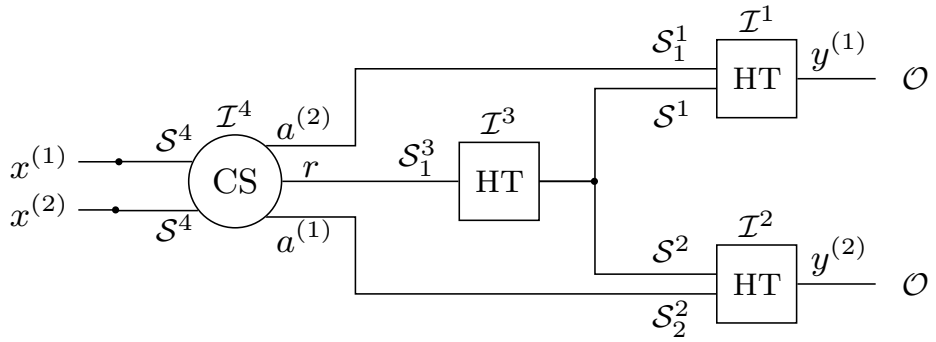

Fig. 5. Illustration of common input table as composition of gadgets. Each variable $x^{(\ell)}$ and $y^{(\ell)}$ contains actually n shares $x_i^{(\ell)}$ and $y_i^{(\ell)}$, for $\ell = \{1, 2\}$.

□

6.4 Generalization to Multiple SBoxes

It is easy to generalize the previous construction to multiple SBoxes, following the same approach as in [CGPZ16]. We describe such generalization in Appendix D. As explained previously, when evaluating ℓ SBoxes in parallel, (for example, $\ell = 16$ for AES), the speed-up factor becomes:

$$\frac{1+\ell}{2 \cdot \ell} \simeq \frac{1}{2}$$

for large ℓ.

6.5 Common input shares and increasing number of output shares

It is natural to try to combine the common input shares approach and the increasing number of outputs shares technique from Section 5. Namely instead of starting with a table T of already n

shares at Line 2 of Algorithm 6, one can start with a single output share as in Line 2 of Algorithm 3. Since the mutualised table evaluation takes as input $n/2$ shares $r_1, \ldots, r_{n/2}$, one would obtain as output of Line 3 of Algorithm 6 a table $T^{(1)}$ with $n/2+1$ output shares, and eventually n shares as previously after lines 5 and 6.

However when applying the common shares technique to the progressive increase of output shares, one does not get a factor 2 improvement in speed as previously, because the mutualised part works with 1 to $n/2+1$ shares (hence with an average of $n/4$ shares), while the two non-mutualised parts work with $n/2+1$ to n shares (hence with $3n/4$ shares on average); in other words, we only mutualize the more efficient part of the table evaluation algorithm. The speed-up ratio when evaluating ℓ SBoxes is therefore:

$$\frac{\frac{n}{4} + \ell \cdot \frac{3n}{4}}{\ell \cdot \frac{n}{4} + \ell \cdot \frac{3n}{4}} = \frac{1+3\ell}{4\ell} \simeq \frac{3}{4}$$

for large ℓ, instead of $1/2$ in the previous section. Since the progressive increase of output shares from Section 5 provide a speed-up ratio of $1/2$, the combined speed-up ratio is therefore $3/8$ compared to the original countermeasure recalled in Section 3. All in all, when taking into account the improved t-SNI security proof from Section 4 that enables to use $n = t+1$ shares instead of $n = 2t+1$ as in [Cor14], since the complexity is quadratic in n, we obtain a speed-up ratio of $1/4 \cdot 3/8 = 3/32$. Hence asymptotically for large n and ℓ, the running-time is decreased by a factor $32/3 = 10.7$.

We provide a formal description of the corresponding algorithm in Appendix E, with as previously a security proof with the t-SNI definition. For simplicity we only perform the analysis for the mutualised computation of $\ell = 2$ SBoxes in parallel. We obtain that to get a t-SNI security proof, we must start at Line 2 of Algorithm 6 with a mutualised table T with 2 shares instead of 1. More generally, when computing ℓ SBoxes in parallel, one should start with ℓ shares instead of 1 to get a security proof.[2] This makes the combination of the two techniques less interesting in practice. While the above analysis still holds asymptotically, because for $n \gg \ell$ the number of initial shares ℓ can be seen as a constant, in practice for AES with $\ell = 16$ that would require a number of shares $n \gg 16$, which is probably unrealistic. In the next section we provide the result of practical implementations for AES and DES, showing that while the two techniques provide a significant speed-up when used separately, the combination does not really provide any further speed-up.

7 Implementation

We have performed a practical implementation of our new countermeasure for both AES and DES, using a 32-bit architecture so that we could apply the large register variant recalled in Appendix A. More precisely we can pack $\delta = 4$ output bytes for AES, and $\delta = 8$ output 4-bit nibbles for DES. When using this variant, only the first table evaluation can be used with increasing number of shares (as in Section 5), while the second table evaluation always works with n shares. Similary, only the first table evaluation can be used with the common shares technique from Section 5.

We have also implemented the Rivain-Prouff countermeasure [RP10] for AES and the Carlet *et al.* countermeasure [CGP+12] for DES; for the latter we have used the technique from [RV13], in which the evaluation of a DES SBox requires only 7 non-linear multiplications, and the improved

[2] More precisely, one should start with $\min(\ell, n)$ shares up to n shares, since using more than n shares as output of the table T would not increase the level of security.

technique from [CRV14], which requires only 4 non-linear multiplications. The performances of our implementations are summarized in Table 1, with $n = t + 1$ shares for security against t-th order attacks (except for the original countermeasure from [Cor14] with $n = 2t + 1$ shares). We provide the penalty factor compared to a non-protected implementation for various values of t.

AES SBox evaluation	Security order t				
	2	3	4	5	6
Rivain-Prouff [RP10], $n = t + 1$	82	123	179	247	326
Randomized table [Cor14], $n = 2t + 1$	1 491	3 174	5 545	8 599	12 253
Randomized table (Section 4), $n = t + 1$	434	870	1 496	2 255	3193
Randomized table, INC (Section 5)	309	596	956	1 400	1 919
Randomized table, CS (Section 6.3)	325	612	1 142	1 637	2 478
Randomized table, CS INC (Section 6.5)	329	548	1 002	1 366	1 982

DES SBox evaluation	Security order t				
	2	3	4	5	6
[CGP+12] with [RV13] polynomials	59	78	101	127	156
[CGP+12] with [CRV14] polynomials	23	30	39	51	64
Randomized table [Cor14], $n = 2t + 1$	49	91	153	227	312
Randomized table (Section 4), $n = t + 1$	22	35	51	72	96
Randomized table, INC (Section 5)	21	32	45	61	80

Table 1. Penalty factor for various high-order evaluation of SBox algorithms, up to security order $t = 6$, with $n = t+1$ shares (except for the original countermeasure in [Cor14] with $n = 2t + 1$ shares). The implementation was done in C on an iMac running a 3.2 GHz Intel processor.

We see that for AES the randomized table algorithms are still less efficient in practice than Rivain-Prouff, which can take advantage of the special algebraic structure of the AES SBox. However, considering the increasing number of output shares technique from Section 5, and the common input shares technique from Section 6.3, both techniques provide a 30% speed-up separately; however combining the two techniques does not provide any further speed-up, for reasons explained in Section 6.5. If we take into account the improved t-SNI security proof which enables to use $n = t + 1$ shares instead of $2t + 1$, we obtain a cumulative speed-up factor of roughly 4.8 for both techniques, compared to [Cor14].

For DES, we see that our countermeasure has the same level of efficiency as the Carlet *et al.* countermeasure, when using the polynomials from [CRV14]. Moreover with the increasing number of output shares technique from Section 5, we get up to 20% speed-up compared to the original algorithm (and a 2.5 speed-up factor when taking into account the $n = t + 1$ shares instead of $n = 2t + 1$).

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/.

[CGP+12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In *FSE*, pages 366–384, 2012.

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

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

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

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 742–763, 2015.

[CRV14] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 170–187, 2014.

[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.

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

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

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

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In *EUROCRYPT*, pages 142–159, 2013.

[RDP08] Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block ciphers implementations provably secure against second order side channel analysis. In *FSE*, pages 127–143, 2008.

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

[RV13] Arnab Roy and Srinivas Vivek. Analysis and improvement of the generic higher-order masking scheme of FSE 2012. In *CHES*, pages 417–434, 2013.

A Variant with Large Registers

As shown in [Cor14], the efficiency of the randomized table countermeasure can be improved by packing multiple SBox outputs into a single register, so that the table shifts can be performed more efficiently at the register level first; for example for AES, 4 outputs of the 8-bit SBox can be stored on the same 32-bit register.

More precisely, given an original SBox $S : \{0,1\}^k \to \{0,1\}^{k'}$, the variant works with registers of size $\omega = 2^{k_2} \cdot k'$ bits, for some parameter $k_2 < k$. One defines a new SBox S' with $k_1 = k - k_2$ bits input and ω bits output, with:

$$S'(a) = S(a \,\|\, 0^{k_2}) \,\|\, \cdots \,\|\, S(a \,\|\, 1^{k_2})$$

for all $a \in \{0,1\}^{k_1}$. Given $x = a\|b$ as input, $S(x)$ is computed in two steps:

1. Let $z \leftarrow S'(a) = S(a\|0^{k_2})\| \cdots \|S(a\|1^{k_2})$
2. Viewing z as a k_2-bit input and k'-bit output table, compute $y \leftarrow z(b) = S(x)$.

Similarly, the n-shared evaluation of S proceeds in two steps. In the first step, the table S' is evaluated using the randomized table countermeasure, working with registers of size $2^{k_2} \cdot k'$ bits instead of only k'; in principle this leads to a 2^{k_2} speed-up factor. The complexity of the first step is therefore $\mathcal{O}(2^{k-k_2})$.

In the second step, the value z is returned in the form of n shares $(z_i)_{1 \le i \le n}$, and viewed as a look-up table. The randomized table countermeasure is again used. The only difference is that the look-up table already comes in shared form. Therefore in the second step the randomized table is initialised as follows:

$$T(u) = (z_1(u), \ldots, z_n(u)) \in (\{0,1\}^{k'})^n$$

for all $u \in \{0,1\}^{k_2}$. The complexity of the second step is therefore $\mathcal{O}(2^{k_2})$. As explained in [Cor14], the total complexity of the countermeasure is minimized when taking $k_2 = k/2$, which gives a complexity $\mathcal{O}(2^{k/2} \cdot n^2)$, the same complexity as the Carlet *et al.* countermeasure. We provide a formal description of the variant in Algorithm 7.

Algorithm 7 Evaluation of $y = S(x)$ with registers of size $\omega = 2^{k_2} \cdot k'$ bits.

Input: $x_1, \ldots, x_n$ such that $x = x_1 \oplus \cdots \oplus x_n$
Output: $y_1, \ldots, y_n$ such that $y = S(x) = y_1 \oplus \cdots \oplus y_n$
for all $u_1 \in \{0,1\}^{k_1}$ **do**
  $\mathrm{tmp} \leftarrow S(u_1 \| 0^{k_2}) \| \cdots \| S(u_1 \| 1^{k_2})$
  $T(u_1) \leftarrow (\mathrm{tmp}, 0, \ldots, 0) \in (\{0,1\}^{\omega})^n$ ▷ $\oplus\big(T(u_1)\big) = S'(u_1)$
end for
for $i = 1$ to $n-1$ **do**
  for all $u \in \{0,1\}^{k_1}$ **do**
    for $j = 1$ **to** n **do** $T'(u)[j] \leftarrow T(u \oplus (x_i \gg k_2))[j]$
  end for
  for all $u \in \{0,1\}^{k_1}$ **do** $T(u) \leftarrow \mathsf{RefreshMasks}\big(T'(u)\big)$
end for
$(y_1, \ldots, y_n) \leftarrow \mathsf{RefreshMasks}\big(T(x_n \gg k_2)\big)$ ▷ $\oplus\big(T(x_n \gg k_2)\big) = S'(x \gg k_2)$
for $j = 1$ to n **do**
  $T(0^{k_2})[j], \ldots, T(1^{k_2})[j] \leftarrow y_j$
end for
for $i = 1$ to $n-1$ **do**
  for all $u \in \{0,1\}^{k_2}$ **do**
    for $j = 1$ **to** n **do** $T'(u)[j] \leftarrow T(u \oplus (x_i \bmod 2^{k_2}))[j]$
  end for
  for all $u \in \{0,1\}^{k_2}$ **do** $T(u) \leftarrow \mathsf{RefreshMasks}\big(T'(u)\big)$
end for
$(y_1, \ldots, y_n) \leftarrow \mathsf{RefreshMasks}\big(T(x_n \bmod 2^{k_2})\big)$ ▷ $\oplus(y_i) = S(x)$
return $y_1, \ldots, y_n$

B Proof of Lemma 3

We first provide the proof intuition. If for a given i with $1 \le i \le n/2$ the adversary requests only one of the variables r_i, $a_{n/2+i} \oplus r_i$, $b_{n/2+i} \oplus r_i$, a'_i or b'_i, then such variable can be perfectly simulated without knowing any of the input shares a_i, b_i, $a_{n/2+i}$ and $b_{n/2+i}$, thanks to the mask r_i. On the other hand, if two such variables (or more) are requested, then we can provide a perfect simulation from the 4 previous input shares, whose knowledge is obtained by adding the two indices i and

$n/2+i$ in $\mathcal{S}$. Therefore we never have to add more than one index in $\mathcal{S}$ per probe, which implies that the size of the subset $\mathcal{S}$ of input shares is upper-bounded by t, as required.

More precisely, we describe hereafter the construction of the set $\mathcal{S} \subset [1, n]$ of input shares, initially empty. For every probed input variable a_i and b_i (for any i), we add i to $\mathcal{S}$. For all $1 \leq i \leq n/2$, we let t_i be the number of probed variables among r_i, $a_{n/2+i} \oplus r_i$, $(a_{n/2+i} \oplus r_i) \oplus a_i$, $b_{n/2+i} \oplus r_i$ and $(b_{n/2+i} \oplus r_i) \oplus b_i$. We then add $\{i, n/2+i\}$ to $\mathcal{S}$ if $t_i \geq 2$. This terminates the construction of $\mathcal{S}$. By construction of $\mathcal{S}$, we must have $|\mathcal{S}| \leq t$ as required.

We now show that the t probed variables can be perfectly simulated from $a_{|\mathcal{S}}$ and $b_{|\mathcal{S}}$. This is clear for the probed input variables a_i and b_i, for all $1 \leq i \leq n$ since $i \in \mathcal{S}$ by construction. It remains to simulate variables r_i, $a_{n/2+i} \oplus r_i$, $b_{n/2+i} \oplus r_i$, a'_i and b'_i for $1 \leq i \leq n/2$. We distinguish two cases.

- If $t_i \geq 2$, then $\{i, n/2+i\} \in \mathcal{S}$, so we can let $r_i \leftarrow \mathbb{F}_{2^k}$ as in the real algorithm and simulate all output and intermediate variables from the knowledge of a_i, $a_{n/2+i}$, b_i and $b_{n/2+i}$.
- If $t_i = 1$, then only a single variable among r_i, $a_{n/2+i} \oplus r_i$, $b_{n/2+i} \oplus r_i$, a'_i and b'_i must be simulated. Since each of those variables is masked by r_i, we can simulate this single variable by generating a random value in $\mathbb{F}_{2^k}$.

This terminates the proof of Lemma 3.

C Proof of Lemma 4

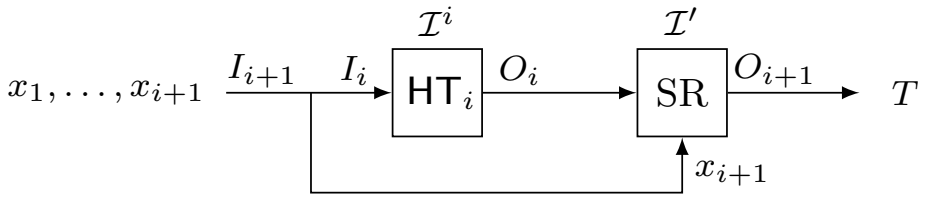

Fig. 6. Illustration of the recursive HT_{i+1} algorithm, with observations $\mathcal{I}^{i+1} = \mathcal{I}^i \cup \mathcal{I}'$.

We consider an algorithm HT_i that takes as input $x_1, \ldots, x_i$ and a table T and outputs a table T^{out} with rows of n shares such that for all u:

$$\bigoplus_{j=1}^{n} T^{out}(u)[j] = T(u \oplus x_1 \oplus \cdots \oplus x_i)$$

To build HT_{i+1} on input $x_1, \ldots, x_{i+1}$ and T, we first apply HT_i on $x_1, \ldots, x_i$ and T, and then we apply another shift by x_{i+1}, then an n-RefreshMask on all the rows of the table.

We must show that for all i, the Gadget HT_i is t-SNI with respect to the input shares x_i, and t-NI with respect to T, meaning that if $t_i + |O_i| < n$, then the t_i probed variables and the output variables $T^{out}(u)[j]_{j \in O_i}$ can be perfectly simulated from $x_{|I_i}$ and $T_{|J_i \cup O_i}$ with $|I_i| \leqslant t_i$ and $|J_i| \leqslant t_i$.

We proceed by induction on i. As previously, it is easy to see that HT_0 satisfies the property since HT_0 is actually the identity function, taking as input T and outputting T. We now assume

that HT_i satisfies the property, and we must show that the property holds for HT_{i+1}, under the condition:

$$t_{i+1} + |O_{i+1}| < n$$

As previously, we let t' be the number of probed values in $\mathsf{HT}_{i+1} \setminus \mathsf{HT}_i$, and we let t_i be the number of probed variables in HT_i. We hence have $t_{i+1} = t_i + t'$.

We first apply Lemma 1 to the SR gadget which corresponds to the shift of the table T by x_{i+1}, followed by a $\mathsf{RefreshMasks}$ of all the rows of the table T (see Figure 6). We obtain that all probed intermediate variables and all output variables corresponding to O_{i+1} can be perfectly simulated from the input variables in O_i, where $O_i = O_{i+1} \cup J$ with $|J| \leq t'$.

We now consider the HT_i gadget. We have from $O_i = O_{i+1} \cup J$:

$$t_i + |O_i| \leq t_i + |O_{i+1}| + t' \leq t_{i+1} + |O_{i+1}| < n$$

Therefore the t-SNI condition is satisfied for Gadget HT_i, and all probed intermediate variables and all output variables corresponding to O_i can be perfectly simulated from $x_{|I_i}$ and $T_{|J_i \cup O_i}$ with $|I_i| \leq t_i$ and $|J_i| \leqslant t_i$. We can write:

$$O_i \cup J_i = (O_{i+1} \cup J) \cup J_i = O_{i+1} \cup J_{i+1}$$

where we let $J_{i+1} := J_i \cup J$. Therefore the simulation can be performed from $T_{|J_{i+1} \cup O_{i+1}}$, where as required:

$$|J_{i+1}| \leqslant |J_i| + t' \leqslant t_i + t' \leqslant t_{i+1}$$

For the construction of I_{i+1}, as in the proof of Theorem 2 we distinguish two cases:

- If $t' = 0$, this means that the SR gadget has not been probed. Since $|O_{i+1}| < n$, thanks to the $\mathsf{RefreshMasks}$ of every row we can simulate all output variables corresponding to O_{i+1} without the knowledge of any input variable. In particular, we don't need to know x_{i+1} to perform the simulation, and we can let $I_{i+1} = I_i$.
- If $|t'| \geq 1$, the knowledge of x_{i+1} is required to perform the simulation, so we let $I_{i+1} = I_i \cup \{i+1\}$.

In both cases we obtain $|I_{i+1}| \leq |I_i| + |t'|$. From $|I_i| \leq t_i$, we obtain:

$$|I_{i+1}| \leq |I_i| + |t'| \leq t_i + |t'| \leq t_{i+1}$$

which shows that HT_{i+1} is satisfies the property. This terminates the proof of Lemma 4.

D Generalization of Common Shares to Multiple SBoxes

Algorithm 8 GeneralizedCommonShares

Input: Shares $(a_i^{(j)})_{1 \leq i \leq n}$ for $1 \leqslant j \leqslant m$
Output: Shares $(r_i)_{1 \leq i \leq n/2}$ and $(b_i^{(j)})_{1 \leq i \leq n/2}$ satisfying $\bigoplus_{i=1}^{n/2} r_i \oplus \bigoplus_{i=1}^{n/2} b_i^{(j)} = \bigoplus_{i=1}^{n} a_i^{(j)}$ for all $1 \leq j \leq m$.

1: **for** $i = 1$ **to** $n/2$ **do**
2: $\quad r_i \leftarrow^{\$} \mathbb{F}_{2^k}$
3: $\quad$ **for** $j = 1$ **to** m **do**
4: $\quad\quad b_i^{(j)} \leftarrow (a_{n/2+i}^{(j)} \oplus r_i) \oplus a_i^{(j)}$
5: $\quad$ **end for**
6: **end for**
7: **return** $(r_i)_{1 \leq i \leq n/2}$ and $(b_i^{(j)})_{1 \leq i \leq n/2}$ for all $1 \leq j \leq m$.

Lemma 5 (t-NI of GeneralizedCommonShares). *Let $(a_i^{(j)})_{1\le i\le n}$ for $1 \leqslant j \leqslant m$ be the input shares of the algorithm* GeneralizedCommonShares*, and let $(r_i)_{1\le i\le n/2}$ and $(b_i^{(j)})_{1\le i\le n/2}$ be the output shares. For any set of t intermediate variables, there exists a subset $\mathcal{S} \subset [1,n]$ with $|\mathcal{S}| \le t$, such that those t variables can be perfectly simulated from $a_{|\mathcal{S}}^{(j)}$ for $1 \le j \le m$.*

Proof. As in [CGPZ16], the proof is a straightforward generalization of the proof of Lemma 5 and is therefore omitted.

Algorithm 9 Generalized Common Table: common masked computation of $y^{(j)} = S(x^{(j)})$ for $1 \le j \le m$

Input: $x_1^{(j)}, \dots, x_n^{(j)}$ for $1 \le j \le m$
Output: $y_1^{(j)}, \dots, y_n^{(j)}$ such that $y_1^{(j)} \oplus \cdots \oplus y_n^{(j)} = S(x_1^{(j)} \oplus \cdots \oplus x_n^{(j)})$ for all $1 \le j \le m$.
1: $(r_i)_{1\le i\le n/2}, (b_i^{(j)})_{1\le i\le n/2} \leftarrow$ GeneralizedCommonShares$((x_i^{(j)})_{1\le i\le n})$
2: **for all** $u \in \{0,1\}^k$ **do** $T(u) \leftarrow \big(S(u), 0, \dots, 0\big) \in (\{0,1\}^{k'})^n$ ▷ $\oplus\big(T(u)\big) = S(u)$
3: $T \leftarrow$ Htable$(T, r_1, \dots, r_{n/2})$
4: **for** $j = 1$ **to** m **do**
5: $\quad T^{(j)} \leftarrow$ Htable$(T, b_1^{(j)}, \dots, b_{n/2-1}^{(j)})$
6: $\quad (y_1^{(j)}, \dots, y_n^{(j)}) \leftarrow$ RefreshMasks$\big(T^{(j)}(b_{n/2}^{(j)})\big)$
7: **end for**
8: **return** $y_1^{(j)}, \dots, y_n^{(j)}$ for $1 \le j \le m$

Theorem 5 (t-SNI of Generalized Common Table). *Let $(x_i^{(j)})_{1\le i\le n}$ for $1 \le j \le m$ be the input of* Generalized Common Table *and $(y^{(j)})_{1\le i\le n}$ be the output. For any set of t intermediate variables and any subset of indices $\mathcal{O}$ with $t + |\mathcal{O}| < n$, there exists a subset $I \subset [1,n]$ with $|I| \le t$, such that those t variables as well as the output shares $(y^{(j)})_{|\mathcal{O}}$ for all $1 \le j \le m$ can be perfectly simulated from $x_{|I}^{(j)}$ for all $1 \le j \le m$.*

Proof. As in [CGPZ16], the proof is a straightforward generalization of the proof of Theorem 4 and is therefore omitted.

E Common Shares with Increased Number of Shares

We define an algorithm HTableInc that takes as input a table T and shifts it by the input shares $x_1, \dots, x_\lambda$, that is the first λ shares, instead of the full n shares; see Algorithm 10. The algorithm also progressively increases the number of shares in the table, from ℓ shares to $\ell + \lambda$ shares at the end.

The following Lemma is analogous to Lemma 4 and proves that the HTableInc algorithm is t-SNI with respect to the input shares x_i, and t-NI with respect to the input table T.

Lemma 6 (t-SNI of HTableInc). *Let $(x_i)_{1\le i\le\lambda}$ and T be the input of* HTableInc *and let T^{out} be the output table, where T has ℓ shares. For any set of t intermediate variables and any subset of indices O, if $t + |O| < \ell + \lambda$, there exist subsets $I \subset [1,\lambda]$ and $J \subset [1,\ell]$ with $|I| \le t$, such that those t variables as well as the output shares $T^{out}_{|O}$ can be perfectly simulated from $x_{|I}$ and $T_{|S}$, where $S = (O \cap [1,\ell]) \cup J$ with $|J| \le t - \lambda$, or $S = J$ with $|I| + |J| \le t + 1$ and $|J| \le t$.*

Algorithm 10 HTableInc

Input: $x_1, \ldots, x_\lambda$ and T with ℓ shares.
Output: $\bigoplus T^{out}(u) = \bigoplus T(u \oplus x_1 \oplus \cdots \oplus x_\lambda)$, with $\ell + \lambda$ shares.

```
for i = 1 to λ do
    for all u ∈ {0,1}^k do
        for j = 1 to ℓ + i − 1 do T'(u)[j] ← T(u ⊕ x_i)[j]
    end for
    for all u ∈ {0,1}^k do
        T(u) ← (T'(u)[1], ..., T'(u)[ℓ + i − 1], 0)
        T(u) ← RefreshMasks_{ℓ+i}(T(u))
    end for
end for
return T
```

Proof. We use similar notations as in the recursive proof of Theorem 2. We consider an algorithm R_λ that takes as input $x_1, \ldots, x_\lambda$ and a table T with ℓ shares and outputs a table T^{out}_λ with $\ell + \lambda$ shares such that for all u:

$$\bigoplus_{j=1}^{\lambda+\ell} T^{out}(u)[j] = \bigoplus_{j=1}^{\ell} T(u \oplus x_1 \oplus \ldots \oplus x_\lambda)[j]$$

To build $R_{\lambda+1}$ on input $x_1, \ldots, x_{\lambda+1}$ and T, we first apply R_λ on $x_1, \ldots, x_\lambda$, and then we apply another shift S by $x_{\lambda+1}$, then a RefreshMasks on every table row of $\ell + \lambda + 1$ shares. We will proceed by induction on λ. We assume that the gadget R_λ is t_λ-SNI, which means that if $t_\lambda + |O_\lambda| < \ell + \lambda$, then the t_λ intermediate variables and the output variables $T(u)[j]_{j \in O_\lambda}$ can be perfectly simulated from $x_{|I_\lambda}$ and $T_{|S_\lambda}$ where $|I_\lambda| \leqslant t_\lambda$ and $S_\lambda = (O_\lambda \cap [1, \ell]) \cup J_\lambda$ with $|J_\lambda| \leq t_\lambda - \lambda$, or $S_\lambda = J_\lambda$ with $|I_\lambda| + |J_\lambda| \leq t_\lambda + 1$ and $|J_\lambda| \leq t_\lambda$. We must show that the gadget $R_{\lambda+1}$ is $t_{\lambda+1}$-SNI, which means that if

$$t_{\lambda+1} + |O_{\lambda+1}| < \ell + \lambda + 1 \tag{5}$$

then the $t_{\lambda+1}$ intermediate variables and the output variables $T^{out}(u)[j]_{j \in O_{\lambda+1}}$ can be perfectly simulated from $x_{|I_{\lambda+1}}$ and $T_{|S_{\lambda+1}}$ where $|I_{\lambda+1}| \leqslant t_{\lambda+1}$ and $S_{\lambda+1} = (O_{\lambda+1} \cap [1, \ell]) \cup J_{\lambda+1}$ with $|J_{\lambda+1}| \leq t_{\lambda+1} - (\lambda + 1)$, or $S_{\lambda+1} = J_{\lambda+1}$ with $|I_{\lambda+1}| + |J_{\lambda+1}| \leq t_{\lambda+1} + 1$ and $|J_{\lambda+1}| \leq t_{\lambda+1}$.

We start by showing that the gadget R_0 satisfies the above property. This is actually straightforward, because R_0 is the identity function, taking as input a table T and outputting T.

We now assume that R_λ satisfies the above property. Let t' be the number of probed values in $R_{\lambda+1} \setminus R_\lambda$. We hence have $t_{\lambda+1} = t_\lambda + t'$. We apply Lemma 2 on $R_{\lambda+1} \setminus R_\lambda$ and we distinguish two cases, depending on the way the set O_λ is built.

∘ *Case* $O_\lambda = (O_{\lambda+1} \cap [1, \ell + \lambda]) \cup S'$ *with* $|S'| \leqslant t' - 1$. In this case, we deduce that

$$|O_\lambda| + t_\lambda \leqslant |O_{\lambda+1}| + |S'| + t_\lambda \leqslant |O_{\lambda+1}| + t' - 1 + t_\lambda \leqslant |O_{\lambda+1}| + t_{\lambda+1} - 1$$

From (5) we obtain:

$$|O_\lambda| + t_\lambda < \ell + \lambda + 1 - 1 = \ell + \lambda$$

Therefore, one can apply the induction step which ensures that $|I_\lambda| \leqslant t_\lambda$ and $S_\lambda = (O_\lambda \cap [1, \ell]) \cup J_\lambda$ with $|J_\lambda| \leq t_\lambda - \lambda$, or $S_\lambda = J_\lambda$ with $|I_\lambda| + |J_\lambda| \leq t_\lambda + 1$ and $|J_\lambda| \leq t_\lambda$. We first consider the construction of $I_{\lambda+1}$. We distinguish two cases:

- If $t' = 0$, then the RefreshMasks in $R_{\lambda+1} \setminus R_\lambda$ are not probed, and from $|O_{\lambda+1}| < \ell + \lambda + 1$ we can perfectly simulate all output variables corresponding to $O_{\lambda+1}$ by generating random values, without the knowledge of any input variable, and therefore $x_{\lambda+1}$ is not needed for the simulation and one can keep $I_{\lambda+1} = I_\lambda$.
- If $t' \geqslant 1$, then the knowledge of $x_{\lambda+1}$ is required for the simulation and we take $I_{\lambda+1} = I_\lambda \cup \{\lambda+1\}$.

In both cases we have as required:

$$|I_{\lambda+1}| \leq |I_\lambda| + t' \leq t_\lambda + t' \leq t_{\lambda+1}$$

We now proceed with the construction of $J_{\lambda+1}$. We distinguish two cases from the previous application of the recursion hypothesis:

- If $S_\lambda = (O_\lambda \cap [1, \ell]) \cup J_\lambda$ with $|J_\lambda| \leq t_\lambda - \lambda$, then we obtain:

  $$S_\lambda = \big(\big((O_{\lambda+1} \cap [1, \ell+\lambda]) \cup S'\big) \cap [1, \ell]\big) \cup J_\lambda = (O_{\lambda+1} \cap [1, \ell]) \cup \big((S' \cap [1, \ell]) \cup J_\lambda\big) \ .$$

  Therefore, we take $S_{\lambda+1} = S_\lambda$ and $J_{\lambda+1} = (S' \cap [1, \ell]) \cup J_\lambda$. We deduce that $|J_{\lambda+1}| \leqslant |S'| + |J_\lambda| \leqslant (t' - 1) + (t_\lambda - \lambda) \leqslant t_{\lambda+1} - (\lambda + 1)$, which implies that the recursion hypothesis is satisfied for $R_{\lambda+1}$.
- If $S_\lambda = J_\lambda$ with $|I_\lambda| + |J_\lambda| \leq t_\lambda + 1$ and $|J_\lambda| \leq t_\lambda$, then we take $S_{\lambda+1} = S_\lambda = J_{\lambda+1} = J_\lambda$. As a consequence, because $|I_{\lambda+1}| \leqslant |I_\lambda| + t'$, we have $|I_{\lambda+1}| + |J_{\lambda+1}| \leqslant t' + |I_\lambda| + |J_\lambda| \leqslant t' + t_\lambda + 1 \leqslant t_{\lambda+1} + 1$ and $|J_{\lambda+1}| = |J_\lambda| \leqslant t_\lambda \leqslant t_{\lambda+1}$, which implies that the recursion hypothesis is satisfied for $R_{\lambda+1}$.

$\circ$ *Case* $O_\lambda = S'$ *with* $|O_\lambda| \leqslant t'$. In this case, since $t_{\lambda+1} < \ell + \lambda + 1$ from (5), we distinguish two cases:

- If $t_{\lambda+1} = \ell + \lambda$, we can take $I_{\lambda+1} = [1, \lambda+1]$ and $J_{\lambda+1} = [1, \ell]$, which gives $|I_{\lambda+1}| + |J_{\lambda+1}| \leq t_{\lambda+1} + 1$ and $|J_{\lambda+1}| \leq t_{\lambda+1}$ as required.
- If $t_{\lambda+1} < \ell + \lambda$, then we deduce that:

  $$|O_\lambda| + t_\lambda \leqslant t' + t_\lambda = t_{\lambda+1} < \ell + \lambda$$

  Therefore, one can use as previously the recurrence assumption which ensures that $|I_\lambda| \leqslant t_\lambda$ and $S_\lambda = (O_\lambda \cap [1, \ell]) \cup J_\lambda$ with $|J_\lambda| \leq t_\lambda - \lambda$, or $S_\lambda = J_\lambda$ with $|I_\lambda| + |J_\lambda| \leq t_\lambda + 1$ and $|J_\lambda| \leq t_\lambda$. As in the previous case, we take $I_{\lambda+1} = I_\lambda$ if $t' = 0$ and $I_{\lambda+1} = I_\lambda \cup \{\lambda + 1\}$ if $t' \geqslant 1$, which gives $|I_{\lambda+1}| \leq t_{\lambda+1}$. It remains to build the set $J_{\lambda+1}$. As before, we distinguish two cases:
  - If $S_\lambda = (O_\lambda \cap [1, \ell]) \cup J_\lambda$ with $|J_\lambda| \leq t_\lambda - \lambda$, then since $O_\lambda = S'$, we can take $S_{\lambda+1} = J_{\lambda+1} = S_\lambda = (S' \cap [1, \ell]) \cup J_\lambda$. Therefore we have $|J_{\lambda+1}| \leqslant |S'| + |J_\lambda| \leqslant t' + t_\lambda - \lambda \leqslant t_{\lambda+1}$. Furthermore, since we always have $|I_{\lambda+1}| \leqslant \lambda + 1$, we deduce that $|I_{\lambda+1}| + |J_{\lambda+1}| \leqslant \lambda + 1 + t' + t_\lambda - \lambda = t' + t_\lambda + 1 \leqslant t_{\lambda+1} + 1$, which means that the recursion hypothesis is satisfied for $R_{\lambda+1}$.
  - If $S_\lambda = J_\lambda$ with $|I_\lambda| + |J_\lambda| \leq t_\lambda + 1$ and $|J_\lambda| \leq t_\lambda$, then we take $S_{\lambda+1} = S_\lambda = J_{\lambda+1} = J_\lambda$. As a consequence, because $|I_{\lambda+1}| \leqslant t' + |I_\lambda|$, we have $|I_{\lambda+1}| + |J_{\lambda+1}| \leqslant t' + |I_\lambda| + |J_\lambda| \leqslant t' + t_\lambda + 1 \leqslant t_{\lambda+1} + 1$ and $|J_{\lambda+1}| = |J_\lambda| \leqslant t_\lambda \leqslant t_{\lambda+1}$, which means that that the recursion hypothesis is satisfied for $R_{\lambda+1}$.

Hence, we proved that if R_λ is t-SNI then so does $R_{\lambda+1}$. This concludes the proof of Lemma 6.

□

We show an extension of Lemma 6 where after the HTableInc we perform the regular Htable algorithm, where the number of output shares in the table remains constant and equal to $\ell + \lambda$.

Lemma 7. *Let $(x_i)_{1\leq i\leq \lambda+\delta}$ and T be the input, where the table T has ℓ output shares. Let $T' \leftarrow \mathsf{HTableInc}(T, x_1, \ldots, x_\lambda)$, and $T^{out} \leftarrow \mathsf{HTable}(T', x_{\lambda+1}, \ldots, x_{\lambda+\delta})$. For any set of t intermediate variables and any subset of indices $\mathcal{O}$, if $t + |\mathcal{O}| < \ell + \lambda$, there exist subsets $I \subset [1, \lambda + \delta]$ and $J \subset [1, \ell]$ with $|I| \leq t$, such that those t variables as well as the output shares $T^{out}_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$ and $T_{|\mathcal{S}}$, where $\mathcal{S} = (\mathcal{O} \cap [1, \ell]) \cup J$ with $|J| \leq t - \lambda$, or $\mathcal{S} = J$ with $|I| + |J| \leq t + 1$ and $|J| \leq t$.*

Proof. We use lemmas 4 and 6 to prove that the composition of the HTable gadget (Gadget 1) with the HTableInc gadget (Gadget 2) allows the entire circuit to be t-SNI.

Let $\mathcal{I} = \mathcal{I}^1 \cup \mathcal{I}^2$ be a set of indices such that $|\mathcal{I}| \leq t$, corresponding to observations of intermediate variables done by the attacker in the two gadgets, and let $\mathcal{O}$ be a set of indices such that $t + |\mathcal{O}| < \ell + \lambda$, corresponding to observations on the outputs made by the attacker.

Gadget HTable. By assumption, we know that $\left|\mathcal{I}^1\right| + |\mathcal{O}| \leq |\mathcal{I}| + |\mathcal{O}| \leq t + |\mathcal{O}| < \ell + \lambda$. Since from Lemma 4, the HTable gadget is t-SNI, this means that there exist two sets of indices $\mathcal{S}^1_1 \subset [\lambda + 1, \lambda + \delta]$ and $\mathcal{S}^1_2 \subset [1, \ell + \lambda]$ such that $\left|\mathcal{S}^1_1\right| \leq \left|\mathcal{I}^1\right|$, $\left|\mathcal{S}^1_2\right| \leq \left|\mathcal{I}^1\right|$ and the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^1_1$ for the x_i's and $\mathcal{S}^1_2 \cup \mathcal{O}$ for the table T'.

Gadget HTableInc. Since $\left|\mathcal{I}^2\right| + \left|\mathcal{S}^1_2 \cup \mathcal{O}\right| \leq \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| + |\mathcal{O}| \leq |\mathcal{I}| + |\mathcal{O}| \leq t + |\mathcal{O}| < \ell + \lambda$, from Lemma 6, there exist two sets of indices $\mathcal{S}^2$ and $\mathcal{S}^2_2$ such that the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^2_2$ for the x_i's, with $\left|\mathcal{S}^2_2\right| \leq \left|\mathcal{I}^2\right|$, and indices in $\mathcal{S}^2$ for table T, where

- $\mathcal{S}^2 = ((\mathcal{O} \cup \mathcal{S}^1_2) \cap [1, \ell]) \cup \mathcal{S}^2_1$ with $\left|\mathcal{S}^2_1\right| \leq \left|\mathcal{I}^2\right| - \lambda$, or
- $\mathcal{S}^2 = \mathcal{S}^2_1$ with $\left|\mathcal{S}^2_1\right| + \left|\mathcal{S}^2_2\right| \leq \left|\mathcal{I}^2\right| + 1$ and $\left|\mathcal{S}^2_1\right| \leq \left|\mathcal{I}^2\right|$.

Each of the previous steps ensures the existence of a simulator for each gadget. Let $I = \mathcal{S}^1_1 \cup \mathcal{S}^2_2$ and $\mathcal{S} = \mathcal{S}^2$. We can then compose these simulators to perfectly simulate the computation of Algorithm 11 from $x_{|I}$ and $T_{|\mathcal{S}}$. Furthermore, from Gadget HTableInc we either have:

- $\mathcal{S} = (\mathcal{O} \cap [1, \ell]) \cup J$ with $J = (\mathcal{S}^1_2 \cap [1, \ell]) \cup \mathcal{S}^2_1$ and $|J| \leq |\mathcal{S}^1_2| + |\mathcal{S}^2_1| \leq \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| - \lambda \leq |\mathcal{I}| - \lambda \leq t - \lambda$, or
- $\mathcal{S} = J = \mathcal{S}^2_1$ with $|I| + |J| \leq |\mathcal{S}^1_1| + |\mathcal{S}^2_2| + |\mathcal{S}^2_1| \leq |\mathcal{S}^1_1| + \left|\mathcal{I}^2\right| + 1 \leq \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + 1 \leq |\mathcal{I}| + 1 \leq t + 1$ and $|J| = |\mathcal{S}^2_1| \leq \left|\mathcal{I}^2\right| \leq t$.

which proves the lemma. □

Finally we describe the Common Table Inc algorithm below, which is similar to the Common Table algorithm described in Section 6.3, but with increasing number of shares. Note that as explained in Section 6.5, for the evaluation of two SBoxes in parallel, the table T starts with 2 shares instead of 1 at Line 2.

Theorem 6 (t-SNI of Common Table Inc). *Let $(x_i^{(\ell)})_{1\leq i\leq n}$ for $\ell \in \{1, 2\}$ be the input of* Common Table Inc *and $(y^{(\ell)})_{1\leq i\leq n}$ be the output. For any set of t intermediate variables and any subset of indices $\mathcal{O}$ with $t + |\mathcal{O}| < n$, there exists a subset $I \subset [1, n]$ with $|I| \leq t$, such that those t variables as well as the output shares $(y^{(1)})_{|\mathcal{O}}$ and $(y^{(2)})_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$ and $y_{|I}$.*

Algorithm 11 Common Table Inc: evaluation of of $y^{(1)} = S(x^{(1)})$ and $y^{(2)} = S(x^{(2)})$

Input: $x_1^{(\ell)}, \ldots, x_n^{(\ell)}$ for $\ell \in \{1, 2\}$
Output: $y_1^{(\ell)}, \ldots, y_n^{(\ell)}$ such that $y_1^{(\ell)} \oplus \cdots \oplus y_n^{(\ell)} = S(x_1^{(\ell)} \oplus \cdots \oplus x_n^{(\ell)})$ for $\ell \in \{1, 2\}$.

1: $(r_i)_{1 \le i \le n/2}, (a_i^{(1)})_{1 \le i \le n/2}, (a_i^{(2)})_{1 \le i \le n/2} \leftarrow \mathsf{CommonShares}(x_i^{(1)}, x_i^{(2)})$
2: **for all** $u \in \{0,1\}^k$ **do** $T(u) \leftarrow \big(S(u), 0\big) \in (\{0,1\}^{k'})^2$ ▷ $\oplus\big(T(u)\big) = S(u)$
3: $T^{(1)} \leftarrow \mathsf{HTableInc}(T, r_1, \ldots, r_{n/2})$
4: $T^{(2)} \leftarrow T^{(1)}$
5: $T^{(1)} \leftarrow \mathsf{HTableInc}(T^{(1)}, a_1^{(1)}, \ldots, a_{n/2-2}^{(1)})$
6: $T^{(2)} \leftarrow \mathsf{HTableInc}(T^{(2)}, a_1^{(2)}, \ldots, a_{n/2-2}^{(2)})$ ▷ $T^{(1)}$ and $T^{(2)}$ now have n shares.
7: $T^{(1)} \leftarrow \mathsf{Htable}(T^{(1)}, a_{n/2-1}^{(1)})$
8: $T^{(2)} \leftarrow \mathsf{Htable}(T^{(2)}, a_{n/2-1}^{(2)})$
9: $(y_1^{(1)}, \ldots, y_n^{(1)}) \leftarrow \mathsf{LinearRefreshMasks}\big(T^{(1)}(a_{n/2}^{(1)})\big)$
10: $(y_1^{(2)}, \ldots, y_n^{(2)}) \leftarrow \mathsf{LinearRefreshMasks}\big(T^{(2)}(a_{n/2}^{(2)})\big)$
11: **return** $y_1^{(\ell)}, \ldots, y_n^{(\ell)}$ for $\ell \in \{1, 2\}$

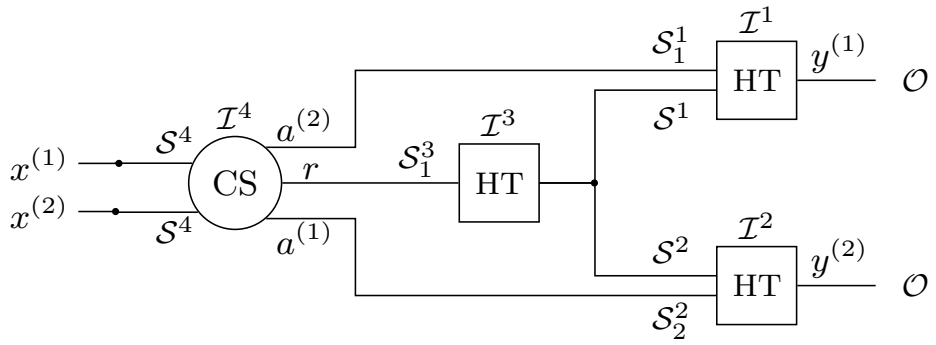

Fig. 7. Illustration of common input table as composition of gadgets. Each variable $x^{(\ell)}$ and $y^{(\ell)}$ contains actually n shares $x_i^{(\ell)}$ and $y_i^{(\ell)}$, for $\ell = \{1, 2\}$.

Proof. We use Lemmas 3 and 7 to prove that the composition of the CommonShares gadget with the three HTable gadgets allows the entire circuit to be t-SNI. We label the gadgets from 1 to 4 starting from right to left (see Figure 7).

Let $\mathcal{I} = \mathcal{I}^1 \cup \mathcal{I}^2 \cup \mathcal{I}^3 \cup \mathcal{I}^4$ be a set of indices such that $|\mathcal{I}| \le t$, corresponding to observations of intermediate variables done by the attacker in the four gadgets, and let $\mathcal{O}$ be a set of indices such that

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

corresponding to observations on the outputs made by the attacker.

Gadget 1. By assumption, we know that $\big|\mathcal{I}^1\big| + |\mathcal{O}| \le |\mathcal{I}| + |\mathcal{O}| \le t + |\mathcal{O}| < n$. Since from Lemma 7, the composition of the HTableInc and HTable gadgets is t-SNI, and since the number of output shares increases by $n/2 - 2$, this means that there exist two sets of indices $\mathcal{S}_1^1$ and $\mathcal{S}^1$ such that the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}_1^1$ with $\big|\mathcal{S}_1^1\big| \le \big|\mathcal{I}^1\big|$, and indices in $\mathcal{S}^1$ where

- **Case 1.a:** $\mathcal{S}^1 = (\mathcal{O} \cap [1, \frac{n}{2} + 1]) \cup \mathcal{S}_2^1$ with $\big|\mathcal{S}_2^1\big| \le \big|\mathcal{I}^1\big| - (\frac{n}{2} - 2)$, or
- **Case 1.b:** $\mathcal{S}^1 = \mathcal{S}_2^1$ with $\big|\mathcal{S}_1^1\big| + \big|\mathcal{S}_2^1\big| \le \big|\mathcal{I}^1\big| + 1$ and $\big|\mathcal{S}_2^1\big| \le \big|\mathcal{I}^1\big|$.

Gadget 2. Similarly, since $\left|\mathcal{I}^2\right| + |\mathcal{O}| \leq |\mathcal{I}| + |\mathcal{O}| \leq t + |\mathcal{O}| < n$, from Lemma 7, there exist two sets of indices $\mathcal{S}^2$ and $\mathcal{S}_2^2$ such that the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}_2^2$ with $\left|\mathcal{S}_2^2\right| \leq \left|\mathcal{I}^2\right|$, and indices in $\mathcal{S}^2$ where

- **Case 2.a:** $\mathcal{S}^2 = (\mathcal{O} \cap [1, \frac{n}{2}+1]) \cup \mathcal{S}_1^2$ with $\left|\mathcal{S}_1^2\right| \leq \left|\mathcal{I}^2\right| - (\frac{n}{2}-2)$, or
- **Case 2.b:** $\mathcal{S}^2 = \mathcal{S}_1^2$ with $\left|\mathcal{S}_1^2\right| + \left|\mathcal{S}_2^2\right| \leq \left|\mathcal{I}^2\right| + 1$ and $\left|\mathcal{S}_1^2\right| \leq \left|\mathcal{I}^2\right|$.

Gadget 3. From gadgets 1 and 2, we have four possible values for $\left|\mathcal{S}^1 \cup \mathcal{S}^2\right|$:

- **Case 1.a/2.a:** We have

$$\begin{aligned}\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| &= \left|(\mathcal{O} \cap [1, n/2+1]) \cup \mathcal{S}_2^1 \cup (\mathcal{O} \cap [1, n/2+1]) \cup \mathcal{S}_1^2\right| \\ &\leqslant \left|\mathcal{O} \cup \mathcal{S}_2^1 \cup \mathcal{S}_1^2\right| \leqslant |\mathcal{O}| + \left|\mathcal{I}^1\right| - n/2 + 2 + \left|\mathcal{I}^2\right| - n/2 + 2 \\ &\leqslant |\mathcal{O}| + \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| - n + 4\end{aligned}$$

  This gives from $|\mathcal{O}| + |\mathcal{I}| < n$:

$$\begin{aligned}\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| &\leqslant |\mathcal{O}| + |\mathcal{I}| - n + 4 \\ &< n - n + 4 = 4\end{aligned}$$

  For $n \geq 4$, this gives:

$$\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| < n/2 + 2$$

  We note that the inequality is also satisfied for $n = 3$. Namely in that case, there is only one common share, and there is no increase in the number of outputs shares in gadgets 1 and 2. Therefore one can apply Lemma 4 to gadgets 1 and 2, and we get $\left|\mathcal{S}_2^1\right| \leq \left|\mathcal{I}^1\right|$ and $\left|\mathcal{S}_1^2\right| \leq \left|\mathcal{I}^2\right|$. As a consequence, we have:

$$\begin{aligned}\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| &\leqslant \left|\mathcal{O} \cup \mathcal{S}_2^1 \cup \mathcal{S}_1^2\right| + \left|\mathcal{I}^3\right| \leqslant |\mathcal{O}| + \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| \\ &\leqslant |\mathcal{O}| + |\mathcal{I}| < n = 3 < n/2 + 2\end{aligned}$$

- **Case 1.a/2.b:** We have:

$$\begin{aligned}\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| &= \left|(\mathcal{O} \cap [1, n/2+1]) \cup \mathcal{S}_2^1 \cup \mathcal{S}_1^2\right| + \left|\mathcal{I}^3\right| \leqslant |\mathcal{O}| + \left|\mathcal{I}^1\right| - n/2 + 2 + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| \\ &\leqslant |\mathcal{O}| + |\mathcal{I}| - n/2 + 2 < n - n/2 + 2 = n/2 + 2\end{aligned}$$

- **Case 1.b/2.a:** Similarly, we have:

$$\begin{aligned}\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| &= \left|\mathcal{S}_2^1 \cup (\mathcal{O} \cap [1, n/2+1]) \cup \mathcal{S}_1^2\right| + \left|\mathcal{I}^3\right| \leqslant \left|\mathcal{I}^1\right| + |\mathcal{O}| + \left|\mathcal{I}^2\right| - n/2 + 2 + \left|\mathcal{I}^3\right| \\ &\leqslant |\mathcal{O}| + |\mathcal{I}| - n/2 + 2 < n - n/2 + 2 = n/2 + 2\end{aligned}$$

- **Case 1.b/2.b:** We have:

$$\begin{aligned}\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| &= \left|\mathcal{S}_2^1\right| + \left|\mathcal{S}_1^2\right| + \left|\mathcal{I}^3\right| \\ &\leqslant \left|\mathcal{I}^1\right| - \left|\mathcal{S}_1^1\right| + 1 + \left|\mathcal{I}^2\right| - \left|\mathcal{S}_2^2\right| + 1 + \left|\mathcal{I}^3\right| \\ &\leqslant \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| - \left|\mathcal{S}_1^1\right| - \left|\mathcal{S}_2^2\right| + 2\end{aligned}$$

  Note that in this case, we cannot assert that $\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| < n/2 + 2$. Therefore we distinguish two cases:

- If $\left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| - \left|\mathcal{S}_1^1\right| - \left|\mathcal{S}_2^2\right| < n/2$, then we obtain as in the previous cases:

$$\left|\mathcal{S}^1 \cup \mathcal{S}^2\right| + \left|\mathcal{I}^3\right| < n/2 + 2$$

  In that case, since the HTableInc algorithm in Gadget 3 has $n/2+2$ output shares (since it takes as input a table T with 2 output shares and processes $n/2$ shares r_i), as in the first 3 cases considered above, we can apply Lemma 6 to Gadget 3, which ensures that there exist two sets of indices $\mathcal{S}_1^3$ and $\mathcal{S}^3$ such that $\left|\mathcal{S}_1^3\right| \leq \left|\mathcal{I}^3\right|$, and the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}_1^3$ and $\mathcal{S}^3$. Since Gadget 3 takes as input a table T with public values only, we only consider the set of indices $\mathcal{S}_1^3$ of input shares.
- If $\left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| - \left|\mathcal{S}_1^1\right| - \left|\mathcal{S}_2^2\right| \geq n/2$, we cannot apply Lemma 6 and we must set $\mathcal{S}_1^3 = [1, n/2]$ to simulate Gadget 3.

Gadget 4. From Lemma 3 there exists a set of indices $\mathcal{S}^4$ such that

$$\left|\mathcal{S}^4\right| \leq \left|\mathcal{I}^4\right| + \left|\mathcal{S}_1^3\right| + \left|\mathcal{S}_1^1\right| + \left|\mathcal{S}_2^2\right|$$

and Gadget 4 can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^4$.

Each of the previous steps ensures the existence of a simulator for each gadget. Letting $I = \mathcal{S}^4$, we can then compose these simulators to perfectly simulate the computation of Algorithm 11 from $x_{|I}$ and $y_{|I}$. It remains to show that $|I| \leqslant t$. Two cases arise depending on Gadget 3.

- If $\left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| - \left|\mathcal{S}_1^1\right| - \left|\mathcal{S}_2^2\right| < n/2$, we have obtained $\left|\mathcal{S}_1^3\right| \leqslant \left|\mathcal{I}^3\right|$. From gadgets 1 and 2, it follows that
$$|I| = \left|\mathcal{S}^4\right| \leqslant \left|\mathcal{I}^4\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| \leqslant t$$
- If $\left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| - \left|\mathcal{S}_1^1\right| - \left|\mathcal{S}_2^2\right| \geq n/2$, we have set $\left|\mathcal{S}_1^3\right| = n/2$, and we still obtain:
$$\begin{aligned} |I| = \left|\mathcal{S}^4\right| &\leqslant \left|\mathcal{I}^4\right| + \left|\mathcal{S}_1^3\right| + \left|\mathcal{S}_1^1\right| + \left|\mathcal{S}_2^2\right| \\ &\leqslant \left|\mathcal{I}^4\right| + n/2 + \left(\left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| - n/2\right) \leqslant \left|\mathcal{I}^4\right| + \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^3\right| \leqslant t \end{aligned}$$

Therefore in both cases we obtain $|I| \leq t$, which terminates the proof. □