Tight Private Circuits: Achieving Probing Security with the Least Refreshing

Sonia Belaïd[1(✉)], Dahmun Goudarzi[1,2(✉)], and Matthieu Rivain[1(✉)]

[1] CryptoExperts, Paris, France
[2] ENS CNRS INRIA and PSL Research University, Paris, France
{sonia.belaid,dahmun.goudarzi,matthieu.rivain}@cryptoexperts.com

Abstract. Masking is a common countermeasure to secure implementations against side-channel attacks. In 2003, Ishai, Sahai, and Wagner introduced a formal security model, named t-probing model, which is now widely used to theoretically reason on the security of masked implementations. While many works have provided security proofs for small masked components, called *gadgets*, within this model, no formal method allowed to securely compose gadgets with a tight number of shares (namely, $t+1$) until recently. In 2016, Barthe *et al.* filled this gap with `maskComp`, a tool checking the security of masking schemes composed of several gadgets. This tool can achieve provable security with tight number of shares by inserting mask-refreshing gadgets at carefully selected locations. However the method is not tight in the sense that there exists some compositions of gadgets for which it cannot exhibit a flaw nor prove the security. As a result, it is overconservative and might insert more refresh gadgets than actually needed to ensure t-probing security. In this paper, we exhibit the first tool, referred to as `tightPROVE`, able to clearly state whether a shared circuit composed of standard gadgets (addition, multiplication, and refresh) is t-probing secure or not. Given such a composition, our tool either produces a probing-security proof (valid at any order) or exhibits a security flaw that directly implies a probing attack at a given order. Compared to `maskComp`, `tightPROVE` can drastically reduce the number of required refresh gadgets to get a probing security proof, and thus the randomness requirement for some secure shared circuits. We apply our method to a recent AES implementation secured with higher-order masking in bitslice and we show that we can save all the refresh gadgets involved in the s-box layer, which results in an significant performance gain.

Keywords: Side-channel · Masking · Composition · Private circuits

1 Introduction

Most cryptographic algorithms are assumed to be secure against the so-called *black-box* attacks, where the adversary is restricted to the knowledge of inputs and outputs to recover the secret key. However, the late nineties revealed a new

T. Peyrin and S. Galbraith (Eds.): ASIACRYPT 2018, LNCS 11273, pp. 343–372, 2018.
https://doi.org/10.1007/978-3-030-03329-3_12

class of attacks, referred to as *side-channel attacks*, that exploit the physical leakages (*e.g.* temperature, power consumption) of components which execute implementations of cryptographic algorithms. Many implementations of symmetric cryptographic algorithms have been broken so far [7,17], raising the need for concrete and efficient protection.

A sound and widely deployed approach to counteract side-channel attacks is the so-called *masking* countermeasure that was simultaneously introduced in 1999 by Chari *et al.* [8] and by Goubin and Patarin [13]. The idea is to split each key-dependent variable x of the implementation into d *shares* $(x_i)_{0 \leq i \leq d-1}$ such that $x = x_0 * \cdots * x_{d-1}$ for some law $*$ and any strict subset of shares is uniformly distributed. The number of degrees-of-freedom $d-1$ of such a sharing is referred to as the *masking order*. When $*$ is the addition on a finite field of characteristic two, the approach is referred to as *Boolean masking*, and when d is additionally strictly greater than 2, the approach is referred to as *higher-order Boolean masking*. Chari *et al.* showed that recombining d noisy shares to recover the secret is then exponentially complex in d which makes the masking order a sound security parameter with respect to side-channel attacks.

In order to design masking schemes and theoretically reason on their security, the community has defined several leakage models. In the most realistic one, the *noisy leakage model* introduced by Rivain and Prouff [19] as a specialization of the *only computation leaks* model [18], the adversary gets a noisy function of each intermediate variable of the cryptographic computation. Unfortunately, this model is not very convenient to build security proofs as it requires complex mutual information computations. A second and widely used leakage model is the *t-probing model* introduced by Ishai, Sahai, and Wagner [15] in which the adversary gets the exact values of t chosen intermediate variables. As it manipulates exact values in a limited quantity, this model is advantageously much more convenient for security proofs. In order to benefit from the advantages of both models, Duc, Dziembowski, and Faust demonstrated in [12] a reduction from the noisy leakage model to the t-probing model. In a nutshell, an implementation that is secure in the t-probing model is also secure in the more realistic noisy leakage model for some level of noise.

In their seminal work [15], Ishai *et al.* proposed a t-probing secure masking scheme for any circuit based on $d = 2t + 1$ shares. This scheme was extended by Rivain and Prouff in [20] with the aim to derive a tight t-probing secure implementation of AES, where *tightness* means that the t-probing security is obtained with the optimal number of $d = t + 1$ shares. In particular, they show that the so-called ISW multiplication gadget actually achieves tight probing security provided that the two input sharings are mutually independent. In order to obtain tight security for the full AES circuit, Rivain and Prouff suggested to insert *refresh gadgets* that renew the randomness of sharings at carefully chosen locations [20]. But the proposed refresh gadget was shown to introduce a flaw in the composition [10]. In 2016, Barthe *et al.* introduced new security notions to fill this gap, namely the t-*non interference* and the t-*strong non interference* [2]. When these notions are met by a set of gadgets, one can easily reason on the probing security of their composition. Informally, a gadget is t-non interfering (or t-NI) if and only if any set of at most t intermediate variables can be perfectly simulated with at most t shares of each input. Since t input shares are trivially independent from the input itself as long as $t < d$, non-interference trivially implies probing security. While this notion was first defined in [2], it was actually already met by most existing gadgets. One step further, a gadget is t-strong non interfering (or t-SNI) if and only if any set of t intermediate variables among which t_{out} are output variables can be perfectly simulated with $t_{int} = t - t_{out}$ shares of each input sharing. This property makes it possible to compose any set of SNI gadgets since it stops the propagation of dependencies. A concrete tool to build probing secure implementations from unprotected implementations is provided [2] which was later called `maskComp`. Following this work, numerous examples of globally probing secure schemes were proposed with a decomposition in identified NI and SNI gadgets [3,11,21]. While these schemes achieve their security goals, each inserted SNI refresh gadget increase the requirement of fresh randomness which is generally expensive to generate. And up to now, no efficient method exists to check the probing security of any given composition of gadgets. As a result, existing tools such as `maskComp` are overconservative and might insert more refresh gadgets than necessary.

Nevertheless, some formal tools have been recently developed to evaluate the probing security of implementations at a given masking order. Among the most efficient ones, Barthe *et al.* developed `maskVerif` [1] and Coron developed `CheckMasks` [9]. Both tools take as input a shared circuit and return a formal security proof when no attack is found. But here again, this evaluation is not tight and false negatives may occur and hence imply the addition of unnecessary refresh gadgets. Moreover, while such tools are very convenient to evaluate the security of concrete implementations, they suffer from an important limitation which is their exponential complexity in the size of the circuit and consequently in the masking order. As a result, these tools are impractical beyond a small number of shares (typically $d = 5$). In a recent work, Bloem *et al.* [5] further developed a new tool to verify the security of masked implementations subject to *glitches*, which is an important step towards provable and practical security of hardware implementations. However this tool still suffers from the same efficiency limitations as the previous ones.

Motivation and Contributions. The method of Barthe *et al.* [2] allows one to safely compose t-NI and t-SNI gadgets and get probing security at any order. Nevertheless, it is not tight and makes use of more refresh gadgets than required. In many contexts, randomness generation is expensive and might be the bottleneck for masked implementations. For instance, Journault Standaert describe an AES encryption shared at the order $d = 32$ for which up to 92% of the running time is spent on randomness generation [16]. In such a context, it is fundamental to figure out whether the number of t-SNI refresh gadgets inserted by Barthe *et al.*'s tool `maskComp` is actually minimal to achieve t-probing security. In this paper, we find out that it is not and we provide a new method which *exactly* identifies the concrete probing attacks in a Boolean shared circuit.

Let us take a simple example. We consider the small randomized circuit referred to as Circuit 1 and illustrated in Fig. 1 with $[\oplus]$ a t-NI sharewise addition, $[\otimes]$ a t-SNI multiplication, and two Boolean sharings $[x_1]$ and $[x_2]$. Applying Barthe *et al.*'s tool `maskComp` on this circuit automatically inserts a t-SNI refresh gadget in the cycle formed by gates $[x_1]$, $[\oplus]$, and $[\otimes]$ as represented in Fig. 2. However, it can be verified that for any masking order t, the initial circuit is t-probing secure without any additional refresh gadget. Therefore, in the following, this paper aims to refine the state-of-the-art method [2] to only insert refresh gadgets when absolutely mandatory for the t-probing security.

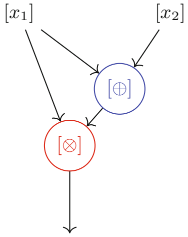


Fig. 1. Graph representation of Circuit 1.

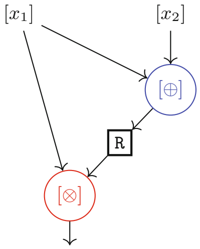


Fig. 2. Graph representation of Circuit 1 after `maskComp`.

More specifically, our contributions can be summarized as follows:

(1) We introduce formal definitions of the probing, non-interfering, and strong-non-interfering security notions for shared circuits based on concrete security games. Although these definitions are no more than a reformulation of existing security notions, we believe that they provide a simple and precise framework to reason about probing security.

(2) From the introduced game-based definitions, we provide a reduction of the probing security of a given *standard shared circuit* –*i.e.* a shared circuit composed of ISW multiplication gadgets, sharewise addition gadgets and SNI refresh gadgets– to the probing security of a simpler circuit of multiplicative depth 1 and for which the adversary is restricted to probe the multiplication inputs (which are linear combinations of the circuit inputs).

(3) We give an algebraic characterization of the final security game, which allows us to express the probing security of any standard shared circuit in terms of linear algebra.

(4) We show how to solve the latter problem with a new exact and proven method. Our method takes the description of any standard shared circuit and either produces a probing-security proof (valid at any order) or exhibits a probing attack (*i.e.* a set of $t < d$ probes that reveal information on the circuit d-shared input for some d). We provide a concrete tool, named

tightPROVE (for tight PRObing VErification), implementing our method in Sage.

(5) We apply tightPROVE to the efficient implementation of the AES s-box developed by Goudarzi and Rivain in [14]. Based on the previous state of the art, this s-box was implemented using one SNI refresh gadget per multiplication gadget (to refresh one of the operands), hence requiring a total of 32 refresh gadgets (which was later on confirmed by the maskComp tool). Our new method formally demonstrates that the same d-shared implementation is actually t-probing secure with *no* refresh gadget for any $d = t + 1$. We provide implementation results and a performance analysis: this new implementation achieves an asymptotic gain up to 43%. The code is provided at https://github.com/CryptoExperts/tightPROVE.

(6) We extend our results to larger circuits by establishing new compositional properties on t-probing secure gadgets. In particular, these new composition properties well apply to the case of SPN-based block ciphers. We also show that they apply to a wide range of Boolean circuits with common gadgets and input sets.

Paper Organization. In Sect. 2, useful notions are introduced, security definitions for composition are formalized through concrete security games, and some useful security results are provided. Section 3 provides our security reduction for standard shared circuits. Section 4 then details our new method to exactly determine the probing security of a standard shared circuit. It also gives an upper bound on the number of required refresh gadgets together with an exhaustive method to make a standard shared circuit achieve tight probing security. In Sect. 5, our new method is extended to apply to larger circuits, and in particular to SPN-based block ciphers, with new compositional properties. Finally, Sect. 6 describes the new tool tightPROVE we implemented to experiment our method on concrete circuits.

2 Formal Security Notions

2.1 Notations

In this paper, we denote by $\mathbb{F}_2$ the finite field with two elements and by $[\![i, j]\!]$ the integer interval $\mathbb{Z} \cap [i, j]$ for any two integers i and j. For a finite set $\mathcal{X}$, we denote by $|\mathcal{X}|$ the cardinality of $\mathcal{X}$ and by $x \leftarrow \mathcal{X}$ the action of picking x from $\mathcal{X}$ independently and uniformly at random. For some (probabilistic) algorithm $\mathcal{A}$, we further denote $x \leftarrow \mathcal{A}(in)$ the action of running algorithm $\mathcal{A}$ on some inputs in (with fresh uniform random tape) and setting x to the value returned by $\mathcal{A}$.

2.2 Basic Notions

A *Boolean circuit* is a directed acyclic graph whose vertices are input gates, output gates, constant gates of fan-in 0 that output constant values, and operation gates of fan-in at most 2 and fan-out at most 1 and whose edges are wires. In this paper we consider Boolean circuits with two types of operation gates: addition gates (computing an addition on $\mathbb{F}_2$) and multiplication gates (computing a multiplication on $\mathbb{F}_2$). A *randomized circuit* is a Boolean circuit augmented with random-bit gates of fan-in 0 that outputs a uniformly random bit.

A *d-Boolean sharing* of $x \in \mathbb{F}_2$ is a random tuple $(x_0, x_1, \ldots, x_{d-1}) \in \mathbb{F}_2^d$ satisfying $x = \sum_{i=0}^{d-1} x_i$. The sharing is said to be *uniform* if, for a given x, it is uniformly distributed over the subspace of tuples satisfying $x = \sum_{i=0}^{d-1} x_i$. A uniform sharing of x is such that any m-tuple of its *shares* x_i is uniformly distributed over $\mathbb{F}_2^m$ for any $m \leq d-1$. In the following, a d-Boolean sharing of a given variable x is denoted by $[x]$ when the sharing order d is clear from the context. We further denote by Enc a probabilistic *encoding* algorithm that maps $x \in \mathbb{F}_2$ to a fresh uniform sharing $[x]$.

A *d-shared circuit* C is a randomized circuit working on d-shared variables. More specifically, a d-shared circuit takes a set of n input sharings $[x_1]$, ..., $[x_n]$ and computes a set of m output sharings $[y_1]$, ..., $[y_m]$ such that $(y_1, \ldots, y_m) = f(x_1, \ldots, x_n)$ for some deterministic function f. A *probe* on C refers to a wire index (for some given indexing of C's wires). An *evaluation* of C on input $[x_1]$, ..., $[x_n]$ under a set of probes $\mathcal{P}$ refers to the distribution of the tuple of wires pointed by the probes in $\mathcal{P}$ when the circuit is evaluated on $[x_1]$, ..., $[x_n]$, which is denoted by $C([x_1], \ldots, [x_n])_{\mathcal{P}}$.

We consider a special kind of shared circuits which are composed of *gadgets*. A gadget is a simple building block of a shared circuit that performs a given operation on its input sharing(s). For instance, for some two-input operation $*$, a $*$-gadget takes two input sharings $[x_1]$ and $[x_2]$ and it outputs a sharing $[y]$ such that $y = x_1 * x_2$. In the paper, we specifically consider three types of gadgets, namely ISW-multiplication gadgets ($[\otimes]$), ISW-refresh gadgets ($[\mathtt{R}]$) and sharewise addition gadgets ($[\oplus]$). The ISW-multiplication gadget, introduced in [15], takes two d-sharings $[a]$ and $[b]$ as inputs and computes the output d-sharing $[c]$ such that $c = a \cdot b$ as follows:

1. for every $0 \leq i < j \leq d-1$, pick uniformly at random a value $r_{i,j}$ over $\mathbb{F}_2$;
2. for every $0 \leq i < j \leq d-1$, compute $r_{j,i} \leftarrow (r_{i,j} + a_i \cdot b_j) + a_j \cdot b_i$;
3. for every $0 \leq i \leq d-1$, compute $c_i \leftarrow a_i \cdot b_i + \sum_{j \neq i} r_{i,j}$.

The ISW-refresh gadget is actually the ISW-multiplication gadget in which the second operand $[b]$ is set to the constant Boolean sharing $(1, 0, \ldots, 0)$. The output $[c]$ is thus a fresh independent sharing of a. Finally, a sharewise addition gadget computes a d-sharing $[c]$ such that $c = a + b$ by letting $c_i \leftarrow a_i + b_i$ for every $0 \leq i \leq d-1$. When called with a second operand equal to the constant Boolean sharing $(1, 0, \ldots, 0)$, such a sharewise addition gadget computes the complementary of its first operand $c = \overline{a}$.

Definition 1. *A standard shared circuit is a shared circuit exclusively composed of ISW-multiplication gadgets, ISW-refresh gadgets, and sharewise addition gadgets as described above.*

2.3 Game-Based Security Definitions

In the following, we recall the *probing*, *non-interfering*, and *strong non-interfering* security notions introduced in [2,15] and we formalize them through concrete security games. Each of these games is defined for a given n-input d-shared circuit C and it opposes an *adversary* $\mathcal{A}$, which is a deterministic algorithm outputting a set of (plain) inputs $x_1, \ldots, x_n$ and a set of probes $\mathcal{P}$, to a simulator $\mathcal{S}$, which aims at simulating the distribution $C([x_1], \ldots, [x_n])_{\mathcal{P}}$.

Probing Security. We first recall the definition from [15]. Our game-based definition is then given with a proposition to state the equivalence of both notions.

Definition 2 (from [15]). *A circuit is t-probing secure if and only if any set of at most t intermediate variables is independent from the secret.*

Probing Security Game. The t-probing security game is built based on two experiments as described in Fig. 3. In both experiments, an adversary $\mathcal{A}$ outputs a set of probes $\mathcal{P}$ (indices of circuit's wires) such that $|\mathcal{P}| = t$ and n input values $x_1, \ldots, x_n \in \mathbb{F}_2$.

In the first (real) experiment, referred to as $\mathsf{ExpReal}$, the chosen input values $x_1, \ldots, x_n$ are mapped into n sharings $[x_1], \ldots, [x_n]$ with encoding algorithm Enc. The resulting encodings are given as inputs to the shared circuit C. The real experiment then outputs a random evaluation $C([x_1], \ldots, [x_n])_{\mathcal{P}}$ of the chosen gates through a t-uple $(v_1, \ldots, v_t)$.

```
$\mathsf{ExpReal}(\mathcal{A}, C)$:
1. $(\mathcal{P}, x_1, \ldots, x_n) \leftarrow \mathcal{A}()$
2. $[x_1] \leftarrow \mathsf{Enc}(x_1), \ldots, [x_n] \leftarrow \mathsf{Enc}(x_n)$
3. $(v_1, \ldots, v_t) \leftarrow C([x_1], \ldots, [x_n])_{\mathcal{P}}$
4. Return $(v_1, \ldots, v_t)$
```

```
$\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$:
1. $(\mathcal{P}, x_1, \ldots, x_n) \leftarrow \mathcal{A}()$
2. $(v_1, \ldots, v_t) \leftarrow \mathcal{S}(\mathcal{P})$
3. Return $(v_1, \ldots, v_t)$
```

Fig. 3. t-probing security game.

In the second experiment, referred to as ExpSim, the probing simulator $\mathcal{S}$ takes the (adversary chosen) set of probes $\mathcal{P}$ and outputs a simulation of the evaluation $C([x_1], \ldots, [x_n])_{\mathcal{P}}$, which is returned by the simulation experiment. The simulator wins the game if and only if the two experiments return identical distributions.

Proposition 1. *A shared circuit C is t-probing secure if and only if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins the t-probing security game defined in Fig. 3, i.e. the random experiments* $\mathsf{ExpReal}(\mathcal{A}, C)$ *and* $\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$ *output identical distributions.*

Proof. From right to left, if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins the t-probing security game defined in Fig. 3, then any set of probes is independent from the secret as $\mathcal{S}$ has no knowledge of the secret inputs. Thus C is trivially t-probing secure by Definition 2. From left to right, if the random experiments $\mathsf{ExpReal}(\mathcal{A}, C)$ and $\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$ do not output identical distributions, then there exists a set of at most t intermediate variables which cannot be perfectly simulated without the knowledge of the input secrets. As a consequence, the circuit is not t-probing secure from Definition 2. □

A shared circuit C which is t-probing secure is referred to as a *t-private circuit*. It is not hard to see that a d-shared circuit can only achieve t-probing security for $d > t$. When a d-shared circuit achieves t-probing security with $d = t + 1$, we call it a *tight private circuit*.

Non-Interfering Security. The non-interfering security notion is a little bit stronger ([2]). Compared to the probing security notion, it additionally benefits from making the security evaluation of composition of circuits easier. We recall its original definition from [2] before we give an equivalent formal game-based definition.

Definition 3 (from [2]). *A circuit is t-non-interfering (t-NI) if and only if any set of at most t intermediate variables can be perfectly simulated from at most t shares of each input.*

Non-Interfering Security Game. The t-non-interfering (t-NI) security game is built based on two experiments as described in Fig. 4. In both experiments, an adversary $\mathcal{A}$ outputs a set of probes $\mathcal{P}$ (indices of circuit's wires) such that $|\mathcal{P}| = t$ and n input sharings $[x_1], \dots, [x_n] \in \mathbb{F}_2^d$.

The first (real) experiment, referred to as $\mathsf{ExpReal}$, simply returns an evaluation of C on input sharings $[x_1]$, ..., $[x_n]$ under the set of probes $\mathcal{P}$.

The second experiment, referred to as ExpSim, is defined for a two-round simulator $\mathcal{S} = (\mathcal{S}^1, \mathcal{S}^2)$. In the first round, the simulator $\mathcal{S}^1$ takes the (adversary chosen) set of probes $\mathcal{P}$ and outputs n sets of indices $\mathcal{I}_1, \dots, \mathcal{I}_n \subseteq \{1, \dots, d\}$, such that $|\mathcal{I}_1| = \dots = |\mathcal{I}_n| = t$. In the second round, in addition to the set of probes $\mathcal{P}$, the simulator $\mathcal{S}^2$ receives the (adversary chosen) input sharings restricted to the shares indexed by the sets $\mathcal{I}_1, \dots, \mathcal{I}_n$, denoted $[x_1]_{\mathcal{I}_1}$, ..., $[x_n]_{\mathcal{I}_n}$, and outputs a simulation of $C([x_1], \dots, [x_n])_{\mathcal{P}}$, which is returned by the simulation experiment. The simulator wins the game if and only if the two experiments return identical distributions.

Proposition 2. *A shared circuit C is* t-non-interfering secure *if and only if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins the t-non-interfering security game defined in Fig. 4, i.e. the random experiments* $\mathsf{ExpReal}(\mathcal{A}, C)$ *and* $\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$ *output identical distributions.*

```
$\mathsf{ExpReal}(\mathcal{A}, C)$:

1. $(\mathcal{P}, [x_1], \dots, [x_n]) \leftarrow \mathcal{A}()$
2. $(v_1, \dots, v_t) \leftarrow C([x_1], \dots, [x_n])_{\mathcal{P}}$
3. Return $(v_1, \dots, v_t)$
```

```
$\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$: *

1. $(\mathcal{P}, [x_1], \dots, [x_n]) \leftarrow \mathcal{A}()$
2. $\mathcal{I}_1, \dots, \mathcal{I}_n \leftarrow \mathcal{S}^1(\mathcal{P})$
3. $(v_1, \dots, v_t) \leftarrow \mathcal{S}^2(\mathcal{P}, [x_1]_{\mathcal{I}_1}, \dots, [x_n]_{\mathcal{I}_n})$
4. Return $(v_1, \dots, v_t)$
```

* For t-NI: $|\mathcal{I}_1| = \dots = |\mathcal{I}_n| = t$.
For t-SNI: $|\mathcal{I}_1| = \dots = |\mathcal{I}_n| = |\mathcal{P}_{int}| \leq t$.

Fig. 4. t-(S)NI security game.

Proof. From right to left, if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins the t-non interfering security game defined in Fig. 3, then any set of probes can be perfectly simulated from sets of at most t shares of each input. Thus C is trivially t-non-interfering from Definition 3. From left to right, if the random experiments $\mathsf{ExpReal}(\mathcal{A}, C)$ and $\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$ do not output identical distributions, then there exists a set of at most t intermediate variables which cannot be perfectly simulated from sets $\mathcal{I}_j$ of input shares whose cardinalities are less than t. As a consequence, the circuit is not t-non interfering secure from Definition 3. □

Strong Non-Interfering Security. The strong non-interfering security is a stronger notion than non-interfering security as it additionally guarantees the independence between input and output sharings. The latter property is very convenient to securely compose gadgets with related inputs.

Definition 4 (Strong non-interfering security from [2]). *A circuit is t-strong non-interfering (t-SNI) if and only if any set of at most t intermediate variables whose t_1 on the internal variables (i.e. intermediate variables except the output's ones) and t_2 on output variables can be perfectly simulated from at most t_1 shares of each input.*

Strong Non-Interfering Security Game. The t-strong-non-interfering (t-SNI) security game is similar to the t-NI security game depicted in Fig. 4. The only difference relies in the fact that the first-round simulator $\mathcal{S}^1$ outputs n sets of indices $\mathcal{I}_1, \dots, \mathcal{I}_n \subseteq \{1, \dots, d\}$, such that $|\mathcal{I}_1| = \dots = |\mathcal{I}_n| = |\mathcal{P}_{int}| \leq t$ where $\mathcal{P}_{int} \subseteq \mathcal{P}$ refers to the probes on internal wires, *i.e.* the probes in $\mathcal{P}$ which do not point to outputs of C.

Proposition 3. *A shared circuit C is* t-strong-non-interfering secure *if and only if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins the t-SNI security game defined in Fig. 4, i.e. the random experiments* $\mathsf{ExpReal}(\mathcal{A}, C)$ *and* $\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$ *output identical distributions.*

Proof. From right to left, if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins the t-non interfering security game defined in Fig. 3, then any set of probes can be perfectly simulated from sets of at most $|\mathcal{P}_{int}| = t_1$ shares of each input. Thus C is trivially t-strong non-interfering from Definition 4. From left to right, if the random experiments $\mathsf{ExpReal}(\mathcal{A}, C)$ and $\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$ do not output identical distributions, then there exists a set of at most t intermediate variables which cannot be perfectly simulated from sets $\mathcal{I}_j$ of input shares whose cardinalities are less than t_1. As a consequence, the circuit is not t-strong non interfering secure from Definition 4. □

2.4 Useful Security Results

This section states a few useful security results. From the above definitions, it is not hard to see that for any shared circuit C we have the following implications:

$$C \,\text{is}\, t\text{-SNI} \;\Rightarrow\; C \,\text{is}\, t\text{-NI} \;\Rightarrow\; C \,\text{is}\, t - \text{probing secure}$$

while the converses are not true. While the ISW-multiplication (and refresh) gadget defined above was originally shown to achieve probing security, it actually achieves the more general notion of strong non-interfering security as formally stated in the following theorem:

Theorem 1 **([2]).** *For any integers d and t such that $t < d$, the d-shared ISW-multiplication gadget $[\otimes]$ and the d-shared ISW-refresh gadget $[\mathtt{R}]$ are both t-SNI.*

The next lemma states a simple implication of the t-SNI notion (which up to our knowledge has never been stated in the literature):

Lemma 1. *Let C be a n-input $(t+1)$-shared t-SNI circuit. Then for every $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$, an evaluation of C taking n uniform and independent $(t+1)$-Boolean sharings $[x_1], \ldots, [x_n]$ as input produces a sharing $[y]$ (of some value $y \in \mathbb{F}_2$ function of $x_1, \ldots, x_n$) which is uniform and mutually independent of $[x_1], \ldots, [x_n]$.*

Proof of Lemma 1 is available in the full version of this paper [4].

3 A Security Reduction

This section provides a reduction for the t-probing security of a standard $(t+1)$-shared circuit C as defined in Sect. 2. Through a sequence of games we obtain a broad simplification of the problem of verifying whether C is probing secure or not. At each step of our reduction, a new game is introduced which is shown to be equivalent to the previous one, implying that for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins the new game if and only if the circuit C is t-probing secure. We get a final game (see Game 3 hereafter) in which only the inputs of the multiplication gadgets can be probed by the adversary and the circuit is *flattened* into an (equivalent) circuit of multiplicative depth one. This allows us to express the probing security property as a linear algebra problem, which can then be solved efficiently as we show in Sect. 4.

In a nutshell, our Game 0 exactly fits the game-based definition of t-probing security given in the previous section. Then, with Game 1, we prove that verifying the t-probing security of a standard shared circuit C is exactly equivalent to verifying the t-probing security of the same circuit C where the attacker $\mathcal{A}$ is restricted to probe inputs of refresh gadgets, pairs of inputs of multiplication gadgets, and inputs and outputs of sharewise additions (i.e., no internal gadgets variables). Game 2 then shows that verifying the t-probing security of a standard shared circuit C with a restricted attacker $\mathcal{A}$ is equivalent to verifying the t-probing security of a functionally equivalent circuit C' of multiplicative depth one where all the outputs of multiplication and refresh gadgets in C are replaced by fresh input sharings of the same values in the rest of the circuit. Finally, with Game 3, we show that we can even restrict the adversary to probe only pairs (x_i, y_j) where x_i (resp. y_j) is the i^{th} share of x (resp. the j^{th} share of y) and such that x and y are operands of the same multiplication in C. These three games are deeply detailed hereafter and proofs of their consecutive equivalence are provided at each step. An overview is displayed on Fig. 5.

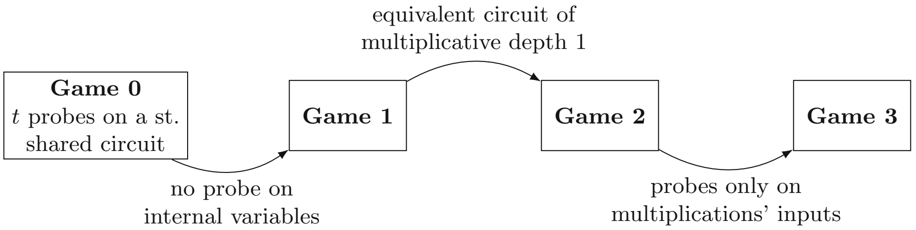


Fig. 5. Overview of the sequence of games.

Game 1. In a nutshell, our first game transition relies on the fact that each probe in a t-SNI gadget can be replaced by 1 or 2 probes on the input sharing(s) of the gadget. In particular, one probe on a refresh gadget is equivalent to revealing one input share, one probe on a multiplication gadget is equivalent to revealing two input shares (one share per input sharings). Formally, in the random experiments $\mathsf{ExpReal}(\mathcal{A}, C)$ and $\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$, the set of probes $\mathcal{P}$ returned by $\mathcal{A}$, noted $\mathcal{P}'$ in the following, has a different form explicitly defined below.

Let us associate an index g to each gadget in the standard shared circuit and denote by $\mathcal{G}$ the set of gadget indices. Let us further denote by $\mathcal{G}_r$, $\mathcal{G}_m$ and $\mathcal{G}_a$ the index sets of refresh gadgets, multiplication gadgets and addition gadgets, such that $\mathcal{G} = \mathcal{G}_r \cup \mathcal{G}_m \cup \mathcal{G}_a$. Then we can denote by $\mathcal{I}_g$ and $\mathcal{J}_g$ the indices of circuit wires which are the shares of the (right and left) input operands of gadget

$g \in \mathcal{G}$ (where $\mathcal{J}_g = \emptyset$ if gadget g is a refresh). Similarly, we denote by $\mathcal{O}_g$ the indices of circuit wires which represent the output of gadget $g \in \mathcal{G}$. From these notations, an admissible set of probes $\mathcal{P}'$ from the adversary in the new game is of the form

$$\mathcal{P}' = \mathcal{P}'_r \cup \mathcal{P}'_m \cup \mathcal{P}'_a$$

where

$$\mathcal{P}'_r \subseteq \bigcup_{g \in \mathcal{G}_r} \mathcal{I}_g$$

$$\mathcal{P}'_m \subseteq \bigcup_{g \in \mathcal{G}_m} \mathcal{I}_g \times \mathcal{J}_g$$

$$\mathcal{P}'_a \subseteq \bigcup_{g \in \mathcal{G}_a} \mathcal{I}_g \bigcup_{g \in \mathcal{G}_a} \mathcal{J}_g \bigcup_{g \in \mathcal{G}_a} \mathcal{O}_g$$

and $|\mathcal{P}'| = t$. That is, each of the t elements of $\mathcal{P}'$ either is a pair of index in $\mathcal{I}_g \times \mathcal{J}_g$ for a multiplication gadget g, or a single index in $\mathcal{I}_g$ for a refresh gadget g, or a single index in $\mathcal{I}_g \cup \mathcal{J}_g \cup \mathcal{O}_g$ for an addition gadget. Note that in the latter case, the index can correspond to any wire in the addition gadget (which is simply composed of $t+1$ addition gates).

Let t_m be the number of probes on multiplication gadgets, *i.e.* $t_m = |\mathcal{P}'_m|$, and t_{ar} the number of probes on refresh or addition gadgets, *i.e.* $t_{ar} = |\mathcal{P}'_a \cup \mathcal{P}'_r|$, so that $t_m + t_{ar} = t$. The evaluation $C([x_1], \ldots, [x_n])_{\mathcal{P}'}$ then returns a q-tuple for $q = 2t_m + t_{ar}$, which is composed of the values taken by the wires of index $i \in \mathcal{P}'_a \cup \mathcal{P}'_r$, and the values taken by the wires of index i and j with $(i, j) \in \mathcal{P}'_m$. The new experiments $\mathsf{ExpReal}_1(\mathcal{A}, C)$ and $\mathsf{ExpSim}_1(\mathcal{A}, \mathcal{S}, C)$, carefully written in Fig. 6, each output a q-tuple and, as before, the simulator wins Game 1 if and only if the associated distributions are identical.

```
ExpReal_1(A, C):
  1. $(\mathcal{P}', x_1, \ldots, x_n) \leftarrow \mathcal{A}()$
  2. $[x_1] \leftarrow \mathsf{Enc}(x_1), \ldots, [x_n] \leftarrow \mathsf{Enc}(x_n)$
  3. $(v_1, \ldots, v_q) \leftarrow C([x_1], \ldots, [x_n])_{\mathcal{P}'}$
  4. Return $(v_1, \ldots, v_q)$
```

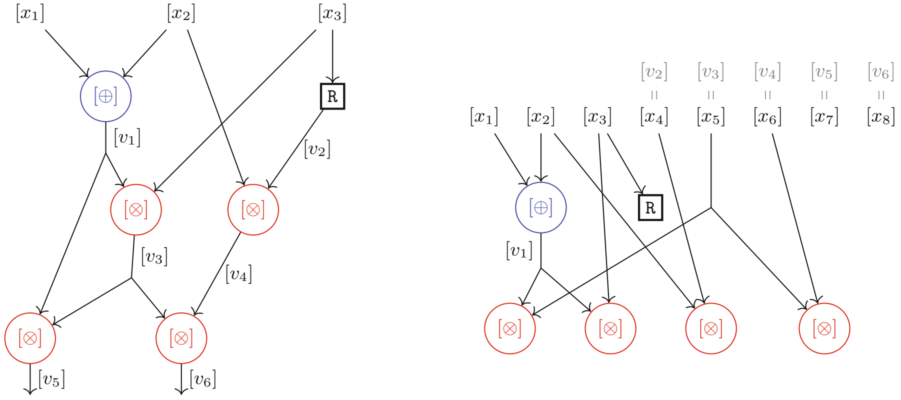
```
ExpSim_1(A, S, C):
  1. $(\mathcal{P}', x_1, \ldots, x_n) \leftarrow \mathcal{A}()$
  2. $(v_1, \ldots, v_q) \leftarrow \mathcal{S}(\mathcal{P}')$
  3. Return $(v_1, \ldots, v_q)$
```

Fig. 6. Game 1.

Proposition 4. *A standard shared circuit C is t-probing secure if and only if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins Game 1 defined above, i.e. the random experiments* $\mathsf{ExpReal}_1(\mathcal{A}, C)$ *and* $\mathsf{ExpSim}_1(\mathcal{A}, \mathcal{S}, C)$ *output identical distributions.*

Proof. Basically, the proof is based on the fact that with the SNI property on the gadgets in our circuit, each probe in a t-SNI gadget can be replaced by 1 or 2 probes on the input sharing(s) of the gadget. The complete proof can be found in the full version of this paper [4].

Game 2. Our second game transition consists in replacing the circuit C by a functionally equivalent circuit C' of multiplicative depth one and with an extended input. In a nutshell, each output of a multiplication or a refresh gadget in C is replaced by a fresh new input sharing of the same value in the rest of the circuit. The new circuit hence takes N input sharings $[x_1], \ldots, [x_n], [x_{n+1}], \ldots, [x_N]$, with $N = n + |\mathcal{G}_m| + |\mathcal{G}_r|$. The two circuits are functionally equivalent in the sense that for every input $(x_1, \ldots, x_n)$ there exists an extension $(x_{n+1}, \ldots, x_N)$ such that $C([x_1], \ldots, [x_n])$ and $C'([x_1], \ldots, [x_N])$ have output sharings encoding the same values. This transformation is further referred to as $\mathsf{Flatten}$ in the following, and is illustrated on Fig. 7.



Circuit C

Circuit C'

Fig. 7. Illustration of the $\mathsf{Flatten}$ transformation.

The resulting Game 2 is illustrated on Fig. 8. Although the additional inputs $x_{n+1}, \ldots, x_N$ are deterministic functions of the original inputs $x_1, \ldots, x_n$, we allow the adversary to select the full extended input $x_1, \ldots, x_N$ for the sake of simplicity. This slight adversarial power overhead does not affect the equivalence between the games.

```
$\mathsf{ExpReal}_2(\mathcal{A}, C)$:
1. $C' \leftarrow \mathsf{Flatten}(C)$
2. $(\mathcal{P}', x_1, \ldots, x_N) \leftarrow \mathcal{A}()$
3. $[x_1] \leftarrow \mathsf{Enc}(x_1), \ldots, [x_N] \leftarrow \mathsf{Enc}(x_N)$
4. $(v_1, \ldots, v_q) \leftarrow C'([x_1], \ldots, [x_N])_{\mathcal{P}'}$
5. Return $(v_1, \ldots, v_q)$
```

```
$\mathsf{ExpSim}_2(\mathcal{A}, \mathcal{S}, C)$:
1. $C' \leftarrow \mathsf{Flatten}(C)$
2. $(\mathcal{P}', x_1, \ldots, x_N) \leftarrow \mathcal{A}()$
3. $(v_1, \ldots, v_q) \leftarrow \mathcal{S}(\mathcal{P}')$
4. Return $(v_1, \ldots, v_q)$
```

Fig. 8. Game 2.

Proposition 5. *A standard shared circuit C is t-probing secure if and only if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins Game 2 defined above, i.e. the random experiments* $\mathsf{ExpReal}_2(\mathcal{A}, C)$ *and* $\mathsf{ExpSim}_2(\mathcal{A}, \mathcal{S}, C)$ *output identical distributions.*

Proof. Basically, the proof is based on the fact that the output encodings of a ISW multiplication are completely independent of its inputs encodings. The complete proof can be found in the full version of this paper [4].

Corollary 1. *A standard shared circuit C is t-probing secure if and only if the standard shared circuit* $\mathsf{Flatten}(C)$ *is t-probing secure.*

Translation to Linear Algebra. At this point, the problem of deciding the t-probing security of a Boolean standard shared circuit C has been equivalently reduced to the problem of deciding the t-probing security of a circuit $C' = \mathsf{Flatten}(C)$ when the attacker is restricted to probes on multiplication and refresh gadgets' inputs, and intermediate variables of sharewise additions. In order to further reduce it, we translate the current problem into a linear algebra problem. In the following, we denote by $x_{i,j}$ the jth share of the ith input sharing $[x_i]$ so that

$$[x_i] = (x_{i,0}, x_{i,1}, \dots, x_{i,t}),$$

for every $i \in [\![1, N]\!]$. Moreover, we denote by $\overrightarrow{x_j} \in \mathbb{F}_2^N$ the vector composed of the jth share of each input sharing:

$$\overrightarrow{x_j} = (x_{0,j}, x_{1,j}, \dots, x_{N,j}).$$

As a result of the $\mathsf{Flatten}$ transformation, each probed variable in the q-tuple $(v_1, \dots, v_q) = C([x_1], \dots, [x_N])_{\mathcal{P}'}$ is a linear combination of the input sharings $[x_1], \dots, [x_N]$. Moreover, since the addition gadgets are sharewise, for every $k \in [\![1, q]\!]$, there is a single share index j such that the probed variable v_k only depends of the jth shares of the input sharings, giving:

$$v_k = \overrightarrow{a_k} \cdot \overrightarrow{x_j}, \tag{1}$$

for some constant coefficient vector $\overrightarrow{a_k} \in \mathbb{F}_2^N$. Without loss of generality, we assume that the tuple of probed variables is ordered w.r.t. the share index j corresponding to each v_k (*i.e.* starting from $j = 0$ up to $j = t$). Specifically, the q-tuple $(v_1, \dots, v_q)$ is the concatenation of $t + 1$ vectors

$$\overrightarrow{v_0} = M_0 \cdot \overrightarrow{x_0}\ , \quad \overrightarrow{v_1} = M_1 \cdot \overrightarrow{x_1}\ , \quad \dots \quad \overrightarrow{v_t} = M_t \cdot \overrightarrow{x_t}, \tag{2}$$

where the matrix M_j is composed of the row coefficient vectors $\overrightarrow{a_k}$ for the probed variable indices k corresponding to the share index j.

Lemma 2. *For any $(x_1, \dots, x_N) \in \mathbb{F}_2^N$, the q-tuple of probed variables $(v_1, \dots, v_q) = C([x_1], \dots, [x_N])_{\mathcal{P}'}$ can be perfectly simulated if and only if the M_j matrices satisfy*

$$\mathrm{Im}(M_0^T) \cap \mathrm{Im}(M_1^T) \cap \dots \cap \mathrm{Im}(M_t^T) = \emptyset.$$

Moreover, if the M_j matrices are full-rank (which can be assumed without loss of generality), then the above equation implies that $(v_1, \ldots, v_q)$ is uniformly distributed.

Proof. Without loss of generality we can assume that the M_j matrices are full-rank since otherwise the probed variables $v_1, \ldots, v_q$ would be mutually linearly dependent and simulating them would be equivalent to simulating any subset $(v_k)_{k \in \mathcal{K} \subseteq [\![1,q]\!]}$ defining a free basis of $(v_1, \ldots, v_q)$, and which would then induce full-rank matrices M_j.

Throughout this proof, we denote $\overrightarrow{x} = (x_1, \ldots, x_N)$. We first show that a non-null intersection implies a non-uniform distribution of $(v_1, \ldots, v_q)$ which is statistically dependent on $\overrightarrow{x}$. Indeed, a non-null intersection implies that there exist a non-null vector $\overrightarrow{w} \in \mathbb{F}_2^N$ satisfying

$$\overrightarrow{w} = \overrightarrow{u_0} \cdot M_0 = \overrightarrow{u_1} \cdot M_1 = \cdots = \overrightarrow{u_t} \cdot M_t. \tag{3}$$

for some (constant) vectors $\overrightarrow{u_0}$, ..., $\overrightarrow{u_t}$. It follows that

$$\sum_{j=0}^{t} \overrightarrow{u_j} \cdot \overrightarrow{v_j} = \sum_{j=0}^{t} \overrightarrow{w} \cdot \overrightarrow{x_j} = \overrightarrow{w} \cdot \overrightarrow{x},$$

which implies that the distribution of the q-tuple $(v_1, \ldots, v_q) = (\overrightarrow{v_0} \parallel \cdots \parallel \overrightarrow{v_t})$ is non-uniform and dependent on $\overrightarrow{x}$.

We now show that a null intersection implies a uniform distribution (which can then be easily simulated). The uniformity and mutual independence between the sharings $[x_1]$, ..., $[x_N]$ implies that we can see $\overrightarrow{x_1}$, ..., $\overrightarrow{x_t}$ as t uniform and independent vectors on $\mathbb{F}_2^N$, and $\overrightarrow{x_0}$ as

$$\overrightarrow{x_0} = \overrightarrow{x} + \overrightarrow{x_1} + \cdots + \overrightarrow{x_t}.$$

The joint distribution of $\overrightarrow{v_1}$, ..., $\overrightarrow{v_t}$ is hence clearly uniform. Then each coordinate of $\overrightarrow{v_0}$ is the result of the inner product $\overrightarrow{r} \cdot \overrightarrow{x_0}$ where $\overrightarrow{r}$ is a row of M_0. By assumption, there exists at least one matrix M_j such that $\overrightarrow{r} \notin \mathrm{Im}(M_j^T)$. It results that $\overrightarrow{r} \cdot \overrightarrow{x_j}$ is a uniform random variable independent of $\overrightarrow{v_1}$, ..., $\overrightarrow{v_t}$ and the other coordinates of $\overrightarrow{v_0}$ (since M_0 is full-rank). Since the latter holds for all the coordinates of $\overrightarrow{x_0}$ we get overall uniformity of $(\overrightarrow{v_0} \parallel \cdots \parallel \overrightarrow{v_t})$ which concludes the proof. □

Lemma 2 allows us to reduce the t-probing security of a standard shared circuit to a linear algebra problem. If an adversary exists that can choose the set of probes $\mathcal{P}'$ such that the transposes of induced matrices M_1, ..., M_t have intersecting images, then the distribution of $(v_1, \ldots, v_q)$ depends on $(x_1, \ldots, x_N)$ and a perfect simulation is impossible (which means that the circuit is not probing secure). Otherwise, the tuple $(v_1, \ldots, v_q)$ can always be simulated by a uniform distribution and the circuit is probing secure. This statement is the basis of our verification method depicted in the next section. But before introducing our verification method, we can still simplify the probing security game as shown hereafter by using Lemma 2.

Game 3. In this last game, the adversary is restricted to probe the multiplication gadgets only. Formally, $\mathcal{A}$ returns a set of probes $\mathcal{P}' = \mathcal{P}'_r \cup \mathcal{P}'_m \cup \mathcal{P}'_a$ such that $\mathcal{P}'_r = \emptyset$ and $\mathcal{P}'_a = \emptyset$. Such a set, denoted $\mathcal{P}''$ is hence composed of t pairs of inputs from $\bigcup_{g \in \mathcal{G}_m} \mathcal{I}_g \times \mathcal{J}_g$. The evaluation $C([x_1], \ldots, [x_n])_{\mathcal{P}''}$ then returns a q-tuple for $q = 2t$. The new experiments $\mathsf{ExpReal}_3(\mathcal{A}, C)$ and $\mathsf{ExpSim}_3(\mathcal{A}, \mathcal{S}, C)$, displayed in Fig. 6, each output a q-tuple and, as before, the simulator wins Game 3 if and only if the associated distributions are identical.

```
ExpReal_3(A, C):
$C' \leftarrow \mathsf{Flatten}(C)$
$(\mathcal{P}'', x_1, \ldots, x_N) \leftarrow \mathcal{A}()$
$[x_1] \leftarrow \mathsf{Enc}(x_1), \ldots, [x_N] \leftarrow \mathsf{Enc}(x_N)$
$(v_1, \ldots, v_q) \leftarrow C'([x_1], \ldots, [x_N])_{\mathcal{P}''}$
Return $(v_1, \ldots, v_q)$
```

```
ExpSim_3(A, S, C):
$C' \leftarrow \mathsf{Flatten}(C)$
$(\mathcal{P}'', x_1, \ldots, x_N) \leftarrow \mathcal{A}()$
$(v_1, \ldots, v_q) \leftarrow \mathcal{S}(\mathcal{P}'')$
Return $(v_1, \ldots, v_q)$
```

Fig. 9. Game 3.

Proposition 6. *A standard shared circuit C is t-probing secure if and only if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins Game 3 defined above, i.e. the random experiments* $\mathsf{ExpReal}_3(\mathcal{A}, C)$ *and* $\mathsf{ExpSim}_3(\mathcal{A}, \mathcal{S}, C)$ *output identical distributions.*

Proof. Basically, the proof is based on the fact that probing a cross products $a_i \cdot b_j$ allows you to gain informations on the two shares a_i and b_j. The complete proof can be found in the full version of this paper [4].

4 Probing-Security Verification for Standard Shared Circuits

In this section, we describe a formal verification method that checks for any $t \in \mathbb{N}$ whether a standard $(t+1)$-shared circuit C achieves t-probing security for every $t \in \mathbb{N}$. Specifically, our tool `tightPROVE` either provides a formal proof that C is t-probing secure (where C is a standard shared circuit with sharing order $t+1$), or it exhibits a *probing attack* against C for the given t, namely it finds a set of probes $\mathcal{P}$ (indices of wires) in the $(t+1)$-shared instance of C, such that $|\mathcal{P}| = t$, for which the evaluation $C([x_1], \ldots, [x_n])_{\mathcal{P}}$ cannot be simulated without some knowledge on the plain input $(x_1, \ldots, x_n)$.

4.1 Linear Algebra Formulation

As demonstrated in the previous section, the t-probing security game for a standard $(t+1)$-shared circuit C can be reduced to a game where an adversary selects a set of probes $\mathcal{P}''$ solely pointing to input shares of the multiplication gadgets of a *flattened* circuit C'. In the following, we will denote by m the number of multiplication gadgets in C (or equivalently in C') and by $g \in [\![1, m]\!]$ the index of a multiplication gadget of C. We will further denote by $[a_g]$ and $[b_g]$ the input sharings of the g-th multiplication gadget so that we have

$$[a_g] = (\overrightarrow{a_g} \cdot \overrightarrow{x_0}, \ldots, \overrightarrow{a_g} \cdot \overrightarrow{x_t}) \text{ and } [b_g] = (\overrightarrow{b_g} \cdot \overrightarrow{x_0}, \ldots, \overrightarrow{b_g} \cdot \overrightarrow{x_t}), \tag{4}$$

for some constant coefficient vectors $\overrightarrow{a_g}, \overrightarrow{b_g} \in \mathbb{F}_2^N$, recalling that $\overrightarrow{x_j}$ denotes the vector with the jth share of each input sharing $[x_1]$, ..., $[x_N]$. In the following, the vectors $\{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$ are called the *operand vectors*.

In Game 3, the adversary chooses t pairs of probes such that each pair points to one share of $[a_g]$ and one share of $[b_g]$ for a multiplication gadget g. Without loss of generality, the set of pairs output by the adversary can be relabeled as a set of triplet $\mathcal{P} = \{(g, j_1, j_2)\}$ where $g \in [\![1, m]\!]$ is the index of a multiplication gadget, j_1 and j_2 are share indices. For any triplet $(g, j_1, j_2) \in \mathcal{P}$ the two input shares $\overrightarrow{a_g} \cdot \overrightarrow{x_{j_1}}$ and $\overrightarrow{b_g} \cdot \overrightarrow{x_{j_2}}$ are added to the $(2t)$-tuple of probed variables to be simulated. This set of triplets exactly defines a sequence of $t+1$ matrices M_0, ..., M_t, defined iteratively by adding $\overrightarrow{a_g}$ to the rows of M_{j_1} and $\overrightarrow{b_g}$ to the rows of M_{j_2} for each $(g, j_1, j_2) \in \mathcal{P}$. Equivalently, the matrix M_j is defined as

$$M_j = \mathsf{rows}(\{\overrightarrow{a_g} \;;\; (g, j, *) \in \mathcal{P}\} \cup \{\overrightarrow{b_g} \;;\; (g, *, j) \in \mathcal{P}\}), \tag{5}$$

for every $j \in [\![0, t]\!]$ where rows maps a set of vectors to the matrix with rows from this set.

Lemma 2 then implies that a probing attack on C consists of a set of probes $\mathcal{P} = \{(g, j_1, j_2)\}$ such that the transposes of the induced M_j have intersecting images. Moreover, since the total number of rows in these matrices is $2t$, at least one of them has a single row $\overrightarrow{w}$. In particular, the image intersection can only be the span of this vector (which must match the row of all single-row matrices) and this vector belongs to the set of operand vectors $\{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$. In other words, there exists a probing attack on C if and only if a choice of probes $\mathcal{P} = \{(g, j_1, j_2)\}$ implies

$$\mathrm{Im}(M_0^T) \cap \mathrm{Im}(M_1^T) \cap \cdots \cap \mathrm{Im}(M_t^T) = \langle \overrightarrow{w} \rangle. \tag{6}$$

for some vector $\overrightarrow{w} \in \{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$. In that case we further say that there is a probing attack on the operand vector $\overrightarrow{w}$.

In the remainder of this section, we describe an efficient method that given a set of vector operands $\{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$ (directly defined from a target circuit C) determines whether there exists a parameter t and a set $\mathcal{P} = \{(g, j_1, j_2)\}$ (of cardinality t) for which (6) can be satisfied. We prove that (1) if such sets $\mathcal{P}$ exist, our method returns one of these sets, (2) if no set is returned by our method then the underlying circuit is t-probing secure for any sharing order $(t+1)$.

4.2 Method Description

The proposed method loops over all the vector operands $\overrightarrow{w} \in \{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$ and checks whether there exists a probing attack on $\overrightarrow{w}$ (*i.e.* whether a set $\mathcal{P}$ can be constructed that satisfies (6)).

For each $\overrightarrow{w} \in \{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$ the verification method is iterative. It starts from a set $\mathcal{G}_1 \subseteq [\![1, m]\!]$ defined as

$$\mathcal{G}_1 = \{g \ ; \ \overrightarrow{a_g} = \overrightarrow{w}\} \cup \{g \ ; \ \overrightarrow{b_g} = \overrightarrow{w}\}. \tag{7}$$

Namely $\mathcal{G}_1$ contains the indices of all the multiplication gadgets that have $\overrightarrow{w}$ as vector operand. Then the set of *free vector operands* $\mathcal{O}_1$ is defined as

$$\mathcal{O}_1 = \{\overrightarrow{b_g} \ ; \ \overrightarrow{a_g} = \overrightarrow{w}\} \cup \{\overrightarrow{a_g} \ ; \ \overrightarrow{b_g} = \overrightarrow{w}\}. \tag{8}$$

The terminology of *free* vector operand comes from the following intuition: if a probing adversary spends one probe on gadget $g \in \mathcal{G}_1$ such that $\overrightarrow{a_g} = \overrightarrow{w}$ to add $\overrightarrow{w}$ to a matrix M_j (or equivalently to get the share $\overrightarrow{w} \cdot \overrightarrow{x_j}$), then she can also add $\overrightarrow{b_g}$ to another matrix $M_{j'}$ (or equivalently get the share $\overrightarrow{b_g} \cdot \overrightarrow{x_{j'}}$) for *free*. The adversary can then combine several free vector operands to make $\overrightarrow{w} \in \mathrm{Im}(M_{j'}^T)$ occur without directly adding $\overrightarrow{w}$ to $M_{j'}$ (or equivalently without directly probing $\overrightarrow{w} \cdot \overrightarrow{x_{j'}}$). This is possible if and only if $\overrightarrow{w} \in \langle \mathcal{O}_1 \rangle$.

The free vector operands can also be combined with the operands of further multiplications to generate a probing attack on $\overrightarrow{w}$. To capture such higher-degree combinations, we define the sequences of sets $(\mathcal{G}_i)_i$ and $(\mathcal{O}_i)_i$ as follows:

$$\mathcal{G}_{i+1} = \{g \ ; \ \overrightarrow{a_g} \in \overrightarrow{w} + \langle \mathcal{O}_i \rangle\} \cup \{g \ ; \ \overrightarrow{b_g} \in \overrightarrow{w} + \langle \mathcal{O}_i \rangle\}, \tag{9}$$

and

$$\mathcal{O}_{i+1} = \{\overrightarrow{b_g} \ ; \ \overrightarrow{a_g} \in \overrightarrow{w} + \langle \mathcal{O}_i \rangle\} \cup \{\overrightarrow{a_g} \ ; \ \overrightarrow{b_g} \in \overrightarrow{w} + \langle \mathcal{O}_i \rangle\}. \tag{10}$$

for every $i \geq 1$. The rough idea of this iterative construction is the following: if at step $i+1$ a probing adversary spends one probe on gadget $g \in \mathcal{G}_{i+1}$ such that $\overrightarrow{a_g} \in \overrightarrow{w} + \langle \mathcal{O}_i \rangle$, then she can add $\overrightarrow{a_g}$ together with some free vector operands of previous steps to M_j in order to get $\overrightarrow{w} \in \mathrm{Im}(M_j^T)$. Then she can also add $\overrightarrow{b_g}$ to another matrix $M_{j'}$, making $\overrightarrow{b_g}$ a new free vector operand of step $i+1$.

Based on these definitions, our method iterates the construction of the sets $\mathcal{G}_i$ and $\mathcal{O}_i$. At setp i, two possible stop conditions are tested:

1. if $\mathcal{G}_i = \mathcal{G}_{i-1}$, then there is no probing attack on $\overrightarrow{w}$, the method stops the iteration on $\overrightarrow{w}$ and continues with the next element in the set of vector operands;
2. if $\overrightarrow{w} \in \langle \mathcal{O}_i \rangle$, then there is a probing attack on $\overrightarrow{w}$, the method stops and returns `True` (with $\overrightarrow{w}$ and the sequence of sets $(\mathcal{G}_i, \mathcal{O}_i)_i$ as proof);

The method returns `True` if there exists a concrete probing attack on a vector $\overrightarrow{w} \in \{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$ for a certain sharing order $t+1$. Otherwise, it will eventually stop with vector operand $\overrightarrow{w}$ since the number of multiplications is finite and since $\mathcal{G}_i \subseteq \mathcal{G}_{i+1}$ for every $i \geq 1$. When all the possible vector operands have been tested without finding a probing attack, the method returns `False`. Algorithm 1 hereafter gives a pseudocode of our method where NextSets denotes the procedure that computes $(\mathcal{G}_{i+1}, \mathcal{O}_{i+1})$ from $(\mathcal{G}_i, \mathcal{O}_i)$ and is implemented in Sect. 6.

Algorithm 1. Search probing attack

Input: A set of vector operands $\{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$

Output: `True` if there is probing attack on some $\overrightarrow{w} \in \{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$ and `False` otherwise

```
for all $\overrightarrow{w} \in \{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g$ do
    $(\mathcal{G}_1, \mathcal{O}_1) \leftarrow \mathsf{NextSets}(\emptyset, \emptyset, \{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g, \overrightarrow{w})$
    if $\overrightarrow{w} \in \langle \mathcal{O}_1 \rangle$ then return True
    for $i = 1$ to $m$ do
        $(\mathcal{G}_{i+1}, \mathcal{O}_{i+1}) \leftarrow \mathsf{NextSets}(\mathcal{G}_i, \mathcal{O}_i, \{\overrightarrow{a_g}, \overrightarrow{b_g}\}_g, \overrightarrow{w})$
        if $\mathcal{G}_{i+1} = \mathcal{G}_i$ then break
        if $\overrightarrow{w} \in \langle \mathcal{O}_i \rangle$ then return True
    end for
end for
return False
```

In the rest of the section we first give some toy examples to illustrate our methods and then provides a proof of its correctness.

4.3 Toy Examples

Two examples are provided hereafter to illustrate our iterative method in the absence then in the presence of a probing attack.

In the very simple example of Fig. 1, two variables are manipulated in multiplications in the circuit C: $\overrightarrow{w}_1 = \overrightarrow{x_1}$ and $\overrightarrow{w}_2 = \overrightarrow{x_1} + \overrightarrow{x_2}$. The set of multiplications $\mathcal{G}$ is of cardinality one since it only contains one multiplication $(\overrightarrow{w}_1, \overrightarrow{w}_2)$. Following the number of variables, the method proceeds at most in two steps:

1. As depicted in Algorithm 1, the method first determines whether there exists a probing attack on $\overrightarrow{w}_1$. In this purpose, a first set $\mathcal{G}_1$ is built, such that $\mathcal{G}_1 = (\overrightarrow{w}_1, \overrightarrow{w}_2)$ and $\mathcal{O}_1 = \overrightarrow{w}_2$. Since $\mathcal{G}_1 \neq \emptyset$ and $\overrightarrow{w}_1 \neq \overrightarrow{w}_2$, then a second set must be built. However, there is no multiplication left, that is $\mathcal{G}_2 = \mathcal{G}_1$ and so there is no attack on $\overrightarrow{w}_1$.
2. The method then focuses on $\overrightarrow{w}_2$. In this purpose, a dedicated set $\mathcal{G}_1$ is built, such that $\mathcal{G}_1 = (\overrightarrow{w}_2, \overrightarrow{w}_1)$ and $\mathcal{O}_1 = \overrightarrow{w}_1$. Since $\mathcal{G}_1 \neq \emptyset$ and $\overrightarrow{w}_2 \neq \overrightarrow{w}_1$, then a second set must be built. However, there is no multiplication left, that is $\mathcal{G}_2 = \mathcal{G}_1$ and so there is no attack on $\overrightarrow{w}_2$ either. Since there is no input variable left, the method returns `False`, which means that there is no possible probing attack on this circuit.

Figure 10 provides a second Boolean circuit. It manipulates five variables $\overrightarrow{w}_i$ as operands of multiplication gadgets: $\overrightarrow{w}_1 = \overrightarrow{x_1}$, $\overrightarrow{w}_2 = \overrightarrow{x_2}$, $\overrightarrow{w}_3 = \overrightarrow{x_3}$, $\overrightarrow{w}_4 = \overrightarrow{x_1} + \overrightarrow{x_2}$, and $\overrightarrow{w}_5 = \overrightarrow{x_2} + \overrightarrow{x_3}$. The set of multiplications $\mathcal{G}$ is of cardinality three with $(\overrightarrow{w}_1, \overrightarrow{w}_2)$, $(\overrightarrow{w}_4, \overrightarrow{w}_5)$, and $(\overrightarrow{w}_3, \overrightarrow{w}_4)$. Following the number of variables, the method proceeds at most in five steps:

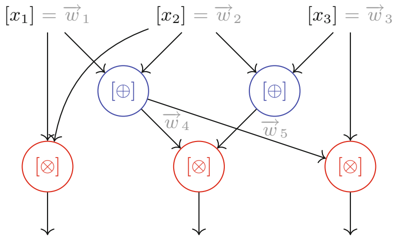


Fig. 10. Graph representation of a second Boolean circuit.

1. The method first determines whether there exists a probing attack on $\overrightarrow{w}_1$. In this purpose, a first set $\mathcal{G}_1$ is built, such that $\mathcal{G}_1 = (\overrightarrow{w}_1, \overrightarrow{w}_2)$ and $\mathcal{O}_1 = \overrightarrow{w}_2$. Since $\mathcal{G}_1 \neq \emptyset$ and $\overrightarrow{w}_1 \neq \overrightarrow{w}_2$, then a second set must be built. $\mathcal{G}_2 = \mathcal{G}_1 \cup \{(\overrightarrow{w}_4, \overrightarrow{w}_5), (\overrightarrow{w}_4, \overrightarrow{w}_3)\}$ since $\overrightarrow{w}_4 = \overrightarrow{w}_1 + \overrightarrow{w}_2$. However, $\overrightarrow{w}_1 \notin \mathcal{O}_2 (=< \overrightarrow{w}_2, \overrightarrow{w}_3, \overrightarrow{w}_5 >)$, so a third set must be built. Since there is no multiplication left, that is $\mathcal{G}_3 = \mathcal{G}_2$, there is no attack on $\overrightarrow{w}_1$.
2. The method then focuses on $\overrightarrow{w}_2$. In this purpose, a dedicated set $\mathcal{G}_1$ is built, such that $\mathcal{G}_1 = (\overrightarrow{w}_2, \overrightarrow{w}_1)$ and $\mathcal{O}_1 = \overrightarrow{w}_1$. Since $\mathcal{G}_1 \neq \emptyset$ and $\overrightarrow{w}_2 \neq \overrightarrow{w}_1$, then a second set must be built. $\mathcal{G}_2 = \mathcal{G}_1 \cup \{(\overrightarrow{w}_4, \overrightarrow{w}_5), (\overrightarrow{w}_4, \overrightarrow{w}_3)\}$ since $\overrightarrow{w}_4 = \overrightarrow{w}_2 + \overrightarrow{w}_1$. And in that case, $\overrightarrow{w}_2 \in \mathcal{O}_2 (=< \overrightarrow{w}_1, \overrightarrow{w}_3, \overrightarrow{w}_5 >)$ since $\overrightarrow{w}_2 = \overrightarrow{w}_3 + \overrightarrow{w}_5$. Thus the method returns `True` and there exists an attack on $\overrightarrow{w}_2 = \overrightarrow{x_2}$ for some masking order t.

4.4 Proof of Correctness

This section provides a proof of correctness of the method. This proof is organized in two propositions which are based on some invariants in Algorithm 1. The first proposition shows that if the method returns `True` for some operand vector $\overrightarrow{w}$ and corresponding sets $(\mathcal{G}_i, \mathcal{O}_i)$ then there exists a probing attack on $\overrightarrow{w}$ (*i.e.* a set $\mathcal{P}$ can be constructed that satisfies (6)). The second proposition shows that if the method returns `False` then there exists no probing attack for any $\overrightarrow{w}$, namely the underlying circuit is t-probing secure as soon as masked variables are masked with $t + 1$ shares.

Proposition 7. *For every $i \in \mathbb{N}$, if $\overrightarrow{w} \in \langle \mathcal{O}_i \rangle$ then there exists $t \in \mathbb{N}$ and $\mathcal{P} = \{(g, j_1, j_2)\}$ with $|\mathcal{P}| = t$ implying $\bigcap_{j=0}^{t} \mathrm{Im}(M_j^T) = \overrightarrow{w}$.*

Proposition 8. *Let $i > 1$ such that $\mathcal{G}_1 \subset \cdots \subset \mathcal{G}_{i-1} = \mathcal{G}_i$ and $\overrightarrow{w} \notin \langle \mathcal{O}_i \rangle$. Then for any $t \in \mathbb{N}$ and $\mathcal{P} = \{(g, j_1, j_2)\}$ with $|\mathcal{P}| = t$ we have $\overrightarrow{w} \notin \bigcap_{j=0}^{t} \mathrm{Im}(M_j^T)$.*

Proofs of Propositions 7 and 8 are available in the full version of this paper [4].

4.5 Towards Efficient Construction of Tight t-Private Circuits

Our formal verification method exactly reveals all the t-probing attacks on standard shared circuits. A sound countermeasure to counteract these attacks is the use of refresh gadgets. We discuss here how to transform a flawed standard shared circuit into a t-private circuit with exactly the minimum number of refresh gadgets.

In a first attempt, we easily show that refreshing the left operands of each multiplication in C is enough to provide t-probing security.

Proposition 9. *A standard shared circuit C augmented with t-SNI refresh gadgets operating on the left operand of each multiplication gadget is t-probing secure.*

In a second attempt, we need to slightly modify Algorithm 1 so that it conducts an analysis on all the possible operands in order to return a complete list of the flawed ones. So far, it stops at the first flaw. With such a list for a standard shared circuit, we can show that refreshing only the flawed operands is enough to provide t-probing security.

Proposition 10. *A standard shared circuit C augmented with t-SNI refresh gadgets operating on each flawed operand, as revealed by our method, of its multiplication gadgets is t-probing secure.*

Proofs of these propositions are available in the full version of this paper [4].

Propositions 9 and 10 provide an upper bound of the required number of refresh gadgets in a standard shared circuit to achieve probing security at any order t. If we denote by m the number of multiplications in a standard shared circuit C and by o the number of flawed operands returned by our method, then C is to be augmented of at most $r = \min(m, o)$ refresh gadgets to achieve probing security at any order t. Given this upper bound, an iterative number of refresh gadgets from 1 to r can be inserted at each location in C in order to exhibit a tight private circuit with a minimum number of refresh gadgets.

5 Further Steps

Now that we are able to exactly determine the t-probing security of standard shared circuits, a natural follow-up consists in studying the t-probing security of their composition. In a first part, we establish several compositional properties, and then we show how they apply to the widely deployed SPN-based block ciphers. We eventually discuss the extension of our results to generic shared circuits.

5.1 Generic Composition

This section is dedicated to the statement of new compositional properties on tight private circuits. In a first attempt, we show that the composition of a t-private circuit whose outputs coincide with the outputs of t-SNI gadgets with another t-private circuit is still a t-private circuit.

Proposition 11. *Let us consider a standard shared circuit C composed of two sequential circuits:*

- *a t-probing secure circuit C_1 whose outputs are all outputs of t-SNI gadgets,*
- *a t-probing secure circuit C_2 whose inputs are C_1's outputs.*

Then, $C = C_2 \circ C_1$ is t-probing secure.

Proof. As the outputs of the first circuit C_1 are the outputs t-SNI gadgets, we get from Lemma 1 that the input encodings of C_1 and the input encodings of C_2 are independent and uniformly distributed. Then, the proof is straightforward from Proposition 5. Basically, the analysis of C's t-probing security can be equivalently reduced to the analysis of the t-probing security of $C' = \mathsf{Flatten}(C)$ in which each output of a t-SNI gadget is replaced by a fresh new input sharing of the corresponding value in the rest of the circuit, *i.e.* C_2. As a consequence, C is t-probing secure if and only if both C_1 and C_2 are t-probing secure, which is correct by assumption. □

In a second attempt, we establish the secure composition of a standard shared circuit that implements a (shared) linear surjective transformation through several sharewise addition gadgets, that we refer to as a t-linear surjective circuit, and a standard t-probing circuit.

Proposition 12. *Let us consider a standard shared circuit C composed of two sequential circuits:*

- *a t-linear surjective circuit C_1, exclusively composed of sharewise additions,*
- *a t-probing secure circuit C_2 whose inputs are C_1's outputs.*

Then, $C = C_2 \circ C_1$ is t-probing secure.

Proof. We consider a standard shared circuit C with input $\overrightarrow{x} = (x_1, \ldots, x_n)$ composed of a t-linear surjective circuit C_1 as input to a t-probing secure circuit C_2. We denote by $\overrightarrow{y} = (y_1, \ldots, y_{n'})$ the set of C_1's outputs, or equivalently the set of C_2's inputs. From Proposition 6, the t-probing security of C can be reduced to the t-probing security of circuit $C' = \mathsf{Flatten}(C)$ for probes restricted to the multiplications' operands. In our context, C_1 is exclusively composed of sharewise additions, so the probes are restricted to C_2. From Lemma 2, any set of probed variables on C_2's multiplications operands $(v_1, \ldots, v_q)$ can be written as the concatenation of the $t+1$ vectors

$$\overrightarrow{v_0} = M_0 \cdot \overrightarrow{y_0}\ , \quad \overrightarrow{v_1} = M_1 \cdot \overrightarrow{y_1}\ , \quad \ldots \quad \overrightarrow{v_t} = M_t \cdot \overrightarrow{y_t},$$

where

$$\mathrm{Im}(M_0^T) \cap \mathrm{Im}(M_1^T) \cap \cdots \cap \mathrm{Im}(M_t^T) = \emptyset. \tag{11}$$

To achieve global t-probing security for C, we need to achieve a null intersection for matrices that apply on C's inputs instead of C_2's inputs. As C_1 implements a linear surjective transformation f, there exists a matrix M_f of rank n' such that

$$\forall\ 0 \leq i \leq t, \quad \overrightarrow{y_i} = M_f \cdot \overrightarrow{x_i}.$$

As a consequence, any set of probes $(v_1, \dots, v_q)$ in C' as defined in Game 3 can equivalently be rewritten as the concatenation of the $t+1$ vectors

$$\overrightarrow{v_0} = M_0 \cdot M_f \cdot \overrightarrow{x_0}\ , \quad \overrightarrow{v_1} = M_1 \cdot M_f \cdot \overrightarrow{x_1}\ , \quad \dots \quad \overrightarrow{v_t} = M_t \cdot M_f \cdot \overrightarrow{x_t}.$$

By contradiction, let us assume that

$$\mathrm{Im}(M_f^T \cdot M_0^T) \cap \mathrm{Im}(M_f^T \cdot M_1^T) \cap \dots \cap \mathrm{Im}(M_f^T \cdot M_t^T) \neq \emptyset,$$

that is, there exists a non-null vector $\overrightarrow{w}$ such that

$$\overrightarrow{w} \in \mathrm{Im}(M_f^T \cdot M_0^T) \cap \mathrm{Im}(M_f^T \cdot M_1^T) \cap \dots \cap \mathrm{Im}(M_f^T \cdot M_t^T).$$

Equivalently, there exists $\overrightarrow{z_0}, \overrightarrow{z_1}, \dots, \overrightarrow{z_t}$ such that

$$\overrightarrow{w} = M_f^T \cdot M_0^T \cdot \overrightarrow{z_0} = M_f^T \cdot M_1^T \cdot \overrightarrow{z_1} = \dots = M_f^T \cdot M_1^T \cdot \overrightarrow{z_t}.$$

From Eq. (11), there exist at least two distinct indices i and j in $\{0, \dots, t\}$, such that

$$M_i^T \cdot \overrightarrow{z_i} \neq M_j^T \cdot \overrightarrow{z_j}.$$

As $\overrightarrow{w} = M_f^T \cdot M_i^T \cdot \overrightarrow{z_i} = M_f^T \cdot M_j^T \cdot \overrightarrow{z_j}$, the difference $M_i^T \cdot \overrightarrow{z_i} - M_j^T \cdot \overrightarrow{z_j}$ belongs to M_f^T's kernel. But from the surjective property of M_f, M_f^T has full column rank n', and thus a null kernel:

$$\dim(\mathrm{Ker}(M_f^T)) = n' - \dim(\mathrm{Im}(M_f^T)) = 0.$$

As a consequence, $M_i^T \cdot \overrightarrow{z_i} - M_j^T \cdot \overrightarrow{z_j} = 0$ and since $M_i^T \cdot \overrightarrow{z_i} \neq M_j^T \cdot \overrightarrow{z_j}$ we have a contradiction which completes the proof. □

Eventually, we claim that two t-private circuits on independent encodings form a t-private circuit as well.

Proposition 13. *Let us consider a standard shared circuit C composed of two parallel t-probing secure circuits which operate on independent input sharings. Then, $C = C_1 \| C_2$ is t-probing secure.*

Proof. As the input sharings are independent, the result is straightforward from Lemma 2. □

5.2 Application to SPN-Based Block Ciphers

An SPN-based block cipher is a permutation which takes as inputs a key k in $\{0,1\}^\kappa$ and a plaintext p in $\{0,1\}^n$ and outputs a ciphertext c in $\{0,1\}^n$, where n and κ are integers. It is defined by successive calls to a round function and by an optional expansion algorithm KS. The round function is a combination of a non linear permutation S and a linear permutation L.

Proposition 14. *Let C be a standard shared circuit implementing an SPN block cipher. And let C_S and C_{KS} be the standard shared (sub-)circuits implementing S and KS respectively. If both conditions*

1. *C_S's and C_{KS}'s outputs are t-SNI gadgets' outputs,*
2. *C_S and C_{KS} are t-probing secure (for any sharing order $t+1$),*

are fulfilled, then C is also t-probing secure.

Note that if S's and KS's outputs are not t-SNI gadgets' outputs, then the linear surjective circuit can be extended to the last t-SNI gadgets' outputs of these circuits without loss of generality.

Proof. As S and KS are t-probing secure, it follows from Proposition 13, that when implemented in parallel on independent input encodings, their composition is t-probing secure as well. Then, as the output of their composition matches the outputs of t-SNI gadgets, then they can be sequentially composed with a t-probing secure circuit from Proposition 11. Finally, the composition of linear surjective circuits with t-probing secure circuits is ensured by Proposition 12, which completes the proof. □

5.3 Extension to Generic Shared Circuits

We discuss hereafter two straightforward extensions of our work. Namely some constraints on gadgets that compose the standard shared circuits can be relaxed, and the considered circuit can easily be extended to work on larger finite fields.

On Standard Shared Circuits. The method presented in this paper through Sects. 3 and 4 aims to accurately establish the t-probing security of a *standard shared circuit* for any sharing order $t+1$. Namely, it is restricted to Boolean shared circuits exclusively composed of ISW-multiplication gadgets, ISW-refresh gadgets, and sharewise addition gadgets. While the assumption on addition gadgets is quite natural, the restrictions made on the multiplication and refresh gadgets can be relaxed. The reduction demonstrated in Sect. 3 only expects the refresh gadgets to be t-SNI secure to ensure the equivalence between Game 1 and the initial t-probing security game. Afterwards, t-probing security is equivalently evaluated on a corresponding *flattened* circuit with probes on multiplications' operands only. Therefore, there is no restriction on the choice of refresh gadgets but their t-SNI security. While multiplication gadgets are also expected to be t-SNI secure for the equivalence between Game 1 and the initial t-probing security game to hold, this feature is not enough. To prove this equivalence, multiplication gadgets are also expected to compute intermediate products between every share of their first operand and every share of their second operand. Otherwise, our method could still establish the probing security of a circuit, but not in a tight manner, meaning that security under Game 3 would imply probing security but insecurity under Game 3 would not imply insecurity w.r.t. the original probing insecurity notion. Our method would hence allowed false negatives, as state-of-the-art methods currently do. Beyond the advantages of providing an exact method, this restriction is not very constraining since not only the widely deployed ISW-multiplication gadgets but also the large majority of existing multiplication gadgets achieve this property.

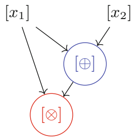
On Circuits on Larger Fields. Since ISW-multiplication gadgets and ISW-refresh gadgets can straightforwardly be extended to larger fields our reduction and verification method could easily be extended to circuits working on larger fields.

6 Application

Following the results presented in previous sections, we developed a tool in sage, `tightPROVE`, that takes as input a standard shared circuit and determines whether or not it is t-probing secure with Algorithm 1. Specifically, the standard shared circuit given as input to `tightPROVE` is expressed as a set of instructions (`XOR`, `AND`, `NOT`, `REFRESH`) with operands as indices of either shared input values or shared outputs of previous instructions. Namely, the `XOR` instructions are interpreted as sharewise addition gadgets of fan-in 2, the `NOT` instructions as sharewise addition gadgets of fan-in 1 with the constant shared input $(1, 0, \ldots, 0)$, the `AND` instructions as ISW-multiplication gadgets of fan-in 2, and the `REFRESH` instructions as ISW-refresh gadgets of fan-in 1. As an application, we experimented `tightPROVE` on several standard shared circuits. First, we analyzed the t-probing security of the small examples of Sect. 4 as a sanity check. Then, we investigated the t-probing security of the AES s-box circuit from [6] and compared the result with what the `maskComp` tool produces. Additionally, we studied the impact of our tool to practical implementations (for both the randomness usage and the performance implications).

6.1 Application to Section 4 Examples

In order to have some sanity checks of our new method on simple standard shared circuits, we applied `tightPROVE` to the examples given in Sect. 4, namely the standard shared circuits depicted in Figs. 1 and 10. Specifically, we first translated the two standard shared circuits into a list of instructions that is given to our tool. For each circuit, the first instruction gives the number of shared inputs. Then, each of the following instruction matches one of the four possible operations among `XOR`, `AND`, `NOT`, and `REFRESH` together with the indices of the corresponding one or two operands. The output of each such operation is then represented by the first unused index. At the end, from the generated list of instructions the tool derives a list of pairs of operands, namely the inputs to the multiplications in the circuit. Finally, Algorithm 1 is evaluated on the obtained list of operands.



→

```
;;  2
XOR 1 2
AND 1 3
```

→

```
list_comb = [1,3]
-----------------------------------
comb =  1
=> NO ATTACK (G2 = G1)
   G: [[(1,3)], []]
   O: [[3], []]
-----------------------------------
comb =  3
=> NO ATTACK (G2 = G1)
   G: [[(1,3)], []]
   O: [[1], []]
-----------------------------------
('No attack found')
```

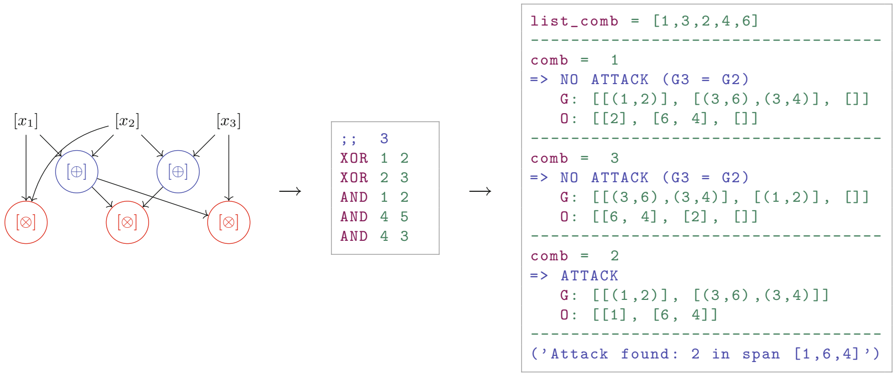
Fig. 11. New method applied on example 1.

The first example is based on a standard shared circuit that takes 2 shared inputs and then performs two operations, namely a sharewise addition (`XOR`) and an ISW-multiplication (`AND`). The `AND` instruction takes two inputs, namely the output of the `XOR` and one of the two inputs of the circuit, which means that there is only two possible target vectors for an attack to be mounted. They are displayed in the list `list_comb`. For both these two vectors successively displayed with variable `comb`, the tool generates their respective sets $\mathcal{G}_1$ and $\mathcal{O}_1$, as defined in Sect. 4. Then since $\mathcal{G}_2$ is equal to $\mathcal{G}_1$ for both vectors, the tool outputs that no attack could be found. The circuit is thus t-probing secure. The complete process is described in Fig. 11.

The second example is based on a standard shared circuit that takes 3 shared inputs and then performs 5 operations, namely 2 sharewise additions (`XOR`) and 3 ISW-multiplications (`AND`). The three `AND` instructions take five distinct inputs, which means that there are five possible target vectors for an attack to be mounted. For the two first target vectors, no attack could be found as the tool expressed all the multiplications in the circuit with two sets $\mathcal{G}_1$ and $\mathcal{G}_2$ without finding any attack. For the third target vector, after the construction of $\mathcal{G}_2$ an attack was found as the target vector belonged to the span of the set $\mathcal{O}_2$. The complete process is described in Fig. 12. Moreover, we verified that by adding a refresh gadget on the operand for which our tool finds an attack prior to the multiplication where it is used, the tool is not able any more to find an attack on the new circuit for this example. The results can be found in the full version of this paper [4].

6.2 Application to AES s-box

At Eurocrypt 2017, Goudarzi and Rivain [14] proposed an efficient software implementation of the s-box of the AES for higher-order masking. Based on



```
;;  3
XOR 1 2
XOR 2 3
AND 1 2
AND 4 5
AND 4 3
```

```
list_comb = [1,3,2,4,6]
-----------------------------------
comb =  1
=> NO ATTACK (G3 = G2)
   G: [[(1,2)], [(3,6),(3,4)], []]
   O: [[2], [6, 4], []]
-----------------------------------
comb =  3
=> NO ATTACK (G3 = G2)
   G: [[(3,6),(3,4)], [(1,2)], []]
   O: [[6, 4], [2], []]
-----------------------------------
comb =  2
=> ATTACK
   G: [[(1,2)], [(3,6),(3,4)]]
   O: [[1], [6, 4]]
-----------------------------------
('Attack found: 2 in span [1,6,4]')
```

Fig. 12. New method applied on example 2.

the Boolean circuit of Boyar *et al.* [6], their implementation evaluates the s-box on a state under bitsliced representation with only 32 AND gates. In order to be t-probing secure without doubling the number of shares in the encoding of sensitive variables, a conservative choice was made to add a refresh gadget prior to each multiplication. As explained in Sect. 1, a major drawback of such a conservative approach is the performance overhead induced by the number of calls to refresh gadgets due to the randomness usage.

In order to obtain efficient implementations of the AES s-box and to be tight on the number of randomness requirement, we have applied our tool to the circuit of the s-box reordered by Goudarzi and Rivain without any refreshing gadget. Interestingly, we obtained that no attack can be found for any masking order. More precisely, the tool first identified 36 distinct target vectors out of the 64 possible operands of multiplication gadgets (it can be easily checked on the circuit found in Sect. 6 of [14]). For each of the 36 target vectors, the corresponding set $\mathcal{G}_1$ is constructed. Then, for every variable the algorithm stops as the respective sets $\mathcal{G}_2$ are always equal to the respective sets $\mathcal{G}_1$. The complete report of the tool results can be found in the full version of this paper [4].

To prove the security of the AES s-box circuit, our tool took only 427 ms. This speed is mainly due to the fact that for each possible target variable, only the set $\mathcal{G}_1$ is computed. For comparison, we looked at the time taken by the `maskVerif` tool of [1]. For a masking order $t = 2$, `maskVerif` found no attack in 35.9 s and for $t = 3$ in approximately 10 h.

For the sake of comparison, we also applied the `maskComp` tool on the same circuit. We obtained that `maskComp` adds refresh gadgets prior to each multiplication in the circuit, transforming it into a new t-NI secure circuit. Since our tool has shown that the circuit is t-probing secure with no refresh gadgets, adding those refresh gadgets implies an overhead in the t-probing security that can lead to less efficient practical implementations. As an illustration, we have implemented the AES s-box circuit in bitslice for a generic masking order to see the impact in performances between a full refresh approach (*i.e.* the conservative

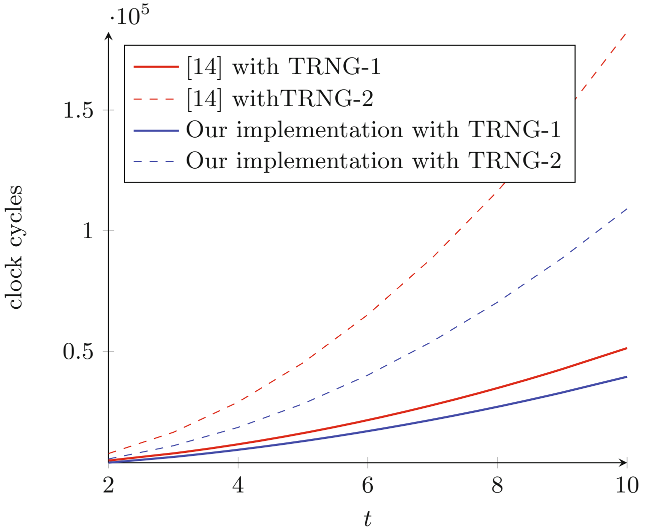


Fig. 13. Timings of a t-probing secure AES s-box implementation.

choice of Goudarzi and Rivain and the result of `maskComp`) and a no refresh approach (our new tool). Each of this two approaches produces a circuit that is at least t-probing secure for any masking order t. Both produced circuit are securely composable with other circuits (for `maskComp` from the proofs given in [2] and for our tool from the result of Sect. 5). To be consistent with the state of the art, the randomness in our implementations can be obtained from a TRNG with two different settings: a first setting with a *free* TRNG that outputs 32-bit of fresh randomness every 10 clock cycles (as in [14]) and a second setting with a constrained TRNG that outputs 32-bit of fresh randomness every 80 clock cycles (as in [16]). The performance results can be found in Table 1. For both approaches, the number of refresh gadgets used and the number of randomness needed are displayed. Then, the timing in clock cycles for both settings are shown. We can see that our tool allows to divide by 2 the number of required randomness and benefits from an asymptotic gain of up to 43% in speed. The comparison of the timings for several masking orders are depicted in Fig. 13.

Table 1. Performance results of the implementation AES s-box depending on the number of refresh gadgets

	Nb. of refresh	Nb. of random	Timing (Set. 1)	Timing (Set. 2)
[14]	32	$32\,t(t-1)$	$408\,t^2 + 928\,t + 1262$	$1864\,t^2 - 528\,t + 1262$
this paper	0	$16\,t(t-1)$	$295.5\,t^2 + 905.5\,t + 872$	$1069\,t^2 + 132\,t + 872$

Acknowledgments. We would like to thank François-Xavier Standaert and Gaëtan Cassiers for their in-depth review and helpful comments.

References

1. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y.: Verified proofs of higher-order masking. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 457–485. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_18
2. Barthe, G., et al.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 116–129. ACM Press, New York, October 2016
3. Barthe, G., et al.: Masking the GLP lattice-based signature scheme at any order. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 354–384. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_12
4. Belaïd, S., Goudarzi, D., Rivain, M.: Tight private circuits: achieving probing security with the least refreshing. IACR Cryptol. ePrint Arch. **2018**, 439 (2018)
5. Bloem, R., Gross, H., Iusupov, R., Könighofer, B., Mangard, S., Winter, J.: Formal verification of masked hardware implementations in the presence of glitches. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 321–353. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_11
6. Boyar, J., Matthews, P., Peralta, R.: Logic minimization techniques with applications to cryptology. J. Cryptol. **26**(2), 280–312 (2013)
7. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
8. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26
9. Coron, J.-S.: Formal verification of side-channel countermeasures via elementary circuit transformations. Cryptology ePrint Archive, Report 2017/879 (2017). http://eprint.iacr.org/2017/879
10. Coron, J.-S., Prouff, E., Rivain, M., Roche, T.: Higher-order side channel security and mask refreshing. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 410–424. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_21
11. Coron, J.-S., Rondepierre, F., Zeitoun, R.: High order masking of look-up tables with common shares. Cryptology ePrint Archive, Report 2017/271 (2017). http://eprint.iacr.org/2017/271
12. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_24
13. Goubin, L., Patarin, J.: DES and differential power analysis the "Duplication" method. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_15
14. Goudarzi, D., Rivain, M.: How fast can higher-order masking be in software? In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 567–597. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_20

15. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
16. Journault, A., Standaert, F.-X.: Very high order masking: efficient implementation and security evaluation. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 623–643. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_30
17. Messerges, T.S.: Using second-order power analysis to attack DPA resistant software. In: Koç, Ç.K., Paar, C. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44499-8_19
18. Micali, S., Reyzin, L.: Physically observable cryptography. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_16
19. Prouff, E., Rivain, M.: Masking against side-channel attacks: a formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_9
20. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_28
21. Zhang, R., Qiu, S., Zhou, Y.: Further improving efficiency of higher order masking schemes by decreasing randomness complexity. IEEE Trans. Inf. Forensics Secur. **12**(11), 2590–2598 (2017)