UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Masking Against Side-Channel Attacks: Security and Performance Improvements

or An Algorithmic Journey from the Probing Model to the Noisy Leakage Model

Dissertation presented by
Gaëtan CASSIERS

for obtaining the Master's degree in
Electrical Engineering

Supervisor
François-Xavier STANDAERT

Readers
Jean-Didier LEGAT, Olivier PEREIRA

Academic year 2017-2018

Abstract

Security is a primordial requirement of the ever more ubiquitous communicating information systems. Modern cryptography is very effective at protecting the communications, but what about the devices themselves? Side-channel attacks are a way to exploit the physical emanations of a chip in order to extract secret information from its inside, for example leading to recovering the secret key used to secure communications.

The masking technique provides a framework to build algorithmic countermeasures against side-channel attacks that proceeds by randomizing the computations. In this way, the link between the physical emanations and the secret information is weakened and extracting the information is expected to be much harder. However, this technique has a high computational cost as it increases by a multiplicative factor the number of computations to be carried out, and this can prevent its deployment given the computational resource constraints of some embedded devices. Furthermore, since masking is a high-level algorithmic protection against low-level physical attacks, its impact on the security level is rather hard to precisely quantify.

The first goal of this master thesis is to analyze the masking principles and to refine their conceptual understanding, both from the qualitative and the quantitative viewpoints, in order to improve the security of concrete schemes. A second goal is to design new masking algorithms that increase the state-of-the-art computational efficiency of existing schemes, in order to extend the applicability range of masking.

For this purpose, we rely on two main models that are often used to analyze the security of masking. In the (more abstract) probing model, we introduce a new security definition which, combined with a proof technique based on analyzing the propagation of the probes within the implementations, allows greatly simplified analyzes of complex masked implementations thanks to strong composability guarantees. We give the first implementations that satisfy the new security definition, and show that it reduces the state-of-the-art computational cost (that we also optimize thanks to integer programming) by about 40 %. In the (more concrete) noisy leakage model, we analyze quantitatively the security levels of various masking algorithms, which leads to the important conclusion that randomness optimizations in the probing model are not sufficient to discuss the global security vs. efficiency trade-off of masking. We then use our findings to design new algorithms that better capture this trade-off and show that they gradually gain in relevance as the order of masking schemes increases.

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor Prof. François-Xavier Standaert for introducing me to the exciting topic of side-channel attacks, and giving me a clear overview of the possible research directions in this domain. He was always available to help me move forward thanks to open-minded conversations and deep reviews of my research results. I would like to thank my classmates Antoine and Charles for the passionate interactions we had, often leading to new ideas. Finally, I would also like to thank Prof. Jean-Didier Legat and Prof. Olivier Pereira for accepting to be readers this thesis.

Contents

List of Symbols and Abbreviations

$\mathbb{F}_q$	Galois field with q elements
$x \xleftarrow{\$} X$	x is drawn from a uniform distribution over the set X, independently of any other random variable.
$\Pr[X]$	Probability of event X
$H(X)$	Entropy of random variable X
$\mathcal{O}(f(\cdot))$	Complexity class upper-bounded by a multiple of f
$o(f(\cdot))$	Complexity class lower-bounded by a multiple of f
$\Theta(f(\cdot))$	Complexity class upper-bounded and lower-bounded by multiples of f
HW	Hamming Weight
$\lvert X \rvert$	Size of the set X
n	Number of (unshared) inputs of a gadget
d	Masking order. The number of shares is $d+1$.
$x_{i,j}^G$	Share j of the input i of gadget G
$x_{*,j}^G$	The share j of all the inputs of gadget G
MI_o	Information leaked for one manipulation of a variable
MI_t	Information on the sensitive variable recovered by an horizontal attack
MI	Mutual Information
PDF	Probability Density Function
SNR	Signal-to-Noise Ratio
AES	Advanced Encryption Standard
SPN	Substitution-Permutation Network
SCA	Side-Channel Attack
DAG	Directed Acyclic Graph
RPM	Random Probing Model
LRPM	Local Random Probing Model
BP	Belief Propagation
SASCA	Soft Analytical Side-Channel Attack
ISW, $\mathsf{SNI\text{-}ISW}$	Multiplication gadget of Ishai, Sahai and Wagner [37]
PINI_1	First PINI multiplication gadget (Algorithm 2)
PINI_2	Second PINI multiplication gadget (Algorithm 5)

Introduction

As the society of information technology develops, and with the advent of the internet of things, numerous embedded devices are deployed, and increasingly placed everywhere, including locations where physical access in not controlled. Furthermore, the devices often need security features: secure communications, authentication of users, etc. For this reason, cryptography became a fundamental building block of such secure systems.

Modern cryptography has been proven to be very successful: few cryptographic algorithms have been unexpectedly broken in the last decades. The root of modern cryptography is the adversarial model. It is often expressed as a security game, where the adversary has access to some information, may sometimes interact with an algorithm and must finally guess some data (such as a secret message). The algorithm attacked by the adversary is considered as secure if, for any adversary, the probability that the adversary wins (i.e. guesses the correct data) is not significantly larger than the one of a random guessing adversary. The main features of such models are first that they make no assumption on the nature of the adversary: the algorithm is secure against any adversary (although some models limit its computational power). Second, the adversary is limited in the information it is given access to. For example, it may have access to some plaintext and to the corresponding ciphertext, but not to the encryption key.

In most cryptographic models, algorithms are viewed as black-boxes: the adversary has only access to some inputs and outputs of the algorithm, and never to the intermediate values [37]. Side-channel attacks [41] have challenged this assumption: algorithms are implemented in devices, and the physical emanation of a device (power consumption, electromagnetic radiation, computation time, etc. [62]), which we next refer to as leakage through side-channels, depends on the execution of the algorithm and on the manipulated data. An adversary that measures the emanations can thus get information about the intermediate values of the algorithm and that can completely break the security [20].

The first countermeasures against those attacks were based on physical characteristics such as decreasing the amplitude of the leakage signal and increasing the noise [42]. These countermeasures are effective, but they are limited in that the increase in hardness for the adversary is roughly proportional to the signal-to-noise ratio (SNR), and thus to the cost of the countermeasure, whereas in cryptography it is commonly expected that countermeasures increase the difficulty of the attacks exponentially with respect to their cost. Research for techniques that obtain this exponential security in a formally proven way and for formalization of adversarial models has thus been carried out [16]. Among the most prominent techniques against side-channel attacks is the family of masking techniques, that split any sensitive variables into $d + 1$ shares (where d is known as the masking order) such that any set of size d of intermediate values in the algorithm is independent of the secret information. It is proven [21] that the security level of masking

is exponential in d, and most masking implementations have $\mathcal{O}(d^2)$ cost. Although these asymptotic complexities are formally sufficient, masking is computationally expensive: it can increase the computation time by orders of magnitude compared to non-protected implementations [56, 31]. Furthermore, the proofs do not necessarily give the actual security level, that depends on many factors such as the SNR of an attack. Those limitations, along with the complexity of the implementation, make it difficult to deploy a masked algorithm in a real-world device. This thesis is a step towards the long-term goal of having masking algorithms that have low computational cost, that are applicable to a wide range of devices/side-channels (in particular those with higher SNR), that can be easily implemented and whose security level can be reliably measured.

A common approach to prove the security of masking schemes is based on the use of multiple adversarial models. A scheme is first proven secure in an abstract model. Given security in the first model and additional evidence, the scheme can be proven secure in a second (more concrete) model, and so on. We work on two of those models, and for each of those, we first build analysis tools to gain a better intuitive understanding of the model and of the existing techniques, and then we design new masking techniques (e.g security definitions, concrete algorithms, etc.).

As a theoretical background (Chapter 1), we first introduce a standard building block of cryptography: the AES block cipher. This algorithm is used in many cryptographic devices and is often a target for side-channel attacks (see e.g. [45]), we will thus use it as an example of algorithm to implement in a masked fashion. Next, we discuss in more details the side-channel attacks, the various adversarial models for those attacks and the proofs of security of the masking technique in those models, as well as the way masking can actually be implemented in a device.

The two main parts of this thesis relate to the analysis and design of masking schemes. The first part (Chapter 2) considers security in the abstract probing security model [37, 56]. In this model, the question of security is well-understood and binary, hence the two main concerns are the question of composability (i.e. how to assemble secure algorithmic building blocks — named gadgets, for example arithmetic operations — securely?) and the computational cost. We first recall the probe propagation framework, which is an intuitive way to prove the security in this model. This allows us to build an automated tool that securely composes existing building blocks (namely, non-interfering and strongly non-interfering gadgets) in a way that optimizes the computational cost. Another contribution is the introduction of a new security definition, which enjoys a very simple composability property (essentially: any composition of gadgets that satisfy this definition is secure). Lastly, we introduce gadgets that satisfy this definition, which immediately leads to improved overall performance.

The second part deals (Chapter 3) with the more realistic noisy leakage model [54]. This model allows us to analyze more precisely the possible attacks, in particular the powerful horizontal attacks [7]. A first contribution is an automated tool that bounds the security level in this model (in a slightly heuristic way, but that is close to the worst-case). Second, this tool is used to analyze gadgets with and without heuristic protection against horizontal attacks, which confirms the effectiveness of the protection. Third,

the previous results and our analysis emphasize the existence of a trade-off between the computational cost and the security level and the need for a joint optimization. Lastly, we design new protections that improve the security level and that go towards security-computation trade-off optimization. Moreover, it is important to note that the security in the noisy leakage model depends on the SNR: increasing the noise level increases the security while at low noise the security vanishes. With previous protections, improving the security level thanks to algorithmic changes (i.e. increasing the masking order d) also increases the minimum noise level at which the implementation is secure, and this noise requirement can be hard or expensive to meet. Our new protections have a noise level requirement independent of the masking order, which in practice translates into lower noise requirement for high security levels.

We finally note that the representation of the device in the models we use is rather abstract. An interesting direction for future research is thus to implement our new algorithms in actual devices and investigate to what extent the hardware non-idealities (such as glitches) reduce the security level.

1. Background

1.1. Block Ciphers

Block ciphers are a building block of modern symmetric cryptography. They are mainly used to build symmetric encryption systems, but they also appear in some hash function, authentication schemes, pseudo-random number generators, etc. Block ciphers are often the main cryptographic algorithm in small devices such as smartcards for which side-channel attacks are particularly relevant. They are thus a natural (and often used) target for experimentation with side-channel attacks and countermeasures [45, 56].

A block cipher is a computationally secure instance of a pseudo-random permutation: it is a function $F : \mathbb{F}_2^s \times \mathbb{F}_2^l \to \mathbb{F}_2^l : (k, x) \mapsto F_k(x)$ such that [39]:

- For any $k \in \mathbb{F}_2^s$, $F_k(\cdot)$ is a bijection over $\mathbb{F}_2^l$.
- F can be *efficiently* (i.e. polynomial-time) evaluated.
- For any probabilistic polynomial-time distinguisher algorithm D,

$$\left|\Pr\left[D^{F_k}(1^l) = 1\right] - \Pr\left[D^{f_l}(1^l) = 1\right]\right| \leq \epsilon(l),$$

  where $k \xleftarrow{\$} \mathbb{F}_2^s$ and f^l is a uniformly randomly chosen bijection over $\mathbb{F}_2^l$. $D^f(1^l)$ denotes the output of the algorithm D when it is given oracle access to f: for any x it can get $f(x)$ but it has no other information about f. $\epsilon(l)$ is a negligible function: for any polynomial P, there exists a M such that for any $m > M$, $\epsilon(m) < p(m)$.

In this formal definition, efficient (or polynomial-time) means that the computational complexity is polynomial in the security parameters l (the block size) and s (the key size).

Building a block cipher that satisfies this definition is not a trivial task. A simple approach that would list in a table the value $F_k(x)$ for any k and x is not be efficient. It is possible to build a block cipher that satisfies all the required properties (including polynomial-time evaluation) [53, 48] whose security is implied by a strong numerical assumption (such as hardness of factorization). The practical efficiency for common values of security parameters (for example $n = s = 128$) is however very poor. Constructions that are based on more ad-hoc assumptions and motivated by a heuristic approach are thus used in practice.

We take the AES block cipher (also known as Rijndael) [19] as the algorithm for which we build implementations protected against side-channel attacks. The AES is one of the

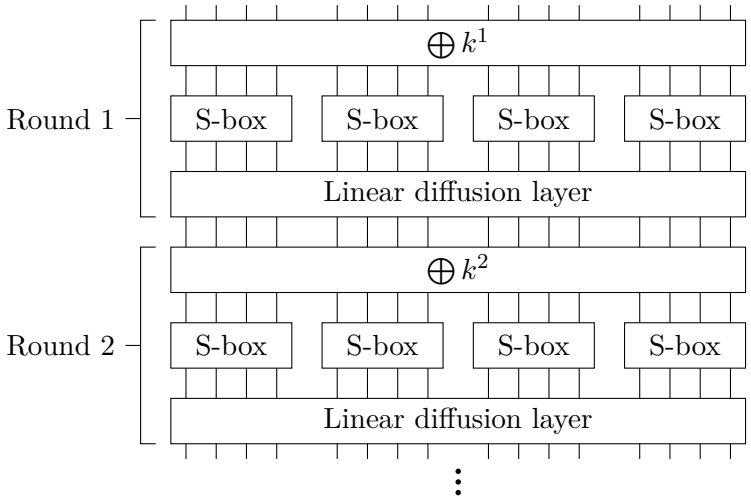


Figure 1.1.: Example of SPN structure with 4-bit S-boxes and $l = 16$ bits.

most used block ciphers, it has 128 bit block size and 128 bit key size.[1] It is based on the Substitution-Permutation Network (SPN) architecture.

1.1.1. SPN Architecture [39]

The idea behind SPN is that although it is impossible to store the explicit description of a large bijection, it is possible to do so for a small bijection. A small bijection, named a S-box, is thus replicated to form a big bijection. The S-box of the AES operates over 8 bits and it is repeated 16 times in parallel to cover the whole block size. A simple parallel combination of S-boxes is however not sufficient, as its structure can be detected from a few input/output pairs. This makes it easily distinguishable from a random permutation. Furthermore, this construction does not depend on the key k.

The SPN architecture (see example in Figure 1.1) solves those issues by serially composing three stages (also known as layers):

- The S-box layer: a parallel combination of S-boxes.
- The diffusion layer: this layer is linear (hence easy to describe and compute), and each of its output bits depends on multiple input bits that are produced different S-boxes (this is called the diffusion property).
- The addition (bitwise `XOR`) of the key.

This composition is not secure by itself, it is thus iterated multiple times, and each iteration is called a round. For instance, the AES has ten rounds.

At each round, the key added differs. A specific algorithm, called the key scheduling, generates the key of each round from the main key. Even though the key scheduling

[1] There also exists 192 bit and 256 bit variants, but we only discuss the 128 bit construction.

algorithm must be protected against side-channel attacks, the way to do it is not discussed in this thesis since it is similar to the protection of the rounds of the block cipher.

AES: field representations The AES is specified as an algorithm that operates over values in $\mathbb{F}_{256}$ [19]. The operation over those values are linear operations (sums, multiplication by a constant) and one non-linear operation in the S-box: field inversion. The linear operations are simple to implement, and the field inversion is tabulated in most implementations.

Another possibility is to consider the AES as operating over bits (values in $\mathbb{F}_2$), the involved operations are thus the basic field operation in $\mathbb{F}_2$: addition (`XOR`) and multiplication (`AND`) [55]. A software implementation using this approach can run many operations in parallel, using the bitwise operations of the processor (on a 32 bit processor, each bit, a `XOR` instruction computes the `XOR` between 32 pairs of bits). The parallelism can be used to run multiple computations of the AES in parallel or to different parts of the same computation (e.g. 16 parallel S-boxes can be computed at the same time). Such implementations are called bitslice implementations.

1.2. Side-channel Attacks

As already mentioned in the introduction, modern cryptography is based the definition of adversarial models. While the usual abstract models which consider algorithms as black-boxes are useful tools that help to build complex systems and to prove their security, they completely ignore attacks related to the physical implementation of the algorithms, which are known as the side-channel attacks (SCA).

While a side-channel attack has already been carried in 1965 [62], Paul Kocher introduced SCA in the public cryptography research community in the nineties [40, 41]. Since this introduction, many kinds of SCA have been discovered. Even if it is difficult to build an exact classification of the SCA, they can be categorized into analysis attacks that do not perturb the operation of the device under attack, and the fault attacks that introduce errors in the computations.

The side-channel analysis attacks exploit the correlation between sensitive variables (i.e. variables that depend on secrets) and information carried through various physical media: computation time, electrical power consumption, electromagnetic radiation, etc., that is called (physical) leakage.

The timing attacks often apply to microprocessor based implementations, and exploit time variability due to branches, cache misses, etc. [40, 27, 13] A very effective countermeasure against those attacks is to write constant time code: never use a sensitive variable as an operand of an operation whose execution time depends on the operand. While this can be hard to put into practice, this countermeasure is simple, well understood and very effective, although it can significantly increase the execution time [50, 11].[2]

[2] Achieving constant time and high performance on complex microprocessors can be complicated due to their numerous non constant-time features (caches, branch predictors, etc.). It is however possible to

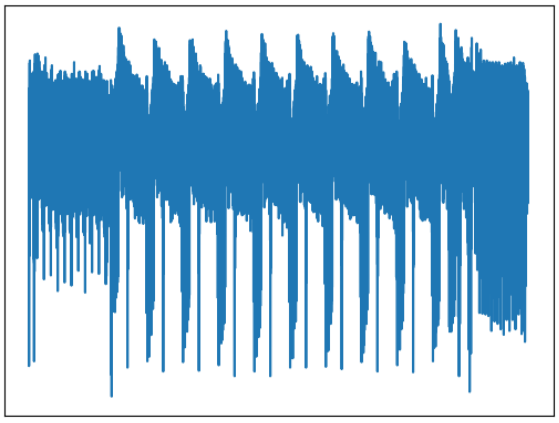


Figure 1.2.: Leakage trace of an AES encryption. The then rounds of the algorithm can be distinguished.

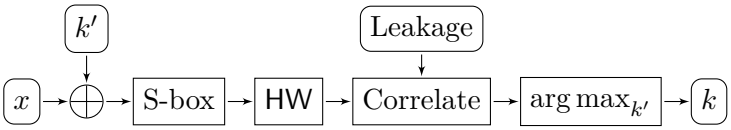


Figure 1.3.: Simple DPA workflow.

The other big class of SCA is the one that was introduced with the differential power analysis (DPA) [41]. This class also includes SCA are similar to the DPA, but exploit other media such as electromagnetic radiation. The basic principle of DPA is that the power consumption of a circuit depends on the values being manipulated. Measuring a time series (called a leakage trace) of the power consumption gives thus a noisy measurement of the sensitive variables.

Figure 1.3 shows an example of a simple key recovery DPA attack for a AES implementation on a microprocessor. First, the adversary collects leakage traces such as the one shown in Figure 1.2: for many known plaintexts (and always the same key), the power consumption is measured. The simple DPA attack is based on the divide-and-conquer principle: it recovers one byte of the key at a time, by exploiting the leakage of the output of one S-box in the first round. The relationship between a specific byte and the leakage trace is unknown, but we can assume a simple model, for example that at least one time sample is correlated with the Hamming Weight (HW) of each manipulated byte (more accurate models could be built by profiling). Let k be a guess of the key byte, we model the leakage $\mathsf{HW}(\mathsf{S}-\mathsf{box}(x \oplus k))$ for all the plaintexts x for which the leakage has been measured. If the key guess is correct, the modeled leakage is correlated to the actual leakage of the output of the S-Box; and there is no correlation otherwise (due to the strong non-linearity of the S-box). The attack is thus simple: for all key guesses and for all sampling time, we compute the correlation between the modeled leakage for all

write simple constant-time code at a sometimes large performance cost.

the plaintexts and the actual leakage samples. The maximum absolute value for this correlation is then chosen, and the corresponding key guess is the final key guess (which is the actual key if the number of recorded traces is sufficient). The attack is applied sequentially to all the S-boxes of the first round to find all the bytes of the key.

Whereas constant time code is a relatively simple and theoretically sound countermeasure for timing attacks, no such thing exists for DPA. Indeed, physical emanations of an implementation are always dependent of the manipulated values, even though physical countermeasures allow to reduce the amount of leaked information [62], and thus increase the number of trace required to break the system. Those countermeasures are often medium-specific: a shielding against electromagnetic radiations will have low impact on the power consumption.

1.3. Masking Countermeasure

Masking is among the most popular countermeasures to prevent side-channel attacks. It has been first introduced in a rather heuristic/ad hoc context, it was then formally proven in an abstract (not very realistic) adversarial model [37], and it has more recently been proven secure in a more realistic model [21].

Representation of algorithms In the following, we discuss the implementation of algorithm that work with values exclusively in Galois fields of characteristic 2 (the most common case being $\mathbb{F}_2$). A first representation for algorithms is a (possibly randomized) function. Furthermore, for the algorithms do not contain branches or loops with a non-constant number of iterations,[3] it is possible (by unrolling the loops) to get the second representation: an arithmetic circuit. This representation is fundamental for the analysis of masking.

A deterministic circuit C is a Directed Acyclic Graph (DAG) whose vertices are (boolean) gates and whose edges are wires. A randomized circuit is a circuit augmented with random gates. A random gate is a gate with fan-in 0 that produces a random output, uniformly and independently of everything else afresh for each invocation of the circuit.

Masking works with the circuit representation of the algorithm to protect: the original (or source) circuit. A masking algorithm is a circuit compiler: it takes as input the original circuit and outputs another circuit, called a masked circuit, that has equivalent functionality but that is secure against SCA.

Masking intuition Given a masking order d, the working principle of masking is to split all the sensitive data manipulated by the source circuit in $d+1$ shares in the masked circuit, and to perform the computations on those shares only [16]. In the masked circuit, each wire of the original circuit (which is a sensitive variable) is replaced by a set of wires (the shares), and each gate is replaced with a gadget (whose operation on the shares is

[3] This is usually the case for cryptographic algorithm, and it is required to be secure against timing attacks.

equivalent to the operation of the gate in the original circuit). Intuitively, this protection makes attacks more difficult: to recover the value of a sensitive variable, the adversary has to know the value of all its shares (this intuition is formalized in Section 1.3.1).

Concretely, various types of masking schemes have been considered in the literature including additive/boolean masking (e.g., [37, 56, 18] and many follow-ups), multiplicative masking [29, 28], affine masking [61, 26], Inner Product masking [2, 3], ... All these proposals come with significant overheads in execution time and randomness consumption. We focus on the case of additive boolean masking, which has been shown recently to provide the best concrete performances thanks to bitslice implementations (see Section 1.6) [31, 38].

In additive boolean masking (hereafter also simply denoted masking), any sensitive variable x is shared into a tuple $(x_0, \ldots, x_d)$. To mask a variable x, the values $x_0, \ldots x_{d-1}$ are generated uniformly at random, and $x_d = x_0 + \cdots + x_{d-1} + x$[4]. Any set of at most d shares is thus independent of x.

1.3.1. Threshold Probing Model

In the current state-of-the-art, masking schemes usually come with a security proof in the so-called probing model [37, 56] (or threshold probing model). In its simplest definition, d-probing security requires that the observation of up to d intermediate variables in the implementation does not reveal anything about the sensitive variables.

Admittedly, a situation where an adversary can probe up to d wires in a circuit is not very common (although it can model attacks carried thanks to micro-probing [52]). A security proof in the probing model is however a first step in the analysis of a masked implementation that must be followed by discussions in more realistic leakage models (see Section 1.5). Yet, it is a necessary first step since an insecurity in the probing model usually leads to powerful concrete attacks.

We use the definitions of algorithms as circuits to define formally the notion of private circuit (which is equivalent to security in the d-probing model [56]). We also define the notion of gadget, which is a masked circuit that implements a simple function such as a group operation.

Definition 1.1 (Private circuit [37])**.** *A private circuit for $f : \mathbb{F}_q^n \to \mathbb{F}_q^m$ is defined by a triple (I, C, O), where*

- *$I : \mathbb{F}_q^n \to \mathbb{F}_q^{n'}$ is a randomized circuit with uniform randomness ρ and called input encoder;*
- *C is a randomized circuit with input in $\mathbb{F}_q^{n'}$, output in $\mathbb{F}_q^{m'}$, and uniform randomness $r \in \mathbb{F}_q^{\alpha}$;*
- *$O : \mathbb{F}_q^{m'} \to \mathbb{F}_q^m$ is a circuit, called output decoder.*

[4] We work only in Galois fields of characteristic 2 (mostly $\mathbb{F}_2$ but also sometimes $\mathbb{F}_q$), and we denote the field operations by $+$ and $\cdot$ (those are `XOR` and `AND` in $\mathbb{F}_2$). We have thus $x = x_0 + \cdots + x_d$.

We say that C is a d-private implementation of f with encoder I and decoder O if the following requirements hold:

- *Correctness: for any input $w \in \mathbb{F}_q^n$, $\Pr[O(C(I(w;\rho);r)) = f(w)] = 1$, where the probability is over the randomness ρ and r;*
- *Privacy: for any $w, w' \in \mathbb{F}_q^n$ and any set P of d wires in C, the distributions $\{C_P(I(w;\rho);r)\}_{\rho,r}$ and $\{C_P(I(w';\rho);r)\}_{\rho,r}$ are identical, with $C_P(I(w;\rho);r)$ the list of the d values on the wires from P.*

From now on, we assume that I and O are the canonical encoder and decoder: I encodes each input b by a block $(b_j)_{0\leq j\leq d}$ of $d+1$ random values with sum b, and O takes the sum of each block of $d+1$ values (this restriction corresponds to the case of additive masking). Each block $(b_j)_{0\leq j\leq d}$ is called a sharing of b and each b_j is called a share of b.

A gadget C implementing a function $f : \mathbb{F}_q^n \to \mathbb{F}_q^m$ is a private implementation of f working with the canonical encoder and decoder. The gadget has n inputs and m outputs, each of which is a vector of $d+1$ shares. We usually denote the inputs of a gadget as $x_{i,j}$ where $i \in \{1,\ldots,n\}$ is the input index and $j \in \{0,\ldots,d\}$ is the share index. Outputs are usually written as $y_{i,j}$, $i \in \{1,\ldots,m\}$. A set A is a set of share indices if $A \subset \{0,\ldots,d\}$. We use the notations $x_{i,A} = \{x_{i,j} : j \in A\}$, $x_{*,A} = \{x_{i,j} : 1 \leq i \leq n, j \in A\}$ and $x_{*,*} = \{x_{i,j} : 1 \leq i \leq n, 0 \leq j \leq d\}$. When it is not clear from the context, we explicitly denote the gadget G to which the inputs or the outputs are related: $x_{i,j}^G$, $y_{i,j}^G$.

The wires in the set P used by the attacker are called the probes and the corresponding values $C_P(I(w;\rho);r)$ the values of the probes. Abusing notation, a probe p is sometimes used to denote the corresponding value.

1.3.2. Composability

It is desirable to have a few elementary gadgets (field operations, etc.) that can then be connected together in order to implement more complex functions such as S-boxes or block ciphers. The resulting composite circuits should indeed also be probing secure. It has however been shown that composing probing secure circuit does not guarantee probing security for the composite circuit [18].

Two solutions have then been introduced to mitigate this issue. A first solution is based on formal tools enabling the automated analysis of implementations, up to certain number of shares and for limited circuit complexity [6]. The second approach (the one we use) is based on the more demanding definitions of Non-Interference (NI) and Strong Non-Interference (SNI) that can be proven generically and guarantee secure composability for any number of shares [5].

Composition We first formally define our notion of a composition of gadgets that implements a composite function.

Definition 1.2 (Composite function)**.** *A function f from $\mathbb{F}_q^m$ to $\mathbb{F}_q^n$ is a sequential composition of functions f_κ for $\kappa = 1,\ldots,\ell$ if the computation of $(y_1,\ldots,y_n) = f(x_1,\ldots,x_m)$*

can be done using the following algorithm:

$$\begin{aligned}(a_{o_{0,1}}, \ldots, a_{o_{0,m}}) &= (x_1, \ldots, x_m) \\ (a_{o_{\kappa,1}}, \ldots, a_{o_{\kappa,n_\kappa}}) &= f_\kappa(a_{i_{\kappa,1}}, \ldots, a_{i_{\kappa,m_\kappa}}) \qquad \text{for } \kappa = 1, \ldots, \ell \\ (y_1, \ldots, y_n) &= (a_{i_{\ell+1,1}}, \ldots, a_{i_{\ell+1,n}})\end{aligned}$$

for some set of connection indices $i_{\kappa,p}$ and $o_{\kappa,p}$.

The functions f_κ are called composing functions*.*

This definition is different of the mathematical composition of functions $f_\ell \circ \cdots \circ f_1$ in that all the outputs of f_1 are not necessarily inputs of f_2: an output of f_1 may (for example) be an input of f_3 and f_4 (it is thus also possible to "re-use" values).

Definition 1.3 (Composite gadget)**.** *Let f be a composite function of functions f_κ for $\kappa = 1, \ldots, \ell$ with connections indices $i_{\kappa,p}$ and $o_{i,p}$. A composite gadget G over $d+1$ shares that implements f is made of gadgets G_j over $d+1$ shares that implement the functions f_κ.*

The connection of the composing gadgets is done as follows, for an evaluation of $(y_1, \ldots, y_n) \leftarrow G(x_1, \ldots, x_m)$ (where the x_i's and y_i's are elements of $\mathbb{F}_q^l$):

$$\begin{aligned}(a_{o_{0,1}}, \ldots, a_{o_{0,m}}) &\leftarrow (x_1, \ldots, x_m) \\ (a_{o_{\kappa,1}}, \ldots, a_{o_{\kappa,n_\kappa}}) &\leftarrow G_\kappa(a_{i_{\kappa,1}}, \ldots, a_{i_{\kappa,m_\kappa}}) \qquad \text{for } \kappa = 1, \ldots, \ell \\ (y_1, \ldots, y_n) &\leftarrow (a_{i_{\ell+1,1}}, \ldots, a_{i_{\ell+1,n}})\end{aligned}$$

where $a_\kappa \in \mathbb{F}_q^l$.

The gadgets G_κ are called composing gadgets*.*

This general definition can be particularized to two interesting cases: the serial composition, where the outputs of the function f_κ are only connected to inputs of the function $f_{\kappa+1}$ (this is thus a mathematical composition $f_\ell \circ \cdots \circ f_1$). The other interesting case in the parallel composition: the composing functions are not interconnected, each input of a composing function is an input of the composite function (the same goes for the outputs).

Composable security properties Since probing secure composing gadgets are not sufficient to guarantee the probing security of a composite gadget (except in the restricted case of parallel composition [8]), we need stronger security definitions. In order to introduce those definitions, we use the simulability framework put forward in [10]. Note that the notion of $(\mathcal{I}, \mathcal{O})$-Non-Interference introduced in [5] is equivalent.

Intuitively, a set of probes in a circuit is simulable using a subset of the input wires of the circuit if, for any value of the inputs, a simulator that has only access to the subset of the inputs can generate values which have the same distribution as the actual probes. The distribution is over the randomness used in the circuit (and in the simulator).

Definition 1.4 (Simulability [10])**.** *Let $P = \{p_1, \ldots, p_l\}$ be a set of l probes of a gadget C. Let $I = \{(i_1, j_1), \ldots, (i_k, j_k)\} \subset \{1, \ldots, n\} \times \{0, \ldots, d\}$ be a set of input wires of C.*

A simulator is a random function $\mathcal{S} : \mathbb{F}_q^k \to \mathbb{F}_q^l$. A distinguisher is a random function $\mathcal{D} : \mathbb{F}_q^l \times \mathbb{F}_q^{n(d+1)} \to \{0, 1\}$.

The set of probes P can be simulated *with the set of input wires I if and only if there exists a simulator $\mathcal{S}$ such that for any distinguisher $\mathcal{D}$ and any inputs $x_{*,*}$, we have*

$$\Pr[\mathcal{D}(C_P(x_{*,*}), x_{*,*}) = 1] = \Pr[\mathcal{D}(\mathcal{S}(x_{i_1,j_1}, \ldots, x_{i_k,j_k}), x_{*,*}) = 1],$$

where the probability is over the random coins in C, $\mathcal{S}$ and $\mathcal{D}$.

We can now define Non-Interfering (NI) gadgets, Tight Non-Interfering (TNI) gadgets and Strong Non-Interfering (SNI) gadgets. We again take the definitions from [10] and denote output probes on a gadget as probes on shares of outputs of the gadget, and internal probes as probes on any wire of the gadget including inputs and outputs.

Definition 1.5. *A gadget is d-NI if and only if every set of at most d internal probes can be simulated with at most d shares of each input.*

NI is a sufficient condition for probing security: a circuit C is d-probing secure if any set of probes P of size d can be simulated with d shares of each input. It is not a necessary condition for probing security, because the distinguisher has access to the input shares which sometimes makes the simulation of probing secure gadgets impossible (e.g., in first-order threshold implementations where non-linear gadgets leverage the input shares in order to reduce the randomness requirements [49]).

The notion of TNI looks a bit stronger than NI but is in fact equivalent to NI (Property 1.1, proved in [10]). Thanks to this property, we can use the TNI definition when we discuss NI gadgets.

Definition 1.6. *A gadget is d-TNI if and only if every set of $t \leq d$ internal probes can be simulated with at most t shares of each input.*

Property 1.1 (d-NI $\Leftrightarrow$ d-TNI)**.** *A gadget is d-NI if and only if it is d-TNI.*

It can be easily proven that the serial composition of NI/TNI composing gadget with only one output is a NI/TNI composite gadget. The same property applies for parallel composition. The probe propagation framework discussed in Section 2.1.1 gives a simple intuition for these properties which are consequences of Property 2.6.

The NI property is however not sufficient for generic composition, which leads to the definition of SNI. The difference with NI is that in the SNI simulation game, the distinguisher has access to output probes "for free": an output probe does not "give access" to the simulator to one input share.

Definition 1.7. *A gadget is d-SNI if and only for if every set $\mathcal{I}$ of t_1 internal probes and every set $\mathcal{O}$ of t_2 output probes such that $t_1 + t_2 \leq d$, the set $\mathcal{I} \cup \mathcal{O}$ of probes can be simulated with t_1 shares of each input.*

Any composite gadget made exclusively of SNI sub-gadgets with only one output (and in which two inputs of a gadget are never connected to the same wires) is itself SNI. This is a consequence of Property 2.7 and it can again be more easily understood in the probe propagation framework of Section 2.1.1.

1.3.3. Well-Known Elementary Gadgets

There are many designs of gadgets that implement elementary field operations and are NI or SNI. The most studied ones are linear gadgets, NI and SNI field multiplication and so-called SNI refresh gadgets (which implement the identity function in a SNI fashion).

Linear gadgets For any linear function f, there is a *trivial implementation* which requires no random gates and consists in applying the function independently to each share: $y_{*,j} = f(x_{*,j})$ for $j = 0, \ldots, d$.

Such an implementation is always NI: the simulator can use the $x_{*,j}$ values for all the j's for which there is a probe in the evaluation of $f(x_{*,j})$. It is however not SNI: to simulate an output share $y_{i,j}$, a simulator needs to have access to the corresponding input shares $x_{*,j}$. Particular cases of interest are the group addition and affine functions (e.g. the `Not` gate)[5] Due to its simplicity and its low cost (no randomness use, $\Theta(d)$ scaling), we will always use the trivial implementation for linear and affine functions.

Field multiplication gadgets In the following, we will consider the multiplication of Ishai, Sahai and Wagner (which we call the ISW multiplication) as a SNI multiplication that is proven secure at arbitrary orders and has randomness cost $d(d+1)/2$ [37] and the multiplication of Belaid et al. as a NI multiplication that is proven secure at arbitrary orders and has randomness cost $\lfloor d^2/4 \rfloor + d$ [10]. The motivation for using randomness as the main cost metric is discussed in Section 1.6.

As an illustration of the implementation a masked gadget, we next describe the ISW multiplication (Algorithm 1). For shared inputs $(x_i)_{i=0,\ldots,d}$ and $(y_j)_{j=0,\ldots,d}$, the gadget computes all the products $x_i \cdot y_j$, which constitutes a $(d+1)^2$-sharing of the output. There is then a compression step to reduce the $(d+1)^2$-sharing into a $(d+1)$-sharing. This step uses random bits to ensure probing security.

Refresh gadgets The SNI refresh gadgets (hereafter "refresh") are sometimes useful to ensure secure composition. For example, it can be proven that refreshing (i.e. adding a refresh to) each of the outputs of a NI gadget makes it SNI. The refresh of Battistello et al. is an example of SNI refresh that is proven secure at arbitrary orders and has randomness complexity in $\mathcal{O}(d \log d)$ [7].

[5] Affine functions also have a simple NI implementation, although they are not linear: for the `Not` function (respectively for the addition of a constant), the `Not` (resp. addition) is applied to the first share.

Algorithm 1 ISW multiplication gadget

Require: shared factors $x, y \in \mathbb{F}_q^{d+1}$ such that $\sum_i x_i = x^*$ and $\sum_i y_i = y^*$
Ensure: output $c \in \mathbb{F}_q^{d+1}$ such that $\sum_i c_i = x^* \cdot y^*$
  for $i = 0$ to d **do**
    for $j = i + 1$ to d **do**
      $r_{ij} \xleftarrow{\$} \mathbb{F}_q$
      $z_{ij} \leftarrow (r_{ij} + x_i \cdot y_j) + x_j \cdot y_i$
      $z_{ji} \leftarrow r_{ij}$;
    end for
  end for
  for $i = 0$ to d **do**
    $c_i \leftarrow x_i \cdot y_i + \sum_{j=0, j\neq i}^{d} z_{ij}$
  end for

1.4. Noisy Leakage Model

The threshold probing model allows for simple proofs of security of gadgets. It is however not a realistic model of attacks where the adversary collects a whole leakage trace (a time series of measurements of the physical value of interest). A more realistic assumption is that the trace is noisy, which comes from the observation that measurements are inherently noisy in the real world, and from the strategies of protection against side-channel attacks that try to reduce the signal-to-noise ratio (SNR) of the measurements [42]. Another model, the noisy leakage model [54], is thus required to formalize those attacks.

The noisy leakage model does not attempt to accurately model the leakage trace in itself (that would be a complex task), it is rather a simple bound on the information that can be extracted from the trace. Let $X = (x_1, \dots x_l)$ be the vector of all variables manipulated by the implementation, the noisy leakage model gives to the adversary access to $(\mathcal{X}_1, \dots, \mathcal{X}_l)$, where $\mathcal{X}_i \leftarrow \phi_i(x_i)$. The random functions ϕ_i can be chosen by the adversary, under two constraints:

- The mutual information[6] is bounded: $\mathrm{MI}(\mathcal{X}_i, x_i) < \epsilon$, where ϵ is a parameter of the model.[7]
- The independence of leakage on variables: the different ϕ_i function are mutually independent.

The validity of those assumptions is discussed next, along with the practical use of the noisy leakage model.

[6] The mutual information (MI) between two random variables A and B is

$$\mathrm{MI}(A, B) = \sum_{a \in R_A} \sum_{b \in R_B} \Pr[A = a, B = b] \log_2 \left(\frac{\Pr[A = a, B = b]}{\Pr[A = a] \Pr[B = b]} \right).$$

where R_A and R_B are the sets of possible values of A and B.

[7] The original noisy leakage model uses a bound on the statistical distance [54], but it has been shown that it is equivalent to a MI bound [22].

Multi-model approach In the current state-of-the-art, proving security in the noisy leakage model is generally achieved in three main steps. First, the circuit is analyzed in the abstract d-probing model. The latter is instrumental in detecting composability flaws due to a lack of refreshing [18]. Concretely, such flaws can be avoided by checking the implementations in order to determine where refreshing gadgets have to be introduced [6], or by using composable gadgets [5].

Next, the circuit is analyzed in the qualitative bounded moment model of Barthe et al. [4], in order to determine the extent to which physical defaults such as glitches re-combine the shares and reduce the security guarantees [44]. Concretely, such flaws can be avoided both by constraining the algorithmic design of the masking schemes (e.g., by using the non-completeness property of threshold implementations [49], which can be analyzed by extending the probing model to capture physical defaults [24]), or by mitigating the problem directly at the hardware level [47].

Third and finally, the circuit is analyzed in the noisy leakage model. First, the leakage of each share has to be sufficiently noisy, which is purely a hardware assumption that has to be verified/falsified empirically [38]. Second, the independence of the leakages of different variables is guaranteed (at least to a sufficient extent) by the analysis in the bounded moment model.

This multi-model approach is theoretically validated by the work of Duc et al., who showed that under the aforementioned noise and independence conditions, security in the noisy leakage model is implied by security in the probing model [21].[8] This approach imposes however strong requirements on the mutual information of the leakage (i.e. on the noise in the device), which can be hard to meet at the physical level. In the following, we first present an attack that shows what happens when the noise level is not sufficient, and then we introduce a model that allows to analyze this attack and to design countermeasures.

1.5. Horizontal Attacks

Classical side-channel attacks for masked implementations use a simple technique: they observe leakage on the different shares of one variable, and then combine these observations to infer the value of the sensitive variable. This can be improved in various ways, which leads to the family of the horizontal attacks [7, 33].

The optimal approach is the Bayesian approach. Let $\mathcal{L}$ be the leakage trace and K the sensitive variable, this approach computes $\Pr[K = k|\mathcal{L}]$ for all possible values k of K (and, for example, takes the one with the maximum probability) using the Bayes rule:

$$\begin{aligned}\Pr[K = k|\mathcal{L}] &= \frac{\Pr[\mathcal{L}|K = k]\Pr[K = k]}{\Pr[\mathcal{L}]} \\ &= \frac{(\sum_x \Pr[\mathcal{L}|X = x]\Pr[X = x|K = k])\Pr[K = k]}{\Pr[\mathcal{L}]}\end{aligned}$$

[8] More precisely, the success rate of an adversary trying to distinguish between two different inputs for the circuit decreases exponentially with the masking order d.

where X is the set of intermediate values. The computational cost of using this formula is exponential in $|X|$, the size of the circuit[9] and makes the corresponding attack impractical for any circuit of significant size.

1.5.1. Soft Analytical Side-Channel Attack

The soft analytical side-channel attack (SASCA) is introduced in [58]; it uses the tools of coding theory. Indeed, decoding an error-correcting code is a problem similar to carrying out a side-channel attack: recover the value of variables when noisy and redundant information is given. In both cases, Bayes formula is optimal but cannot be used due to its computational cost. In a nutshell, the SASCA models the circuit under attack as a code whose inputs are the sensitive variables and the outputs are all the intermediate variables, and then runs the well-known belief propagation algorithm (BP) to perform the decoding.

Factor graph and BP algorithm SASCA are based on the construction of a factor graph built from a set of relationships between intermediate values. For the masked implementations of block ciphers, we typically consider three kinds of relationships: sums: $x = x_1 + \cdots + x_k$, products of two elements: $x = x_1 \cdot x_2$ and bijections: $y = g(x)$ where g is a bijection. The factor graph is a bipartite graph made of variable nodes (one for each intermediate result within the leaking implementation) and function nodes (one for each relationship), with an edge if an intermediate result appears in a relationship. Furthermore, to each variable node is associated an intrinsic information — an estimated probability density function (PDF) — which represents the leakage that can be observed on the corresponding intermediate result.

The principle of the BP algorithm is to exploit the relationships between variables in order to constraint the PDF estimates: estimates based only on the leakage are usually not coherent with the relationships. The algorithm works by sending messages (PDF estimates) on the edges of the factor graph. Those messages are called extrinsic information. The BP algorithm alternates two steps until the algorithm converges. The first step sends messages from variable nodes to function nodes, and the second step sends messages from function nodes to variable nodes. The message sent by a variable node to a function node is a combination of the intrinsic information and the extrinsic information coming from function nodes at the previous step, for all the function nodes the variable nodes is connected to, except the function node that is the destination of the message being computed.[10] Likewise, the message sent by a function node to a variable node is a combination of all the incoming messages to the function node, except the message coming from the destination variable node (the actual combination depends on the relationship represented by the function node, and the position — as operand or as result — of the variable). Once the algorithm has converged, the intrinsic and extrinsic

[9] That can be reduced to the number of random values used in the circuit, but it does not change the conclusion: the attack is not feasible.

[10] All messages are initialized to an a priori (e.g., uniform) PDF.

information at each node is combined to get the final estimated PDF. We refer to [58] for the details.

The next part of the attack depends on the context. For example, if the goal is to recover the key of a block cipher, key enumeration techniques can be used [59].

1.5.2. Theoretical Analysis

From a theoretical viewpoint, the performance of horizontal attacks is captured by the concept of "*noise rate*", which essentially corresponds to the level of noise increase (or shares' information reduction) that is required so that masking still provides an exponential security improvement. The noise rate of Ishai et al.'s original multiplication algorithm is known to be in $\mathcal{O}(1/d)$ [37, 54, 21, 22]. Recent works by Andrychowicz et al. and Goudarzi et al. have shown that it is possible to reach noise rates in $\mathcal{O}(1/\log(d))$ or even $\mathcal{O}(1)$ [1, 30]. Yet, the latter results rely working in larger fields (typically such that $|\mathbb{F}|$ is in $o(d)$).

The noise rate has two issues: first, it is often used as an asymptotic tool, which does not give the actual security level of a circuit. Second, and more importantly, it is difficult to compute it on algorithms that include countermeasures against horizontal attacks (for instance, the countermeasure of Battistello et al. [7] is only based on a heuristic argument).

The local random probing model (LRPM) has been recently [35] introduced as a way to remedy these drawbacks.

Local Random Probing Model The Local Random Probing Model (LRPM) [35] is a way to model the SASCA. It is a variation of the Random Probing Model (RPM) by Duc et al. [21].[11] Compared to the noisy leakage model, the LRPM restricts the way the information leakage of an implementation is exploited to local propagation rules inspired by the belief propagation algorithm, which allows simple concrete evaluations of actual multiplication algorithms (or even more complex circuits).

The main difference between the BP used by the SASCA and the LRPM is that the messages are no longer PDF estimates, but mutual information values that represent the amount of information contained in the PDF estimates. A first consequence is that the intrinsic information is not a PDF, but the leakage observed on the variable. The second consequence is a modification of the rules to compose messages [35]:

- For messages of the first step (from variables to functions): the message sent is the sum of the intrinsic and the extrinsic information, limited to $\mathrm{MI} = 1$.[12] This follows from the fact that combining estimates for a variable at most sums the information of the estimates.

[11] The RPM is not discussed in detail in this thesis, however there is a short explanation of it in Appendix A.

[12] The limitation comes from the fact that $\mathrm{MI} = 1$ implies no uncertainty on the leaking variable, if the MI unit is the field element.

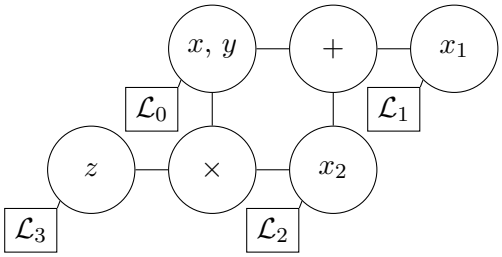


Figure 1.4.: Factor graph with six nodes. $\mathcal{L}$ labels indicate intrinsic information.

- For messages of the second step (from functions to variables): using the random probing model, it is shown in [35] that the information on the result of a sum or on the operand of any operation (sum or product) is bounded by the product of the information sent by the other concerned variables.

  Another case can happen though, which was not discussed in [35] (since they do not use the result of multiplications in other operations). Namely, the information on the result of a multiplication. We discuss it in Appendix A. Simply stated, in our case (where we multiply two elements in $\mathbb{F}_2$), the resulting MI is the average of the MI on the operands.

- The case of bijection relationship is special: since MI on one of the variables related by a bijection translates into information on the other variable, the two nodes can be merged and the bijection function node can be removed.

- The final combination of information (once the algorithm has converged) is a sum of all the intrinsic and extrinsic MI for each variable node.

For illustration, we next give an example of factor graph in Figure 1.4. The corresponding equations for LRPM are the following:

$$x = \mathsf{g}(y), \qquad x = x_1 + x_2, \qquad z = x_2 \cdot y,$$

The messages sent at the first step of the BP algorithm are:

$$\begin{aligned}
\mathrm{MI}_{x,y\to+} &\leftarrow \min\left(1, \mathcal{L}_0 + \mathrm{MI}_{\times\to x,y}\right), & \mathrm{MI}_{x,y\to\times} &\leftarrow \min\left(1, \mathcal{L}_0 + \mathrm{MI}_{+\to x,y}\right),\\
\mathrm{MI}_{x_1\to+} &\leftarrow \min\left(1, \mathcal{L}_1\right) = \mathcal{L}_1, & \mathrm{MI}_{x_2\to\times} &\leftarrow \min\left(1, \mathcal{L}_2 + \mathrm{MI}_{+\to x_2}\right),\\
\mathrm{MI}_{x_2\to+} &\leftarrow \min\left(1, \mathcal{L}_2 + \mathrm{MI}_{\times\to x_2}\right), & \mathrm{MI}_{z\to\times} &\leftarrow \min\left(1, \mathcal{L}_3\right) = \mathcal{L}_3,
\end{aligned}$$

and those sent at the second step are:

$$\begin{aligned}
\mathrm{MI}_{+\to x,y} &\leftarrow \mathrm{MI}_{x_1\to+} \cdot \mathrm{MI}_{x_2\to+}, & \mathrm{MI}_{\times\to x,y} &\leftarrow \mathrm{MI}_{x_2\to\times} \cdot \mathrm{MI}_{z\to\times},\\
\mathrm{MI}_{+\to x_1} &\leftarrow \mathrm{MI}_{x_2\to+} \cdot \mathrm{MI}_{x,y\to+}, & \mathrm{MI}_{\times\to x_2} &\leftarrow \mathrm{MI}_{x,y\to\times} \cdot \mathrm{MI}_{z\to\times},\\
\mathrm{MI}_{+\to x_2} &\leftarrow \mathrm{MI}_{x,y\to+} \cdot \mathrm{MI}_{x_1\to+}, & \mathrm{MI}_{\times\to z} &\leftarrow \left(\mathrm{MI}_{x,y\to\times} + \mathrm{MI}_{x_2\to\times}\right)/2.
\end{aligned}$$

The two steps are alternated until the algorithm converges.

Finally, we note that the LRPM is inherently slightly heuristic, as it uses the BP algorithm. Since SASCA are among the (if not the) most efficient way to perform horizontal attacks in the current state-of-the-art [33], the bounds on the information that can be extracted thanks to SASCA (provided by the LRPM) can be viewed as a good approximation of the wort-case security level in the noisy leakage model.

1.6. Implementation and Performance

Most block cipher algorithms can be expressed as circuits with values in $\mathbb{F}_2$ (the bitslice implementation). The masking can thus operate in $\mathbb{F}_2$: shares are in $\mathbb{F}_2$ and addition is `XOR`. The AES is a particular case: it has also a very natural and efficient representation in $\mathbb{F}_{256}$, and masking with shares in $\mathbb{F}_{256}$ has thus been investigated. It has however been shown that this masked implementation in $\mathbb{F}_{256}$ is computationally more expensive than the bitslice implementations [31], on which we will focus.

Masking implies significant overheads in terms of time and randomness complexity (usually quadratic in the number of shares [34]). Concretely, it has been observed that the randomness complexity dominates as the number of shares in the masking schemes increases [31, 38], due to the cost of (pseudo-)randomness generation; we therefore use the randomness complexity as our main cost metric in the following. As a result, significant efforts have been devoted to the reduction of the randomness complexity of masked multiplications [10, 9, 23].

2. Composability and Performance Improvements in the Probing Model

In this chapter, we start from the observation that the composability properties of NI and SNI gadgets are not trivial and that they have previously only been used and analyzed in simple contexts: one AES S-box (implemented in $\mathbb{F}_{256}$, hence single input and single output), multiplication and addition gadgets (two inputs, one output). In particular, the bitslice implementations produce more complex gadgets (bitslice S-box are multi-input and multi-output). Our contributions in this respect are threefold:

First, we introduce a definition of Multiple-Input Multiple-Output Strong Non-Interference (MIMO-SNI), which extends the existing definition of SNI to the bitslice case. The latter is instrumental to formally analyze a compositional strategy proposed in [31] and re-used in [38], which is to use only SNI multiplications and to systematically refresh one of their inputs with a SNI gadget. We next call it the "greedy strategy" due to its high randomness requirements.

Second, we show how the greedy strategy can be optimized by representing the circuit to mask as a "computation graph" and describing how to express the definitions of NI, SNI and MIMO-SNI as graph properties. For simple circuits (such as the AES S-box operating in $\mathbb{F}_{256}$), the number of solutions is sufficiently small for exhaustive search (and we confirm the recent results of [10]). For more complex circuits, exhaustive analysis is impossible thus we rely on integer programming. As an illustration, we launch our optimization on the bitslice S-box of Boyar, Matthews and Peralta [12] that is used in [31, 38] and can reduce the number of SNI gadgets to 41 (with a lower bound of 34) compared to the 64 of the greedy strategy. In view of the significant (asymptotically dominating) impact of these SNI gadgets in the overall performances of a masked AES implementation, it directly leads to comparable performance gains.

Third, we observe that the definition of MIMO-SNI is still theoretically (over)demanding. We suggest an alternative approach based on "probe isolation" which rather ensures that such paths cannot be exploited by an adversary (by verifying that the propagated probes on these paths can be simulated with the same input shares). The latter Probe-Isolating Non-Interference (PINI) enables a major simplification of the security analyzes of complex masked circuits, since it allows the composition of any linear gadget with PINI multiplications. We also use this definition to exhibit concrete performance improvements, by proposing and proving two PINI multiplication gadgets, which reduce the randomness requirements of the previous optimized bitslice S-box mixing SNI multiplications and refreshes.

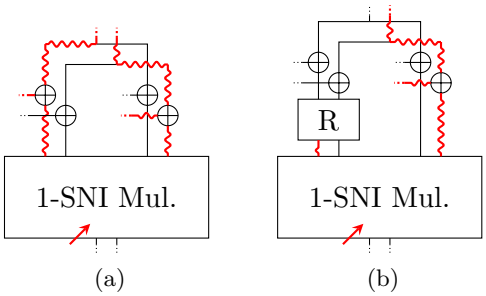


Figure 2.1.: Implementation of $(x+y)(x+z)$. The circuit is made of a SNI multiplication with linear circuits at the input, masked at order $d = 1$. The circuits illustrate (a) the limitation of SNI input composability and (b) the greedy strategy. The arrows indicate the adversarial probes (there is thus one internal probe in the multiplication) and the red snake wires are the propagated probes. The R box is a SNI refresh.

2.1. Composition of Bitslice Gadgets

The previous definitions of NI and SNI gadgets (Section 1.3.2) have been shown to be very effective to avoid compositional issues such as put forward in [18]. For example, the specialization of the $(\mathcal{I}, \mathcal{O})$-Non-Interference they provide is perfectly suited to analyze the amount of refreshing required to implement an AES S-box with operations in $\mathbb{F}_{256}$ [10]. Yet, one possible limitation of these abstractions is that they implicitly assume gadgets with single inputs and single outputs. In this section, we first argue that such definitions are therefore not sufficient to capture the security of certain composite gadgets such as encountered in masked bitslice implementations. We then propose a new definition of Multiple-Input Multiple-Output (MIMO) SNI gadgets as a useful ingredient to analyze the security of any composite gadget. We finally put forward that secure composite gadgets are naturally obtained by the proposal in [31] of using (only) SNI multiplications with one input refreshed in a SNI manner (although we will use an additional ingredient for the full proof of this fact, discussed in Section 2.3.4).

2.1.1. Simulation Framework and Greedy Strategy

We start by providing intuition about the simulation framework we use and the greedy strategy with the simple circuit example of Figure 2.1a which performs a multiplication of dependent values (masked at order $d = 1$). There is one adversarial (internal) probe in the SNI multiplication gadget, and we will try to prove that the probe is not an attack (i.e., it is independent of the actual inputs) by demonstrating that it is possible to simulate it using at most one share of each of the inputs.[1]

[1] Note that this does not prove that the circuit is 1-probing-secure. Proving the probing security would require to analyze all the possible sets of probes. A more efficient way of making that proof is discussed in Sections 2.1.3 and 2.2.

According to the SNI definition, it is possible to perfectly simulate the adversarial probe by knowing one share of each of the inputs of the SNI multiplication. Let those required shares be the red snake wires in the circuit (the set of wires shown is an arbitrary example, the shares required by the simulator depend of course on the position of the adversarial probe). Those wires are called *propagated probes*: if it is possible to simulate the propagated probes, then the adversarial probe can be simulated. We can propagate the probes one step further: a probe at the output of an addition can be simulated with probes on the two inputs of the addition. This gives four propagated probes at the input of the circuit, which probes all the shares of one of the inputs. Because of that, we cannot prove that the circuit is probing secure.

The circuit of Figure 2.1b has the same functionality as the circuit of Figure 2.1a, but is implemented using the greedy strategy: there is a SNI refresh gadget on one of the inputs of the multiplication gadget. The propagated probe at the output of the refresh gadget can be simulated using no input of the gadget (thanks to the SNI property), which makes the circuit 1-NI (and thus 1-probing secure).

The technique of proof used in this section is similar to the one used in [10] to prove the security of the implementation of a masked AES S-box in $\mathbb{F}_{256}$. We call this technique *probe propagation*: it is based on the idea of replacing adversarial probes with propagated probes that can be used to simulate the adversarial probes, and then iterating the process until the propagated probes are all at the inputs of the circuit. The conclusion is then easy.

The propagation of probes happens backwards in the circuit (probes on the outputs of a gadget propagate into probes on the inputs of the gadget). The definitions of NI and SNI can be expressed as the following rules in the probe propagation framework.

Probe propagation rules:

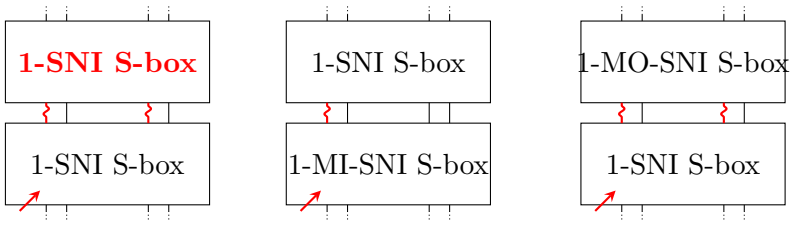
- For a NI gadget with n_o probes on shares of its output[2] and n_i probes inside the gadget, there is a propagated probe on $n_o + n_i$ shares of each input (this comes from Property 1.1).
- For a SNI gadget with n_o probes on output shares and n_i probes inside the gadget, there is a propagated probe on n_i shares of each input. Hence, the SNI gadgets (and in particular SNI refresh) stop the propagation of probes.

There are then two probe propagation conditions that guarantee security against the considered adversarial probes.

Probe propagation security conditions:

1. For any NI or SNI gadget, the number of output probes must be at most d. This follows from the definitions of NI and SNI.

[2] We only consider here the gadgets with one output such as a multiplication. Multiple output NI gadgets (such as bitslice S-box) are not investigated since the stronger SNI property is itself is not sufficient for security of a block cipher implementation.



(a) SNI S-boxes: not secure (b) First solution: MI-SNI S-boxes (c) Second solution: MO-SNI S-boxes

Figure 2.2.: First problem: serial composition of two S-boxes. Example for $d = 1$ and 2-bit S-boxes. The arrows indicate the adversarial probes and the red snake wires are the propagated probes. Red boldface label for the S-box indicates that the first probe propagation security condition is not satisfied.

2. For any input of the circuit, there cannot be propagated probes on all the $d + 1$ shares.

2.1.2. SNI is Not Enough (e.g., for Bitslice Implementations)

Say we are now interested in the situation where a non-linear gadget has multiple inputs and multiple outputs, such as an S-box with a bitslice implementation. To simplify the discussion, we take the case of 2-bit S-boxes (i.e., each S-box has two inputs and two outputs) masked at order $d = 1$, but our reasoning applies to any size of S-boxes (such as the 8 bit S-boxes of the AES) and any order.

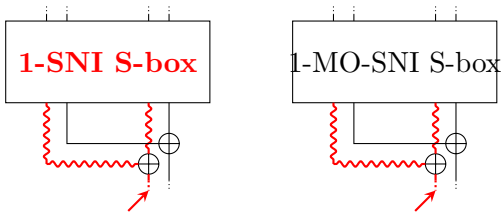
A first example of this context is the serial composition of two S-boxes (in which each of the outputs of the first S-box is connected to one input of the second S-box), depicted in Figure 2.2a.

This leads to the following problem: if the S-boxes are d-SNI and the adversary has d probes in the second S-box (remember $d = 1$ in our example), the propagated probes can cover up to d shares of each of the inputs of this S-box. There is thus a total of up to $2d$ propagated probes on outputs of the first S-box. Since the number of probes discussed in the first probe propagation security condition is the total number of probes on output shares, the condition is violated in this example.

Next, the situation gets even worse when linear gadgets come into play. A practical example is the AES S-boxes and MixColumns operations: 4 parallel 8-bit S-boxes followed by a 32-bit linear layer, followed again by 4 parallel 8-bit S-boxes. We will however use simpler examples to explain the two additional problems that arise in this case.

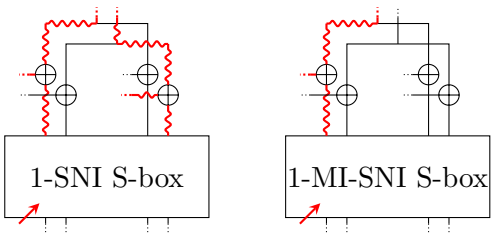
For the second problem, we consider a linear operation between two outputs of one S-box (depicted in Figure 2.3a). The adversary has d probes on one output of the linear operation. The probes propagate to $2d$ probes on the output of the S-box. The first probe propagation security condition is again not respected.

The third problem depicted in Figure 2.4a is the close to the one discussed previously for SNI multiplications (Figure 2.1): a linear layer at the input of the S-box is such that



(a) SNI S-box: not secure (b) Solution: MO-SNI

Figure 2.3.: Second problem: S-box with linear layer at the output. Example for $d = 1$ and 2 bit S-box.



(a) SNI S-box: not secure (b) Solution: MI-SNI

Figure 2.4.: Third problem: S-box with linear layer at the input. Example for $d = 1$ and 2 bit S-box.

the probes propagated by the S-box propagate through the linear layer and reach all the shares of an input. This violates the second probe propagation security condition.

Summarizing, these examples show a limitation in the definition of SNI for the analysis of bitslice S-boxes that does not appear for implementations of the AES S-box in $\mathbb{F}_{256}$ (which have only one input and one output). In the following, we show how to fix this issue by adapting $(\mathcal{I}, \mathcal{O})$-Non-Interference to this slightly more general context.[3]

2.1.3. Multiple-Input Multiple-Output SNI

Intuition

Based on the previous examples, natural directions to extend the definition of SNI and ensure composability in a bitslice implementation context are twofold:

- First, require the simulator to work with at most d input shares instead of d input

[3]Not being composable does not directly imply the existence of an attack in the probing model, in particular for low security orders that may be tested exhaustively with formal methods [6]. Yet, as the number of shares increases, the sufficient condition of security that composable gadgets provide becomes increasingly useful.

shares on each input. This property is called Multiple-Input Strong Non-Interference (MI-SNI) and adds a third *probe propagation rule*:

3. For a MI-SNI gadget with n_o probes on output shares and n_i internal probes, there is a total of n_i propagated probes on the shares of the inputs.

- Second, allow the adversary to probe up to d shares of each output, instead of d output shares in total. This property is called Multiple-Output Strong Non-Interference (MO-SNI) and adds a third *probe propagation security condition* (which is the first condition adapted to the MO-SNI gadgets):

  3. For any MO-SNI gadget, the number of probed shares on each output must be at most d.

Either of these extensions (MI-SNI or MO-SNI) solves the first problem (see Figures 2.2b and 2.2c), while the second problem is only solved by MO-SNI (see Figure 2.3b) and the third problem is only solved by MI-SNI (see Figure 2.4b). As a result, and since we want composability with any composite gadget, we need both properties for S-boxes: Multiple-Input Multiple-Output Strong Non-Interference (MIMO-SNI).[4]

Formalization

We now formalize the definitions and prove that MIMO-SNI enjoys a useful composability property.

Definition 2.1 (MI-SNI)**.** *A gadget is d-MI-SNI if and only for if every set $\mathcal{I}$ of t_1* ***internal probes*** *and every set $\mathcal{O}$ of t_2* ***output probes*** *such that $t_1 + t_2 \leq d$, the set $\mathcal{I} \cup \mathcal{O}$ of probes can be simulated* ***with at most*** *t_1* ***input shares****.*

Definition 2.2 (MO-SNI)**.** *Let O_i be a set of share indices for $i = 1, \ldots, m$. A gadget is d-MIMO-SNI if and only if any set $\mathcal{I}$ of t_1* ***internal probes*** *and any sets O_i such that there exists a t_2* ***that satisfies*** *$t_1 + t_2 \leq d$* ***and*** *$|O_i| \leq t_2$* ***for*** *$i = 1, \ldots, m$, the set of probes $\mathcal{I} \cup y_{1,O_1} \cup \cdots \cup y_{m,O_m}$ can be simulated* ***with at most*** *t_1* ***shares of each input****.*

MIMO-SNI can be defined as satisfying both MI-SNI and MO-SNI. For completeness, we give the equivalent explicit definition:

Definition 2.3 (MIMO-SNI)**.** *Let O_i be a set of share indices for $i = 1, \ldots, m$. A gadget is d-MIMO-SNI if and only if any set $\mathcal{I}$ of t_1* ***internal probes*** *and any sets O_i such that there exists a t_2* ***that satisfies*** *$t_1 + t_2 \leq d$* ***and*** *$|O_i| \leq t_2$* ***for*** *$i = 1, \ldots, m$, the set of probes $\mathcal{I} \cup y_{1,O_1} \cup \cdots \cup y_{m,O_m}$ can be simulated* ***with at most*** *t_1* ***input shares****.*

From these definitions, we derive the following composition rules, which are used to prove the main result (Property 2.5).

[4] We note that this definition can also be applied to multiplication gadgets, and that the SNI refresh & SNI multiplication composition of the greedy strategy is MIMO-SNI In fact, only the MI-SNI part is relevant since a multiplication gadget has only one output. This is not surprising since the problem solved by MI-SNI is essentially the same as the one solved by the greedy strategy (see Figures 2.1 and 2.4).

Property 2.1. *Serial composition of d-MO-SNI gadgets is d-MO-SNI.*

Proof. We make the proof for two gadgets by describing the simulator. The general case follows by induction. The probes to be simulated can be split in

- t_1 internal probes in the first gadget,
- t_2 internal probes in the second gadget,
- output probes y_{i,O_i},

such that $t_1 + t_2 + |O_i| \leq d$ for all i. The probes in the second gadget and the output probes can be simulated with t_2 shares of each of its inputs. These shares and the probes in the first gadget can be simulated with at most t_1 shares of each of its inputs. □

Property 2.2. *Parallel composition of d-MIMO-SNI gadgets is d-MIMO-SNI.*

Proof. The simulator for the composite gadget simply runs the simulator for each internal gadget. The total number of input shares that are requested is at most the number of internal probes. □

Property 2.3. *Serial composition of a linear & a d-MIMO-SNI gadget is d-MO-SNI.*

Proof. Let t_1 be the number of internal probes in the linear gadget and t_2 the number of internal probes in the MIMO-SNI gadget. The simulator uses the simulator of the MIMO-SNI gadget to simulate the internal probes of the MIMO-gadget and the output probes. It has then to simulate t_2 output shares of the linear gadget and the t_1 internal probes. Let A the set of share indices corresponding to all the values to be simulated. The simulator can request to know all the shares $x_{*,A}$ and can thus trivially simulate the linear gadget. □

Property 2.4. *Serial composition of a d-MO-SNI & a linear gadget is d-SNI.*

Proof. Let B be the set of share indices of the output probes and of the probes in the linear gadget. The simulator of the MO-SNI gadget is used to simulate its internal probes and all the output shares whose index is in B. Simulation of the remaining probes is then trivial. □

We can now prove our main composition theorem about Substitution Permutation Networks (SPN), which shows that our definitions enable to build d-SNI (hence d-probing secure) SPN. We consider only SPN whose S-boxes are grouped into layers that operate over the full state.

Property 2.5. *A SPN whose S-boxes are d-MIMO-SNI is d-SNI.*

Proof. The SPN can be viewed as a alternating serial composition of linear and S-box layers, whose first and last layers are linear (this is without loss of generality since linear layers can be the identity function). A S-box layer is a parallel composition of S-boxes.

The S-box layers are d-MIMO-SNI thanks to Property 2.2. A round, the serial composition of one linear layer followed by one S-box layer is d-MO-SNI thanks to Property 2.3. Serial composition of rounds is d-MO-SNI thanks to Property 2.1. The combination of the final linear layer with the other rounds is d-SNI thanks to Property 2.4. □

We insist that this proof is limited to SPN with full layers of d-MIMO-SNI S-boxes. It is for example not yet a proof of the greedy strategy proposed in [31], which uses MIMO-SNI multiplications within an S-box that does not have such a regular (full-layer) structure. We defer this proof to Section 2.3.4 where the introduction of "probe isolation" will allow much simplified analyses.

For a gadget with only one input, MI-SNI is equivalent to SNI and for a gadget with only one output, MO-SNI is equivalent to SNI. Hence, MIMO-SNI is equivalent to SNI for gadgets with one input and one output, such as the AES S-box implemented over $\mathbb{F}_{256}$. Property 2.5 thus applies to an AES implementation using the S-box in [10].

2.2. Optimized S-box Implementations

The previous section gave a general framework for proving the security of refreshing strategies in masked (bitslice) block cipher implementations using MIMO-SNI S-boxes. In this section, we tackle the problem of minimizing the amount of SNI gadgets in the implementation of such an S-box in order to reduce their (e.g., randomness) cost.

For this purpose, we first show how to express this optimization based on the properties of a graph describing the computations to perform. We then apply this optimization to the AES S-box in $\mathbb{F}_{256}$ (confirming the results in [10]) and to the bitslice AES S-box of Boyar, Matthews and Peralta [12], bringing significant improvements over the greedy strategy in [31].

2.2.1. Connecting Composability to Computation Graph Properties

Let a masked S-box be given as a mix of elementary operations such as additions, multiplications and refreshes. We can define a high-level computation graph modeling this S-box as a DAG whose vertices represent operations and edges represent intermediate values. The operations can take any number of incoming edges (usually one or two) and produce one outgoing edge. Besides the aforementioned field operations and refreshes, such a computation graph may include three other types of vertices:

- split vertices take one incoming edge and can produce any number of outgoing edge(s). They model multiple uses of an intermediate value;
- input vertices have no incoming edge, one outgoing edge and the edges connected to these vertices are called input edges;
- output vertices have one incoming edge, no outgoing edge and the edges connected to these vertices are called output edges.

For simplicity, we next assume that all the additions and multiplications in our computation graphs are NI, and we model SNI gadgets as NI ones followed by a SNI refresh. Given an optimized computation graph, an implementer can then replace NI multiplications followed by a SNI refresh by (sometimes more efficient) SNI multiplications. We insist that this modeling is without loss of generality since it is equivalent from the probing model point of view and the respective (e.g., randomness) costs of the different gadgets of a private circuit are parameters of the optimizations in the next subsections.

The computation graph model is a formalization of the probe propagation framework discussed in Section 2.1.1. Capitalizing on the remark that SNI refresh gadgets stop the propagation of probes, we can simple remove them (and their incident edges) from the graph to build a *simplified graph*. The probes inside the refresh gadgets can be reported to gadgets connected to their input, hence the simplified graph is equivalent to the original graph regarding security in the probing model.

Definition 2.4 (Simplified computation graph)**.** *The simplification of the computation graph G is the graph that is obtained from G by removing all SNI refresh vertices and their incident edges.*

Since there are at most d adversarial probes in the circuit, a simple security condition is that each probe propagates backwards to a single input through a single path. In this case, there will be at most d probes for each input and the second probe propagation security condition will be satisfied. Furthermore, there will be at most d probes at the output of each gadget, since the probe propagation graph is a DAG (it is the reversal of the computation graph) so that the first probe propagation security condition will be satisfied too.

We can relax this constraint and impose that no probe can propagate backwards from a node to another one through two different paths, while still satisfying the probe propagation security conditions. In other words, for any pair of vertices there should be at most one (directed) path between them.

It can be seen that the latter is a necessary condition: if probes can propagate backwards through two paths from a node A to a node B and if the adversary has d probes on the output of A, $d+1$ shares of the output of A could be required for the simulation.

We now formalize this security condition with the following properties (which generalize the proof that the AES inversion is d-SNI in [10]).

Property 2.6. *Let G be a composite gadget. If the gadget is implemented with only NI gadgets and SNI refreshes, and if for any pair of vertices u, v in the corresponding simplified computation graph there exists at most one path from u to v, then the gadget is NI.*

Proof. For each edge i in the computation graph, there is a number of adversarial probes a_i, a number of propagated probes p_i and a total number of probes s_i. The sum of the a_i's is at most d. For all i, $s_i = a_i + p_i$. For each edge, the number of propagated probes is:

- 0 if the node at the end of the edge is a refresh;
- the number of corresponding output probes if it is an output edge;
- the sum of the total number of probes of the outgoing edges of the vertex at the end of the edge otherwise (i.e., for split or NI operations).

If for each input edge i, a simulator knows s_i well-chosen shares, then it can simulate all the probes of the adversary by using the simulator for each gadget in order to get the required intermediate values.

The probes inside a NI gadget are not considered since they can equivalently be replaced with probes on output shares of the gadget.

We now prove that the hypothesis implies that for all input edges i, $s_i \leq d$. This proves that the gadget is NI thanks to the previous observation.

We use a small lemma for this purpose: for all edges i, $s_i = \sum_j \alpha_{ij} a_j$ where α_{ij} is the number of paths from the output node of i to the input node of j in the simplified computation graph. This can be proven by backwards induction on the graph: if all the children of a node satisfy this property, it is also satisfied for the node itself if the node is a refresh, split or NI operation. Input and output nodes are trivial.

The main hypothesis implies that $\alpha_{ij} \leq 1$ for all edges i and share indices j, hence $s_i \leq \sum_j a_j \leq d$. □

Property 2.7. *Let G be a composite gadget. If the gadget satisfies Property 2.6 and if for any input node u and any output node v, there is no path from u to v, then the gadget is SNI.*

Proof. Looking at the proof of Property 2.6, we observe that the coefficients $\alpha_{ij} = 0$ for all input edges i and output edges j. Hence, for all input edges i, $s_i \leq t_1$ where t_1 is the number of internal probes. □

Property 2.8. *Let G be a composite gadget. If the gadget satisfies Property 2.7 and if for any pair of output nodes u_1, u_2 there is no node v such that there is a path from v to u_1 and a path from u_2 to v, then the gadget is MO-SNI.*

Proof. We have to prove that for all edges i, $s_i \leq d$. Using the lemma from the proof of Property 2.6, we have that for any edge i, and input edges j, all but one α_{ij} are zero (i.e., $\sum_j \alpha_{ij} \leq 1$). This implies that $s_i \leq t_1 + t_2 \leq d$, taking the definitions of t_1 and t_2 from the definition of MO-SNI. For input edges i, the proof of Property 2.8 applies. □

Property 2.9. *Let G be a composite gadget. If the gadget satisfies Property 2.8 and if for any pair of input nodes u_1, u_2 there is no node v such that there is a path from v to u_1 and a path from v to u_2, then the gadget is MIMO-SNI.*

Proof. In addition to Property 2.8, we want to prove that $\sum_{i \in I_e} s_i \leq t_1$ where I_e is the set of input edges.

We know that for all j, $\sum_{i \in I_e} \alpha_{ij} \leq 1$ and for output edges j, $\sum_{i \in I_e} \alpha_{ij} = 0$. Hence, $\sum_{i \in I_e} s_i \leq \sum_{j \notin O_e} a_j = t_1$ where O_e is the set of output edges. □

2.2.2. Optimizing the AES S-box in $\mathbb{F}_{256}$

Using the previous graph formalization, we built a tool[5] that checks if a circuit is (MIMO-)SNI. If we want to build a SNI S-box with the multiplication chain from [10], there are 16 wires on which we could insert a refresh. This number is sufficiently small to make a exhaustive search, which confirms the result of [10] and shows that it is the only solution with only three refresh elements (up to the permutation of refresh gadgets with the $(\cdot)^{2^\alpha}$ power gadgets): two refresh gadgets and one SNI multiplication.[6] It also shows that two refresh gadgets is the minimum possible, even with all multiplications implemented as SNI gadgets.

2.2.3. Optimizing the Bitslice AES S-box of Boyar et al.

We now optimize the implementation of a bitslice AES S-box. We take the logic circuit by Boyar et al. in [12] and search where it requires adding SNI refresh elements to get a MIMO-SNI implementation.

The circuit is made of three parts: a top linear transformation, a middle non-linear transformation and a bottom linear transformation. Since our goal is to have a probing secure implementation of the AES, we do not actually need to have a fully MIMO-SNI S-box. Having only the middle non-linear transformation MIMO-SNI is enough since the top and bottom linear transformations can be considered as combined with the other linear operations of the AES (i.e., ShiftRow, MixColumns and AddRoundKey) when applying the MIMO-SNI composability property.

The non-linear transformation is made of 30 `XOR` gates and 32 `AND` gates, hence it contains more than $2 \cdot (30 + 32) = 124$ wires. This means that it is impossible to apply the exhaustive search used in Section 2.2.2. We therefore reformulate our graph optimization problem into a linear programming problem, for which there exists numerous solvers. The latter does not guarantee that we can find an optimal solution with a reasonable amount of resources, but solvers have efficient heuristics to find good solutions and can prove lower bounds for the solution. Since we take care that our representation as an optimization problem admits as acceptable solutions all the possible implementations of the considered logic circuit, we are able to provide upper and lower bounds on the cost of the optimal implementation.

We write the linear optimization problem in the following way. A binary variable e_i is associated to each edge i of the graph, indicating if it is cut (i.e., if a refresh is inserted). All the paths in the graph are then computed and a binary variable p_j is assigned to each path j, again indicating if it is cut. A path is cut if any edge in the path is cut. It implies a first general constraint $p_j \leq \sum_i e_i$ (the sum is over the edges in the path).

We can then add the various constraints related to Non-Interference properties. First, to enforce NI, for each pair of vertices (u, v) all but one paths from u to v must be cut.

[5] This open-source tool [15] is written using the python programming language and the `networkx` library [36].

[6] Actually, [10] mentions two SNI multiplications are needed, but it was observed by Jean-Sébastien Coron that one is enough during Adrian Thillard's PhD defense.

Let J be the set of paths from u to v, $\sum_{j\in J} p_j \geq |J| - 1$. Next, to enforce SNI, when u is an input node and v an output node, the constraint becomes $\sum_{j\in J} p_j \geq |J|$. For the MI part we need: for any node u, let J be the set of paths from any input node to u, $\sum_{j\in J} p_j \geq |J| - 1$. Finally, for the MO part we need: for any node u, let J be the set of paths from u to any output node, $\sum_{j\in J} p_j \geq |J| - 1$.

The objective function to be minimized is a weighted sum of the e_i variables. The weigh associated to each variable is the cost of adding a refresh on the corresponding edge. This cost can be any metric, such as the amount of random bits required, the computation time, etc. Since each edge has a distinct associated cost parameter, this is the point where we can take into account that the cost of adding a SNI refresh gadget may not be the same as replacing a NI multiplication with a SNI multiplication.

This simple way of writing our problem has two limitations. First, there are many paths in the computation graph (in the order of magnitude of 10^5 for the AES S-box) which leads to many variables and constraints in the optimization problem. This can be mitigated by grouping paths into clusters that share common parts and associating them to a single variable.

The second issue is related to split nodes: there are multiple trees of split nodes that represent the split of a value in more than two parts, and all these representations do not give equivalent possibilities for inserting refresh elements. Furthermore, no tree can provide all the optimization degrees of freedom. Since it would be impractical to run the optimization for all the possible trees, we instead modified the optimization problem. Each split node is replaced by a set of split nodes that form a fully connected DAG and constraints are set to ensure that a constant number of added edges is cut, which ensures that the added edges do not distort the objective function.

We ran this optimization[7] with uniform cost for all edges (which, as discussed in Section 2.4, is reasonable for very high order making considering the current state-of-the-art for elementary gadget implementations). This gave a solution with 41 SNI elements and a lower bound of 34 SNI elements. The implementation of Goudarzi and Rivain in [31] uses two SNI elements per `AND` gate, totaling 64 SNI elements. Our optimized S-box is given in Appendix D.

2.3. Probe-Isolating Non-Interference

In this section, we complement the previous optimization by introducing a new kind of Probe-Isolating Non-Interference (PINI) definition, and proving that it enjoys a very simple composition property. Furthermore, we show that any linear gadget satisfies this new definition, and we exhibit two multiplication gadgets that also satisfies it. It gives us the ability to directly implement any boolean function with PINI gadgets. This is for example in contrast with the more complex analysis of [10] (or the one in the previous section), which requires a careful combination of NI and SNI gadgets.

[7] The optimization setup was done with the tool [15] discussed in Footnote 5 and with the PuLP modelization library [46]. The optimization solving was run with two solvers: CBC [46] and IBM CPLEX; both gave the same result.

2.3.1. Intuition

The main idea behind this new definition is to take into account not the number of probes (or of required inputs for simulation), but instead their position (i.e., the shares' index). The whole circuit can then be cut into $d+1$ circuit shares that are not interconnected, except for non-linear gadgets. If we neglect those gadgets, the circuit is d-probing secure: the adversary can only probe d of the circuit shares, hence it has no information about one circuit share, which contains one share of each input. PINI gadgets behave in the probing model as if they had no connection between circuit shares, which enables to implement non-linear functions while keeping the previous intuition of circuit sharing valid.[8]

Intuitively, the (MIMO-)SNI approach using the computation graph model of Section 2.2.1 cuts paths in the computation graph to avoid having distinct paths between two nodes. By contrast, the probe-isolating approach allows those paths while making sure that they are redundant from the adversarial viewpoint, by ensuring that the propagated probes on both paths can be simulated with the same shares.

2.3.2. Probe Propagation Framework

In the probe propagation framework, probes propagate through PINI gadgets in a way that respects circuit shares isolation.

Internal probes in PINI gadgets cannot be trivially associated to a circuit share, since there is no actual circuit share isolation inside (non-linear) PINI gadgets (the isolation is only simulated). However, we can let those probes carry the same intuition as the output probes: each internal (adversarial) probe gives knowledge to the adversary of at most one circuit share. This preserves the intuition that the adversary has knowledge of at most d of the $d+1$ circuit shares.

We thus add a fourth *probe propagation rule*:

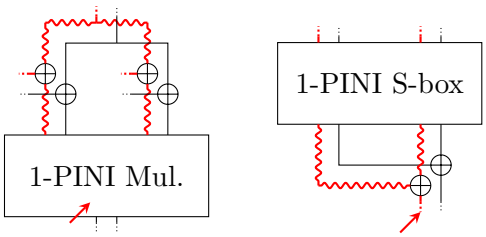
4. Each output probe on a PINI gadget, propagates to all the input shares that are in the same circuit share as the output probe. Each internal probe propagates to all the input shares that are in one additional circuit share (this circuit share may depend on the position of the probes).

The probe isolation principles also impact security conditions: we count the number of circuit shares probed at the output, instead of the number of probes. There is thus a fourth *probe propagation security condition*:

4. For any PINI gadget, the number of circuit shares touched by output probes must be at most d.

[8] To some extent the probe isolation idea can be connected to the Domain-Oriented Masking implementation approach described in [32]. However, the latter focuses on the local security of gadgets without discussing composability issues.

The probe isolation principle is also implicitly used in the seminal work of Ishai et al. [37], but they use $2d+1$ masking.



(a) Multiplication (b) S-box

Figure 2.5.: PINI circuits, masked at order $d = 1$. The arrows indicate the adversarial probes and the red snake wires are the propagated probes.

The way PINI works is illustrated in Figure 2.5, which takes two cases where SNI is not sufficient (Figures 2.1a and 2.3a) and shows how PINI solves the problem.

In Figure 2.5a, there is one internal probe which propagates to one share of each input of the multiplication as it is the case for (S)NI multiplications. However, the propagated probes have the same share index (they are in the same circuit share), hence probe propagation through the linear operation does not violate the second probe propagation security condition.

In Figure 2.5b, the two propagated probes at the output of the S-box have the same share index, hence it does not violate the fourth probe propagation security condition.

2.3.3. Probe-Isolating NI Gadgets & Composability

In this section, we give the formal definition of PINI, and prove security and composability properties. Furthermore, we exhibit PINI addition and multiplication gadgets.

We first show the link between the notion of circuit share and the notations of Section 1.3.1. For a gadget with inputs $x_{i,j}$ and outputs $y_{i,j}$, all the shares of one input are denoted as $x_{i,*}$ and all the input shares that are in the same circuit share are all the input share that have the same share index j: $x_{*,j}$. The same goes for outputs.

In the following definition, the set A is the set of share indices (i.e., circuit shares) that are probed through output probes, and B is the set of circuit shares requested to simulate the internal probes.

Definition 2.5 (Probe-Isolating Non-Interference)**.** *Let G be a gadget over $d+1$ shares and P a set of d_1 probes on wires of G (called internal probes). Let A be a set of d_2 share indices. G is d-Probe-Isolating Non-Interfering (d-PINI) if and only if for all P and A such that $d_1 + d_2 \leq d$, there exists a set B of at most d_1 indices such that probes on the set of wires $P \cup y^G_{*,A}$ can be simulated with the wires $x^G_{*,A\cup B}$.*

The following property shows that PINI satisfies the probing security requirement up to a slight additional requirement of independent input encodings. The latter is generally not a strong constraint, since we will use this property for large circuits, such as complete block ciphers, but it prevents the use of the optimizations described in [23].

Property 2.10 (PINI implies probing security)**.** *A d-PINI gadget (with $d+1$ sharing) is d-probing secure if the encodings of the inputs are independent of each other.*

Proof. Any set of at most d probes can be simulated with at most d shares of each input. Thanks to the independent input encodings, this set of simulation input probes is independent of all the input values. □

We now look at composability properties for PINI gadgets.

Property 2.11 (PINI composability)**.** *A composite gadget made of d-PINI composing gadgets is d-PINI.*

Proof. All internal probes are either inside a gadget, or they are on a share of a a_j wire, in which case it can be considered as internal to one gadget for which a_j is an input or an output. Hence, if the set P of internal probes is of size d_1, there exists sets P_κ of probes internal to G_κ and of sizes $d_{1,\kappa}$ for $\kappa = 1, \ldots, \ell$ such that $P_1 \cup ... \cup P_\ell = P$ and $d_{1,1} + ... + d_{1,\ell} = d_1$.

Let $A_\ell = A$ the set of d_2 indices for output probes shares. For $\kappa = \ell$ to 1, we apply the definition of d-PINI: there exists a set of share indices B_κ of size at most $d_{1,\kappa}$ such that the probes P_κ and output probes $y^{G_\kappa}_{*,A_\kappa}$ can be simulated with the input probes $x^{G_\kappa}_{*,A_{\kappa-1}}$, where $A_{\kappa-1} = A_\kappa \cup B_\kappa$.

By induction, we have $A_{\kappa-1} = A \cup B_\kappa \cup \cdots \cup B_\ell$. This implies that $|A_{\kappa-1}| \leq d_2 + \sum_{p=\kappa}^{\ell} d_p \leq d_2 + d_1 \leq d$, hence the PINI conditions are satisfied for each gadget.

Let $B = A_0 \setminus A$, we have $|B| \leq |B_1 \cup \cdots \cup B_\ell| \leq d_1$. Finally, looking at the whole circuit, we observe that the wires $x_{*,A\cup B}$ allow to simulate all the required probes. □

We next prove that linear gadgets are PINI.

Property 2.12. *The trivial implementation of a linear function is d-PINI.*

Proof. Take B as the set of all share indices for which there is a probe in P. □

We next introduce in Algorithm 2 a new multiplication gadget (PINI_1) for $d+1$ shares with $d(d+1)/2$ randomness requirement. It is a small variation of the ISW multiplication [37], the only modification being the introduction of what we call the "masked shares multiplication trick".

In the original ISW algorithm (Algorithm 1), there is a computation of the randomized product $\alpha_{ij} = r_{ij} + x_i \cdot y_j$ (the variable r_{ij} is a random bit used for the compression stage of the algorithm). A probe on the intermediate product $x_i \cdot y_j$ violates the PINI definition: its simulation requires the knowledge of two input shares from distinct circuit shares. The masked shares multiplication trick solves this problem by replacing the previous computation with $\alpha_{ij} = r_{ij} + x_i \cdot r + x_i \cdot (r + y_j)$, where r is a fresh random bit for each use of the trick.

Furthermore, it is possible to re-use the r_{ij} random bit as the r bit for α_{ij} without breaking the security in order to reduce the randomness complexity. The computation is thus $\alpha_{ij} = r_{ij} + x_i \cdot r_{ij} + x_i \cdot (r_{ij} + y_j)$, which can be simplified to $\alpha_{ij} = \bar{x_i} \cdot r_{ij} + x_i \cdot (r_{ij} + y_j)$.

Algorithm 2 PINI multiplication gadget over $d+1$ shares (PINI_1)

Require: shared factors $a, b \in \mathbb{F}_q^{d+1}$ such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$
Ensure: output $c \in \mathbb{F}_q^{d+1}$ such that $\bigoplus_i c_i = a \cdot b$

```
for i = 0 to d do
  for j = i + 1 to d do
    r_ij ←$ F_q;
    r_ji ← r_ij;
  end for
end for
for i = 0 to d do
  for j = 0 to d do
    if i ≠ j then
      s_ij ← b_j ⊕ r_ij;
      p^0_ij ← ā_i · r_ij;
      p^1_ij ← a_i · s_ij;
      z_ij ← p^0_ij ⊕ p^1_ij;
Ensure: z_ij = r_ij ⊕ a_i · b_j
    end if
  end for
end for
for i = 0 to d do
  c_i ← a_i · b_i ⊕ ⨁^d_{j=0,j≠i} z_ij;
end for
```

Algorithm 3 Input shares chooser for the simulator of PINI multiplication

Require: Set of probes $y^G_{*,A} \cup P$
  $X \leftarrow \emptyset$;
  for $i = 0$ to d **do**
    if a_i, $\overline{a_i}$, b_i, $a_i \cdot b_i$ or c_i is probed **then**
      $X \leftarrow X \cup \{i\}$;
    else if there exists k such that $\bigoplus_{j=1}^{k} z_{ij}$ is probed **then**
      $X \leftarrow X \cup \{i\}$;
    end if
    for $j = 0$ to d **do**
      if there are at least two probes on intermediate values of computation of z_{ij} (these values are r_{ij}, p^k_{ij}, s^k_{ij} and z_{ij}) **then**
        $X \leftarrow X \cup \{i, j\}$;
      else if there is one probe on an intermediate value of the computation of z_{ij} **then**
        if $i \in X$ or $j \in X$ **then**
          $X \leftarrow X \cup \{i, j\}$;
        else
          $X \leftarrow X \cup \{i\}$;
        end if
      end if
    end for
  end for
  $B \leftarrow X \setminus A$;
Ensure: $|B| \leq |P|$

Property 2.13. *The multiplication gadget PINI_1 (Algorithm 2) is d-PINI.*

Proof. We prove that the values assigned to the probes by the simulator described in Algorithm 4 are indistinguishable from the multiplication gadget probes.

This behavior of the simulator is identical to the behavior of the gadget, except for values z_{ij}, s_{ij} and p^1_{ij} for which $i \notin X$ (X is generated by Algorithm 4).

In these cases, if z_{ij} or a sum in which it appears is probed, then there is no probe on z_{ji} (or their intermediate values, or a sum in which it appears) or on intermediate values of the computation of z_{ij}, hence r_{ij} is only observable to the distinguisher through z_{ij}. This means that z_{ij} is seen by the distinguisher as a uniform random variable independent of all other variables, which is what the simulator generates.

For probes on s_{ij}, the same argument applies: r_{ij} is only observable to the distinguisher through s_{ij}, hence s_{ij} is seen by the distinguisher as a uniform independent random variable. To simulate p^1_{ij}, the simulator simulates s_{ij} as previously (and the same argument applies), then computes p^1_{ij} in the same manner as Algorithm 2. □

We next build a PINI adaptation of the NI multiplication gadget of Belaid et al. [10] in order to reduce the randomness complexity. Indeed, the gadget of Belaid et al. has a lower asymptotic randomness complexity compared to the ISW gadget ($\lfloor d^2/4 \rfloor + d$ vs. $d(d+1)/2$ random bits). Its structure is similar to the one of the ISW multiplication, the main difference being more re-use of some random bits at the `Compression` stage (the staging of multiplication algorithms is detailed in Section 3.4). The masked shares multiplication trick can thus be applied to this algorithm. However, since there is less randomness used at the compression stage, it can not simply be re-used for the trick, as it was previously the case.

A naive implementation of the trick (a fresh random bit r for each masked shares multiplication) would require $d(d+1)$ random field elements: it would make the algorithm less efficient than the previous PINI_1 multiplication which uses $d(d+1)/2$ random field elements. Nevertheless, we show that the multiplication is still PINI if only $d+1$ fresh random bits $s_0, \ldots, s_d$ are generated and the random field elements used for the trick are $r = s_i + s_j$ (for the computation of $x_i y_j$). As a result, the randomness complexity of this new PINI multiplication gadget (PINI_2, see Algorithm 5) is $\lfloor d^2/4 \rfloor + 2d + 1$.

Intuitively, it can be seen that this reuse of randomness does not break the security: the security could break if the adversary had multiple probes that depend on the same s_i. Since each use of an s_i is masked with a s_j, there can be problems only when the adversary also has multiple probes that depend on s_j. In this case, the number of adversarial probes is such that the simulator can have access to the inputs x_i and y_j and can thus run a perfect simulation. An example of such a situation are probes in the computation of $x_{i_1} y_{j_1}$, $x_{i_1} y_{j_2}$, $x_{i_2} y_{j_1}$ and $x_{i_2} y_{j_2}$ for some indices i_1, i_2, j_1 and j_2.

Property 2.14. *The masked multiplication gadget PINI_2 (Algorithm 5) is d-PINI.*

The proof follows from the previous discussion and the proof that the multiplication of Belaid et al. [10] is NI. The full details are given in Appendix B.

Algorithm 4 Simulator of probes for the PINI_1 gadget (Algorithm 2)

Require: Set of probes $y^G_{*,A} \cup P$
  Run Algorithm 3 and get X and B.
Require: Knowledge of input shares $x^G_{*,A\cup B} = x^G_{*,X}$.
  for $0 \leq i \leq d$ **do**
    for $0 \leq j \leq d$ **do**
      if $i \in X$ and $j \in X$ **then**
        Compute w^k_{ij}, s^k_{ij} and z_{ij} as specified by the algorithm of the multiplication gadget;
      else if $i \notin X$ **then**
        Leave z_{ij} and its intermediate values unassigned as they are not involved in any probe;
      else
Ensure: $i \in X$ and $j \notin X$.
Ensure: Only one intermediate values of the computation of z_{ij} is probed, or a sum in which z_{ij} appears.
Ensure: z_{ji} or its intermediate values do not appear in any probe.
        if z_{ij} or a sum in which z_{ij} appears is probed **then**
          $z_{ij} \xleftarrow{\$} \mathbb{F}_q$;
        else if s_{ij} is probed **then**
          $s_{ij} \xleftarrow{\$} \mathbb{F}_q$;
        else if p^0_{ij} is probed **then**
          $r_{ij} \xleftarrow{\$} \mathbb{F}_q$;
          $p^0_{ij} \leftarrow \overline{a_i} \cdot r_{ij}$;
        else if p^1_{ij} is probed **then**
          $s_{ij} \xleftarrow{\$} \mathbb{F}_q$;
          $p^1_{ij} \leftarrow a_i \cdot s_{ij}$;
        end if
      end if
    end for
  end for
  Compute (partial) sums of assigned z_{ij}, products $a_i b_i$ and associated c_i.
Ensure: All the probed values are now assigned.

Algorithm 5 PINI_2 multiplication over $d+1$ shares.

Require: shared factors $a, b \in \mathbb{F}_q^{d+1}$ such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$
Ensure: output $c \in \mathbb{F}_q^{d+1}$ such that $\bigoplus_i c_i = a \cdot b$
  for $i = 0$ to d **do**
    $s_i \xleftarrow{\$} \mathbb{F}_q$;
    for $j = 0$ to $d - i - 1$ by 2 **do**
      $r_{i,d-j} \xleftarrow{\$} \mathbb{F}_q$;
    end for
  end for
  for $j = d - 1$ down to 1 by 2 **do**
    $r_j \xleftarrow{\$} \mathbb{F}_q$;
  end for
  for $i = 0$ to d **do**
    for $j = i + 1$ to d **do**
      $s_{i,j} \leftarrow s_i + s_j$;
      $p^0_{i,j} \leftarrow a_i \cdot s_{i,j}$;
      $p^1_{i,j} \leftarrow a_i \cdot (b_j + s_{i,j})$;
      $p^2_{i,j} \leftarrow b_i \cdot s_{i,j}$;
      $p^3_{i,j} \leftarrow b_i \cdot (a_j + s_{i,j})$;
    end for
    $c_{i,d} \leftarrow a_i \cdot b_i$;
    for $j = d$ down to $i + 2$ by 2 **do**
      $t_{i,j} \leftarrow r_{i,j} + p^0_{i,j} + p^1_{i,j} + p^2_{i,j} + p^3_{i,j} + r_{j-1} + p^0_{i,j-1} + p^1_{i,j-1} + p^2_{i,j-1} + p^3_{i,j-1}$;
Ensure: $t_{ij} = r_{ij} + a_i \cdot b_j + a_j \cdot b_i + r_{j-1} + a_i \cdot b_{j-1} + a_{j-1} \cdot b_i$
      $c_{i,j-2} \leftarrow c_{i,j} + t_{i,j}$;
    end for
    if $i \not\equiv d \mod 2$ **then**
      $t_{i,i+1} \leftarrow r_{i,i+1} + p^0_{i,i+1} + p^1_{i,i+1} + p^2_{i,i+1} + p^3_{i,i+1}$;
      $c_{i,i} \leftarrow c_{i,i+1} + t_{i,i+1}$;
      if $i \equiv 1 \mod 2$ **then**
        $c_{i,0} \leftarrow c_{i,i} + r_i$;
      else
        $c_{i,0} \leftarrow c_{i,i}$;
      end if
    else
      for $j = i - 1$ down to 0 **do**
        $c_{i,j} \leftarrow c_{i,j+1} + r_{j,i}$;
      end for
    end if
    $c_i \leftarrow c_{i,0}$;
  end for

2.3.4. Relationship Between PINI and MIMO-SNI

We first observe that MIMO-SNI implies PINI.

Property 2.15. *Any d-MIMO-SNI gadget is d-PINI.*

Proof. In the definition of PINI, take B as the share indices of t_1 input shares required by the simulator. □

This leads immediately to the following composability result.

Corollary 2.16. *A composite gadget whose composing gadgets are only d-MIMO-SNI gadgets and linear gadgets is d-probing secure.*

Proof. Direct from Property 2.15, Property 2.12, Property 2.11 and Property 2.10. □

We note that this corollary now applies to the greedy strategy (AES implementation proposed in [31]): using only SNI multiplications and systematically refreshing one of their inputs guarantees the input and output independence conditions which are required to move from SNI to MIMO-SNI and therefore PINI gadgets.

2.4. Implementation Cost

We conclude the chapter by discussing and comparing the expected performances of our two proposals for the implementation of a (masked, bitslice) composable AES S-box: the MIMO-SNI optimization and the PINI approach. As mentioned in Section 1.6, we use the randomness complexity (in bits) as our main cost metric for this purpose (and since the computational complexity is essentially similar for the gadgets we consider that all have to compute the full matrix of shares' products). To verify the validity of this assumption, we compare the execution time of the various algorithms on a microprocessor.

We report the state-of-the-art randomness cost for the gadgets used in Table 2.1. Using these cost formulas, we first observe that for sufficiently high orders ($d \geq 17$), the multiplication of Belaid et al. followed by the refresh of Battistello et al. has a lower cost than the multiplication of Ishai, Sahai and Wagner, which justifies the assumptions made for the optimization in Section 2.2.3.[9]

Using those costs, we can evaluate the cost for a full bitslice, masked and composable AES S-box for three different approaches: the greedy strategy, the MIMO-SNI optimized S-box, and the S-box that uses PINI multiplications. The bitslice S-box has 32 multiplications, hence the greedy strategy requires 32 SNI multiplications and 32 SNI refresh gadgets. The optimized MIMO-SNI S-box requires 41 refresh elements, and 12 of those refresh the output of a multiplication (hence can be implemented with a SNI multiplication). The PINI implementation simply requires 32 PINI multiplication gadgets.

[9] Our optimization can be easily adapted to take into account the actual costs at lower orders, but since the relative costs differ for every order, finding the optimal implementation would require re-running the optimization for each order.

Order d	NI mul.	SNI refresh	SNI mul.	PINI mul.
1	1	1	1	1 [PINI_1]
2	2 [10]	3	3	3 [PINI_1]
3	4 [10]	4 [4]	6	6 [PINI_1]
4	5 [10]	8	10	10 [PINI_1]
5	11	12	15	15 [PINI_1]
6	15	14 [4]	21	21 [PINI_1]
7	19	20	24 [4]	27 [PINI_2]
$d \geq 8$	$\mathcal{C}_{mul,NI}$	$\mathcal{C}_{ref}$	$\mathcal{C}_{mul,SNI}$	$\mathcal{C}_{mul,\mathsf{PINI}_2}$

$$
\begin{aligned}
\mathcal{C}_{mul,NI} &= \left\lfloor d^2/4 \right\rfloor + d \text{ [10]} \\
\mathcal{C}_{ref} &= (d+1)\left(\log_2(d+1) - 1\right) \text{ [7]} \\
\mathcal{C}_{mul,SNI} &= \min\left(d(d+1)/2, \mathcal{C}_{mul,NI} + \mathcal{C}_{ref}\right) \\
\mathcal{C}_{mul,\mathsf{PINI}_1} &= d(d+1)/2 \\
\mathcal{C}_{mul,\mathsf{PINI}_2} &= \left\lfloor d^2/4 \right\rfloor + 2d + 1
\end{aligned}
$$

Table 2.1.: Randomness cost of best known gadgets at various orders. Numbers in the table with no reference are instantiations of the formula valid at all orders. The formula for $\mathcal{C}_{ref}$ is valid only if $d+1$ is a power of 2. It is more complicated in other cases but it is still $\mathcal{O}(d \log d)$. The implementation of the SNI multiplication is either the ISW multiplication [37] or a NI multiplication followed by a SNI refresh.

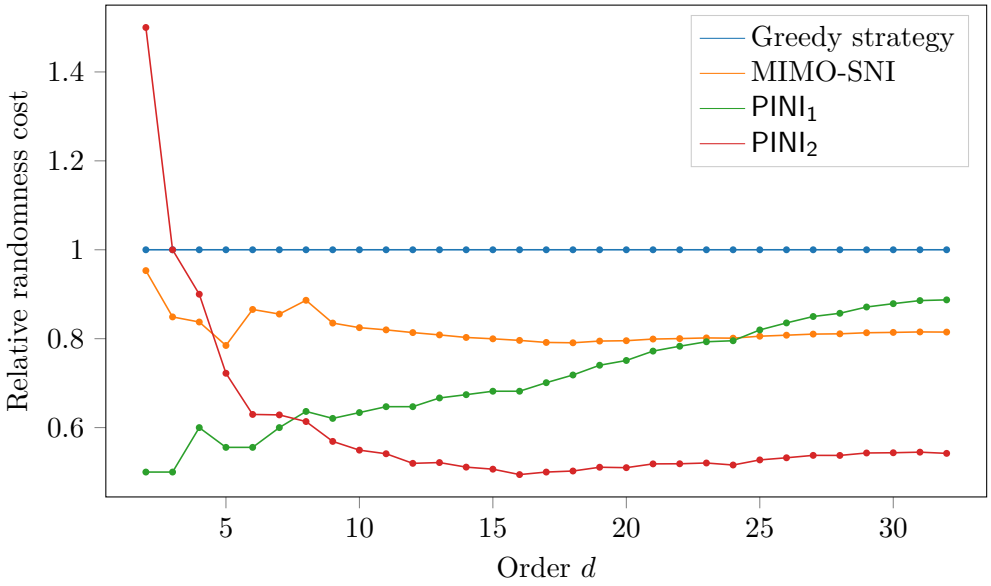


Figure 2.6.: Randomness cost for a bitslice, masked and composable AES S-box implementation. The cost is measured relatively to the cost of the greedy strategy.

The total randomness cost is shown in Figure 2.6. First, we can see that the MIMO-SNI optimization reduces costs by approximately 20 % over the greedy strategy. The PINI_1 multiplication is even better at low orders ($d \leq 23$), with cost reduction up to 50 %. The PINI_2 multiplication however is better at higher orders. Overall, taking the best PINI implementation at each order, we get a cost reduction of 40 % to 50 % for the PINI strategy compared to the greedy strategy.

It is interesting to note that the generic PINI_1 construction is more efficient at low orders than the other SNI solutions, even though those solutions use automatically optimized gadgets.

We conclude by discussing the performance in a broader sense by looking at the execution time of a software implementation. We take the RC = 80 scenario from [38]: for their algorithm, the linear layers cost 1 % of execution time, the arithmetic operations of the S-box 7 % and the randomness generation 92 %. Their algorithm can be analyzed to find the execution time (expressed as percentage of their total runtime) of one arithmetic operation (including load/store overheads) and the time to generate one random bit. Knowing the number of arithmetic operations and amount randomness required for each of our algorithms, we compute their execution time, which gives Figure 2.7.[10]

Since the randomness generation time is dominant, our goal of reducing randomness requirements is justified. The main difference is that for the same randomness consumption, the PINI multiplication gadgets have about two times more operations that their

[10] The only assumption of this estimation method is that the arithmetic operation execution time and randomness generation time are constant across algorithms. The estimation error is thus small and independent of many algorithmic parameters such as the masking order.

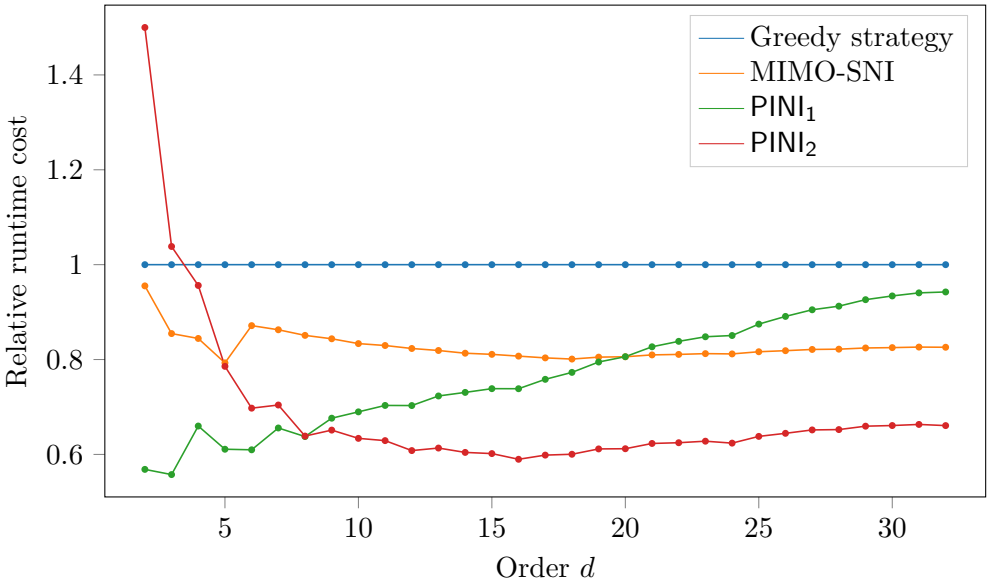


Figure 2.7.: Software runtime cost for a bitslice, masked and composable AES S-box implementation. The cost is measured relatively to the cost of the greedy strategy.

NI/SNI counterparts, which reduces slightly their score compared to the one based only on randomness complexity.

Finally, this comparative result (i.e., the choice between the MIMO-SNI optimization and PINI multiplications) is strongly dependent on the randomness complexity of the (state-of-the-art) gadgets used. In this respect, it is worth emphasizing that in addition to its better performance with current gadgets, one core advantage of the PINI definition is that it allows trivial proofs of complex circuits with one single type of gadget, namely PINI multiplications (while the optimization has to deal with NI multiplications, SNI multiplications and SNI refreshes).

3. Improved Security against Horizontal Attacks with Reduced Cost

By pushing the reduction of randomness complexity towards optimal, one ignores the increased risk that the noise of each share can be reduced by an adversary, and therefore that this optimization can eventually be detrimental to the concrete security level of an implementation. From the practical viewpoint, this has been demonstrated by the horizontal attacks in [7, 33]. From a theoretical viewpoint, it is captured by the concept of "*noise rate*" discussed in Section 1.5.2.

Again motivated by the excellent performances of additive masking in small fields, we therefore investigate the noisy leakage security of masked multiplications that can work in $\mathbb{F}_2$. Our starting point for this purpose is an algorithm by Battistello et al. [7] (next denoted as SNI-H) which was shown to have improved resistance against horizontal attacks based on qualitative arguments (i.e., the authors showed that the number of times each share is manipulated is reduced compared to the ISW algorithm — that we hereafter denote as SNI-ISW to emphasize the connection between the algorithms).

We push the understanding of this algorithm one step further by analyzing it, along with other (SNI and PINI) gadgets without specific protection against horizontal attacks, in the recently proposed Local Random Probing Model (LRPM, see Section 1.5.2) which bounds the resistance of an implementation against SASCA. This allows us to obtain good intuition about the impact of the gadget design on their resistance against SASCA, and to obtain a more quantitative view of this resistance. In particular, we are able to confirm the relevance of qualitative analysis of Battistello et al. quantitatively, confirming a noise rate in $\mathcal{O}(1/\log(d))$.

We then propose new algorithms (derived from the one of Battistello et al., but also from PINI multiplications and the original multiplication by Ishai et al.) with improved resistance against horizontal attacks. Those lead to a significantly better security in low-noise contexts, and our results suggest that their noise rate is $\mathcal{O}(1)$. Eventually, we discuss the "*randomness complexity vs. noise rate*" trade-off based on the different algorithms analyzed, suggesting gradual performance overheads as the noise rate evolves from $\mathcal{O}(1/d)$ (unprotected gadgets such as SNI-ISW or PINI_1 and PINI_2) to $\mathcal{O}(1/\log(d))$ and $\mathcal{O}(1)$.

3.1. Methodology

Our goal is to use the LRPM of [35] (introduced in Section 1.5.2) in order to measure the resistance of various masked gadgets to horizontal attacks. We analyze masked multiplication gadgets at various orders, which are relevant targets of investigation since

they usually correspond to the more complex parts of a masked implementation (both in terms of security and efficiency).

The LRPM requires the following inputs:

1. The factor graph of the target gadget (or implementation), which is a graph of relationships between all the variables manipulated.

2. The MI between each variable in the graph and the leakages.

For the first point, the relationship on the variables that appear in the gadget first include the ones that are explicit in the gadget (i.e. each operation of the gadget creates a relationship). There are also variables which do not appear in the gadget such as the (unmasked) sensitive variables: for example an input x that only appears through it shares $x_0, \ldots, x_d$ in the gadget. We include it as well in the factor graph with the sharing relationship $x = x_0 + \cdots + x_d$. If the gadget implements the multiplication $z = x \cdot y$ (where x, y and z are the sensitive variables), we furthermore assume that the inputs x and y are related by a bijection and the output z is also in bijection with the input x.[1].

Another implicit relationship appears when refresh gadgets are used: the sum of the inputs of a refresh is equal to the sum of its outputs. This relationship is important to consider when a circuit uses refresh gadgets internally (which as will be seen next, may be relevant to improve security against horizontal attacks). We illustrate this with a simple circuit: $y = \mathsf{R}(x)$, where $\mathsf{R}(\cdot)$ is a refresh gadget, $x = (x_0, \ldots, x_d)$ an input sharing and $y = (y_0, \ldots, y_d)$ an output sharing. Let us assume that there are many manipulations of the shares y_i. In this case, the BP algorithm can propagate this information to the shares x_i, but (especially if the random bits in the refresh and the x_i are manipulated few times and the refresh has a high logic depth — for example by iterating many times a simpler refresh algorithm) the information extracted on x_i can be much smaller than the information available on y_i. A SASCA targeting the input sensitive variable based on the relationship $x^* = x_0 + \cdots + x_d$ will therefore lead to a much lower information than actually available, by exploiting the relationship $x^* = y_0 + \cdot + y_d$. For this example, the solution is simple: insert in the factor graph not only the equation $x^* = x_0 + \cdots + x_d$, but also $x^* = y_0 + \cdots + y_d$. We generally apply a similar strategy for all the (possibly more complex) gadgets we analyze. Let us take as an example the refresh trees of Figure 3.3. For each refresh gadget, we associate a new variable and insert in the factor graph the facts that (*i*) the sum of the input variables of the refresh is equal to this variable, and (*ii*) the sum of the variables associated to the two refresh gadgets is also equal to the associated variable.

For the second point, we first define the *manipulation* of a variable as its use as an operand of an operation or its apparition as the result of an operation. We assume that each manipulation leaks some amount MI_o of information and that the leakages of all manipulations are independent, hence we approximate the total leakage on any

[1] The bijection assumption is a worst-case assumption that is frequently encountered in practice (e.g., if we take the AES S-box implemented in $\mathbb{F}_{256}$). We insist that its impact on our conclusions is anyway limited: removing this assumption generally implies a reduction of MI_t by a factor of approximately 2.

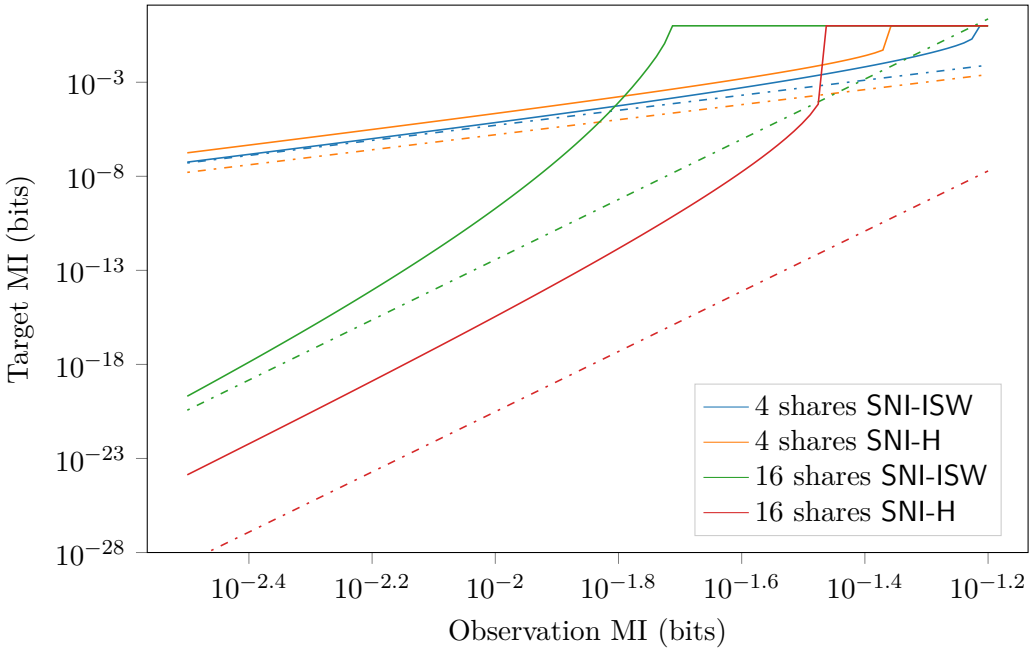


Figure 3.1.: Target MI as a function of observation MI, for the SNI-ISW multiplication gadget [37] and the SNI-H multiplication gadget [7]. The continuous line is the bound on the target MI that can be extracted by the BP algorithm and the dashed line is a lower bound for the target MI computed assuming that only the leakages on input shares are exploited. The results for the NI multiplication gadget of Belaid et al. [10] are identical to the SNI-ISW curves.

variable as $m\mathrm{MI}_o$ where m is the number of manipulations. The latter corresponds to the Independent Operations Leakages (IOL) assumption validated in [33]. For each random variable, we additionally assume that there is one manipulation associated to its generation.

The extraction of those two inputs (factor graph and leakage MI), and the execution of the BP algorithm are automated by a tool [14], whose input is a high-level algorithmic description of the considered gadget. The tool is mainly written in Python, except for the performance-critical part (the BP algorithm) which is written in the high-performance Rust programming language in order to get reasonable computation time (all the graphs of this chapter can be generated in a few dozens of seconds).

3.2. Analysis of SNI Multiplication Gadgets

We first apply our methodology to the SNI-ISW gadget, at orders $d = 3$ and $d = 15$. Results are reported in Figure 3.1, leading to the following conclusions.

In general, for low observation MI (i.e., MI_o) the trend of the target MI (i.e., MI_t) is an asymptote of slope d (as expected for dth order security). For larger MI_o, we see that

the curves leave the asymptote until they reach $\mathrm{MI}_t = 1$. There are thus two regions in the graph: the low MI_o and high MI_o. The boundary between the regions varies depending on the curves (see for example SNI-ISW for $d = 3$ and $d = 15$ in Figure 3.1), which essentially reflects the noise rate. The location of the boundary and the slope of the asymptote are the two parameters that determine the security level of a gadget. In the following, since our focus is on horizontal attacks (and the slope of the asymptote is always equal to d since we only consider d-probing secure gadgets anyway), we will focus on the noise rate, captured as the location of the boundary – concretely, we use the point where $\mathrm{MI}_t = 1$ for this purpose.

More specifically, we see that the location of the boundary strongly depends on the number of shares and security order for the SNI-ISW gadget. This comes from the fact that this gadget manipulates each input share $d + 1$ times, which explains why the boundary at order $d = 3$ is better than at order $d = 15$ for large MI_o. We note that these different boundaries correspond to the noise rate of $\mathcal{O}(1/d)$ that is expected for the SNI-ISW multiplication.[2]

By contrast, for the gadget of Battistello et al. [7] that we also report on the figure (denoted as SNI-H and for which the details will be given later in the paper), the difference between the boundaries is much smaller. This is due to a different refreshing strategy used in this gadget, that is aimed to prevent horizontal attacks. More precisely, since each input share is manipulated only $\mathcal{O}(\log(d))$ times, the SNI-H gadget gains interest over the SNI-ISW one as d increases, thanks to its better noise rate (e.g., for $d = 15$). At lower orders (e.g., $d = 3$), the SNI-H gadget remains worse than the SNI-ISW gadget due to the additional leakages that its embedded refresh operations imply (which are not compensated by the improved noise rate in this case).

Going deeper into the analysis, we additionally plot lower bounds (represented with dashed lines on the figure) which are computed by applying the methodology under the hypothesis that only the leakages of the input shares are observed and exploited. It gives curves $\mathrm{MI}_t = \min\left(1, (n_o \mathrm{MI}_o)^d\right)$, where n_o is the number of observations of each input share. When the asymptote of the curve is close to the lower bound, which happens for SNI-ISW at low MI_o, the BP algorithm is not able to extract much information from the internal leakages of the gadget. Our interpretation is that the information from internal leakage is too small for precisely estimating the random values in the gadget, and the propagation of this information is "blocked" by the use of random bits in the multiplication. At larger MI_o, the information leaked on the internal variables becomes sufficient to (partially) recover the random bits, hence this information can be propagated to the input shares.

Interestingly, and while the lower bound is tight for the SNI-ISW gadget in the high noise region, it is not for the SNI-H gadget (even though the slope is still d). The latter suggests that there are more useful leakages on intermediate variables in this case, such as the outputs of the embedded refresh gadgets.

These analyzes confirm quantitatively the qualitative conclusion from [7] that the

[2] Formally, the proofs in [22] require a noise rate of $\mathcal{O}(1/d^2)$, but as discussed in [22] the latter is assumed to due to the proof that is not completely tight.

repeated manipulation of some shares is the main weakness exploited by horizontal attacks. They cause a horizontal shift of the information theoretic curves in Figure 3.1, and impose a noise level that increases with the order of the implementation. If the noise level does not increase with the order, the shift directly translates into a security loss on MI_t of a factor $\mathcal{O}(d^d)$ for the SNI-ISW gadget (which is reduced to $\mathcal{O}((\log(d))^d)$ if the countermeasure of Battistello et al. is applied). In other words, this experiments highlights that the use of the SNI-ISW (resp., SNI-H) multiplication gadget at high order requires a large amount of noise such that $\mathrm{MI}_o \propto 1/d$ (resp., $\mathrm{MI}_o \propto 1/\log(d)$).

3.3. Analysis of PINI Multiplication Gadgets

The application of our methodology to the PINI multiplication gadgets is reported in Figure 3.2. For the PINI_1 gadget, we see that the global trend is very close to the one of the SNI-ISW gadget, except that there is an horizontal shift of a factor of 2 in the asymptotic region. This is due to the multiple manipulations of the input shares during the computation of $\bar{a} \cdot r + a \cdot (r + b)$ (instead of $a \cdot b$ for SNI-ISW). Note that one of those manipulations is a use of $\bar{a}$, which is equivalent to a manipulation of a from the probing security viewpoint, but not from the horizontal attacks viewpoint, which explains why the bound is not tight.

For the PINI_2 gadget, we see that the asymptotic performance is slightly better than the one of PINI_1, which is due to less manipulations of input shares. However, there are $\mathcal{O}(d^2)$ operations which involve only input shares and $d + 1$ random bits (the s_i in Algorithm 5), instead of the $\mathcal{O}(d^2)$ random bits for SNI-ISW. This has only limited impact at low MI_o (where total leakage on the random bits is sufficiently low, so they can be considered as unknown to the adversary). But intermediate MI_o, the accumulated leakage on these random bits makes them known to the adversary, which causes the "staircase of asymptotes" aspect of the curve.

Since both PINI multiplication gadgets have $\mathcal{O}(d + 1)$ manipulations of the input shares, they have a noise rate in $\mathcal{O}(1/d)$. Therefore, their use at high order faces the same problem as the SNI-ISW gadget: it requires a large amount of noise (i.e., $\mathrm{MI}_o \propto 1/d$).

3.4. Design of Improved Multiplication Gadgets

In this section, we first build a framework that allows to describe the SNI-ISW and SNI-H multiplication gadgets in a similar way. We then investigate how the scheme of Battistello et al. (SNI-H) can be improved against horizontal attacks. Next, we build PINI multiplication gadgets secure against horizontal attacks by applying similar ideas. Finally, we build new gadgets that trade a bit of resistance to horizontal attacks for reduced randomness complexity.

Generic description of SNI-ISW multiplication gadgets. The SNI multiplication gadgets can be decomposed in three stages: the `MatGen` stage produces a $(d + 1) \times (d + 1)$ matrix of pairs of refreshed shares of x and y, the `Product` stage computes the product

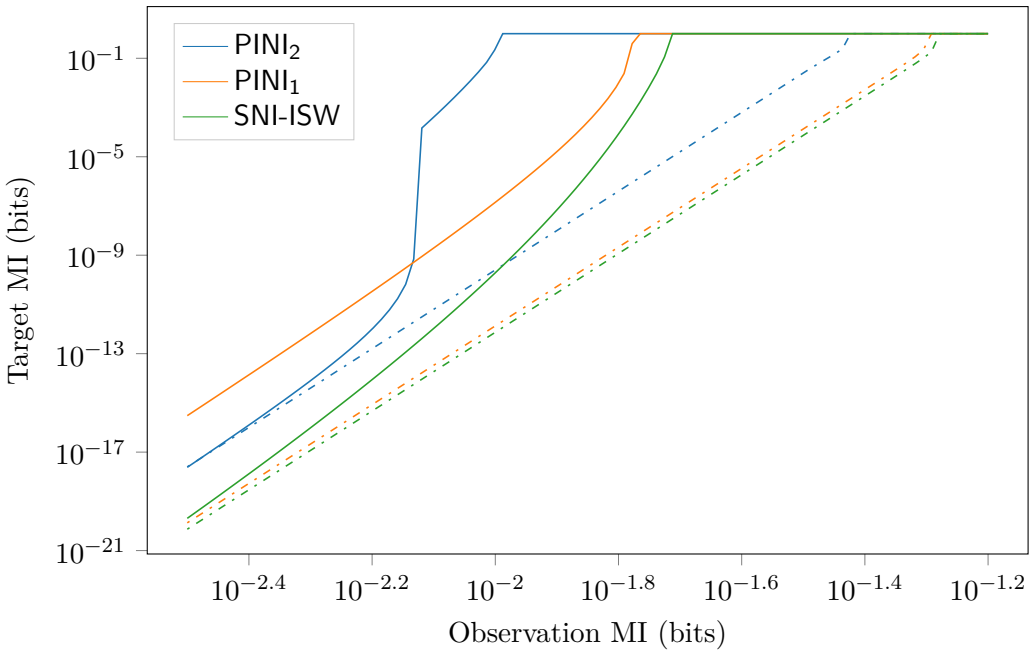


Figure 3.2.: Target MI as a function of observation MI, for the SNI-ISW multiplication gadget [37], the PINI_1 and PINI_2 multiplication gadgets. The continuous line is the result of the belief propagation algorithm for the MI and the dashed line is a lower bound for the target MI computed assuming only the leakage on input shares are exploited. From now on, the results are shown only for order 15, but the conclusions applies at all orders.

of those stages, which gives a $(d+1)^2$ sharing of the product, and the `Compression` stage compresses the $(d+1)^2$ sharing into an $d+1$ sharing output. This process is illustrated by the following equation (for $d+1=4$).

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \begin{bmatrix} y_4 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \xrightarrow{\texttt{MatGen}} \begin{bmatrix} (x_{0,0}, y_{0,0}) & (x_{0,1}, y_{1,0}) & (x_{0,2}, y_{2,0}) & (x_{0,3}, y_{3,0}) \\ (x_{1,0}, y_{0,1}) & (x_{1,1}, y_{1,1}) & (x_{1,2}, y_{2,1}) & (x_{1,3}, y_{3,1}) \\ (x_{2,0}, y_{0,2}) & (x_{2,1}, y_{1,2}) & (x_{2,2}, y_{2,2}) & (x_{2,3}, y_{3,2}) \\ (x_{3,0}, y_{0,3}) & (x_{3,1}, y_{1,3}) & (x_{3,2}, y_{2,3}) & (x_{3,3}, y_{3,3}) \end{bmatrix} \ldots$$

$$\ldots \xrightarrow{\texttt{Products}} \begin{bmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \alpha_{0,3} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \alpha_{3,0} & \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{bmatrix} \xrightarrow{\texttt{Compression}} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad (3.1)$$

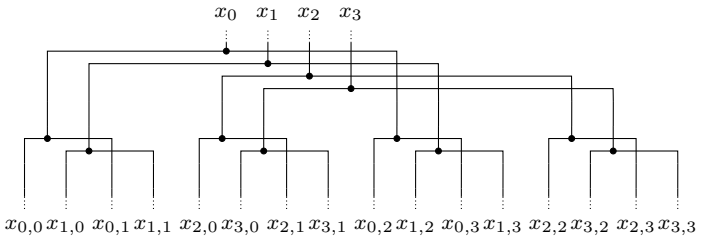
This general construction applies to all the multiplication algorithms discussed so far.[3] An example instantiation is the SNI-ISW multiplication, described in Algorithm 6. The `MatRef` stage is described in Algorithm 7, while the two other stages are interleaved in the other operations. Algorithm 7 is itself generic, in the SNI-ISW case it is simply $x_{i,j} = x_i$ and $y_{i,j} = y_j$. Starting from the SNI-ISW gadget, it suffices to change the `Compression` stage to get the NI multiplication of Belaid et al. Another example is the PINI_1 multiplication: it only differs from the SNI-ISW algorithm by its `Products` stage. In our constructions of SNI multiplication gadgets, only the `MatGen` stage changes.

Each of the stages may use some random bits (and sometimes, as an optimization, the same random bits a re-used in different stages). In the `MatGen` stage, the randomness cost comes from the use of refresh gadgets (called internal refresh gadgets) in it. A good example of randomness in the `Products` stage is the PINI_2 gadget (Algorithm 5, variables $s_0, \ldots, s_d$). Finally, the `Compression` stage is the only one where randomness is used in the SNI-ISW multiplication.

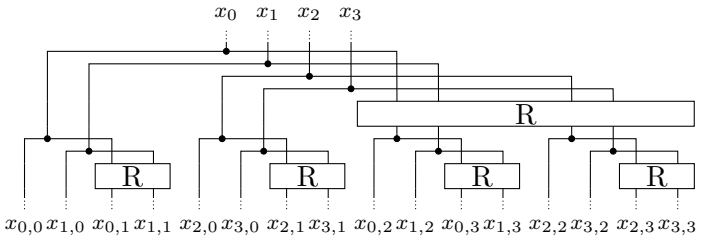
Countermeasure of Battistello et al. [7] The heuristic countermeasure of Battistello et al. makes use of refresh gadget in the `MatGen` stage to avoid repeated manipulation of the same variables. It proceeds using a recursive algorithm which is illustrated in Figure 3.3b (for $n = 4$). This algorithm is applied to the sharing of x and to the sharing of y to get the matrix of pairs of shares. Algorithm 7 formally describes this operation. It is again a generic construction, which uses two algorithms as parameters: `Refresh1` and `Refresh2`. The SNI-H case corresponds to the case where `Refresh2` is a SNI refresh (such as the refresh of Battistello et al. [7]), and `Refresh1` is an identity gadget (the inputs are simply wired to the outputs). The SNI-ISW gadget can be instantiated by taking `Refresh1` and `Refresh2` as identity gadgets (Figure 3.3a).

Improved SNI gadgets. The representation of the SNI-H tree (Figure 3.3b) compared to the tree of SNI-ISW (Figure 3.3a) suggest a scheme where more refresh gadget are added:

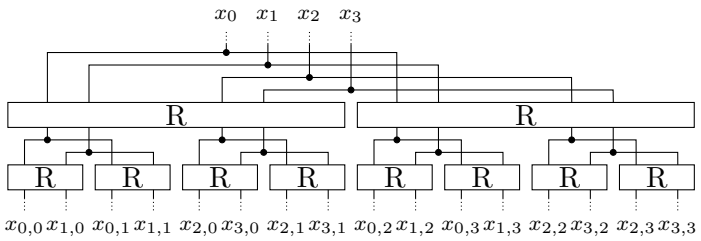
[3] At least in principle. The algorithms sometimes blend the boundaries between the stages for optimization purposes.



(a) No refreshing — SNI-ISW



(b) Battistello et al. refreshing — SNI-H



(c) Double refreshing — SNI-H+

Figure 3.3.: Example of the generalized `MatGen` algorithm (Algorithm 7) for $d + 1 = 4$. The algorithm is basically two trees of internal refresh gadgets (one for sharings of x, one for those of y). The input shares are $x_0, \ldots, x_3$. The $x_{i,j}$ are the outputs of the `MatGen` stage (see Equation (3.1)).

Algorithm 6 Generalized SNI multiplication gadget

Require: shared factors $x, y \in \mathbb{F}_q^n$ such that $\sum_i x_i = x^*$ and $\sum_i y_i = y^*$
Ensure: output $c \in \mathbb{F}_q^{d+1}$ such that $\sum_i c_i = x^* \cdot y^*$

```
M ← MatRef((x_0, ..., x_d), (y_0, ..., y_d)) {See Algorithm 7.}
for i = 0 to d do
  for j = 0 to d do
    (x_ij, y_ij) ← (M)_ij
  end for
end for
for i = 0 to d do
  for j = i + 1 to d do
    r_ij ←$ F_q
    z_ij ← (r_ij + x_ij · y_ij) + x_ji · y_ji
    z_ji ← r_ij;
  end for
end for
for i = 0 to d do
  c_i ← x_ii · y_ii + Σ_{j=0, j≠i}^{d} z_ij
end for
```

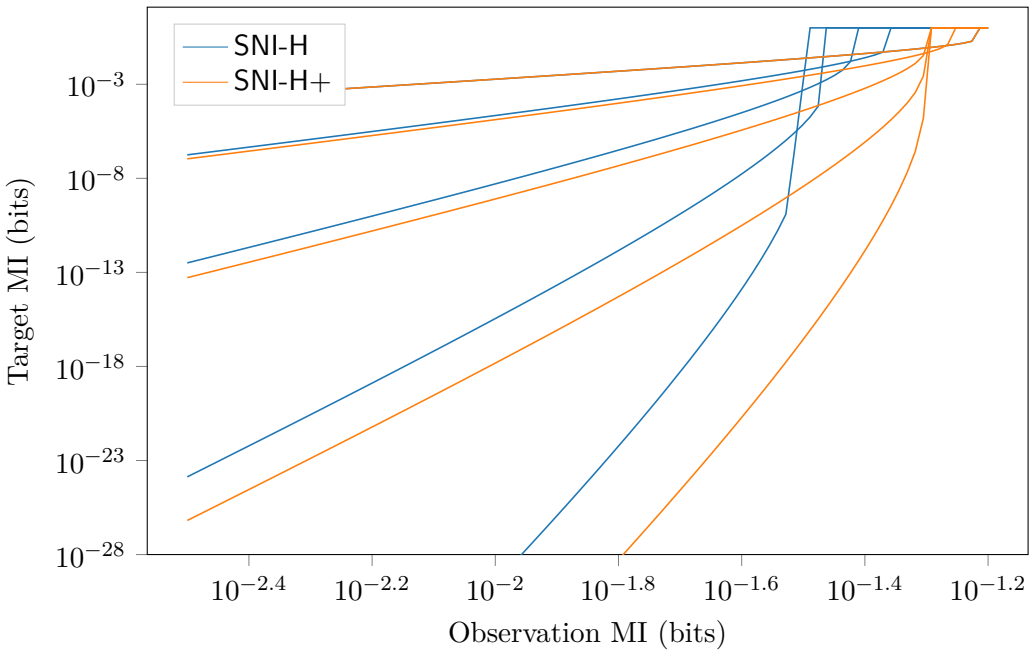


Figure 3.4.: Target MI as a function of observation MI, for the SNI-H and SNI-H+ multiplication gadgets at orders $d = 1, 3, 7, 15$ and 31.

Algorithm 7 `MatGen` (If $d+1$ is a power of 2.)

Require: Refresh algorithms `Refresh1` and `Refresh2`

Require: Shared factors $x, y \in \mathbb{F}_q^{d+1}$ such that $\bigoplus_i x_i = x$ and $\bigoplus_i y_i = y$

Ensure: Output $M \in \left(\mathbb{F}_q^2\right)^{(d+1)\times(d+1)}$ such that $\bigoplus_{i,j} x_{ij} \cdot y_{ij} = x \cdot y$, where $(x_{ij}, y_{ij}) = M_{ij}$.

if d+1=1 **then**

$M \leftarrow [(x_1, y_1)]$

else

$X^{(1)} \leftarrow (x_1, \ldots x_{(d+1)/2})$

$X^{(2)} \leftarrow (x_{(d+1)/2}, \ldots x_{d+1})$

$Y^{(1)} \leftarrow (y_1, \ldots y_{(d+1)/2})$

$Y^{(2)} \leftarrow (y_{(d+1)/2}, \ldots y_{d+1})$

$X^{(1,1)} \leftarrow \texttt{Refresh1}(X^{(1)})$; $X^{(1,2)} \leftarrow \texttt{Refresh2}(X^{(1)})$

$X^{(2,1)} \leftarrow \texttt{Refresh1}(X^{(2)})$; $X^{(2,2)} \leftarrow \texttt{Refresh2}(X^{(2)})$

$Y^{(1,1)} \leftarrow \texttt{Refresh1}(Y^{(1)})$; $Y^{(1,2)} \leftarrow \texttt{Refresh2}(Y^{(1)})$

$Y^{(2,1)} \leftarrow \texttt{Refresh1}(Y^{(2)})$; $Y^{(2,2)} \leftarrow \texttt{Refresh2}(Y^{(2)})$

$M^{(1,1)} = \texttt{MatGen}(X^{(1,1)}, Y^{(1,1)})$

$M^{(1,2)} = \texttt{MatGen}(X^{(1,2)}, Y^{(1,2)})$

$M^{(2,1)} = \texttt{MatGen}(X^{(2,1)}, Y^{(2,1)})$

$M^{(2,2)} = \texttt{MatGen}(X^{(2,2)}, Y^{(2,2)})$

$M \leftarrow \begin{bmatrix} M^{(1,1)} & M^{(1,2)} \\ M^{(2,1)} & M^{(2,2)} \end{bmatrix}$

end if

return M

Table 3.1.: Refreshing gadgets used for the SNI family of gadgets. BatRef is the SNI refresh gadget of Battistello et al. ([7], Algorithm 6). SimpleRef is a NI refresh using d random bits (Gadget 1 of [5]). The Identity gadget wires directly inputs to outputs.

	`Refresh1`	`Refresh2`
SNI-ISW	Identity	Identity
SNI-H	Identity	BatRef
SNI-H+	BatRef	BatRef
SNI-H*	SimpleRef	SimpleRef

SNI-H+ (Figure 3.3c), which instantiates Algorithm 7 with `Refresh1` and `Refresh2` as SNI refresh gadgets. (Table 3.1 summarizes the various instances of SNI multiplication gadgets.) In this scheme the number of manipulations of each variable is independent of the number of shares d, hence we can expect lower MI_t.

The performance of this gadget is shown in Figure 3.4. We observe that the curves for SNI-H+ are shifted to the right compared to the SNI-H curves. Whereas the boundary between the two regions of the SNI-H curves shifts to the left as the order increases (in a $\mathcal{O}(\log(d))$ manner), the location of the boundary is almost constant for SNI-H+ (except for a small shift at orders $d \leq 6$).

An interesting note here is that the SNI-H+ gadget is the first one we see for which increasing the order systematically increases the security level for all values of MI_o (except for $d \leq 7$ and $\mathrm{MI}_o \geq 5 \times 10^{-2}$ bit). This confirms the intuition that having a constant number of share manipulations translates in having a constant MI_o security threshold: the location of the transition between the two regions of the graph is almost independent of the order. We can draw a connection between these observations and the theory of noise rates [1]: the SNI-ISW multiplication has a $\mathcal{O}(1/d)$ noise rate, SNI-H exhibits a behavior of $\mathcal{O}(1/\log(d))$ noise rate, and we conjecture that SNI-H+ (and also SNI-H*, p. 63) has an effective $\mathcal{O}(1)$ noise rate.

Improved PINI gadgets. The principle of "embedded refresh" used for SNI-H+ can be combined with the "masked multiplication" trick of the PINI gadgets. This can be applied directly to the PINI_2 gadget (this gives PINI_2-H+) but the PINI_1 gadget requires some slight modifications[4] which gives PINI_3-H+ (Algorithm 8). The PINI_3 and PINI_3-H variants are of lesser interest and will not be discussed.

In Figure 3.5, we observe that the move from PINI_1 and PINI_2 to PINI_3-H+ and PINI_2-H+ has the general effect of shifting curves to the right (as it happens for the move from SNI-ISW to SNI-H+) while preserving the distinguishing features. In particular, the

[4] The negation of the inputs $\bar{a}$ cannot be computed once for each a_i due to the use of refreshed a_i. Instead, this must be computed before each product $R_{ij}(a_i, b_j)$. In order to avoid the $2(d+1)^2$ leakage on (refreshed) input shares (which decreases significantly the security level), we replace the evaluation of $r \cdot \bar{a}$ with $r \cdot a + r$ (which uses the same number of arithmetic operations but trades leakage on a for leakage on r).

Algorithm 8 Generalized PINI3 multiplication gadget

Require: shared factors $x, y \in \mathbb{F}_q^{d+1}$ such that $\sum_i x_i = x^*$ and $\sum_i y_i = y^*$
Ensure: output $c \in \mathbb{F}_q^{d+1}$ such that $\sum_i c_i = x^* \cdot y^*$

$M \leftarrow \mathsf{MatRef}((x_0, \ldots, x_d), (y_0, \ldots, y_d))$ {See Algorithm 7.}
for $i = 0$ to d **do**
  for $j = i + 1$ to d **do**
    $r_{ij} \xleftarrow{\$} \mathbb{F}_q$
    $r_{ji} \leftarrow r_{ij}$;
  end for
end for
for $i = 0$ to d **do**
  for $j = 0$ to d **do**
    $(x_{ij}, y_{ij}) \leftarrow (M)_{ij}$
    if $i \neq j$ **then**
      $z_{ij} \leftarrow (x_{ij} \cdot r_{ij} + r_{ij}) + x_{ij} \cdot (y_{ij} + r_{ij})$
Ensure: $z_{ij} = r_{ij} + x_{ij} \cdot y_{ij}$
    end if
  end for
end for
for $i = 0$ to d **do**
  $c_i \leftarrow x_{ii} \cdot y_{ii} + \sum_{j=0, j\neq i}^{d} z_{ij}$
end for

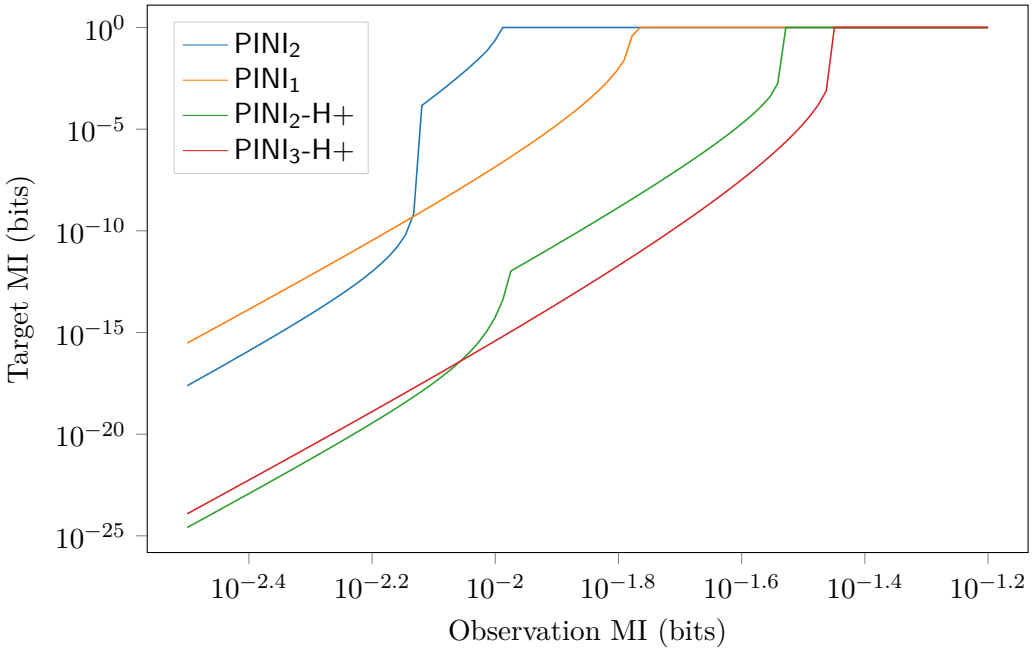


Figure 3.5.: Target MI as a function of observation MI, for various PINI multiplication gadgets at order $d = 15$.

PINI_2-H+ curve still has the "staircase of asymptotes" aspect which reduces significantly the security level in the region of most interest (large MI_o), we will not discuss it further.

In Figure 3.6, we observe that the asymptotic behavior of the PINI_3-H+ gadget is identical to the one of SNI-H+. The location of the transition between regions is also almost independent of the order. The main difference between the curves for the two gadgets is that the PINI_3-H+ curve is shifted of a factor of 2 to the left compared to SNI-H+, due to more manipulations of the shares (required by the "masked multiplication" trick), as it is the case when comparing PINI_1 and SNI-ISW.

The greedy strategy is another way to build a PINI multiplication gadget (see Sections 2.1 and 2.3.4): the PINI multiplication gadget is based on a SNI multiplication gadget for which one of the inputs is refreshed (with a SNI refresh). We can instantiate this idea with the SNI-ISW gadget and the refresh of Battistello et al., we call it GreedyMult: $\mathsf{GreedyMult}(x, y) = \mathsf{SNI\text{-}ISW}(\mathsf{BatRef}(x), y)$ (and its variations such as GreedyMult-H+ which uses SNI-H+, etc.). The performance GreedyMult-H+ is shown in Figure 3.6: as expected, it is the same as that of SNI-H+ (except for low orders and large MI_o) since it based on SNI-H+ and the refresh gadget has a low leakage. In particular, GreedyMult-H+ has a gain of a factor of 2 for MI_o compared to PINI_3-H+ since it manipulates its shares only one instead of twice in the `Product` stage.[5]

[5] The GreedyMult gadget is not discussed in detail in the following since it is almost identical to the SNI-ISW gadget, except for its cost (Section 3.5) and that it is PINI.

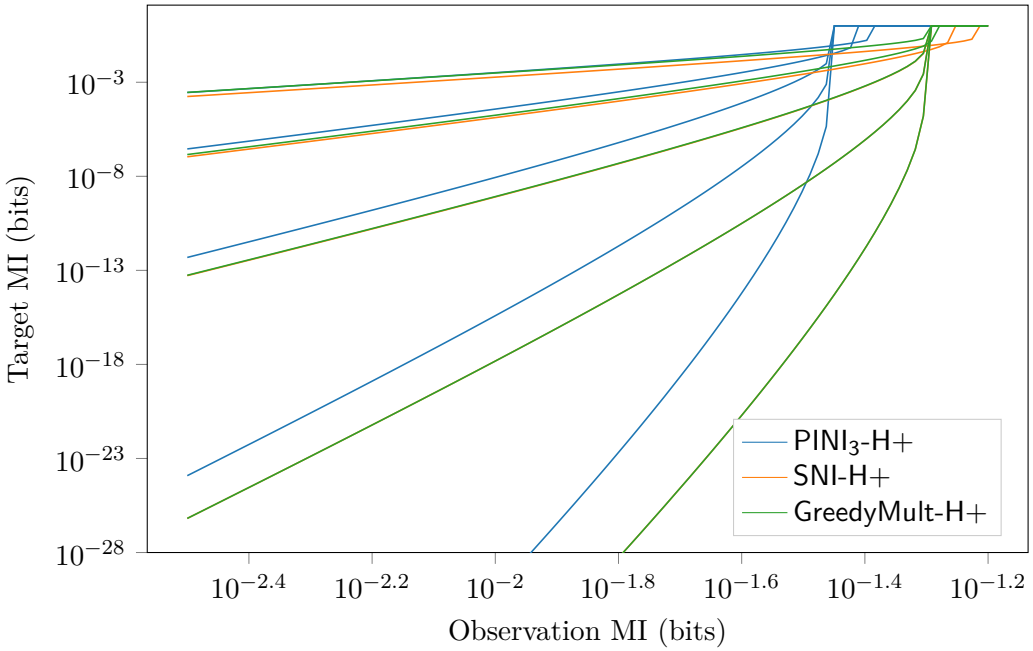


Figure 3.6.: Target MI as a function of observation MI, for the SNI-H+, PINI$_3$-H+ and GreedyMult-H+ multiplication gadget at orders $d = 3, 7, 15, 31$. The curves for SNI-H+ and GreedyMult-H+ are partially superimposed.

Security vs randomness cost trade-off. Battistello et al. use a SNI refresh with randomness complexity $\mathcal{O}(d \log(d))$ in the `MatGen` stage. Since the refresh is not used to prove composability in the d-probing model, the SNI property is not required.[6] We hence analyze the case where this SNI refresh (Algorithm 6 of [7]) is replaced with a simple refresh using d random bits (Gadget 1 of [5]): SNI-H* and PINI$_3$-H* (adapted versions of SNI-H+ and PINI$_3$-H+, see Table 3.1).

We observe that the security level of the these new gadgets is almost the same as the one of the gadgets ones based on the refresh of Battistello et al. (see Figure 3.7).

Regarding the randomness cost of the gadgets, all the gadgets have a randomness complexity of $\mathcal{O}(d^2)$ (this is immediately visible for the `Compression` stage, and proven in Appendix C for the H+ `MatGen`), and the use of H* `MatGen` versus H+ results in a reduced randomness cost of about 50 %. We defer detailed performance comparison to Section 3.5.

Multi-sharing Sensitive Variable Extraction At this point, we empirically validate the argument of Section 3.1 that it is useful to add equations about the input/output relationships of refresh gadgets. Removing those relationships from the factor graph

[6] We note that this property might help to formally prove the counter-measure, but it is out of the scope of this article.

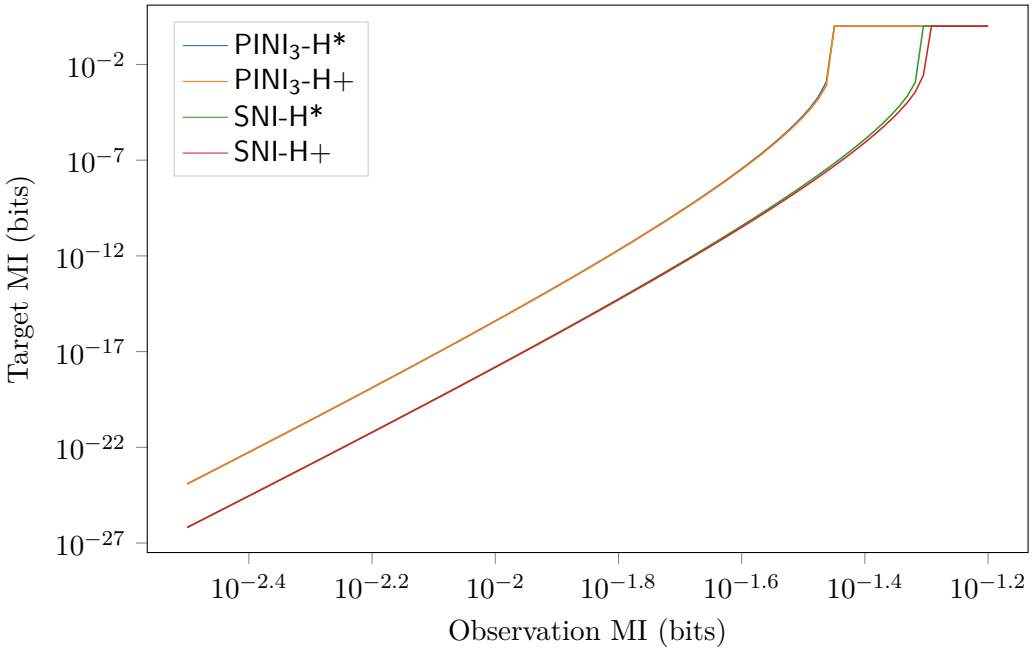


Figure 3.7.: Target MI as a function of observation MI, for the SNI-H+, SNI-H*, PINI$_3$-H+ and PINI$_3$-H* multiplication gadgets at order $d = 15$.

gives the curve "SNI-H+ naive" of Figure 3.8[7] We observe that the naive attack causes a significant loss to the adversary: a shift of the curve of a factor of 4 to the left (this shift depends on the order and the technique of protection against horizontal attacks used).

The conclusion is that this technique should be considered when running SASCA attacks, as it is very powerful and does not add stronger hypotheses on the capabilities of the adversary.[8]

3.5. Implementation Cost

We evaluate the runtime cost of the multiplication gadget in Figure 3.9, using the estimation framework of Section 2.4.

Four main groups of curves can be distinguished on the plot:

- the gadgets not protected against horizontal attacks: SNI-ISW, PINI$_1$, PINI$_2$, Greedy-Mult;
- the H+ gadgets, strongly protected against horizontal attacks with a cost up to six times larger than the SNI-ISW multiplication gadget;

[7] We show the SNI-H+ gadget as it is the one that is the most sensitive to the removal of the input/output refresh relationships (the impact is similar for PINI$_2$-H+ and PINI$_3$-H+).

[8] The only disadvantage of this technique is that it adds cycles in the factor graph, which might harm the convergence of the belief propagation algorithm.

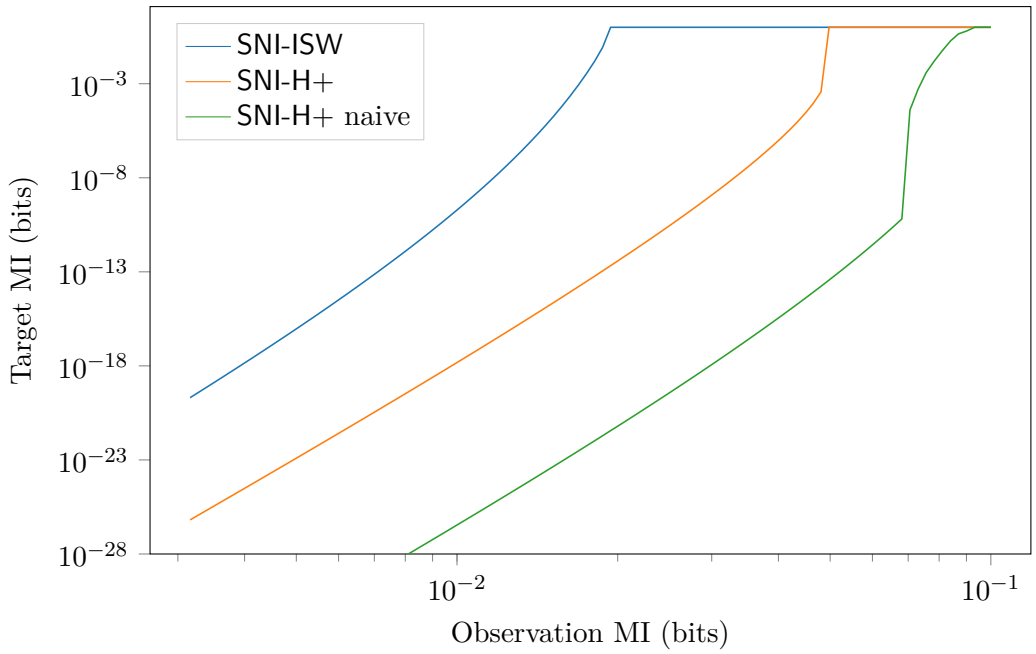


Figure 3.8.: Target MI as a function of observation MI, for the SNI-H+ multiplication gadget at order $d = 15$. For the SNI-H+ naive curve, the target MI is extracted through the sums of the input and output shares only (naive attack). For the SNI-H+ curve, the improved attack is considered (equations about sums of refresh input/outputs are added). The SNI-ISW curve (which is the same for both attacks) is also shown as a reference point.

- the H* gadgets, costing almost half the cost of the H+ gadgets (but which have almost the same security level as the H+ gadgets;
- the H gadgets which are only a bit less costly compared to the H* gadgets but have significantly worse resistance to horizontal attacks.

First, we note that the H and H+ gadget families are not very interesting at high order: the H* combines advantages of both, as it has roughly the same runtime cost as the H family and almost the same security level as the H+ family.

There are thus two main options: in order to get protection against horizontal attacks, the H* family should be chosen. If easy composition (i.e. PINI) is required, then SNI-H* is ruled out. The GreedyMult-H* is has a slightly higher runtime cost compared to PINI_3-H*, but it has a significant security advantage (it can tolerate a MI_o that is twice as large). If easy composition is not required, then the best choice is PINI_1 or PINI_2 depending on the order.

Finally, we note that our results are limited to one multiplication gadget and have no proven composability properties, they are thus only valid in the heuristic local random probing model. Extending the belief propagation algorithm to a whole block cipher is computationally expensive, thus finding a way to have composability is an interesting open question.

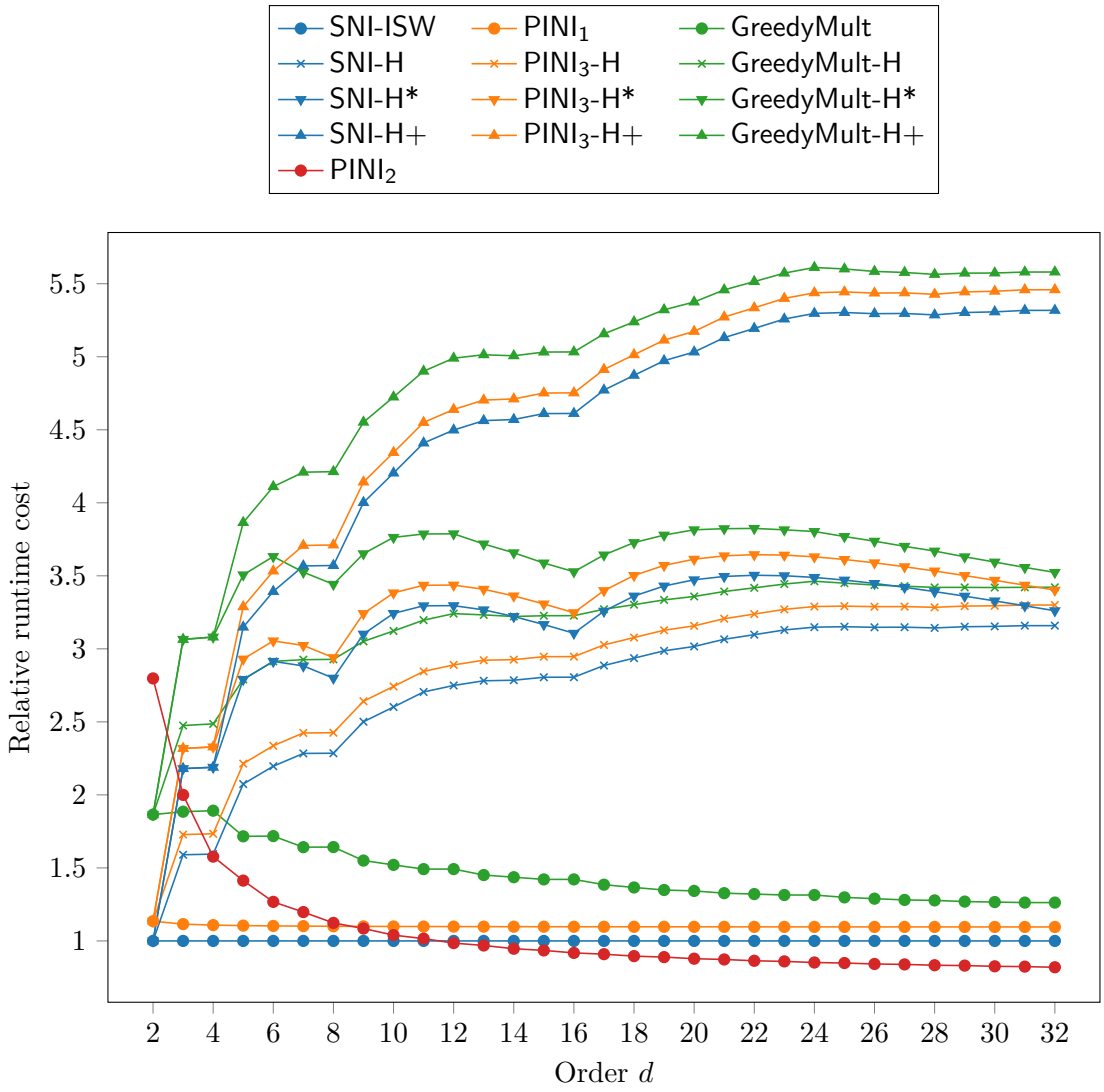


Figure 3.9.: Runtime cost (relative to the SNI-ISW multiplication gadget) of masked multiplication gadgets.

Conclusion

In this thesis, we analyze and improve the security and the performances of masked implementations in two different models: the probing model and the noisy leakage model, in the context of a multi-model approach of side-channel security. In the probing model, the composability properties of the new PINI definition greatly simplify the building of complex circuit, and our new constructions lead to a runtime reduction of about 40 % at any security level compared to the state-of-the-art. In the noisy leakage model, we investigate the impact of horizontal attacks thanks to the LRPM. From a theoretical viewpoint, we show that security against horizontal attacks becomes increasingly important as the number of shares (and claimed security order) in a masking scheme increases, and optimizations based only on reducing the randomness complexity are not sufficient in this context, joint randomness and security level optimization has to be performed. From a practical viewpoint, we show that horizontal attacks are relevant as soon as the implementation is secure at high order (typically, $d \geq 6$), and can cause an exponential security reduction if the noise rate is not adapted. Furthermore, we prove the soundness of the countermeasure of Battistello et al., and we design new countermeasures that improve the security level and/or reduce the randomness complexity (hence the execution time). In particular, we build a gadget that has a constant noise rate (for a runtime cost about 6 times larger that the one of a gadget not protected against horizontal attacks), which allows to increase the masking order without increasing the physical noise. The main consequence of this is that it is possible to increase the security of systems against side-channel attacks without requiring changes to the physical protection: algorithmic changes are sufficient.

Our investigations lead to important questions regarding the concrete relevance of (horizontal) side-channel attacks and the adversarial capabilities that are considered as realistic. For example, horizontal attacks require first to extract information for each of the targeted intermediate variables (at least $d + 1$ shares), which leads us to distinguish two main cases: the open-source setting (i.e., when all implementation details are given to the adversary) and the closed-source/black-box setting (i.e., when these implementation details are not public). In the open source setting, applying the horizontal attacks is direct with SASCA.

If the design is closed-source, the adversary can only do black-box profiling. It is thus not obvious that the adversary can solve those two prerequisites: first build the factor graph (i.e. identify the algorithm used) and second identify the leakage points for each manipulation of each variable. It might thus seem that the combination of the shares' leakages that is the root of the exponential security reduction implied by horizontal attacks requires knowing the code source for being easily exploitable (this is the case in [7, 33]). However, we have seen that the main weakness to horizontal attacks is repeated

manipulation of the same variables. Focusing on the second prerequisite, we first note that if the adversary is not able to mount a higher order attack, the d-probing security is sufficient [4]. However, if higher order attacks are possible (i.e. the adversary can identify at least one leakage point for at last $d+1$ intermediate variables), then they can easily identify all the leakage points of each of those variables since the leakages at those points are correlated. The full SASCA attack is not applicable, but a simple higher-order attack that reduces noise by exploiting the leakage of all the manipulation of the shares[1] can already achieve exponential security reduction. The MI_t – MI_o curves however might be different (i.e. give better security levels) in this setting, see for example the discussion of page 63).

Nevertheless, ignoring horizontal side-channel attacks would be a risky choice since it would imply considering the implementation details as a long-term secret (while standard countermeasures against side-channel attacks do not protect against reverse-engineering). Yet, it may be relevant for some applications with high cost constraints, where moderate / mid-term security is acceptable.

Finally, our work brings two open problems. First, our conclusions regarding horizontal attacks are based on quantitative but heuristic evaluations in the LRPM. Those evaluations are based on the assumption that it is impossible to mount attacks that exploit global information without incurring the cost of a full Bayes computation. It would be interesting to either confirm or invalidate this assumption. That is, obtaining tight proofs in the noisy leakage model or finding attacks (perhaps exploiting machine learning / deep learning [43]) that are computationally tractable and have better efficiency than SASCA.

Second, our evaluations in the LRPM are limited to one gadget, and they do not have simple composability properties such as the ones we have in the probing model. Finding composability results in a model where horizontal attacks are considered (such as the noisy leakage model or the random probing model) is an open problem.

[1] Indeed, only the manipulation of the targeted shares themselves can be used in this setting and not, as it would be the case in open-source setting, values that are in bijection with the shares (such as those that appear when a linear operation is applied to the sharing).

A. BP Rule for Result of Multiplication

We first recall the formal context of the random probing model (RPM) in which the rules of the LRPM are inferred. In the random probing model, the adversary gets for each intermediate variable X a probe P such that

$$P = \begin{cases} \mathrm{X} & \text{with probability } \epsilon \\ \perp & \text{with probability } 1-\epsilon \end{cases},$$

and all probes are independent ($\perp$ means no information).

We can thus compute the MI for this probe:

$$\begin{aligned}
\mathrm{MI}(P,X) &= \sum_p \sum_x \Pr[X=x, P=p] \log_{|\mathbb{F}|}\left(\frac{\Pr[X=x,P=p]}{\Pr[X=x]\Pr[P=p]}\right) \\
&= \sum_x \Pr[X=x, P=\perp] \log_{|\mathbb{F}|}\left(\frac{\Pr[X=x,P=\perp]}{\Pr[X=x]\Pr[P=\perp]}\right) \\
&\quad + \sum_x \Pr[X=x, P=x] \log_{|\mathbb{F}|}\left(\frac{\Pr[X=x,P=x]}{\Pr[X=x]\Pr[P=x]}\right) \\
&= (1-\epsilon)\sum_x \Pr[X=x] \log_{|\mathbb{F}|}\left(\frac{\Pr[X=x]\,\epsilon}{\Pr[X=x]^2\,\epsilon}\right) \\
&= \epsilon H(X) \\
&= \epsilon = \Pr[P=X]
\end{aligned}$$

assuming that X has a uniform distribution.

Let us now assume that $Y = X_1 \cdot X_2$, and that P_1, P_2 are probes of X_1, X_2 (Y is not directly probed). We write $\mathrm{MI}_1 = \mathrm{MI}(P_1, X_1)$ and $\mathrm{MI}_2 = \mathrm{MI}(P_2, X_2)$. The extrinsic information on Y is $\mathrm{MI}_Y = \mathrm{MI}(Y, (P_1, P_2))$. The value of Y is known if either $P_1 = X_1 = 0$, or $P_2 = X_2 = 0$, or $(P_1, P_2) = (X_1, X_2)$, in those cases $\mathrm{MI}_Y \leq 1$. Otherwise, there is not information on Y: $\mathrm{MI}_Y = 0$.

We hence compute

$$\begin{aligned}
\mathrm{MI}_Y &\leq \Pr[P_1 = X_1] \cdot (\Pr[X_1 = 0] + \Pr[X_1 \neq 0] \cdot \Pr[P_2 = X_2]) \\
&\quad + \Pr[P_1 \neq X_1] \cdot \Pr[P_2 = X_2] \cdot \Pr[X_2 = 0] \\
&= \mathrm{MI}_1 \left(\frac{1}{|\mathbb{F}|} + \left(1 - \frac{1}{|\mathbb{F}|}\right)\mathrm{MI}_2\right) + (1 - \mathrm{MI}_1)\,\mathrm{MI}_2\,\frac{1}{|\mathbb{F}|} \\
&= \frac{\mathrm{MI}_1 + \mathrm{MI}_2}{|\mathbb{F}|} + \mathrm{MI}_1\,\mathrm{MI}_2\left(1 - \frac{2}{|\mathbb{F}|}\right).
\end{aligned}$$

In the bitslice case, $|\mathbb{F}| = 2$, thus $\mathrm{MI}_Y \leq (\mathrm{MI}_1 + \mathrm{MI}_2)/2$, which is the result announced in Section 1.5.2.

B. PINI$_2$ Security Proof

In order to prove that the multiplication gadget described in Algorithm 5 is PINI, we prove that for each set of adversarial probes, there is a set $I \cup J \subset \{0, \ldots, d\}$ of share indices whose size is at most the number of probes and that knowledge of the input shares with those indices allows to simulate the probes. Section B.1 describes the algorithm to build the sets I and J and Section B.2 describes the simulation algorithm.

The values that can be probed are: a_i, b_i, s_i, $r_{i,j}$, r_j, $s_{i,j}$, $p^k_{i,j}$, $t_{i,j}$, $c_{i,j}$ and intermediate values in the computation of $p^k_{i,j}$ and $t_{i,j}$.

When discussing probes over $t_{i,j}$ and their intermediate values, we distinguish two cases: probes over intermediate values which do not involve r_{j-1} (hence sums of at most five terms, including $t_{i,i+1}$) denoted collectively $t^-_{i,j}$, and probes which involve r_{j-1} (sums of strictly more than five terms and $t_{i,j}$ for $j \neq i+1$), denoted collectively $t^+_{i,j}$. Probes over any $s_{i,j}$ or intermediate values of $p^k_{i,j}$ (including $p^k_{i,j}$ itself) are denoted collectively $p^*_{i,j}$.

B.1. Building I

Let I be the empty set. For each adversarial probe, we add at most one element (a share index) to I.

Part 1: global probes. We cover here probes a_i, b_i, s_i, $r_{i,j}$, r_j, $t^-_{i,j}$, $t^+_{i,j}$, $c_{i,j}$.

1. For any observed variable $c_{i,j}$ (which includes $c_i = c_{i,0}$), if $i \notin I$ then add i to I, else if $i > 0$ then add $i-1$ to I. That step ensures that the set I satisfies the PINI condition that the input shares required by the simulator for a given output probe have the same share index.

2. For any observed variable a_i or b_i or s_i, add i to I.

3. For any observed variable r_j add j to I.

4. For any observed variable $r_{i,j}$, if $i \notin I$ add i to I, else if $j \notin I$, add j to I. Otherwise add $j-1$ to I.

5. For any observed $t^-_{i,j}$, if $i \notin I$, add i to I, else if $j \notin I$, add j to I. Otherwise add $j-1$ to I.

6. Iterate the following until all observed $t^+_{i,j}$ have been processed:

- Select an observed and not yet processed $t^+_{i,j}$ such that $i \in I$. If there is no such $t^+_{i,j}$, select an observed and not yet processed $t^+_{i,j}$ such that there are two observed $t^+_{i,j}$. If there is no such $t^+_{i,j}$, select any observed and not yet processed $t^+_{i,j}$.
- Process the selected $t^+_{i,j}$, which means: if $j-1 \notin I$, add $j-1$ to I. Otherwise, if $i \notin I$, add i to I, otherwise add j to I.

Part 2: locals. We cover the remaining probes: $p^*_{i,j}$.

If there are at least two probes in $p^*_{i,j}$ for a given pair (i,j), then add i and j to I, and these probes are ignored for the rest of the simulation algorithm.

Let G be a graph whose nodes are all the share indices i. There is an edge between i and j if there is a probe on $p^*_{i,j}$ (there is only one probe, since we previously handled the case where $p^*_{i,j}$ is probed more than once). For each connected component G' of G, if G' contains a node which is in I or if G' contains a cycle then add all the nodes of G' to J. Otherwise, for each edge (i,j) in G', add i to J.

It can be shown that the number of elements in J is at most the number of probes $p^*_{i,j}$ thanks to the following property.

Property B.1. *For any connected graph with v vertices and e edges, $v \leq e+1$ with equality only if the graph does not contain any cycle (i.e. is a tree).*

The required input shares are $I \cup J$. This set satisfies the PINI conditions since its size is at most the number of probes and for each output probe c_i, $i \in I \cup J$.

B.2. Simulation

We now prove that a simulator knowing the input shares with share index in $I \cup J$ can perfectly simulate the probes.

We first show how to simulate the global probes (a_i, b_i, s_i, $r_{i,j}$, r_j, $t^-_{i,j}$, $t^+_{i,j}$, $c_{i,j}$), and then show how to simulate the local probes ($p^*_{i,j}$).

B.2.1. Global Probes

Observations. Before simulating, we make the following observations:

(i) all variables whose expression involves $r_{i,j}$ are $r_{i,j}$, $t^-_{i,j}$, $t^+_{i,j}$, $c_{i,k}$, $c_{j,k}$.
(ii) all variables whose expression involves r_{j-1} are r_{j-1}, $t^+_{k,j}$, $c_{j-1,k}$, $c_{k,l}$.
(iii) if two variables of the form $t^+_{i,j}$ are probed, then $i, j-1 \in I$.
(iv) if $k \in I$ before step 6 and $t^+_{i,j}$ and $t^+_{k,j}$ are probed, then $i, k, j-1 \in I$.
(v) if any two of i, k, j, $j-1$ are in I before step 6 and $t^+_{k,j}$ and $t^+_{i,j}$ is probed, then $i, k, j, j-1 \in I$.
(vi) if $k \in I$ or $j \in I$ before step 6 and $t^+_{k,j}$ is probed and $t^+_{i,j}$ is probed twice, then $i, k, j, j-1 \in I$.
(vii) if $t^+_{k,j}$ is probed twice and $t^+_{i,j}$ is probed twice, then $i, k, j, j-1 \in I$.

Description of the simulation algorithm. Any probe on a_i or b_i can be assigned to the correct value. For any probe on variables s_i, r_j or $r_{i,j}$, the variable is assigned to a fresh random, as it is the case in the real algorithm.

For $t^-_{i,j}$ probes, if $i, j \in I \cup J$, then the probe can be computed as it is the case in the real algorithm. Otherwise, it is assigned a fresh random.

For $t^+_{i,j}$ probes, if $i, j, j-1 \in I \cup J$, then the probe can be computed as it is the case in the real algorithm. Otherwise, it is assigned a fresh random.

For $c_{i,j}$ probes, the intermediates $t_{i,k}$, $r_{k,i}$ and r_i are simulated as described above and the sum is computed.

Indistinguishability proof. The only cases where the simulation is different from the real algorithm is the simulation of values $t^-_{i,j}$ for which i or j is not in $I \cup J$ and the simulation of $t^+_{i,j}$ for which i or j or $j-1$ is not in $I \cup J$. In those two cases, we show that $t^{\pm}_{i,j}$ appear to the adversary as randoms independent of any other input and probe.

Furthermore, probes on $c_{i,j}$ may involve $t_{i,k}$ for which i, k or $k-1$ is not in $I \cup J$. We prove that any $t_{i,k}$ can be perfectly simulated (i.e. is independent of any input and any probed except $c_{i,j}$ if i, k or $k-1$ is not in I) if $c_{i,j}$ is probed.

This proves indistinguishability of the whole simulation.

Independence of $t^-_{i,j}$. The variable $t^-_{i,j}$ (hereafter denoted t) involves the random $r_{i,j}$. We show that if $r_{i,j}$ is not independent of all the other probes and inputs, then $i, j \in I \cup J$. Using Observation (i), we analyze the possibilities for probes on values which depend on $r_{i,j}$.

- If $r_{i,j}$ is probed: step 4 (for $r_{i,j}$) of the algorithm building I adds[1] i to I and step 5 (for t) adds j.
- If $r_{i,j}$ appears in probed $c_{i,k}$: step 1 (for $c_{i,k}$) adds i to I and step 5 (for t) adds j.
- If $r_{i,j}$ appears in probed $c_{j,k}$: step 1 (for $c_{i,k}$) adds j to I and step 5 (for t) adds i.
- If $r_{i,j}$ appears in another probed $t^-_{i,j}$ (denoted t'): step 5 for t and t' adds i and j to I.
- If $r_{i,j}$ appears in a probed $t^+_{i,j}$ (denoted t'): step 5 for t adds i to I. However, t' involves the random r_{j-1}. Observation (ii) gives the variables which depend on r_{j-1}.
  - If r_{j-1} is probed, then step 3 adds $j-1$ to I, hence $i, j \in I$ (j is added by step 6 for t').
  - If $c_{j-1,k}$ is probed, then step 1 adds $j-1$ to I, hence $i, j \in I$.
  - If a $t^+_{k,j}$ (denoted t'') is probed, then step 6 for t' adds $j-1$ or j and step 6 for t'' adds $j-1$, k or j.

[1] All occurrences of "adds" in actually mean "adds if not yet added"

The variable t'' involves the random $r_{k,j}$. Using again Observation (i), we can analyze the probes which depend on $r_{k,j}$.

  * If $r_{k,j}$, $t^-_{k,j}$ or $c_{k,l}$ is probed, then k is added to I before step 6, which implies that step 6 for t' and t'' adds $j-1$ and j to I.
  * If another $t^+_{k,j}$ is probed (denoted t'''), then step 6 ensures $k, j, j-1 \in I$.
  * If $c_{j,l}$ is probed, then $j \in I$.
  * Otherwise, no such variable is probed, hence t'' is seen as a random independent of r_{j-1}, which in turn implies that t' is seen as a random independent of $r_{i,j}$ and thus t is also independent.

- If r_{j-1} appears in a probed $c_{k,l}$ (with $k \neq i$, $k \neq j-1$: those case have already been discussed), $r_{k,j}$ also appears in $c_{k,l}$. Step 1 for $c_{k,l}$ adds k to I and step 5 for t adds i. We can now discuss the probes which depend on $r_{k,j}$.
  * If $r_{k,j}$ or $t^-_{k,j}$ is probed, then j is added to I.
  * If $c_{k,l'}$ is probed then either $c_{k,l}$ is a sub-sum of $c_{k,l'}$ or the other way around. In any of these situations, these sums only observe $r_{k,j} + r_{j-1}$, hence r_{j-1} is independent of these sums in the view of the adversary, since $r_{k,j}$ is not observed elsewhere. This implies that t' is seen as independent of $r_{i,j}$ and thus t also looks independent.
  * If a $t^+_{k,j}$ (denoted t'') is probed, then step 6 for t' and t'' ensures $j, j-1 \in I$.
  * If $c_{j,l'}$ is probed, then $j \in I$.
  * Otherwise, no such variable is probed, hence $c_{k,l}$ is seen as a random independent of r_{j-1}, which in turn implies that t' is seen as a random independent of $r_{i,j}$ and thus t is also independent.
- Otherwise, no such variable is probed, hence t' is seen as a random independent of $r_{i,j}$, which in turn implies that t is seen as independent of any probe of input.

- Otherwise, no such variable is probed, hence $r_{i,j}$ (and thus t) is independent of all probes and inputs.

Independence of $t^+_{i,j}$. The variable $t^+_{i,j}$ (hereafter denoted t) involves the randoms $r_{i,j}$ and r_{j-1}. We show that if none of $r_{i,j}$ and r_{j-1} is independent of all the other probes and inputs, then $i, j, j-1 \in I \cup J$. Using Observations (i) and (ii), we analyze the possibilities for probes on values which depend on $r_{i,j}$ and r_{j-1}.

We look at probes on values which involve $r_{i,j}$:

- If $r_{i,j}$, $t^-_{i,j}$ or $c_{i,k}$ is probed, then $i \in I$
- If $t^+_{i,j}$ is probed, then $i, j-1 \in I$, thanks to Observation (iii).
- If $c_{j,k'}$ is probed, then $j \in I$.

- Otherwise, no such variable is probed, hence t contains an unobserved fresh random, and hence appears as a random independent of all probes and inputs.

  In the following, we will thus not consider anymore this possibility.

We look at probes on values which involve r_{j-1} (using Observation (ii)):

- If r_{j-1} or $c_{j-1,k}$ is probed, then step 3 or 1 adds $j-1$ to I, and then step 6 for t has the same behavior as if t would be a $t^-_{i,j}$ probe. The previous proof then applies.

- If there is a $t^+_{k,j}$ probe (denoted t'), then this probe involves $r_{k,j}$.
  - If $r_{k,j}$, $t^-_{k,j}$ or $c_{k,l}$ is probed, then $k \in I$ after step 5. We then analyze the probes for $r_{i,j}$.
    - If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or $c_{j,k'}$ is probed, then Observation (v) applies: $i, k, j, j-1 \in I$.
    - If $t^+_{i,j}$ is probed, then Observation (vi) applies.
    - Otherwise, $r_{i,j}$ is observed only through t, hence t is an independent random in the view of the adversary.
  - If $t^+_{k,j}$ is probed, then there are two $t^+_{k,j}$ probes.
    - If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or $c_{j,k'}$ is probed, then i or j is in I before 6 and Observation (vi) applies: $i, k, j, j-1 \in I$.
    - If $t^+_{i,j}$ is probed, then Observation (vii) applies.
    - Otherwise, $r_{i,j}$ is observed only through t, hence t is an independent random in the view of the adversary.
  - If $c_{j,l}$ is probed, then step 1 adds j to I.
    - If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or is probed, then $i, j \in I$ before step 6, hence step 6 for t adds $j-1$ to I.
    - If $t^+_{i,j}$ is probed, then Observation (iii) implies $i, j, j-1 \in I$.
    - If $c_{j,k'}$ then after step 1, $j, j-1 \in I$. Step 6 adds i to I.
    - Otherwise, $r_{i,j}$ is observed only through t, hence t is an independent random in the view of the adversary.
  - Otherwise, $r_{k,j}$ is observed only through t', hence r_{j-1} is independent of t'.

- If $c_{k,l}$ is probed, then $k \in I$ after step 1 and this probe involves $r_{k,j}$.
  - If $r_{k,j}$ or $t^-_{k,j}$ is probed, then $j \in I$ after step 5.
    - If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or is probed, then $i, j \in I$ before step 6, hence step 6 for t adds $j-1$ to I.
    - If $t^+_{i,j}$ is probed, then Observation (iii) implies $i, j, j-1 \in I$.
    - If $c_{j,k'}$ then after step 1, $j, j-1 \in I$. Step 6 adds i to I.

* Otherwise, $r_{i,j}$ is observed only through t, hence t is an independent random in the view of the adversary.

– If $t^+_{k,j}$ is probed:

* If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or $c_{j,k'}$ is probed, then Observation (v) applies: $i, k, j, j-1 \in I$.
* If $t^+_{i,j}$ is probed, then Observation (vi) applies.
* Otherwise, $r_{i,j}$ is observed only through t, hence t is an independent random in the view of the adversary.

– If $c_{j,l}$ is probed, then $j \in I$ before step 6.

* If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or is probed, then $i, j \in I$ before step 6, hence step 6 for t adds $j-1$ to I.
* If $t^+_{i,j}$ is probed, then Observation (iii) implies $i, j, j-1 \in I$.
* If $c_{j,k'}$ then after step 1, $j, j-1 \in I$. Step 6 adds i to I.
* Otherwise, $r_{i,j}$ is observed only through t, hence t is an independent random in the view of the adversary.

– Otherwise, $r_{k,j}$ is observed only through $c_{k,l}$, hence r_{j-1} is independent of $c_{k,l}$.

- Otherwise, there is no probe on any value that involves r_{j-1} (except on t), then t is seen as an independent random.

Independence of $t_{i,j}$ if $c_{i,k}$ is probed. If $c_{i,k}$ is probed, then $i \in I$ (before step 4).

If $j \neq i+1$, the variable $t_{i,j}$ involves the randoms $r_{i,j}$ and r_{j-1}. If any of those is not observed through any probe, $t_{i,j}$ is independent of any other input or probe (except $c_{i,k'}$).

- If r_{j-1} is observed through r_{j-1}, $t^+_{k,j}$ or $c_{j-1,k'}$, then $j-1 \in I$.
  – If $r_{i,j}$ is observed through $r_{i,j}$, $t^-_{i,j}$ or $c_{j,l}$, then $j \in I$, which means $t_{i,j}$ can be simulated as it is done in the algorithm.
  – If $r_{i,j}$ is observed through $t^+_{i,j}$, then $j-1 \in I$. In this case, $r_{i,j}$ is only observed through the sum $r_{i,j} + p^0_{i,j} + p^1_{i,j} + p^2_{i,j} + p^3_{i,j}$, which itself looks like an independent random and the remaining part of $t_{i,j}$ can be simulated as it is done in the algorithm.
  – The other possibility for $r_{i,j}$ to be observed is a probe on $c_{i,l}$. In this case all observations of $r_{i,j}$ are through $t_{i,j}$, which can then be simulated as an independent random.

- If r_{j-1} is observed through a $c_{k',l}$, then $k' \in I$ and this $c_{k',l}$ probe involves $t_{k',j}$. This variable involves in turn $r_{k',j}$.
  – If $r_{k',j}$, $t^-_{k',j}$ or $c_{j,l'}$ is observed, then $k', i, j \in I$. We now list the possible probes for $r_{i,j}$.

* If $r_{i,j}$, $t^-_{i,j}$, $t^+_{i,j}$ or $c_{j,l}$ is probed, then $j-1 \in I$.
* The other possibility for $r_{i,j}$ to be observed is a probe on $c_{i,l}$. In this case all observations of $r_{i,j}$ are through $t_{i,j}$, which can then be simulated as an independent random.

– If $t^+_{k',j}$ is probed, then $j-1 \in I$. Possible probes for $r_{i,j}$:
* If $r_{i,j}$, $t^-_{i,j}$ or $c_{j,l}$ is probed, then $j-1 \in I$.
* If $t^+_{i,j}$ is probed, then Observation (v) applies.
* The other possibility for $r_{i,j}$ to be observed is a probe on $c_{i,l}$. In this case all observations of $r_{i,j}$ are through $t_{i,j}$, which can then be simulated as an independent random.

– Finally, if $r_{k',j}$ is only observed through a $c_{k',l'}$ (and $c_{k',l}$), then those observations only observe $r_{k',j}$ and r_{j-1} through the sum $r_{k',j} + r_{j-1}$, which implies that r_{j-1} is independent of any other probe or input, which proves independence of $t_{i,j}$.

We now discuss the simulation of $t_{i,i+1}$ if $c_{i,k}$ is probed. The variable $t_{i,i+1}$ involves the random $r_{i,i+1}$. If $r_{i,i+1}$ is probed through $r_{i,i+1}$, $t^-_{i,i+1}$, $c_{i+1,l}$, then $i, i+1 \in I$ and the simulation can be done as it is the case in the algorithm. If $r_{i,i+1}$ is probed through a $c_{i,k'}$ or not probed, then $r_{i,i+1}$ is only observed through $t_{i,i+1}$, which can thus be simulated as an independent random.

B.2.2. Local Probes

Description of the simulation algorithm. For probes on $p^*_{i,j}$ we always have $i \in I \cup J$. In the case where $j \in I \cup J$ or if the probe does not involve a_j or b_j, the simulation is done as it is the case in the real algorithm. Otherwise the term $a_j + s_{i,j}$ or $b_j + s_{i,j}$ or $s_{i,j}$ is taken as a fresh random, and the remaining part of the computations are done is it is the case in the real algorithm.

Indistinguishability proof. The only cases where the simulation is different from the real algorithm is the simulation of values $p^*_{i,j}$ which involve $a_j + s_{i,j}$ or $b_j + s_{i,j}$ or $s_{i,j}$ and for which j is not in $I \cup J$. In this case, we show that $a_j + s_{i,j}$ or $b_j + s_{i,j}$ or $s_{i,j}$ appears to the adversary as a random independent of any other input and probe, which proves the indistinguishability.

For locals $p^*_{i,j}$: if i or j is in I, then i and are in $I \cup J$, and a perfect simulation is trivial. Otherwise, if i and j are in J, a perfect simulation is also trivial.

The remaining case is $j \notin I \cup J$. This implies $i \notin I$ and $i \in J$.

Independence of $a_j + s_{i,j}$ or $b_j + s_{i,j}$ or $s_{i,j}$ for which $j \notin I \cup J$. We know that s_j can only be observed through s_j, $p^*_{k,j}$, $t^-_{k,j}$, $t^+_{k,j+1}$, $p^*_{j,k}$, $t^-_{j,k}$, $t^+_{j,k}$, and similarly for s_i.

For all probed $t^-_{\cdot,\cdot}$ and $t^+_{\cdot,\cdot}$ variables (denoted t), simulation of globals shows that either $j \in I$ (which implies that $i \in I \cup J$ and simulation of $p^*_{i,j}$ can be done as it is the case in the algorithm), or t is independent of any other probe or input, including $p^*_{i,j}$.

If s_i or s_j were observed, there would be i or j in I. Therefore, s_i and s_j are only observed through probes in $p^*_{i,k}$, $p^*_{k,i}$ and $p^*_{k,j}$ ($p^*_{j,k}$ being probed would imply $j \in J$).

Furthermore, there is at most one $p^*_{i,j}$ probe to be considered here for each (i, j), since otherwise there would be $i, j \in I \cup J$.

Summarizing, we have shown that for any $p^*_{i,j}$ probe (denoted p) for which i or j is not in $I \cup J$, the variable $s_{i,j}$ in p is independent of any probe or input except p and other $p^*_{k,l}$ probes.

We now prove the independence between all $p^*_{i,j}$ probes.

All probes $p^*_{i,j}$ possibly not independent of each other are in the same connected component G' of G (G is defined in Section B.1). This component is a tree (otherwise $i, j \in J$). Let k be a leave of the tree (which, by definition, is connected to only one edge). This means that the only observation of s_k goes through the edge connected to k (hence a probe $p^*_{k,l}$, or $p^*_{l,k}$, denoted p). The random $s_{k,l} = s_k + s_l$ is thus independent of any probe except p, and independent of the random s_l. This implies that s_l is independent of p.

The leaves of the tree and their incident edges can thus be safely ignored from now on. Removing them gives a new tree and the same reasoning can be applied to this new tree. Iterating this procedure completes the proof of independence.

C. SNI-H Costs $\Theta(d^2)$ Random Bits.

In this section, we compute precisely the randomness complexity of the multiplication gadget of Battistello et al. (SNI-H): $\Theta(d^2)$ (previous bound was $\mathcal{O}(d^2 \log d)$).

Let R_{d+1} be the randomness cost of the refresh of Battistello et al. for a $d+1$-sharing input (where $d+1$ is a power of 2). Let C_{d+1} be then randomness cost of the refreshing part of the SNI-H multiplication[1].

Inspection of the algorithm gives $C_2 = 0$ and $C_{d+1} = 4C_{(d+1)/2} + 4R_{(d+1)/2}$ (for $d+1 > 1$). Let us take $2^i = d+1$ and define $C'_i = C_{2^i}$ and $R'_i = R_{2^i}$. We can rewrite the recurrence equation as $C'_i = 4C'_{i-1} + 4R'_{i-1}$ (with $C'_1 = 0$), which gives

$$C'_i = \sum_{j=1}^{i-1} 4^{i-j} R'_j.$$

Using that[2] $R_{d+1} = (d+1)\left(\log_2(d+1) - \frac{1}{2}\right)$ and hence $R'_i = \left(i - \frac{1}{2}\right) 2^i$, we find

$$C'_i = 4^i \sum_{j=1}^{i-1} 2^{-j} \left(j - \frac{1}{2}\right) = 4^i \left(\sum_{j=1}^{i-1} j 2^{-j} - \frac{1}{2} \sum_{j=1}^{i-1} 2^{-j}\right).$$

Using the identities

$$\sum_{k=1}^{n} z^k = z \frac{1 - z^n}{1 - z}$$

and

$$\sum_{k=1}^{n} k z^k = z \frac{1 - (n+1)z^n + n z^{n+1}}{(1-z)^2},$$

we get

$$C'_i = 4^i \left(\frac{3}{2} - \frac{1+2i}{2^i}\right).$$

This implies

$$\frac{1}{2} 4^i \leq C'_i \leq \frac{3}{2} 4^i,$$

which leads to

$$\frac{1}{2}(d+1)^2 \leq C_{d+1} \leq \frac{3}{2}(d+1)^2$$

and to the conclusion $C_{d+1} = \Theta(d^2)$ (more precisely, $C_{d+1} = 3/2(d+1^2) - \mathcal{O}(d \log_2 d)$ whereas the `Compression` step costs $d(d+1)/2$ random bits).

We note that this computation gives also the cost of the refreshing part of SNI-H+, which is twice the refreshing cost of SNI-H.

[1] The non-refreshing part costs $d(d+1)/2$ random bits.

[2] This can be shown by building a recurrence equation thanks to inspection of the algorithm.

D. Optimized AES S-box Implementation

$\mathrm{R}(\cdot)$ means a SNI refresh gadget. The inputs are $x_0, \ldots, x_7$ and the outputs are $s_0, \ldots, s_7$.

Top linear layer:

$t_0 = x_1 + x_2$
$t_1 = x_4 + y_{12}$
$y_1 = t_0 + x_7$
$y_2 = y_1 + x_0$
$y_3 = y_5 + y_8$
$y_4 = y_1 + x_3$
$y_5 = y_1 + x_6$
$y_6 = y_{15} + x_7$
$y_7 = x_7 + y_{11}$
$y_8 = x_0 + x_5$
$y_9 = x_0 + x_3$
$y_{10} = y_{15} + t_0$
$y_{11} = y_{20} + y_9$
$y_{12} = y_{13} + y_{14}$
$y_{13} = x_0 + x_6$
$y_{14} = x_3 + x_5$
$y_{15} = t_1 + x_5$
$y_{16} = t_0 + y_{11}$
$y_{17} = y_{10} + y_{11}$
$y_{18} = x_0 + y_{16}$
$y_{19} = y_{10} + y_8$
$y_{20} = t_1 + x_1$
$y_{21} = y_{13} + y_{16}$

Middle non-linear layer:

$x_{7,0} = \mathrm{R}(x_7)$
$y_{1,0} = \mathrm{R}(y_1)$
$y_{2,0} = \mathrm{R}(y_2)$
$y_{3,0} = \mathrm{R}(y_3)$
$y_{4,0} = \mathrm{R}(y_4)$
$y_{5,0} = \mathrm{R}(y_5)$
$y_{6,0} = \mathrm{R}(y_6)$
$y_{7,0} = \mathrm{R}(y_7)$
$y_{8,0} = \mathrm{R}(y_8)$
$y_{9,0} = \mathrm{R}(y_9)$
$y_{10,0} = \mathrm{R}(y_{10})$
$y_{11,0} = \mathrm{R}(y_{11})$
$y_{12,0} = \mathrm{R}(y_{12})$
$y_{13,0} = \mathrm{R}(y_{13})$
$y_{14,0} = \mathrm{R}(y_{14})$
$y_{15,0} = \mathrm{R}(y_{15})$
$y_{16,0} = \mathrm{R}(y_{16})$
$y_{17,0} = \mathrm{R}(y_{17})$
$t_2 = y_{12,0} \cdot y_{15,0}$
$t_{2,0} = t_2$
$t_3 = y_{6,0} \cdot y_{3,0}$
$t_4 = t_3 + t_{2,0}$
$t_5 = y_{4,0} \cdot x_{7,0}$
$t_6 = t_5 + t_{2,0}$
$t_7 = y_{13,0} \cdot y_{16,0}$
$t_{7,0} = t_7$
$t_8 = y_{1,0} \cdot y_{5,0}$
$t_9 = t_8 + t_{7,0}$
$t_{10} = y_{7,0} \cdot y_{2,0}$
$t_{11} = t_{10} + t_{7,0}$
$t_{12} = y_{9,0} \cdot y_{11,0}$
$t_{13} = y_{17,0} \cdot y_{14,0}$
$t_{14} = t_{13} + t_{12}$
$t_{14,0} = t_{14}$
$t_{15} = y_{10,0} \cdot y_{8,0}$
$t_{16} = t_{15} + t_{12}$
$t_{17} = t_4 + t_{14,0}$
$t_{18} = t_6 + t_{16}$
$t_{19} = t_9 + t_{14,0}$
$t_{20} = t_{11} + t_{16}$
$t_{21} = t_{17} + y_{20}$
$t_{21,0} = \mathrm{R}(t_{21})$
$t_{22} = t_{18} + y_{19}$
$t_{22,0} = \mathrm{R}(t_{22})$
$t_{23} = t_{19} + y_{21}$
$t_{23,0} = t_{23}$
$t_{24} = t_{20} + y_{18}$
$t_{24,0} = \mathrm{R}(t_{24})$
$t_{25} = t_{21,0} + t_{22,0}$
$t_{26} = t_{23,0} \cdot t_{21,0}$
$t_{26,0} = \mathrm{R}(t_{26})$
$t_{27} = t_{24,0} + t_{26,0}$
$t_{27,0} = t_{27}$
$t_{28} = t_{27} \cdot t_{25}$
$t_{29} = t_{28} + t_{22}$
$t_{29,0} = \mathrm{R}(t_{29})$
$t_{29,1} = \mathrm{R}(t_{29})$
$t_{30} = t_{24,0} + t_{23,0}$
$t_{31} = t_{26,0} + t_{22,0}$
$t_{32} = t_{31} \cdot t_{30}$
$t_{33} = \mathrm{R}(t_{32}) + t_{24,0}$
$t_{33,0} = \mathrm{R}(t_{33})$
$t_{34} = t_{23,0} + t_{33}$
$t_{35} = t_{27,0} + t_{33,0}$
$t_{36} = t_{35} \cdot t_{24}$
$t_{36,0} = \mathrm{R}(t_{36})$
$t_{37} = t_{34} + t_{36,0}$
$t_{37,0} = \mathrm{R}(t_{37})$
$t_{38} = t_{27,0} + t_{36,0}$
$t_{39} = t_{38} \cdot t_{29,0}$
$t_{40} = t_{39} + t_{25}$
$t_{40,0} = \mathrm{R}(t_{40})$
$t_{41} = t_{37,0} + t_{40,0}$
$t_{41,0} = t_{41}$
$t_{42} = t_{29,1} + t_{33,0}$
$t_{42,0} = \mathrm{R}(t_{42})$
$t_{43} = t_{29,1} + t_{40,0}$
$t_{44} = t_{33,0} + t_{37}$
$t_{45} = t_{41,0} + t_{42}$
$t_{45,0} = t_{45}$
$z_0 = y_{15,0} \cdot \mathrm{R}(t_{44})$
$z_1 = t_{37,0} \cdot y_{6,0}$
$z_2 = x_{7,0} \cdot t_{33,0}$
$z_3 = y_{16,0} \cdot t_{43}$
$z_4 = y_{1,0} \cdot t_{40,0}$
$z_5 = t_{29,1} \cdot y_{7,0}$
$z_6 = t_{42,0} \cdot y_{11,0}$
$z_7 = t_{45,0} \cdot y_{17,0}$
$z_8 = t_{41,0} \cdot y_{10,0}$
$z_9 = y_{12,0} \cdot t_{44}$
$z_{10} = y_{3,0} \cdot t_{37}$
$z_{11} = y_{4,0} \cdot t_{33}$
$z_{12} = y_{13,0} \cdot t_{43}$
$z_{13} = y_{5,0} \cdot t_{40,0}$
$z_{14} = t_{29,0} \cdot y_{2,0}$
$z_{15} = t_{42,0} \cdot y_{9,0}$
$z_{16} = t_{45,0} \cdot y_{14,0}$
$z_{17} = y_{8,0} \cdot t_{41}$
$o_0 = z_0$
$o_1 = z_1$
$o_2 = z_2$
$o_3 = \mathrm{R}(z_3)$
$o_4 = \mathrm{R}(z_4)$
$o_5 = z_5$
$o_6 = \mathrm{R}(z_6)$
$o_7 = \mathrm{R}(z_7)$
$o_8 = \mathrm{R}(z_8)$
$o_9 = \mathrm{R}(z_9)$
$o_{10} = \mathrm{R}(z_{10})$
$o_{11} = z_{11}$
$o_{12} = \mathrm{R}(z_{12})$
$o_{13} = z_{13}$
$o_{14} = z_{14}$
$o_{15} = z_{15}$
$o_{16} = \mathrm{R}(z_{16})$
$o_{17} = \mathrm{R}(z_{17})$

Bottom linear layer:

$t_{46} = o_{15} + o_{16}$
$t_{47} = o_{10} + o_{11}$
$t_{48} = o_5 + o_{13}$
$t_{49} = o_9 + o_{10}$
$t_{50} = o_2 + o_{12}$
$t_{51} = o_2 + o_5$
$t_{52} = o_7 + o_8$
$t_{53} = o_0 + o_3$
$t_{54} = o_6 + o_7$
$t_{55} = o_{16} + o_{17}$
$t_{56} = o_{12} + t_{48}$
$t_{57} = t_{50} + t_{53}$
$t_{58} = o_4 + t_{46}$
$t_{59} = o_3 + t_{54}$
$t_{60} = t_{46} + t_{57}$
$t_{61} = o_{14} + t_{57}$
$t_{62} = t_{52} + t_{58}$
$t_{63} = t_{49} + t_{58}$
$t_{64} = o_4 + t_{59}$
$t_{65} = t_{61} + t_{62}$
$t_{66} = o_1 + t_{63}$
$t_{67} = t_{64} + t_{65}$
$s_0 = t_{59} + t_{63}$
$s_1 = \mathrm{Not}(t_{64} + s_3)$
$s_2 = \mathrm{Not}(t_{55} + t_{67})$
$s_3 = t_{53} + t_{66}$
$s_4 = t_{51} + t_{66}$
$s_5 = t_{47} + t_{65}$
$s_6 = \mathrm{Not}(t_{56} + t_{62})$
$s_7 = \mathrm{Not}(t_{48} + t_{60})$

References

[1] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. "Circuit Compilers with O(1/\log (n)) Leakage Rate". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 586–615. ISBN: 978-3-662-49895-8. DOI: 10.1007/978-3-662-49896-5_21. URL: https://doi.org/10.1007/978-3-662-49896-5_21.

[2] Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. "Inner Product Masking Revisited". In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 486–510. ISBN: 978-3-662-46799-2. DOI: 10.1007/978-3-662-46800-5_19. URL: https://doi.org/10.1007/978-3-662-46800-5_19.

[3] Josep Balasch et al. "Consolidating Inner Product Masking". In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 724–754. ISBN: 978-3-319-70693-1. DOI: 10.1007/978-3-319-70694-8_25. URL: https://doi.org/10.1007/978-3-319-70694-8_25.

[4] Gilles Barthe et al. "Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 535–566. ISBN: 978-3-319-56619-1. DOI: 10.1007/978-3-319-56620-7_19. URL: https://doi.org/10.1007/978-3-319-56620-7_19.

[5] Gilles Barthe et al. "Strong Non-Interference and Type-Directed Higher-Order Masking". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 116–129. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978427. URL: http://doi.acm.org/10.1145/2976749.2978427.

[6] Gilles Barthe et al. “Verified Proofs of Higher-Order Masking”. In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 457–485. ISBN: 978-3-662-46799-2. DOI: 10.1007/978-3-662-46800-5_18. URL: https://doi.org/10.1007/978-3-662-46800-5_18.

[7] Alberto Battistello et al. “Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme”. In: *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Vol. 9813. Lecture Notes in Computer Science. Springer, 2016, pp. 23–39. ISBN: 978-3-662-53139-6. DOI: 10.1007/978-3-662-53140-2_2. URL: https://doi.org/10.1007/978-3-662-53140-2_2.

[8] Sonia Belad, Dahmun Goudarzi, and Matthieu Rivain. “Tight Private Circuits: Achieving Probing Security with the Least Refreshing”. In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 439. URL: https://eprint.iacr.org/2018/439.

[9] Sonia Belad et al. “Private Multiplication over Finite Fields”. In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Springer, 2017, pp. 397–426. ISBN: 978-3-319-63696-2. DOI: 10.1007/978-3-319-63697-9_14. URL: https://doi.org/10.1007/978-3-319-63697-9_14.

[10] Sonia Belad et al. “Randomness Complexity of Private Circuits for Multiplication”. In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 616–648. ISBN: 978-3-662-49895-8. DOI: 10.1007/978-3-662-49896-5_22. URL: https://doi.org/10.1007/978-3-662-49896-5_22.

[11] Joseph Bonneau and Ilya Mironov. “Cache-Collision Timing Attacks Against AES”. In: *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. Lecture Notes in Computer Science. Springer, 2006, pp. 201–215. ISBN: 3-540-46559-6. DOI: 10.1007/11894063_16. URL: https://doi.org/10.1007/11894063_16.

[12] Joan Boyar, Philip Matthews, and René Peralta. “Logic Minimization Techniques with Applications to Cryptology”. In: *J. Cryptology* 26.2 (2013), pp. 280–312. DOI: 10.1007/s00145-012-9124-7. URL: https://doi.org/10.1007/s00145-012-9124-7.

[13] Billy Bob Brumley and Nicola Tuveri. “Remote Timing Attacks Are Still Practical”. In: *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*. Ed. by Vijay Atluri and Claudia Daz. Vol. 6879. Lecture Notes in Computer Science. Springer, 2011, pp. 355–371. ISBN: 978-3-642-23821-5. DOI: 10.1007/978-3-642-23822-2_20. URL: https://doi.org/10.1007/978-3-642-23822-2_20.

[14] Gaëtan Cassiers. *lrpm-bounder: Target MI bound computation in the LRPM*. original-date: 2018-06-03T21:13:03Z. June 3, 2018. URL: https://github.com/cassiersg/lrpm-bounder (visited on 06/03/2018).

[15] Gaëtan Cassiers. *mimopsni: Randomness cost optimization tool for masked S-Box implementation*. original-date: 2018-06-03T20:38:37Z. June 3, 2018. URL: https://github.com/cassiersg/mimopsni (visited on 06/03/2018).

[16] Suresh Chari et al. “Towards Sound Approaches to Counteract Power-Analysis Attacks”. In: *Advances in Cryptology - CRYPTO ’99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 398–412. ISBN: 3-540-66347-9. DOI: 10.1007/3-540-48405-1_26. URL: https://doi.org/10.1007/3-540-48405-1_26.

[17] Jean-Sébastien Coron and Jesper Buus Nielsen, eds. *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*. Vol. 10210. Lecture Notes in Computer Science. 2017. ISBN: 978-3-319-56619-1. DOI: 10.1007/978-3-319-56620-7. URL: https://doi.org/10.1007/978-3-319-56620-7.

[18] Jean-Sébastien Coron et al. “Higher-Order Side Channel Security and Mask Refreshing”. In: *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*. Ed. by Shiho Moriai. Vol. 8424. Lecture Notes in Computer Science. Springer, 2013, pp. 410–424. ISBN: 978-3-662-43932-6. DOI: 10.1007/978-3-662-43933-3_21. URL: https://doi.org/10.1007/978-3-662-43933-3_21.

[19] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4. URL: https://doi.org/10.1007/978-3-662-04722-4.

[20] Jean-François Dhem et al. “A Practical Implementation of the Timing Attack”. In: *Smart Card Research and Applications, This International Conference, CARDIS ’98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*. Ed. by Jean-Jacques Quisquater and Bruce Schneier. Vol. 1820. Lecture Notes in Computer Science. Springer, 1998, pp. 167–182. ISBN: 3-540-67923-5. DOI: 10.1007/10721064_15. URL: https://doi.org/10.1007/10721064_15.

[21] 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*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. Lecture Notes in Computer Science. Springer, 2014, pp. 423–440. ISBN: 978-3-642-55219-9. DOI: 10.1007/978-3-642-55220-5_24. URL: https://doi.org/10.1007/978-3-642-55220-5_24.

[22] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. "Making Masking Security Proofs Concrete - Or How to Evaluate the Security of Any Leaking Device". In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 401–429. ISBN: 978-3-662-46799-2. DOI: 10.1007/978-3-662-46800-5_16. URL: https://doi.org/10.1007/978-3-662-46800-5_16.

[23] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. "Amortizing Randomness Complexity in Private Circuits". In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 781–810. ISBN: 978-3-319-70693-1. DOI: 10.1007/978-3-319-70694-8_27. URL: https://doi.org/10.1007/978-3-319-70694-8_27.

[24] Sebastian Faust et al. "Composable Masking Schemes in the Presence of Physical Defaults and the Robust Probing Model". In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 711. URL: http://eprint.iacr.org/2017/711.

[25] Marc Fischlin and Jean-Sébastien Coron, eds. *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016. ISBN: 978-3-662-49895-8. DOI: 10.1007/978-3-662-49896-5. URL: https://doi.org/10.1007/978-3-662-49896-5.

[26] Guillaume Fumaroli et al. "Affine Masking against Higher-Order Side Channel Analysis". In: *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. Lecture Notes in Computer Science. Springer, 2010, pp. 262–280. ISBN: 978-3-642-19573-0. DOI: 10.1007/978-3-642-19574-7_18. URL: https://doi.org/10.1007/978-3-642-19574-7_18.

[27] Qian Ge et al. "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware". In: *J. Cryptographic Engineering* 8.1 (2018), pp. 1–27.

DOI: 10.1007/s13389-016-0141-6. URL: https://doi.org/10.1007/s13389-016-0141-6.

[28] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. "Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings". In: *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. Lecture Notes in Computer Science. Springer, 2011, pp. 240–255. ISBN: 978-3-642-23950-2. DOI: 10.1007/978-3-642-23951-9_16. URL: https://doi.org/10.1007/978-3-642-23951-9_16.

[29] Jovan Dj. Golic and Christophe Tymen. "Multiplicative Masking and Power Analysis of AES". In: *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 198–212. ISBN: 3-540-00409-2. DOI: 10.1007/3-540-36400-5_16. URL: https://doi.org/10.1007/3-540-36400-5_16.

[30] Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain. "How to Securely Compute with Noisy Leakage in Quasilinear Complexity". In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 929. URL: http://eprint.iacr.org/2017/929.

[31] Dahmun Goudarzi and Matthieu Rivain. "How Fast Can Higher-Order Masking Be in Software?" In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 567–597. ISBN: 978-3-319-56619-1. DOI: 10.1007/978-3-319-56620-7_20. URL: https://doi.org/10.1007/978-3-319-56620-7_20.

[32] Hannes GroSS, Stefan Mangard, and Thomas Korak. "An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order". In: *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*. Ed. by Helena Handschuh. Vol. 10159. Lecture Notes in Computer Science. Springer, 2017, pp. 95–112. ISBN: 978-3-319-52152-7. DOI: 10.1007/978-3-319-52153-4_6. URL: https://doi.org/10.1007/978-3-319-52153-4_6.

[33] Vincent Grosso and François-Xavier Standaert. "Masking Proofs Are Tight and How to Exploit it in Security Evaluations". In: *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 385–412. ISBN: 978-3-319-78374-1. DOI: 10.1007/978-3-319-78375-8_13. URL: https://doi.org/10.1007/978-3-319-78375-8_13.

[34] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. “Masking vs. multiparty computation: how large is the gap for AES?” In: *J. Cryptographic Engineering* 4.1 (2014), pp. 47–57. DOI: 10.1007/s13389-014-0073-y. URL: https://doi.org/10.1007/s13389-014-0073-y.

[35] Qian Guo, Vincent Grosso, and François-Xavier. *Modeling Soft Analytical Side-Channel Attacks from a Coding Theory Viewpoint*. Cryptology ePrint Archive, Report 2018/498. https://eprint.iacr.org/2018/498. 2018.

[36] Aric Hagberg, Pieter Swart, and Daniel S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[37] Yuval Ishai, Amit Sahai, and David A. Wagner. “Private Circuits: Securing Hardware against Probing Attacks”. In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 463–481. ISBN: 3-540-40674-3. DOI: 10.1007/978-3-540-45146-4_27. URL: https://doi.org/10.1007/978-3-540-45146-4_27.

[38] Anthony Journault and François-Xavier Standaert. “Very High Order Masking: Efficient Implementation and Security Evaluation”. In: *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 623–643. ISBN: 978-3-319-66786-7. DOI: 10.1007/978-3-319-66787-4_30. URL: https://doi.org/10.1007/978-3-319-66787-4_30.

[39] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. ISBN: 9781466570269.

[40] Paul C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *Advances in Cryptology - CRYPTO ’96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113. ISBN: 3-540-61512-1. DOI: 10.1007/3-540-68697-5_9. URL: https://doi.org/10.1007/3-540-68697-5_9.

[41] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis”. In: *Advances in Cryptology - CRYPTO ’99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397. ISBN: 3-540-66347-9. DOI: 10.1007/3-540-48405-1_25. URL: https://doi.org/10.1007/3-540-48405-1_25.

[42] Po-Chun Liu, Hsie-Chia Chang, and Chen-Yi Lee. “A Low Overhead DPA Countermeasure Circuit Based on Ring Oscillators”. In: *IEEE Trans. on Circuits and Systems* 57-II.7 (2010), pp. 546–550. DOI: 10.1109/TCSII.2010.2048400. URL: https://doi.org/10.1109/TCSII.2010.2048400.

[43] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. "Breaking Cryptographic Implementations Using Deep Learning Techniques". In: *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*. Ed. by Claude Carlet, M. Anwar Hasan, and Vishal Saraswat. Vol. 10076. Lecture Notes in Computer Science. Springer, 2016, pp. 3–26. ISBN: 978-3-319-49444-9. DOI: 10.1007/978-3-319-49445-6_1. URL: https://doi.org/10.1007/978-3-319-49445-6_1.

[44] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. "Side-Channel Leakage of Masked CMOS Gates". In: *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*. Ed. by Alfred Menezes. Vol. 3376. Lecture Notes in Computer Science. Springer, 2005, pp. 351–365. ISBN: 3-540-24399-2. DOI: 10.1007/978-3-540-30574-3_24. URL: https://doi.org/10.1007/978-3-540-30574-3_24.

[45] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. "Successfully Attacking Masked AES Hardware Implementations". In: *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. Lecture Notes in Computer Science. Springer, 2005, pp. 157–171. ISBN: 3-540-28474-5. DOI: 10.1007/11545262_12. URL: https://doi.org/10.1007/11545262_12.

[46] Stuart Mitchell, Michael OSullivan, and Iain Dunning. "PuLP: a linear programming toolkit for python". In: *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf* (2011).

[47] Amir Moradi and Oliver Mischke. "Glitch-free implementation of masking in modern FPGAs". In: *2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012*. IEEE, 2012, pp. 89–95. ISBN: 978-1-4673-2341-3. DOI: 10.1109/HST.2012.6224326. URL: https://doi.org/10.1109/HST.2012.6224326.

[48] Moni Naor, Omer Reingold, and Alon Rosen. "Pseudo-Random Functions and Factoring". In: *Electronic Colloquium on Computational Complexity (ECCC)* 8.064 (2001). URL: http://eccc.hpi-web.de/eccc-reports/2001/TR01-064/index.html.

[49] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches". In: *J. Cryptology* 24.2 (2011), pp. 292–321. DOI: 10.1007/s00145-010-9085-7. URL: https://doi.org/10.1007/s00145-010-9085-7.

[50] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache Attacks and Countermeasures: The Case of AES". In: *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*. Ed. by David Pointcheval. Vol. 3860. Lecture Notes in Computer Science. Springer, 2006, pp. 1–20. ISBN: 3-540-31033-9. DOI: 10.1007/11605805_1. URL: https://doi.org/10.1007/11605805_1.

[51] Elisabeth Oswald and Marc Fischlin, eds. *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015. ISBN: 978-3-662-46799-2. DOI: 10.1007/978-3-662-46800-5. URL: https://doi.org/10.1007/978-3-662-46800-5.

[52] Sri Parameswaran and Tilman Wolf. "Embedded systems security - an overview". In: *Design Autom. for Emb. Sys.* 12 (2008), pp. 173–183.

[53] Josef Pieprzyk. "How to Construct Pseudorandom Permutations from Single Pseudorandom Functions". In: *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*. Ed. by Ivan Damgård. Vol. 473. Lecture Notes in Computer Science. Springer, 1990, pp. 140–150. ISBN: 3-540-53587-X. DOI: 10.1007/3-540-46877-3_12. URL: https://doi.org/10.1007/3-540-46877-3_12.

[54] Emmanuel Prouff and Matthieu Rivain. "Masking against Side-Channel Attacks: A Formal Security Proof". In: *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 142–159. ISBN: 978-3-642-38347-2. DOI: 10.1007/978-3-642-38348-9_9. URL: https://doi.org/10.1007/978-3-642-38348-9_9.

[55] Chester Rebeiro, A. David Selvakumar, and A. S. L. Devi. "Bitslice Implementation of AES". In: *Cryptology and Network Security, 5th International Conference, CANS 2006, Suzhou, China, December 8-10, 2006, Proceedings*. Ed. by David Pointcheval, Yi Mu, and Kefei Chen. Vol. 4301. Lecture Notes in Computer Science. Springer, 2006, pp. 203–212. ISBN: 3-540-49462-6. DOI: 10.1007/11935070_14. URL: https://doi.org/10.1007/11935070_14.

[56] Matthieu Rivain and Emmanuel Prouff. "Provably Secure Higher-Order Masking of AES". In: *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer Science. Springer, 2010, pp. 413–427. ISBN: 978-3-642-15030-2. DOI: 10.1007/978-3-642-15031-9_28. URL: https://doi.org/10.1007/978-3-642-15031-9_28.

[57] Tsuyoshi Takagi and Thomas Peyrin, eds. *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017. ISBN: 978-3-319-70693-1. DOI: 10.1007/978-3-319-70694-8. URL: https://doi.org/10.1007/978-3-319-70694-8.

[58] Nicolas Veyrat-Charvillon, Benot Gérard, and François-Xavier Standaert. "Soft Analytical Side-Channel Attacks". In: *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. Lecture Notes in Computer Science. Springer, 2014, pp. 282–296. ISBN: 978-3-662-45610-1. DOI: 10.1007/978-3-662-45611-8_15. URL: https://doi.org/10.1007/978-3-662-45611-8_15.

[59] Nicolas Veyrat-Charvillon et al. "An Optimal Key Enumeration Algorithm and Its Application to Side-Channel Attacks". In: *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*. Ed. by Lars R. Knudsen and Huapeng Wu. Vol. 7707. Lecture Notes in Computer Science. Springer, 2012, pp. 390–406. ISBN: 978-3-642-35998-9. DOI: 10.1007/978-3-642-35999-6_25. URL: https://doi.org/10.1007/978-3-642-35999-6_25.

[60] Michael J. Wiener, ed. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999. ISBN: 3-540-66347-9. DOI: 10.1007/3-540-48405-1. URL: https://doi.org/10.1007/3-540-48405-1.

[61] Manfred von Willich. "A Technique with an Information-Theoretic Basis for Protecting Secret Data from Differential Power Attacks". In: *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings*. Ed. by Bahram Honary. Vol. 2260. Lecture Notes in Computer Science. Springer, 2001, pp. 44–62. ISBN: 3-540-43026-1. DOI: 10.1007/3-540-45325-3_6. URL: https://doi.org/10.1007/3-540-45325-3_6.

[62] Yongbin Zhou and Dengguo Feng. "Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing". In: *IACR Cryptology ePrint Archive* 2005 (2005), p. 388. URL: http://eprint.iacr.org/2005/388.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**