Composable and efficient masking schemes for side-channel secure implementations

Gaëtan Cassiers

June 2022

École polytechnique de Louvain
Université catholique de Louvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. David Bol	*UCLouvain*
Pr. Jean-Sébastien Coron	*University of Luxembourg*
Dr. François Dupressoir	*University of Bristol*
Pr. Sebastian Faust	*TU Darmstadt*
Pr. Stefan Mangard	*TU Graz*
Pr. Olivier Pereira	*UCLouvain*
Pr. François-Xavier Standaert	*UCLouvain*

Gaëtan Cassiers is a Research Fellow of the Belgian Fund for Scientific Research (FNRS-F.R.S.).

Acknowledgments

This thesis is the result of four years of research in the cryptography research group at UCLouvain. Research is sometimes depicted as a solitary activity, my experience has been quite the contrary, and I want to thank all the people who helped me during the last years.

A special thanks to my supervisor François-Xavier Standaert who introduced me to research and guided me in the discovery of the cryptologic research community. He has been very supportive of my work, balancing guidance towards interesting research topics and freedom to explore my own directions.

I am thankful to my jury members for reading this thesis and for discussing with me, both during the private defense and in other occasions. In particular, thanks to Jean-Sébastien Coron for providing me with much food for thought through his works, to Stefan Mangard and Sebastian Faust for being in my thesis committee, and to François Dupressoir for his thorough comments on the manuscript. Finally, I would like to thank David Bol for being the president of the jury, and to Olivier Pereira for having been an inexhaustible source of cryptographic questions and comments during the course of my thesis.

Thanks to all my colleagues for making my experience in the Crypto Group unforgettable. I am lucky to have met there many open-minded people with diverse backgrounds who where always ready for an enriching discussion (with a coffee or a beer), or for a refreshing sport session. I have also had the chance of co-authoring many research papers with colleagues from the Crypto Group, thanks for these collaborations and for all the things you taught me.

Lastly, I would like to thank my friends and my family. You endured bravely my obscure ideas on security and my frustrated rants on the publication process of academic journals and conferences.

Abstract

Modern cryptography has been widely deployed in the last decades, allowing any computing device to secure its communications. Facing the strength of the cryptographic algorithms, attackers turned to their implementations. In particular, side-channel attacks have been shown to threaten the security of cryptographic embedded devices by exploiting information leaked through physical characteristics such as power consumption or electromagnetic radiation. In order to mitigate side-channel attacks, implementations use countermeasures, among which masking is the most investigated choice.

In this thesis, we design and analyze masking schemes with the goal of having efficient masking schemes with strong and well-understood security. Efficiency is an important criterion for masking schemes, since they sometimes have orders of magnitude performance overheads over an implementation without countermeasures. Regarding security, our approach is to prove the security of the masking scheme as comprehensively as possible by analyzing complete masked algorithms in leakage models which are well-aligned with the characteristics of real-world leakage.

We first work in the threshold probing model, which is the simplest and most widely used side-channel leakage model. Our focus is the problem of composability, that is, the possibility to build complex algorithms from small building blocks while preserving the security. Concretely, we introduce a new composable security definition, design efficient constructions that satisfy it, and use them to build more complex circuits.

Next, we adapt our results from the threshold probing model to the robust probing model. This model extends the threshold probing model by taking into account physical phenomena named glitches and transitions. These phenomena that appear in concrete hardware implementations violate the independence assumption of the threshold probing model.

Finally, the threshold and robust probing models fix a bound on the amount of leakage available to the adversary, while in practice, the amount of leakage depends on the amount of computation performed by the implementation. This issue is taken into account by the random probing model, for which we introduce a new composable analysis technique that gives a better security bound for simple and efficient masking schemes, compared to state-of-the-art techniques.

Contents

1 Introduction

Cryptography is the science of information security. More precisely, it is a branch of computer science devolved to the study of mathematical techniques for securing information communication, information storage and computing systems against adversarial behavior. The traditional model of cryptography involves multiple parties that communicate over an insecure channel, and the main goal of the parties is to achieve confidential and authenticated communication: the cryptographic protocol used by the parties should ensure that, despite being able to observe (and sometimes even manipulate) the messages sent on the channel, the adversary should be unable to recover the information communicated by the parties or to inject fake information. In this model, the parties are modeled as *black-boxes*: the adversary may not observe their computations, nor manipulate them.

Modern cryptography has been very successful at securing communication: many encryption schemes are decades old, widely used and have no known weakness. This success has been enabled by the adoption of Kerckhoffs's principle, which can be summarized as: *a cryptosystem should not be secret to be secure*. Modern cryptographic algorithms and protocols follow this principle by being public, and their only secret elements are secret keys (often large numbers) which are selected randomly during their initialization. Kerckhoffs's principle was originally introduced to make the security of a cryptosystem more robust, following it has the side effect of allowing public international standardization of cryptographic schemes such as AES [Dwo+01], RSA [Mor+16] or TLS [Res18]. These standards have been widely implemented in computing devices, such that nowadays anybody can securely communicate over the internet.

Despite their security in the black-box model, many cryptosystem implementations have been broken due to their vulnerability to *physical attacks*. In particular, *side-channel attacks* (SCA) are based on the possibility for the adversary to observe the machine executing a cryptographic algorithm and to extract from these observations more information than what the machine sends on the insecure channel. The observations can be various physical quantities, such as execution time [Koc96], power consumption [KJJ99], electromagnetic radiation [QS01], sound [GST17], etc. Security against attacks that exploit such leakages is sometimes called

security in the *gray-box* model: the adversary gathers more information than only the messages produced, yet it does not have full knowledge of every value computed by the device (this is known as *white box* model). Side-channel leakage is harmful to cryptographic implementations due to the information it brings on the intermediate state of the algorithms. This state, unlike the output of the algorithm, may depend on the secret key in a very simple manner, e.g., it can be the sum of the key and a message sent over the insecure channel. Then, once the adversary recovers the key, all the security of the cryptosystem vanishes.

In the gray-box model, the scope for the analysis of a cryptosystem is not only the cryptographic algorithm and the communication protocol: it also includes the implementation, from the low-level algorithms and data representations used, down to the detailed structure of the circuit that executes the algorithms.

Kerckhoffs's principle can therefore be specialized to the gray-box model: *countermeasures against side-channel attacks must remain effective even if the adversary has full knowledge of all the implementation details.* This assumption is not common in today's industrial practice [JWI20], where "security by obscurity" is often used, as it makes weaknesses harder to detect and exploit (by forcing the attacker to perform tedious reverse-engineering work). Similarly to the adoption of public cryptographic algorithms, the impact of moving to implementations that satisfy gray-box Kerckhoffs's principle will likely be wider deployment of secure implementations and long-term improvements in security level. Indeed, such implementations can be made public and therefore widely available, and easy to collaborate on (e.g., with an open-source development model). Moreover, the assumption that an adversary cannot learn details of an implementation is hard to satisfy in practice, given profiling attacks that use external measurements on a system to model its internals. These attacks get only stronger over time, and may eventually be as powerful as the attacks that exploit direct knowledge of the implementation [Bro+22]. In that respect, applying Kerckhoffs's principle can be viewed as a safe simplifying shortcut (instead of making assumptions on what the adversary can or cannot learn with profiling).

However, moving to public implementations is challenging. First, current state-of-the-art public implementations with side-channel protections have often large performance overheads compared to unprotected implementations.[1] Second, even with large overheads, the state-of-the-art

[1] It is hard to judge how large these overheads are compared to state-of-the-art secret implementations in a public work, due to the (intrinsic) lack of public information on such implementations.

countermeasures do not always effectively protect against adversaries with extensive knowledge of the implementation, either due to missing or broken security claims [Moo+19], or due to the difficulty of satisfying the physical assumptions of countermeasure's security proofs [BS21].

With the long-term goal of getting Kerckhoffs's principle for the gray-box model adopted as an industry standard practice, this thesis contributes novel side-channel countermeasure designs and security analyses. Concretely, we focus on the masking countermeasure to protect against leakage caused by the electrical current flow in transistors, which is the source of most known side-channels: power consumption, electromagnetic radiation, etc.

Our goal is to have countermeasures with comprehensive security proofs, i.e., proofs that cover the entire implementation (and not a small part of it) while relying on realistic physical assumptions that can be easily verified. Moreover, we try to maximize the performance of the countermeasures, under these security constraints.

We do not consider the timing side-channel in this thesis. Although timing attacks are often very powerful, the so-called "constant-time programming" countermeasure is a well-known and well-understood solution. Constant-time programming is an example of countermeasure that achieves provable security in the gray-box model [TOS10], and has become a standard industry practice (although it is not always properly implemented) [Jan+22].

In the remaining part of this introduction, we give a simple model of the power consumption leakage of electronic circuits, then we briefly introduce the main countermeasures against power side-channel attacks. We finally introduce the main contributions and the structure of this work.

1.1 Side-channel leakage characterization

Origin and structure of leakage Most modern digital electronic circuits are based on synchronous CMOS logic design. In such a design, two types of logic gates are used. First, combinational logic gates implement Boolean functions, and their output value updates as soon as their input changes, with a small propagation delay. Second, the register (also known as flip-flop) is the most common sequential logic gate. A register acts as a memory element, which enables to logically split the computation of synchronous circuits into a discrete sequence of clock cycles. At each clock cycle, all registers sample their input signal, store it, and assert the stored value on their output wire.

When a clock cycle begins (i.e., at the rising edge of the clock), the registers set their outputs to a new value. The combinational logic gates connected to the registers subsequently update their output value, which is fed to other logic gates, leading to the propagation of a new state in the whole circuit. This propagation is not instantaneous: every combinational logic gate and every wire incurs a propagation delay. As a result, the logic gates may perform transient incorrect computations when some (but not all) of their inputs are updated to the new values, leading to incorrect output values that may propagate in the circuit. These ephemeral incorrect computations are known as *glitches* and eventually disappear since synchronous logic circuits do not contain combinational logic loops.

The current consumption of CMOS gates can be split into a static part and a dynamic part. The static part is the current consumption when the inputs and output values are stable, it depends on the inputs values (and also on the stored value for registers). This static current is also known as CMOS leakage current (not to be confused with side-channel information leakage). Next, the dynamic part is the current consumption of a logic gate caused by a change of value at its inputs. This dynamic current depends mainly on the switching of the output of the gate: whether it remains constant, switches from 0 to 1, or from 1 to 0. In side-channel leakage modeling, this phenomenon is known as a *transition*. Therefore, the side-channel leakage of a circuit depends on all the values it computes, on the transitions between these values, on the glitches in the combinational logic, and even on the transitions between the glitches values [Mül+22]. The overall side-channel (electromagnetic or power) measurements depend on the sum[2] of current consumption of a set of gates (power leakage depends on all the gates in the circuit while near-field electromagnetic leakage depends on fewer gates).

Noise The leakage of a circuit contains random fluctuations called *noise*, which is typically modeled as the addition of a random value to the deterministic leakage discussed previously. The noise has multiple sources, such as the intrinsic noise of the logic gates, the one caused by the necessarily imperfect measurement of the leakage [BUS21], or even the leakage of parts of the circuit which are not properly modeled in an attack, hence are considered as computational noise [SÖP04].

[2] This assumption is not fully correct: there are other interactions between the logic gates in a circuit such as electromagnetic coupling between wires or interactions through supply voltage drops which are called *coupling* leakage phenomena [LBS19]. The study of countermeasures in the cases where couplings have a significant impact is out of the scope of this thesis.

Metrics The sequence of leakage values measured during a (multi-cycle) execution of a circuit is named a *leakage trace*. The leakage is often characterized by the signal-to-noise ratio (SNR) of each point in the trace with respect to a variable in the attacked algorithm, or by the mutual information (MI) between the trace and the variable [DFS19]. When mounting a side-channel attack, the main figure of merit is the number of leakage traces required for success.

1.2 SCA countermeasures

Numerous countermeasures have been proposed to protect against side-channel attacks. We group them in three broad categories: physical, algorithmic and computational countermeasures.

Physical The first solution proposed by engineers to protect against side-channel attacks was to increase the noise in the circuit to reduce the SNR (hence the MI), which proportionally increases the number of traces needed to mount an attack [DFS19]. This can be done at the electronic design level by adding noise generation circuits, but they have the disadvantage of increasing the power consumption (in inverse proportion to the SNR when reaching low SNR). Another solution is to act on the signal, by reducing the impact of the secret values on the power consumption, such as with the use of dual-rail logic styles [TV04]. This countermeasure also consumes power, but it can produce a larger SNR reduction than a noise generation circuit with the same power consumption.

SNR reduction is however limited: in face of ever more powerful adversaries (i.e., that are able to collect a very large number of traces), the noise level increase (which is the only remaining solution after signal reduction has been applied) always ends up being limited by the power consumption budget. The main issue here is that the strength of the countermeasure (measured in number of attack traces) scales linearly with the SNR reduction, and thus with the cost of the countermeasure.

Hiding Adding randomization in the execution increases protection against attacks that are based on simple averaging or correlations [KJJ99; BCO04]. The hiding techniques are based on randomizing the time at which operations processing sensitive data are executed, leading to a randomized position of the informative leakage in the traces. The main hiding techniques are the insertion of random delay (e.g., by means of dummy processor instructions) and shuffling. Shuffling can be used when

many identical and independent operations have to be performed, such as the S-boxes of a block cipher, and consists in randomizing the order in which the operations are executed [Azo+22]. Let k be the maximum delay (for random delay) or the number of independent operations (for shuffling). The security obtained by hiding is at best an increase of the number of attack traces by a factor k.

Masking Another countermeasure against SCA is to change the computations that are run on a device. In essence, masking is a countermeasure that replaces every sensitive wire x by a secret sharing of it. That is, x is replaced by the sharing $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{d-1})$ from which x can be re-computed, and which is randomized in such a way that any tuple of $t < d$ shares is independent of x. The parameter t is named *masking order*, and implementations for which $t = d - 1$ achieve optimal masking order. In addition to sharing the sensitive values, the masking of a circuit requires replacing the original gates of the circuit with so-called masked *gadgets*, which perform on sharings the operations implemented by the original gates on wires and typically have a quadratic cost in the number of shares.

Provided that each share leaks independently, recovering a sensitive value in a masked implementation with optimal order implies the recovery of the value of all its shares. If each share can be recovered with probability p, then the sensitive value can be recovered with probability $\mathcal{O}\left(p^{t+1}\right)$, hence the number of traces required for mounting an attack is $\mathcal{O}\left(1/p^{t+1}\right)$, which makes t a good security parameter. Practically, this shows that the security level achieved by masking also depends on the MI, which determines the value of p. Masking can therefore be effectively combined with SNR reduction countermeasures, as well as with hiding techniques which have the effect of reducing the MI [Azo+22].

Execution count limitation The last countermeasure against SCA we discuss is based on the idea of limiting the total number of executions that manipulate a given secret, thereby limiting the maximum number of traces that can be acquired by an adversary. Such limitations may be implemented at varied abstraction levels, and with various effectiveness depending on the target application. This technique is already used for other reasons than SCA, such as in application-level limits (such as PIN trials limitations) and protocol-level limitations (e.g., continuous key agreement in the double ratchet protocol [ACD19]).

Leakage-resilient authenticated encryption schemes [PSV15] integrate this technique by using re-keying: for each encryption, a different epheme-

ral keystream is derived from the long-term key, hence the long-term key is used only a few times to encrypt a long message. The ephemeral keys from the keystream are themselves used only a few (e.g., two) times, which strongly limit the SCA adversary and therefore allows using weak and cheap SCA countermeasures (e.g., no masking or hiding) to protect the algorithms that manipulate these keys. The algorithms that manipulate the long-term keys may however be used many times, typically two times (or even less [Cas+19]) the total number of messages encrypted with a single key, and may require strong protection (e.g., masking combined with low SNR). Such a *leveled implementation* approach can reach a high security level for a moderate implementation cost [Bel+20c].

1.3 Outline of the thesis

The topic of this thesis is the conception of masking schemes, their analysis and their application to implementations of cryptographic algorithms. Our goal is to design efficient masking schemes that can be applied to arbitrary computations with good security guarantees. For this purpose, we design masking schemes with composable security properties, which means that large circuits can be built from simple masked gadgets, and that the security of the full computation is proven as a consequence of the security properties of the gadgets. Composability is therefore the key ingredient that allows us to claim security of complex masked computations.

To be relevant, the security proofs must have realistic hypotheses, meaning that the leakage model used for the proof must correspond to the characteristics of the real leakage. We achieve this by studying the security of the masking schemes in three leakage models: (1) the threshold probing model that is the simplest but least connected to real leakage; (2) the robust probing model that extends the threshold probing model to take into account glitch and transition leakage; and (3) the random probing model that better reflects the impact of the amount of computation (which practically impacts the MI between the trace and the variables) on the security.

Chapter 2 first introduces masking in more details, along with the relevant leakage and security models, as well as the challenges related to the composition of masked computations. Next, Chapter 3 presents state-of-the-art masking schemes and their analysis in the threshold probing model, including our new "probe-isolating non-interfering" (PINI) scheme. Then, Chapter 4 adapts the PINI scheme to ensure its security in the robust probing model, leading to the "hardware private circuit" (HPC)

masking schemes. Finally, Chapter 5 introduces the "probe distribution table" (PDT) which is a new methodology to analyze the security of masking schemes in the random probing model. It also uses the so-called *local random probing model* [Guo+20] heuristic to evaluate and optimize masking schemes.

1.4 Other contributions

We briefly describe the results obtained during the work that led to this thesis but are not discussed in this document.

A first series of works relate to the Spook cipher, an authenticated encryption with associated data algorithm that was submitted to the NIST lightweight cryptography competition. This algorithm is designed for side-channel security: it is well-suited to leveled implementations and its tweakable block cipher primitive is easy to mask. I wrote multiple implementations of Spook and contributed to its design by bringing expertise in performance trade-offs for implementation of various targets (high- end low-end CPUs, FPGA and ASIC), with and without side-channel countermeasures [Bel+20d; Bel+20c; MCS22b]. I also collaborated to the design of SpookChain, of a variant of Spook that encrypts message sequences, that further improves the efficiency when protected against side-channel attacks by reducing the number of masked primitive calls [Cas+19].

Beyond the masked implementations of Spook, I also collaborated to the design of masked hardware AES implementations based on the HPC2 masking scheme (see Chapter 4) [MCS22a].

In another pair of works [Wan+20a; Wan+20b], I collaborated to the study of code-based masking. These schemes offer interesting performance optimizations by packing multiple sensitive values in a single sharing, and are also a research track for implementations that are robust against both side-channel and fault attacks [BS97].

Finally, I worked on the evaluation of implementations, implementing in [BCS21] an attack against the ASCAD public dataset [Ben+20], which is a set of power measurements of a first-order masked implementation of the AES running on a micro-controller. More theoretically, I studied the convergence of the profiling stage of SCA, which consists in learning a statistical model of the attacked implementation. In [Mas+22], we provide bounds on how much an attack can improve if its profiling uses more training data. In a side-channel security evaluation context, this tells when to stop collecting profiling data.

2 Masking

Masking aims at protecting computations against side-channel leakage. These computations are modeled as arithmetic circuits that take as input *sensitive wires* (also named *sensitive variables*) whose value must remain secret (circuits with some non-sensitive inputs are discussed in Section 4.1). In the circuit, any wire that depends on a sensitive wire is also assumed to be sensitive.

Given a circuit to protect, a *masking compiler* generates a masked circuit, that implements the same functionality as the original circuit, where each input and output wire is replaced by a *sharing*, a tuple of d wires whose value is a randomized encoding of the original value. A wire can be transformed into a sharing using a randomized encoder circuit, the decoder performs the transformation in the other direction.

The chapter introduces the fundamental notions of masking that are used in this thesis. We first describe the arithmetic circuit model we use, then we explain the generic masking framework introduced in [ISW03]. Next, we discuss the three security models used in this work, and we finally present one of the challenges in the analysis of masking schemes: the composition of small masked circuits to build more complex computations.

2.1 Circuit model

Masking is often analyzed on stateless randomized arithmetic circuits over a finite field $\mathbb{F}_q$. These are modeled as directed acyclic graphs (DAG) whose vertices are abstract gates and whose edges are abstract wires. The gates are of the following types:

- input gates: with fan-in 0 and fan-out 1, modeling the inputs of the circuit,
- output gates: with fan-in 1 and fan-out 0, modeling the outputs of the circuit,
- arithmetic gates: with arbitrary fan-in (usually one or two) and fan-out 1, modeling arithmetic operations in the circuit,

- copy gates: with fan-in 1 and fan-out 2, used when a value appears as input of multiple gates,
- random gates: with fan-in 0 and fan-out 1, they output a field element taken uniformly at random.

A circuit with n_i input gates and n_o output gates implements the randomized function $f : \mathbb{F}_q^{n_i} \to \mathbb{F}_q^{n_o}$ (we assume that input and output gates are ordered) whose output is computed by evaluating the circuit for the given inputs, with uniformly random values assigned to the outputs of random gates.

While this model is abstract and does not directly correspond to real-world circuits, it is a good starting point to analyze computations implemented both in dedicated hardware (see Section 4.1) or in general-purpose processors (see Section 4.6).

In the following, we study circuit compositions. Let A and B be two circuits whose input and output gates are ordered. If the number of outputs of A is equal to the number of inputs of B, the sequential composition of A and B, denoted $B \circ A$, is the circuit built as follows. Its set of gates is the union of the sets of gates of A and B and the wires connecting them are the same as in the circuits A and B, except for the output gates of A and the input gates of B which are discarded, and the wires connected to each (output of A, input of B) pair are merged. The list of inputs (resp. outputs) of $B \circ A$ is the list of inputs of A (resp. outputs of B). Next, the parallel composition of the circuits A and B, denoted $A||B$, is the circuit whose sets of gates are wires are the unions of the ones of A and B. The list of inputs (resp. outputs) of $A||B$ is the concatenation of the lists of inputs (resp. outputs) of A and B. By A^n, we denote the parallel composition of n repetitions of A: $A^n = A||A||\dots||A$ (parallel composition is associative).

2.2 Masking schemes

2.2.1 Masking compiler

Let us now formalize the notion of masking a circuit. A masking scheme is the combination of a *masking compiler*, an *encoder* Enc and a deterministic *decoder* Dec. The encoder takes as input a sensitive value x and outputs a sharing $\mathbf{x} = (\mathbf{x}_0, \dots, \mathbf{x}_{d-1})$ of it, while the decoder performs the opposite operation (it *unmasks* the sharing): $\mathsf{Dec} \circ \mathsf{Enc}$ implements the identity. Each share $\mathbf{x}_i$ in the sharing $\mathbf{x}$ has an index $i \in [\![0, d[\![$,[1]

[1] $[\![a, b[\![$ denotes the set $[a, b) \cap \mathbb{N}$.

which is its position in the sharing. When provided with a circuit C with n_i inputs and n_o outputs, the masking compiler outputs a circuit C' such that C and $\mathsf{Dec}^{n_o} \circ C' \circ \mathsf{Enc}^{n_i}$ implement the same (randomized) function.

Many masking styles can be defined in this formalism. We next give some of them, based on their decoder:

- Arithmetic masking in a finite field $\mathbb{F}_q$, where

$$\mathsf{Dec}(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}) = \mathbf{x}_0 + \cdots + \mathbf{x}_{d-1} \ .$$

- Boolean masking is the special case of arithmetic masking when the field is $\mathbb{F}_2$. It is the most studied masking style, as it is very efficient to implement in hardware: every field operation can be implemented with a single logic gate.

- Inner Product masking [DF12] works with a vector $\mathbf{l}$ in $\mathbb{F}_q$ such that

$$\mathsf{Dec}(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}) = \mathbf{l}_0\mathbf{x}_0 + \cdots + \mathbf{l}_{d-1}\mathbf{x}_{d-1} \ ,$$

  and the vector $\mathbf{l}$ is selected randomly before each execution.

- Code-based masking [Pou+17; Wan+20b] is a generalization of arithmetic and inner product masking in $\mathbb{F}_q$: a sharing represents a vector of sensitive variable, and

$$\mathsf{Dec}(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}) = A\mathbf{x}$$

  where $A \in \mathbb{F}_q^{k \times d}$ and $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{d-1})$.

- Multiplicative masking in a finite field $\mathbb{F}_q$ is such that

$$\mathsf{Dec}(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}) = \mathbf{x}_0 \cdot \ldots \cdot \mathbf{x}_{d-1}.$$

For any decoder Dec, there exists a *uniform encoder*: for any x in the image of Dec, the output of the uniform encoder $\mathsf{Enc}(x)$ is a *uniform sharing* of x, that is, it follows a uniform distribution over the set

$$\{(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}) \text{ s.t. } x = \mathsf{Dec}(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1})\} \, .$$

Lemma 1. *If* $(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1})$ *is a uniform arithmetic sharing of* x*, then any tuple of* $d-1$ *shares* $\mathbf{x}_i$ *has a uniform distribution over* $\mathbb{F}^{d-1}$.

Proof. Let us consider the tuple $X = (\mathbf{x}_0, \ldots, \mathbf{x}_{d-2})$ (this is wlog since the addition is commutative, hence we may shuffle the order of the shares). For any value of $X \in \mathbb{F}^{d-2}$ and $x \in \mathbb{F}$, there exists a single $\mathbf{x}_{d-1}$ such that $x = \mathsf{Dec}(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1})$, which is $\mathbf{x}_{d-1} = x - \mathbf{x}_0 - \cdots - \mathbf{x}_{d-2}$. Therefore, $\mathsf{Dec}(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1})$ outputs a uniform distribution over the set

$$\left\{ \left(\mathbf{x}_0, \ldots, \mathbf{x}_{d-2}, x - \sum_{i=0}^{d-2} \mathbf{x}_i \right) \right\},$$

therefore X is uniformly distributed. □

In this work, we focus on arithmetic masking: all further discussions apply only to arithmetic masking.

2.2.2 Masked gadgets

Masking compilers replace gates (or small subcircuits) with so-called *gadgets*, which are subcircuits that implement the same functionality as the original gate (or sub-circuit) in a masked fashion.

Definition 1 (Gadget)**.** *A gadget* G *is a sub-circuit composed of a set of gates, the set of wires that connect the inputs and outputs of those gates (named internal wires* $\mathcal{W}$*), a set of wires only connected to those gate's inputs (named input wires* $\mathcal{I}$*), and a subset of its gates* $\hat{\mathcal{O}}$ *(named output gates). The input wires (resp. output gates) are grouped in sharings* $\left(\mathbf{x}^i\right)_{i=0,\ldots,m-1}$ *(resp.* $\left(\mathbf{y}^i\right)_{i=0,\ldots,n-1}$*).* G *is d-shares if all its input and output sharings have d shares. Furthermore,* G *implements the function f (with respect to* Enc *and* Dec*) if, for any* x^i *(for* $i = 0, \ldots, m-1$*) and any sampling* $\mathbf{x}^i \leftarrow \mathsf{Enc}\left(x^i\right)$*, the outputs* $\mathbf{y}^i$ *of* G *satisfy*

$$\left(\mathsf{Dec}\left(\mathbf{y}^0\right), \ldots, \mathsf{Dec}\left(\mathbf{y}^{n-1}\right)\right) = f\left(x^0, \ldots, x^{m-1}\right) .$$

Composition The definition of the parallel and serial composition of gadgets is similar to the composition of circuits, using an ordering of the inputs and outputs induced by the sharings: $(\mathbf{x}_0^0, \ldots, \mathbf{x}_{d-1}^0, \mathbf{x}_0^1, \ldots, \mathbf{x}_{d-1}^{m-1})$.

Definition 2 (Gadget parallel composition)**.** *Let* G_0 *(resp.* G_1*) be a gadget with the input sharings* $\mathbf{x}^0, \ldots, \mathbf{x}^{m_0-1}$ *(resp.* $\mathbf{y}^0, \ldots, \mathbf{y}^{m_1-1}$*) and the output sharings* $\mathbf{z}^0, \ldots, \mathbf{z}^{n_0-1}$ *(resp.* $\mathbf{w}^0, \ldots, \mathbf{w}^{n_1-1}$*). The gadget* $\mathsf{G} = \mathsf{G}_0 || \mathsf{G}_1$ *is a parallel composition of* G_0 *and* G_1 *if it is a union of the subcircuits of* G_0 *and* G_1*, and if its tuple of input sharings is a shuffling of* $(\mathbf{x}^0, \ldots, \mathbf{x}^{m_0-1}, \mathbf{y}^0, \ldots, \mathbf{y}^{m_1-1})$ *and its tuple of output sharings is a shuffling of* $(\mathbf{z}^0, \ldots, \mathbf{z}^{n_0-1}, \mathbf{w}^0, \ldots, \mathbf{w}^{n_1-1})$.

Definition 3 (Gadget sequential composition)**.** *Let* G_0 *(resp.* G_1*) be a gadget with the input sharings* $\mathbf{x}^0, \ldots, \mathbf{x}^{l-1}$ *(resp.* $\mathbf{y}^0, \ldots, \mathbf{y}^{m-1}$*) and the output sharings* $\mathbf{z}^0, \ldots, \mathbf{z}^{m-1}$ *(resp.* $\mathbf{w}^0, \ldots, \mathbf{w}^{n-1}$*). The sequential composition of* G_0 *and* G_1 *is the gadget* $\mathsf{G} = \mathsf{G}_1 \circ \mathsf{G}_0$ *that is the union of the subcircuits of* G_0 *and* G_1*, where the input wires corresponding to each input sharing* $\mathbf{y}^i$ *of* G_1 *are connected to the gates of* G_0 *corresponding to the output sharing* $\mathbf{z}^i$ *(respecting the order of the shares). The input sharings of* G *are* $(\mathbf{x}^0, \ldots, \mathbf{x}^{m-1})$ *and its output sharings are* $(\mathbf{y}^0, \ldots, \mathbf{y}^{n-1})$*.*

More generally, we say that a composite gadget G is the composition of composing gadgets (also named "sub-gadgets") $(\mathsf{G}_i)_{i=0,\ldots,n-1}$ if G can be built from the gadgets G_i using parallel and sequential compositions. These compositions often use *copy gadgets* which duplicate sharing by copying each share, when a variable represented by a sharing is used as operand in multiple operations.

Moreover, *identity gadgets* are sometimes needed to make a gadget operate on a part of the state in a sequential composition. An identity gadget, denoted ID, has a single input sharing and contains one identity gate per input share, which is also an output gate, such that the output sharing of the gadget has the same value as its input sharing. The insertion of identity gadgets in a circuit does not change the computations it performs, but allows writing any circuit made of gadgets under the form of an *extended sequential* composition.

Definition 4 (Extended sequential composition)**.** *The gadget* G *is an extended sequential composition of the gadgets* $(\mathsf{G}_i)_{i=0,\ldots,n-1}$ *if it can be written as*

$$\mathsf{G} = \left(\mathsf{G}_0 || \mathit{ID}^{k_0}\right) \circ \ldots \circ \left(\mathsf{G}_{n-1} || \mathit{ID}^{k_{n-1}}\right)$$

for some k_i*, where* ID^k *represents a parallel composition of* k *identity gadgets.*

Let us now present various kinds of gadgets.

Arithmetic gates The simplest masking compilers work at the level of single arithmetic gates, and typically implement the following gadgets.

- Affine gadgets: by definition, a linear function in the masking field is homomorphic with respect to (arithmetic) Dec. Therefore, the masking of a linear operation can be obtained by applying the operation to each share independently, and this technique can be extended to any affine function by adding a constant value to one

Algorithm 1 Sharewise implementation of a n-inputs and m-outputs affine function $f : \mathbb{F}_q^n \to \mathbb{F}_q^m$.

Input: Sharings $\mathbf{x}^0, \ldots, \mathbf{x}^{n-1}$.
Output: Sharings $\mathbf{y}^0, \ldots, \mathbf{y}^{m-1}$ such that $(y^0, \ldots, y^{m-1} = f(x^0, \ldots, x^{n-1})$.

1: $(\mathbf{y}_0^0, \ldots, \mathbf{y}_0^{m-1}) \leftarrow f(\mathbf{x}_0^0, \ldots, \mathbf{x}_0^{n-1})$
2: **for** $i = 1$ to $d - 1$ **do**
3: $\quad (\mathbf{y}_i^0, \ldots, \mathbf{y}_i^{m-1}) \leftarrow f(\mathbf{x}_i^0, \ldots, \mathbf{x}_i^{n-1}) - f(0, \ldots, 0)$

share, as shown in Algorithm 1.[2] The overhead of this *sharewise implementation* of affine gadgets is $\mathcal{O}(d)$.

- Multiplication gadgets: the best-known multiplication gadget has been introduced in [ISW03]. We give in Algorithm 2 a slightly modified version of the original gadget, in order to balance more evenly the amount of computations for each output share. Despite multiple works designing other multiplication gadgets, all of them have $\mathcal{O}\left(d^2\right)$ complexity in arithmetic and random gates: these gadgets all compute the product of every share of the first input and every share of the second input [Bel+16; Bar+20; Wan+20a].

- Refresh gadgets: these gadgets are not useful for the functionality of the circuit, since they implement the identity function. However, they re-randomize their input sharing, which can improve the security of the circuit, e.g., by avoiding composition issues (see Section 2.4.1).

Another kind of gadget worth mentioning is the conversions between arithmetic masking on different fields. These are usually built from simpler gadgets in the two fields, as done in Section 3.6.

In the particular case of Boolean masking, the 1-input and 2-input affine gadgets are the NOT, XOR and XNOR gates, while the multiplication gadget corresponds to the AND gate. Any non-linear 2-input gate can be implemented by combining the AND gate and some NOT gates (using De Morgan laws). The cost of these gadgets is essentially the same as the one of an AND gadgets, since the NOT gadget has a very low cost (a single gate).

[2] We describe the gadgets in an algorithmic way for the sake of readability, whereas formally we describe arithmetic circuits. An algorithmic description can be mapped to the circuit by unrolling the loops in the algorithm and adding a gate for each operation (and connecting the wires accordingly).

Algorithm 2 ISW Multiplication gadget with d shares (adapted from [ISW03]).

Input: Sharings $\mathbf{x}$, $\mathbf{y}$
Output: Sharing $\mathbf{z}$ such that $z = x \cdot y$.

1: **for** $i = 0$ to $d-1$ **do**
2: **for** $j = i+1$ to $d-1$ **do**
3: $r_{ij} \xleftarrow{\$} \mathbb{F}_q$; $r_{ji} \leftarrow -r_{ij}$
4: **for** $i = 0$ to $d-1$ **do**
5: $\mathbf{z}_i \leftarrow \mathbf{x}_i \mathbf{y}_i$
6: **for** $j = 0$ to $d-1, j \neq i$ **do**
7: $\mathbf{z}_i \leftarrow \mathbf{z}_i + \left(\mathbf{x}_i \mathbf{y}_j + r_{ij}\right)$

This work is mainly oriented towards masking compilers based on masking compilers whose gadgets implement simple arithmetic gates.

Table-based Lookup tables are arithmetic gates with $q+1$ inputs (when working in $\mathbb{F}_q$) whose output is one of the first q inputs, selected according to the value of the last input. While these gates can be implemented using elementary logic gates, there exist more efficient implementations of them, both in dedicated hardware and in software implementations. Multiple masking schemes based on lookup tables have been proposed in the literature, and sometimes improve performance compared to other techniques thanks the independence of the implementation cost from the (algebraic) complexity of the implemented function (for a given number of inputs) [Cor14; KSM22].

More complex functions The Threshold Implementations (TI) [NRR06] are Boolean masking schemes whose gadgets implement fairly complex Boolean functions (e.g., S-boxes). Each of these gadgets is tailored to a specific function, and is not a special case of a more general (e.g., table-based) construction. The TIs are presented with more details in Section 4.3.4.

2.3 Masking security

2.3.1 Threshold probing model

Masking schemes are most often studied in the t-threshold probing model, which assumes that the adversary is able to probe (i.e., get the value of) t

wires of its choice in the masked gadget [ISW03]. Informally, a gadget is t-probing secure if, for any such set of probes, the marginal distribution of the probes is independent of the sensitive values represented by the input shares of the gadget.

Definition 5 (Probe set)**.** *A set of probes P in a gadget G is a subset of its wires $\mathcal{A} := \mathcal{W} \cup \mathcal{I}$ and its gates. We denote by $\mathsf{G}_P(\mathbf{x})$ the tuple containing the values of the wires and of the outputs of the gates in P during the execution of G with inputs $\mathbf{x}$ (where P is viewed as a tuple with an arbitrary order for the elements).*

Definition 6 (Safe probe set)**.** *A set of probes P in a gadget G is safe for G if the distribution of $\mathsf{G}_P\left(\mathsf{Enc}\left(x^0\right), \ldots, \mathsf{Enc}\left(x^{m-1}\right)\right)$ is the same for every value of $(x^0, \ldots, x^{m-1})$ when Enc is the uniform encoder. If a set of probes is not safe, it is a* successful attack*.*

Definition 7 (Threshold probing security)**.** *A gadget G is t-probing secure if any set of probes of size t is safe for G.*

2.3.2 Robust probing

Informally, the threshold probing model makes sense when each intermediate value computed on a wire leaks independently, making the attack harder as the number of independent leakages to analyze grows. However, as discussed in Section 1.1, the glitches [MPG05], transitions [MDS02] and couplings [Cnu+17] break this independence: the leakage from a physical wire now depends on the values carried by multiple wires of the arithmetic circuit. The robust t-threshold probing model [Fau+18] takes these *physical non-idealities* into account, by giving to the adversary t *extended probes* that represent the physical wires by leaking multiple values at once. The robust probing model, with its variants (i.e., how the list of extended probes is built), is introduced in more details in Section 4.2.

2.3.3 Random probing model

In the p-random probing model [ISW03; DDF19], every wire in the circuit, independently of the other wires, leaks its value with probability p (otherwise it produces no leakage). The set of wires that leak can then be classified as either safe or as a successful attack, and the security level in the random probing model is the probability that the set of leaking wires is a successful attack.

Definition 8 (Random probing leakage)**.** *The p-random probing leakage of a set of wires $\mathcal{W}$, is a random variable $\mathcal{L}_p(\mathcal{W}) \subseteq \mathcal{W}$ such that every wire $w \in \mathcal{W}$ belongs to $\mathcal{L}_p(\mathcal{W})$ with probability p (independently for each wire). Extending the notation, for a gadget G, we denote $\mathcal{L}_p(\mathsf{G}) := \mathcal{L}_p(\mathcal{A})$, where $\mathcal{A} := \mathcal{I} \cup \mathcal{W}$ is the set of all the wires in the circuit (input wires and internal wires). As an exception the input wire of an identity gate does not leak: $\mathcal{L}_p(\mathsf{ID}) = \varnothing$.*[3]

Definition 9 (Random probing security)**.** *The security level of a gadget G in the p-random probing model (or the p-random probing security of G) is the probability ξ (over the randomness in $\mathcal{L}_p$) that the set $\mathcal{L}_p(\mathsf{G})$ is a successful attack.*

We sometimes consider the random probing security with respect to one of the input sharings of a gadget, which amounts to changing the definition of a safe probe set (Definition 6) to keep all non-sensitive inputs x^i set to 0.

Random probing security is a quantitative notion rather than a qualitative one (such as (robust) threshold probing security). It can be therefore quantified in bits, as commonly done for cryptographic algorithms, with the following formula: $-\log_2(\xi)$.

Compared to the threshold probing model, the random probing model is more accurate, e.g., being able to capture the *horizontal* side-channel attacks, which are attacks that try to exploit all the available leakage, instead of focusing only on a few points in leakage traces (hence target a few computations) [Bat+16]. In horizontal attacks, the more a variable is involved in computations (even if it is the same computation repeated multiple times), the more it leaks. This is easy to understand in the random probing model, while repeated computation has no security impact in the threshold probing model.

2.3.4 Practical security

The (robust) threshold probing model and the random probing model are pretty abstract, hence it is natural to question they relevance: how do they relate to the practical security of circuits?

[3] Since identity gates are only used in identity gadgets for the analysis of gadget composition but does not translate into any actual element in the implementation (physical wire/gate, or CPU instruction), they cause no leakage.

This exception does not cover copy gates, for the sake of consistency with previous works [DFZ19; Bel+20a; BRT21]. Concretely, if a value is used twice, we assume that 3 leaking wires carry it (which is arguably not very realistic).

Reduction to realistic models The noisy leakage model is fairly close to description in Section 1.1 of the leakage of an electronic circuit. Let $(\mathbf{X}_i)_{i=1,\dots,n}$ be random variables representing the values of all the wires in the circuit. In the noisy leakage model, the leakage trace is $(\mathcal{L}_i(\mathbf{X}_i))_{i=1,\dots,n}$, where each $\mathcal{L}_i$ is an independent randomized function, which is supposed to be δ-noisy [DDF19; DFS19]: $\delta = \mathsf{SD}(\mathbf{X}_i; \mathbf{X}_i | \mathcal{L}_i(\mathbf{X}_i))$. The security in the δ-noisy leakage model is characterized by the statistical distance between the output of an adversary that gets access to noisy leakage (for δ-noisy leakage function of its choice), and the output of a black-box simulator.

This leakage model corresponds well to the real leakage of circuits: every manipulated value leads to some leakage, which, when measured, is mixed with noise. Moreover, security in the noisy leakage model can be reduced to the random probing model: if a circuit has a security level ξ p-random probing model, then it has the same security level in the $(p/|\mathbb{F}|)$-noisy leakage model [DDF19]. This shows that the random probing model has a strong connection to an arguably more realistic leakage model. However, both the random probing model and the noisy leakage model do not take into account the physical non-idealities discussed in Section 2.3.2 which often break the assumption of independence of the leakage functions $\mathcal{L}_i$. To the best of our knowledge, no robust version of the random probing (or noisy leakage) has ever been proposed to take into account these non-idealities.

Regarding the threshold probing model, it has been shown in [DDF19] that t-threshold probing security implies p-random probing security: p-random probing security [DDF19]: for any t and p, one may derive the probability that $\left|\mathcal{L}_p(\mathsf{G})\right| \leq t$, which is a lower bound for the p-random probing security level. Concretely, using the Chernoff bound, if $t \geq pn$ (where n is the number of wires in the circuit), the security level in the p-random probing model is at least $-\log_2(\xi) \geq -2\frac{(t-pn)^2}{n} / \log_2(e)$. For gadgets with $n \in \mathcal{O}\left(d^2\right)$, such as the ISW multiplication (Algorithm 2), the constraint $t \geq pn$ implies that must p scale in $1/d$. This scaling of the single-wire leakage probability (related the noise level, as explained in the next section) with the number of shares of the gadget is known as the *noise rate*.

This reduction provides a good theoretical foundation and highlights the link between the parameters of both models, however its bounds are fairly loose compared to the best known attacks (upon which the choice of concrete parameters is based in practice) [DFS19], motivating direct analyzes in the random probing model (see Chapter 5). Moreover, the random probing model relies on the independent leakage assumption,

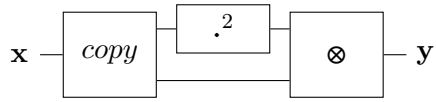


Figure 2.1: Masked computation of x^3 in $\mathbb{F}_{2^k}$. The copy and squaring $(\cdot^2)$ gadgets are sharewise while the multiplication is implemented with the ISW multiplication gadget.

hence physical non-idealities are not taken into account in the reduction.

Empirical evidence Despite the limited guarantees offered by the noisy leakage reduction, the (robust) probing model is widely used and has been instrumental in the design of practically secure masking schemes. Intuitively, t-probing security ensures that the adversary has to estimate at least $t + 1$ variables to recover the secret. This can be done through estimation of higher-order moments of distributions, which is formalized in the bounded moment leakage model [Bar+17] or using techniques such as soft analytical side-channel attacks (SASCA) [VGS14], formalized in the local random probing model (LRPM, see Section 5.3). In both cases, recovery of the secret becomes exponentially harder as the masking order increases.

Beyond giving a proof of the practical security level of an implementation, a t-(robust)-probing security proof can help by restricting the scope of side-channel security evaluations. Indeed, under a leakage independence assumption, the evaluator knows that the best possible attack consists in targeting $t + 1$ variables (typically full sharings), which facilitates the evaluation [Bro22]. Furthermore, using a relevant variant of the robust probing model, the independent leakage assumption can be partially verified in the model, providing additional guarantees prior to the evaluation.

2.4 Masking composability problem

Let us now focus on the threshold probing model. We discuss two challenges in the design of t-probing secure implementations: the probing security does not compose, and the verification of t-probing security for large circuits is difficult.

2.4.1 Composition may break probing security

The composition of t-probing secure gadgets may not be t-probing secure. An example of this is show in Figure 2.1. The issue lies in the ISW

multiplication gadget, which is $d-1$-probing secure, but require its input sharings to be independent. On the other hand, the squaring is linear and therefore implemented sharewise. As a result, for the first multiplication, one input sharing is $(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1})$ while the other input is $\left(\mathbf{x}_0{}^2, \ldots, \mathbf{x}_{d-1}{}^2\right)$. A probe in this multiplication may therefore observe $\mathbf{x}_i\mathbf{x}_j{}^2$, which depends on two shares of x. Additionally probing the remaining $d-2$ shares of x gives a set of probes whose distribution depends on x, and therefore breaks the $d-1$-probing security.

2.4.2 Verification scaling problem

The lack of composability in a system can be circumvented if there is a practical way to perform direct verification of the full system. For t-probing security, however, no such efficient verification has been proposed. The naive approach would be to check that the distribution of each tuple of t wires in the masked circuit is independent of all secrets, which raises two efficiency problems. First, the number of such tuples is very large (super-exponential in t): $\binom{w}{t}$ where w, the number of wires in the circuit, is usually in $\mathcal{O}\left(nt^2\right)$ (n is the number of non-linear gates in the non-masked circuit). Second, the naive way of checking that the distribution of a tuple is independent of a variable requires computing its full distribution. The cost of this computation if $\mathcal{O}\left(2^k\right)$, where k is the number of input shares and random gates in the circuit.

Many optimized tools have been designed (see Section 3.5) to solve this problem, but they still suffer from the fast-growing number of tuples to check. As a result, none of them is able to practically process large circuits (e.g., a block cipher execution) at higher orders (i.e., $t > 1$).

The only solution known to this date for the verification of t-probing security of large circuits is to use composable building blocks: gadgets that satisfy security properties stronger that threshold probing security and can therefore be securely composed.

3 Composition in the threshold probing model

The threshold probing model introduced in Section 2.3.1 is the simplest model for the analysis of masking schemes. As a result, threshold probing security is often the starting point for ensuring the security of a masked implementation, and many t-probing secure masking schemes have been proposed in the literature. In this chapter, we discuss state-of-the-art schemes and introduce new ones, focusing on their performance characteristics and their composability.

The performance of a masking scheme depends on multiple parameters: the number of shares needed to reach a given security order, the number of gadgets needed to implement a given function (e.g., whether refresh gadgets have to be used), and finally the individual performance of the gadgets that are used. The performance of the individual gadgets can be characterized by the number and kind of arithmetic operations they perform (e.g., a multiplication is more expensive than an addition in large fields, but not in $\mathbb{F}_2$), and by their usage of fresh randomness.

Regarding the composability, we only consider masking schemes that are composable, that is, schemes for which there exists an efficient algorithm to mask a circuit. Non-composable masking schemes are schemes for which only the masking of some small circuits is known, or for which the masking algorithm (i.e., the masking *compiler*, or the security verification algorithm) has a high (e.g., exponential) complexity in the size of the circuit or in the masking order. Among composable masking schemes, the so-called *trivially composable* ones mask a circuit by simply replacing each gate with a gadget of the corresponding type (e.g., AND, XOR, etc.). The other composable schemes follow more complex approaches, for example to identify the locations where they require the insertion of refresh gadgets.

This chapter presents in a common formalism multiple threshold probing secure masking schemes, how their security can be automatically verified, and how to use them to build more complex circuits. The main novel contribution of this chapter is the probe isolating non-interference notion (PINI) [CS20] and its use to build conversions gadgets [BC22].

Table 3.1: Composable threshold-probing secure masking schemes: high-level characteristics.

Masking scheme	Shares	Refresh	Multiplication	Trivial composition
ISW (Section 3.2.5)	$2t+1$	no	ISW	yes
Refreshed-copy (Section 3.2.3)	$t+1$	many	NI*	yes
maskComp (Section 3.3)	$t+1$	some	NI*	no
Tight Private Circuits (Section 3.4)	$t+1$	few	ISW	no
PINI (Section 3.2.2)	$t+1$	no	PINI1/2	yes

* The ISW multiplication and the multiplication of [Bel+16] are Non-Interferent (NI).

As shown in Table 3.1, the PINI masking schemes have excellent characteristics compared to the state-of-the-art: they are trivially composable, require the minimum number of shares and do not use refresh gadgets. Moreover, the PINI multiplication gadgets have good performance (see Section 3.2.6).

This chapter is organized as follows. We first recall the notion of simulatability in Section 3.1 and introduce the PINI definition and masking schemes in Section 3.2.2. We then present under a unified formalism some state-of-the-art trivially composable masking schemes, for which we also provide new security proofs and performance comparisons in the remaining part of Section 3.2. Next, we discuss other composable masking schemes in Section 3.3 and Section 3.4. We then discuss the tools that automate the security verification of masked gadgets in Section 3.5, focusing mainly on the maskVerif tool [Bar+19].[1] Finally, we conclude this chapter by showing an application of the PINI masking scheme to compute conversions between arithmetic and Boolean sharings in Section 3.6. The new conversion algorithms enjoy much simpler security proofs that the state-of-the-art algorithms in addition to better performance.

3.1 Simulatability

We use the simulatability framework [Bel+16] as a basis for our composable security notions. Intuitively, a set of probes is simulatable knowing some input shares if there exists a simulator (which is specific to that set of probes) that (knowing the said input shares) can generate simulated probes that have the same statistical distribution as the true probes. Simulatability means that the true probes are independent of the input

[1] I developed an alternative implementation of the verification algorithms of [Bar+19], which led to the correction of critical bugs (circuits wrongly reported as secure) and suboptimal heuristics in maskVerif. This improved the reliability and accuracy of maskVerif.

(a) If all inputs are deterministic, they are needed to simulate the output. **(b) When a fresh random is XORed, the output is independent of inputs.** **(c) Multiplication with an input is not bijective, thus input is needed.** **(d) With two probes, the random is not fresh.**

Figure 3.1: Simulatability examples. All outputs (bold, red) are probed. Inputs needed for simulation are circled in blue.

values that are not given to the simulator (see illustration in Figure 3.1). As extreme cases, if probes can be simulated using no input values, then the probes are independent on the inputs of the gadgets. On the other hand, any set of probes can trivially be simulated using all the input values: the simulator can run the gadget itself. Note that the notion of $(\mathcal{I}, \mathcal{O})$-Non-Interference introduced in [Bar+16] is equivalent.

Definition 10 (Simulatability)**.** *Let P be a set of l probes in a gadget G. Let $\mathcal{I}'$ be a set of k input shares of G. A simulator is a randomized function $\mathcal{S} : \mathbb{F}_q^k \to \mathbb{F}_q^l$. The set of probes P can be* simulated *with the set of input wires $\mathcal{I}'$ if and only if there exists a simulator $\mathcal{S}$ such that for any value of the input shares $\mathbf{x}$ of G, the distributions $\mathsf{G}_P(\mathbf{x})$ and $\mathcal{S}\left(\mathbf{x}_{|\mathcal{I}'}\right)$ are equal, where the probability is over the random coins in G and $\mathcal{S}$ and $\mathbf{x}_{|\mathcal{I}'}$ denotes the restriction of $\mathbf{x}$ to the k elements designated by $\mathcal{I}'$. The* simulatability set *of P denoted $\mathcal{S}^{\mathsf{G}}(P)$, is the smallest set of input shares of G such that P can be simulated with $\mathcal{S}^{\mathsf{G}}(P)$.*

Proposition 1. *If a set of probes P in a d-shares gadget G can be simulated with a set input wires $\mathcal{I}'$ that contains at most $d-1$ shares of each input sharing, then P is safe.*

Proof. For any input sharing $\mathbf{x}^i$ of G, let us denote by $\mathbf{x}^i_{|\mathcal{I}'}$ the shares of $\mathbf{x}^i$ that belong to $\mathbf{x}_{|\mathcal{I}'}$, hence $\left|\mathbf{x}^i_{|\mathcal{I}'}\right| \leq d-1$. Let $\mathbf{x}^i \leftarrow \mathsf{Enc}\left(x^i\right)$ for some sensitive value x^i, Lemma 1 implies that $\mathbf{x}^i_{|\mathcal{I}'}$ is independent of x^i. Therefore, $\mathcal{S}\left(\mathbf{x}_{|\mathcal{I}'}\right)$ are independent of all sensitive variables x^i. The same holds for $\mathsf{G}_P(\mathbf{x})$, therefore P is safe. □

Lemma 2. *The simulatability set is monotonic: let P and P' be sets of probes in a gadget G, then $P \subseteq P'$ implies $\mathcal{S}^{\mathsf{G}}(P) \subseteq \mathcal{S}^{\mathsf{G}}\left(P'\right)$*

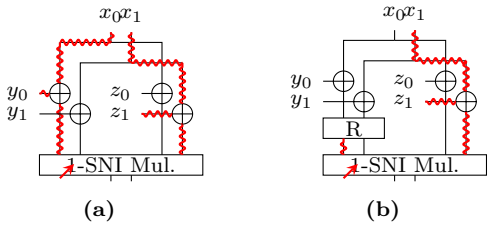


Figure 3.2: Implementation of $(x+y)(x+z)$ masked with $d=2$ shares, with input sharings (x_0,x_1), (y_0,y_1) and (z_0,z_1). The circuit is made of a 1-SNI multiplication with linear circuits at its inputs (two trivial implementations of the addition). The circuits illustrate (a) the limitation of SNI input composability and (b) a simple fix. 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 1-SNI refresh gadget.

Proof. P can be simulated using the shares $\mathcal{S}^{\mathsf{G}}\left(P'\right)$: simulate the probes in P', then output only the simulated probes that belong to P. □

3.1.1 Probe propagation

Let us illustrate how simulatability enables to analyze the security of gadget compositions with the idea of *probe propagation*. Our example is the simple circuit example of Figure 3.2a. This gadget is a two-share masked circuit that implement the function $f(x,y,z) = (x \oplus y)(x \oplus z)$, and it is the composition of three gadgets: two addition gadgets (implemented sharewise), and an ISW multiplication gadget. We assume that there is a single probe (the red arrow) in the multiplication gadget, and that this probe can be simulated with one share of each input sharing (we prove this later on). Let those shares be the red snake wires in the circuit, that we next call *propagated probes* (while the original probe is an *adversarial probe*). Next, we remark that each of these shares is the output of an addition gadget, in which it is computed as the sum of two input shares, and it can therefore be simulated by knowing these. We have now propagated back the original probe one additional step, reaching the input of the gadget under consideration.

In Figure 3.2a, a single probe may require both shares of x to be simulated. However, in Figure 3.2b, the refresh gadget is such that one can simulate a single output probe using no input: each output share is uniformly distributed. Therefore, a probe in the multiplication of this composite gadget can be simulated using at most one share of each input sharing. If we show this for all possible positions of the probe (in the refresh gadget and in the addition gadgets), we can then conclude that

the composite gadget is 1-probing secure.

Summarizing, the main idea of the probe propagation framework is to prove the security of an implementation by replacing the adversarial probes with propagated probes that can be used to simulate the adversarial probes, and by iterating the process until the propagated probes are all at the inputs of the circuit. The conclusion is then easy. More precisely, the propagation of probes always happens backwards in the circuit (probes on the outputs of a gadget are simulated by probes on the inputs of the gadget).

3.1.2 Simulation composition

The probe propagation framework can be formalized with the following propositions that analyze the simulatability of parallel and sequential gadget compositions.

Proposition 2 (Simulatability of parallel gadget composition)**.** *Let the gadget* $\mathsf{G} = \mathsf{G}_0 || \mathsf{G}_1$ *be a parallel composition of* G_0 *and* G_1 *whose sets of input shares are respectively* $\mathbf{x}$ *and* $\mathbf{y}$*. Let* P_0 *(respectively* P_1*) be a set of probes in* G_0 *(resp.* G_1*), we have*

$$\mathcal{S}^{\mathsf{G}}(P_0 \cup P_1) = \mathcal{S}^{\mathsf{G}_0}(P_0) \cup \mathcal{S}^{\mathsf{G}_1}(P_1) \ .$$

Proof. Let us first remark that the distributions $\mathsf{G}_{0P_0}(\mathbf{x})$ and $\mathsf{G}_{P_0}(\mathbf{x}, \mathbf{y})$ are equal for any value of $\mathbf{x}$ and $\mathbf{y}$, as well as $\mathsf{G}_{1P_1}(\mathbf{y})$ and $\mathsf{G}_{P_1}(\mathbf{x}, \mathbf{y})$. This implies that $\mathcal{S}^{\mathsf{G}_0}(P_0) = \mathcal{S}^{\mathsf{G}}(P_0) \subseteq \mathcal{S}^{\mathsf{G}}(P_0 \cup P_1)$ and $\mathcal{S}^{\mathsf{G}_1}(P_1) = \mathcal{S}^{\mathsf{G}}(P_1) \subseteq \mathcal{S}^{\mathsf{G}}(P_0 \cup P_1)$.

Moreover, the distributions of $\mathsf{G}_{P_1}(\mathbf{x}, \mathbf{y})$ and $\mathsf{G}_{P_2}(\mathbf{x}, \mathbf{y})$ are independent since each of them depends only on a distinct set of random gates (the ones in the corresponding subgadget). The simulator $\mathcal{S}$ for G that combines the simulators $\mathcal{S}^0$ and $\mathcal{S}^1$ is therefore correct, and $\mathcal{S}^{\mathsf{G}}(P_0 \cup P_1) \subseteq \mathcal{S}^{\mathsf{G}_0}(P_0) \cup \mathcal{S}^{\mathsf{G}_1}(P_1)$. □

Proposition 3 (Simulatability of sequential gadget composition)**.** *Let the gadget* $\mathsf{G} = \mathsf{G}_1 \circ \mathsf{G}_0$ *be the sequential composition of* G_0 *and* G_1 *whose sets of input shares are respectively* $\mathbf{x}$ *and* $\mathbf{y}$*. Let* P_0 *(respectively* P_1*) be a set of probes in* G_0 *(resp.* G_1*), we have*

$$\mathcal{S}^{\mathsf{G}}(P_0 \cup P_1) \subseteq \mathcal{S}^{\mathsf{G}_0}\left(P_0 \cup \mathcal{S}^{\mathsf{G}_1}(P_1)\right) \ .$$

Proof. Let $\mathcal{I}_0 = \mathcal{S}^{\mathsf{G}_0}\left(P_0 \cup \mathcal{S}^{\mathsf{G}_1}(P_1)\right)$ and $\mathcal{I}_1 = \mathcal{S}^{\mathsf{G}_1}(P_1)$. By definition, there exists a simulator $\mathcal{S}_1$ that simulates P_1 with $\mathcal{I}_1$ and a simulator $\mathcal{S}_0$ that simulates $P_0 \cup \mathcal{I}_1$ with $\mathcal{I}_0$. The simulator $\mathcal{S}$ is built as follows:

using the input shares $\mathcal{I}_0$, run $\mathcal{S}_0$ to simulate the probes in P_0 and the values of $\mathcal{I}_1$, then use these values and the simulator $\mathcal{S}_1$ to simulate the probes in P_1. Let us remark that in G, the distribution of $\mathsf{G}_{P_1}(\mathbf{y})$ only depends on the randomness in G_1 and on $\mathbf{y}_{|\mathcal{I}_1}$, it is therefore independent of the distribution of $\mathsf{G}_{P_0}(\mathbf{x})$ when conditioned on $\mathbf{y}_{|\mathcal{I}_1}$. The simulation of $P_0 \cup P_1$ by $\mathcal{S}$ is therefore correct. □

3.1.3 Simulation-based probing security

We considered so far only the simulation of a single set of probes, concluding that it is safe if it can be simulated with at most $d-1$ shares of each sharing. Therefore, if any set of at most t probes satisfies this condition, the considered gadget is t-probing-secure.

Definition 11 (NI)**.** *A d-shares gadget G is t-non-interferent (t-NI) if any set of probes P such that $|P| \leq t$ can be simulated using at most t shares of each input sharing.*

Proposition 4. *Any t-NI gadget with at least $t+1$ shares is t-probing secure.*

Proof. Any set of at most t probes can be simulated with t shares of each input sharing and is therefore safe, by Proposition 1. □

The converse is not true: there exists probing-secure gadgets which are not NI, such as the first-order Toffoli gate of [Fau+18], which is similar to the ISW multiplication, but where the shares of the input to be added are used in place of the randomness. Indeed, in Algorithm 3, any intermediate variable except the outputs (that is, $\mathbf{x}_i\mathbf{y}_i$, $\mathbf{x}_i\mathbf{y}_{1-i}$ and $\mathbf{x}_i\mathbf{y}_{1-i} + \mathbf{z}_i$ for $i = 0, 1$) can be simulated using at most one share of each input sharing, hence a probe on one of these variables is safe. For the outputs, since the sharing $\mathbf{z}$ is uniform and independent of the sharings $\mathbf{x}$ and $\mathbf{y}$, the share $\mathbf{w}_0 = \mathbf{x}_0\mathbf{y}_0 + \mathbf{x}_0\mathbf{y}_1 + \mathbf{z}_0$ is uniformly distributed, and independent of any sensitive value (and likewise for $\mathbf{w}_1$). On the other hand, Algorithm 3 is not 1-NI: simulating $\mathbf{w}_0$ requires knowledge of $\mathbf{x}_0$, $\mathbf{y}_0$, $\mathbf{y}_1$ and $\mathbf{z}_0$.

Algorithm 3 First-order Toffoli gate with two shares.

Input: Sharings $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$
Output: Sharing $\mathbf{w}$ such that $w = xy + z$.

1: $\mathbf{w}_0 \leftarrow \mathbf{x}_0\mathbf{y}_0 + (\mathbf{x}_0\mathbf{y}_1 + \mathbf{z}_0)$
2: $\mathbf{w}_1 \leftarrow \mathbf{x}_1\mathbf{y}_1 + (\mathbf{x}_1\mathbf{y}_0 + \mathbf{z}_1)$

The use of simulatable gadgets solves the second part of masking verification scaling problem introduced in Section 2.4.2: given a composite gadget, one may enumerate all sets of possible probes, then use the probe propagation framework (as formalized by Propositions 2, 3 and 4): compute the inputs required to simulate the (propagated) probes on each composing gadget, then conclude that the composition is (or is not) NI. Most of the work in this verification strategy happens at the level of the composing gadgets. Therefore, by building complex circuits from small composing gadgets, most of the analysis is done on small gadgets which have few input shares and random gates, which eases the computation of distributions. Furthermore, the analysis techniques presented in Section 3.5 can compute the simulatability set without resorting to the computation of full distributions.

This approach does however not solve the other issue with masking verification, which is the need for enumerating all possible sets of probes. We next discuss composition strategies that tackle this problem.

3.2 Trivial compositions

We first consider *trivially composable* strategies. Masking schemes based on this strategy define a set of gadgets that can be arbitrarily composed, using recursively parallel and serial composition to build larger circuits that retain some security.

The first strategy uses only sharewise gadgets and is therefore limited, but it provides the intuitions for our probe-isolating non-interference (PINI) strategy [CS20] which is introduced next. We then move to older strategies: NI-based composition [Bar+16], refreshed multiplication [GR17], and the original ISW scheme [ISW03]. We present each of them and our contributions to the analysis and implementation of these schemes.

3.2.1 Sharewise gadgets

We first take as example a masking scheme whose gadgets are the sharewise implementations of affine functions (with all gadgets having the same number of shares), which includes copy and identity gadgets.[2] We show that such composite gadgets are NI.

[2] Such a composition is admittedly not useful since the composition of affine function is an affine function itself, but the discussion on this example lays the ground for more interesting schemes.

Proposition 5. *The composition of sharewise d-shares gadgets is $d-1$-NI.*

Proof. Let us first show that a single sharewise gadget G is NI. The gadget G can be partitioned in d non-connected sub-circuits G_i for $i = 0, \ldots, d-1$ (next named gadget shares) such that all input and output shares with index i belong to G_i. The sub-circuits G_i are next called *circuit shares* (this is similar to the definition of *domain* in [GMK16a]). If P is a set of $d-1$ probes, there exists $i^* \in [\![0, d[\![$ such that there is no probe in G_{i^*}. Therefore, the probes P can be simulated by knowing all input shares whose index belongs to $[\![0, d[\![\setminus \{i^*\}$. Since a composition of sharewise gadgets is still sharewise, the proof directly applies in the general case. □

3.2.2 PINI composition

While sharewise gadgets are trivially composable and cheap to implement ($\mathcal{O}(d)$ overhead), they can only implement affine functions. The *probe-isolating non-interference* (PINI) composition strategy [CS20] is based on the idea of *simulating* share isolation, even in non-linear gadgets. That is, even though a gadget is not share-isolating, output shares and internal probes can be simulated using the same input shares as if it was share-isolating. In this way, the core observation that having more (simulated) circuit shares than probes ($d > t$) implies t-probing security is still applicable.

Definition 12 (PINI)**.** *A d-shares gadget G is t-probe-isolating non-interferent (t-PINI) if, for any set $A \subseteq [\![0, d[\![$ and any set of probes P such that $|A| + |P| \leq t$, there exists a set $B \subseteq [\![0, d[\![$, $|B| \leq |P|$ such that the probes P and all output shares of G with index in A can be simulated by the inputs of G with index in $A \cup B$.*

We remark that any t-PINI gadget is also t-NI, since the PINI definition is strictly stronger than NI. Furthermore, a t-PINI gadget is t'-PINI for any $t' \leq t$ by definition, and a d-shares $d-1$-PINI gadget is t-PINI for any $t \geq d$: B can be chosen such that $A \cup B = [\![0, d[\![$. Such a gadget is therefore t-PINI for any t, hence we simply say that it is PINI.

Proposition 6 (Sharewise PINI)**.** *Any sharewise gadget is PINI.*

Proof. The proof is similar to the proof of Proposition 5: based on the same partition of the gadget into gadget shares, the set B of size $|P|$ can be identified such that all probes in P can be simulated using the input shares with index in B. Then, the output shares with index in A can be simulated using the input shares with index in A. □

Theorem 1 (PINI composition)**.** *The parallel, sequential and extended sequential composition of t-PINI gadgets is t-PINI.*

Proof. Let us first consider parallel composition of G_1 and G_2 ($\mathsf{G} = \mathsf{G}_1 \| \mathsf{G}_2$), with probe sets P_1 and P_2 in each of them, respectively. Let $A \subseteq [\![0, d[\![$ and $P = P_1 \cup P_2$ be such that $|A| + |P| \leq t$. The set B_1 (resp. B_2) is defined using the t-PINI simulator for G_1 (resp. G_2), with probes P_1 (resp. P_2) and output share indexes A. As a result, $|B_i| \leq |P_i|$ for $i = 1, 2$, therefore $|B| = |B_1 \cup B_2| \leq |P_1 \cup P_2|$ since $P_1 \cap P_2 = \emptyset$, where $B = B_1 \cup B_2$. The two simulators can be combined with Proposition 2 to simulate the probes P and output shares of G with index in A with the input shares of G whose index belongs to $A \cup B$.

For the serial composition, let $\mathsf{G} = \mathsf{G}_2 \circ \mathsf{G}_1$, and P_1, P_2 defined as previously. Using the t-PINI simulator for G_2 gives B_2 such that $|B_2| \leq |P_2|$ and all probes in P_2 as well as all output shares of G_2 (which are the output shares of G) can be simulated with the input shares of G_2 whose indexes belong to $A \cup B_2$. A similar reasoning with G_1, but with output share indexes $A \cup B_2$ gives B_1 (with $|B_1| \leq |P_1|$) such that the probes in P_1 and all output shares of G_1 with index in $A \cup B_2$ can be simulated with the input shares of G_1 (which are the input shares of G) with index in $A \cup B_2 \cup B_1$. Similarly to the parallel composition case, we let $B = B_1 \cup B_2$ and remark that $|B| \leq |P|$. Finally, both simulators are combined with Proposition 3, resulting in the simulation of all the probes in P and all the output shares of G whose index belongs to A. This simulation is performed with the input shares of G with index in $A \cup B$.

Extended sequential composition is a mix of parallel and sequential compositions which preserve the PINI property, with the addition of identity gadgets which are PINI (as a consequence of Proposition 6). □

While this proof is more complex than the one of Proposition 5, it exhibits the interest of simulatability-based notions: they enable to prove the security of the composition of any gadgets that satisfy them. It also brings us closer to the implementation of arbitrary masked arithmetic circuits, the only missing piece being a PINI multiplication gadget.

Such a gadget is shown in Algorithm 4. It is very close to the ISW multiplication (Algorithm 2), the only difference being the replacement of $\mathbf{x}_i\mathbf{y}_j + r_{ij}$ with $(\mathbf{x}_i + 1)\, r_{ij} + \mathbf{x}_i (\mathbf{y}_j - r_{ij})$. Both computations give the same result, but the first one allows to probe $\mathbf{x}_i\mathbf{y}_j$, which leaks two input shares with distinct share indices, while the second one avoids it by masking $\mathbf{y}_j$ with $-r_{ij}$ prior to the multiplication.

Proposition 7. *Algorithm 4 with d is $d-1$-PINI.*

Algorithm 4 PINI1 multiplication gadget with d shares.

Input: Sharings $\mathbf{x}$, $\mathbf{y}$
Output: Sharing $\mathbf{z}$ such that $z = x \cdot y$.

1: **for** $i = 0$ to $d-1$ **do**
2: **for** $j = i+1$ to $d-1$ **do**
3: $r_{ij} \xleftarrow{\$} \mathbb{F}_q$; $r_{ji} \leftarrow -r_{ij}$
4: **for** $i = 0$ to $d-1$ **do**
5: $\mathbf{z}_i \leftarrow \mathbf{x}_i\mathbf{y}_i$
6: **for** $j = 0$ to $d-1, j \neq i$ **do**
7: $\mathbf{z}_i \leftarrow \mathbf{z}_i + \big((\mathbf{x}_i + 1)\, r_{ij} + \mathbf{x}_i \big(\mathbf{y}_j - r_{ij}\big)\big)$

Proof. Without loss of generality, we assume that the only probes in P are probes either on the single terms $(\mathbf{x}_i + 1)r_{ij}$ and $\mathbf{x}_i(\mathbf{y}_j - r_{ij})$, or probes on sums of these terms (including the term $\mathbf{x}_i\mathbf{y}_i$). This can be done since (1) a probe on $\mathbf{x}_i$, $\mathbf{x}_i + 1$ or $\mathbf{y}_i$ can be excluded from the set of probes, running the simulator and then adding i to B; (2) to simulate a probe on $\mathbf{x}_i(\mathbf{y}_j - r_{ij})$, we simulate $\mathbf{y}_j - r_{ij}$, make the latter probe easier to simulate than the former one; (3) likewise, we simulate r_{ij} as an intermediate step to the simulation of $(\mathbf{x}_i + 1)r_{ij}$. Let P_{ij} contain all the single term probes $(\mathbf{x}_i + 1)r_{ij}$ and $\mathbf{x}_i(\mathbf{y}_j - r_{ij})$ for every i and j, and let P_i' contain all the probed sums involved in the computation of $\mathbf{z}_i$ for very i, which ensures that the sets P_{ij} and P_i form a partition of P.

The simulator initializes B' to the set A, and adds to it every i for which $P_i' \neq \varnothing$. Then, for every pair (i, j) such that $i < j$, let $c_{ij} = \big|\{i, j\} \cap B'\big| + \big|P_{ij}\big| + \big|P_{ji}\big|$. If $c_{ij} \geq 2$, i and j are added to B if they don't belong to it already. Otherwise, if $P_{ij} \neq \varnothing$, i is added to B', while j is added to B' if $P_{ji} \neq \varnothing$. This ensures that $|B| \leq |P|$, where $B = B' \setminus A$.

Let us now show how to simulate the probes. For every probe in P_{ij}, $\mathbf{x}_i$ is known. If $j \in B'$ or if the probe to simulate is $(\mathbf{x}_i + 1)r_{ij}$, then r_{ij} is computed (as $-r_{ji}$ if that variable is already computed, otherwise it is taken uniformly at random), and then the probe can be simulated following the computations of the gadget. Otherwise, $j \notin B'$ and the probe to simulate is $\mathbf{x}_i(\mathbf{y}_j - r_{ij})$. Since $j \notin B'$, no other probe depends on r_{ij} or r_{ji}, and furthermore no intermediate sum of $\mathbf{z}_j$, nor $\mathbf{z}_j$ itself has to be simulated. Therefore, r_{ij} is not observed by the adversary except through the probe $\mathbf{x}_i(\mathbf{y}_j - r_{ij})$, which can therefore be simulated as $\mathbf{x}_i r_{ij}$, where r_{ij} is taken uniformly at random. Finally, for the outputs

and the intermediate sums, the terms $\mathbf{x}_i\mathbf{y}_j$ are always easy to simulate since they only have to be simulated when $i \in B'$. For every other term $(\mathbf{x}_i + 1)r_{ij} + \mathbf{x}_i(\mathbf{y}_j - r_{ij})$ that should be simulated, either $i, j \in B'$ and simulation is trivial, or r_{ij} is only observed through this term. In the latter case, since the term is equal to $\mathbf{x}_i \cdot \mathbf{y}_j + r_{ij}$, it can be simulated as a fresh random. □

In [CS19], we introduce another PINI multiplication gadget (PINI2), with a randomness usage reduced compared to Algorithm 4: $\lfloor (d-1)^2/4 \rfloor + 2d - 1$ instead of $d(d-1)/2$. Concretely, PINI2 uses less random elements than PINI1 only when $d \geq 8$, which makes it relevant only when a very high number of shares is needed, such as in low-noise platforms [BS21].

3.2.3 NI with refreshed-copy

Barthe et al. introduced in [Bar+16] a composition strategy based on the weaker NI notion. Indeed, single-output NI gadgets are trivially composable: any parallel or sequential of such gadgets is NI. In the probe propagation framework, when considering the backwards propagation of probes, the gadgets form a multitree through which each probe propagates without being duplicated. Using a NI multiplication and sharewise gadgets, this allows to perform any computation, although not efficiently: since copy gates are not single-output, any intermediate value in the computation can be used only once, and re-computing an intermediate value every time it is used leads to an overhead exponential in the circuit depth.

Algorithm 5 ISW-based SNI refresh gadget.

Input: Sharing $\mathbf{x}$
Output: Sharing $\mathbf{y}$ such that $y = x$.
1: $\mathbf{y} \leftarrow \mathsf{ISW}(\mathbf{x}, (1, 0, \ldots, 0))$ ▷ Algorithm 2

Algorithm 6 Refreshed copy gadget.

Input: Sharing $\mathbf{x}$
Output: Sharings $\mathbf{y}$ and $\mathbf{z}$ such that $z = y = x$.
1: $\mathbf{y} \leftarrow \mathsf{RefreshSNI}(\mathbf{x})$ ▷ e.g., Algorithm 5
2: $\mathbf{z} \leftarrow \mathbf{x}$

This limitation was circumvented by using use a *refreshed-copy* gadget which is trivially composable with single-output NI gadgets, as shown in [Bar+16] (Proposition 4). The refreshed-copy gadget takes as input

one sharing and outputs two sharings: one copy of the input sharing, and a refreshed copy (Algorithm 6). The refresh gadget must be strongly non-interferent (SNI). In the probe propagation framework, probes inside a SNI gadget propagate to the inputs (similarly to NI), but the output probes do not propagate at all. Therefore, a SNI refresh gadget can be viewed as a "propagation stopper".

Definition 13 (SNI)**.** *A d-shares gadget* G *is t-strong-non-interferent (t-SNI) if any set of probes P and output shares O such that $|P| + |O| \leq t$ can be simulated using at most $|P|$ shares of each input sharing.*

When based on a SNI refresh, the refreshed-copy gadget preserves (from the point of view of probe propagation) the multitree structure since the refreshed branch does not propagate the probes. There exists multiple SNI refresh gadgets. Barthe at al. built theirs from the ISW multiplication, by setting one input sharing to a constant sharing of 1 (see Algorithm 5). We introduce a new SNI refresh with a reduced complexity: $\mathcal{O}(d \log d)$ additions and randomness while Algorithm 5 has a complexity of $\mathcal{O}\left(d^2\right)$. We postpone the introduction and security proof of this gadget to Section 4.3.3 and Algorithm 23, since it is SNI in the more general glitch-robust probing model, which implies that it is SNI in the threshold probing model.

The main drawback of the refreshed-copy composition strategy is the use of refresh gadgets when a variable is used more than once, even in the middle of a linear layer. In computations with many linear operations and few non-linear ones (such as some lightweight block ciphers [Bel+20d] or lattice-based schemes [Bos+18]), this causes a large computational and randomness overhead.

3.2.4 Refreshed multiplication

Goudarzi and Rivain [GR17] avoided the aforementioned overheads by introducing a composition strategy that uses refresh gadgets only at the input of multiplication gadgets. More precisely, it is a trivial composition based on sharewise gadgets and the ISW multiplication. For affine operations (including copy), a sharewise gadget is used, while multiplications are implemented with Algorithm 7: one of the inputs of the multiplication is refreshed (using the SNI refresh derived from the ISW multiplication), then an ISW multiplication gadget performs the actual multiplication. This strategy was introduced without proof, and we prove its security by showing that the "refreshed multiplication"

gadget (Algorithm 7) is PINI.[3]

Algorithm 7 Refreshed multiplication gadget.

Input: Sharings $\mathbf{x}$, $\mathbf{y}$
Output: Sharing $\mathbf{z}$ such that $z = x \cdot y$.
1: $\mathbf{t} \leftarrow \mathsf{RefreshSNI}(\mathbf{x})$
2: $\mathbf{z} \leftarrow \mathsf{Mul\text{-}LPINI}(\mathbf{t}, \mathbf{y})$ ▷ E.g., ISW multiplication

Setting the stage for the next chapter, we actually prove a more general result, allowing the usage of any SNI refresh gadget and any limited-PINI (LPINI, Definition 14) multiplication (this notion is weaker than SNI for multiplication gadgets). Intuitively, the SNI refresh stops the propagation of probes from its outputs to its inputs, hence only the probes in the refresh propagate to the input of the refreshed multiplication, which ensures that the PINI requirements are satisfied for the refreshed input sharing (input sharing $\mathbf{x}$ in Algorithm 7). For the other input sharing, if the inner multiplication gadget is NI, each internal probe propagates to a single share, which again satisfies the PINI requirements. However, a probe on an output share of a refreshed multiplication gadget can be propagated into a probe with a different share index on the input sharing $\mathbf{y}$, which violates the PINI definition. Therefore, we need a slightly stronger constraint on the inner multiplication gadget to fix this.

Definition 14 (t-Limited-PINI)**.** *Let* G *be a* d*-shares gadget,* I *a set of its input sharings,* P_1 *a set of internal probes and* A *a set of share indexes (in* $[\![0, d[\![$*) such that* $|P_1| + |A| \leq t$*. Let* P_2 *be the set of all output shares of* G *whose index is in* A*. The gadget* G *is* t*-Limited-PINI (*t*-LPINI) with respect to* I *if, for any* P_1 *and* A*, there exists a set* B *of at most* $|P_1|$ *shares indexes such that the set of probes* $P_1 \cup P_2$ *can be simulated using the shares with indexes in* $A \cup B$ *for input sharings not in* I*, and at most* t *shares of each input sharing in* I*.*

Remark. t-LPINI stands between t-PINI and t-NI: if the set I is empty, t-LPINI is the same as t-PINI; if it contains all the input sharings, t-LPINI is t-NI.

Let us now prove that if the inner multiplication gadget in Algorithm 7 is LPINI with respect to the input sharing $\mathbf{y}$ ($I = \{\mathbf{y}\}$), then Algorithm 7 is PINI.

[3] Regarding performance, Algorithm 7 uses more randomness than the PINI1 multiplication (Algorithm 4), where the overhead (up to 2x) depends on performance of the RefreshSNI gadget used. For other operations, PINI1 costs up to twice as much as Algorithm 7.

Lemma 3. *Let G_m be a m-input t-LPINI gadget with respect to the set of input sharings I ($= [\![0, k[\![$, wlog), and G_r be a t-SNI refresh gadget. Let G be the gadget built by using G_r to refresh the inputs of G_m not in I: $\mathsf{G} = (\mathsf{Id}\| \dots \|\mathsf{Id}\|\mathsf{G}_r\| \dots \|\mathsf{G}_r) \circ \mathsf{G}_m$ (where Id is the identity gadget). The gadget G is (glitch-robust) t-PINI.*

Proof. Let us consider a set of probes P_m in G_m and a set P_r^i of probes in each refresh gadget G_r^i, as well as a set of shares B such that $|P_m| + \sum_{i=k}^{m-1} \left|P_r^i\right| \leq t$. The probes P_m and output shares of G_m with index in B can be simulated using the input shares of G_m with index in $A_m \cup B$ (for some A_m such that $|A_m| \leq |P_m|$) of the sharings that belong to I, and with a set of shares S^i (such that $\left|S^i\right| \leq |P_m| + |B|$) for each other input sharing i of G_m. Then, for each refresh gadget G_r^i, the output shares S^i and the probes P_r^i can be simulated using the input shares with index in A_i, where $|A_i| \leq \left|P_r^i\right|$. Summarizing, using Proposition 2 and Proposition 3, it comes that all probes and required output shares can be simulated with the input shares with index in $A \cup B$, where $A = A_m \cup \bigcup_{i=k}^{m-1} A_i$. □

Corollary 1. *The refreshed multiplication gadget (Algorithm 7) is PINI.*

Proof. This is a direct application of Lemma 3. □

In Algorithm 7, any SNI refresh algorithm from the previous section, while for the L-PINI multiplication, we next show that the ISW multiplication can be used, since it is $(d-1)$-SNI and has a single output.

Proposition 8. *Any single-output t-SNI gadget G is t-LPINI with respect to any set containing one of its input sharings.*

Proof. Let P_1 and P_2 follow the L-PINI definition. Since G has a single input, $|P_1| + |P_2| \leq t$, therefore both sets can be simulated using at most $|P_1|$ shares from each input sharing. It is then clear that G is t-LPINI with respect to a set containing one input sharing by setting B to the indices of the shares of that sharing that are required by the SNI simulator. □

3.2.5 ISW composition

Let us now finally give a new simulation-based proof for the original masking scheme of ISW. This scheme is a trivial composition of sharewise gadgets and the ISW multiplication. While this masking scheme is trivially composable, it does not achieve optimal masking order: it requires $d = 2t + 1$ shares to reach t-probing security.

Our new proof is based on the observation that the ISW multiplication gadget, while not PINI, satisfies a close variant, where the constraint $|B| \leq |P|$ is changed to $|B| \leq 2|P|$. This new notion, that we here call duplicated-PINI enjoys the same composability properties as PINI but it does not imply NI. However, duplicated-PINI implies t-probing security when the number of shares is at least $2t + 1$.

Definition 15. *A d-shares gadget G is duplicated t-probe-isolating non-interferent (duplicated t-PINI) if, for any set $A \subseteq [\![0, d[\![$ and any set of probes P such that $|A| + 2|P| \leq 2t$, there exists a set $B \subseteq [\![0, d[\![$, $|B| \leq 2|P|$ such that the probes P and all output shares of G with index in A can be simulated by the inputs of G with index in $A \cup B$.*

Proposition 9. *The ISW multiplication gadget (Algorithm 2) with d shares is duplicated $\lfloor (d-1)/2 \rfloor$-PINI.*

Proof. Since the gadget is $d-1$-SNI, the outputs with index in A and the probes in P can be simulated using at most $|P|$ shares of each input sharing. Let B be the set of indices of these input shares, which implies that $|B| \leq 2|P|$. □

Proposition 10. *Any sharewise gadget is duplicated t-PINI for any t.*

Proof. This proof is identical to the proof of Proposition 6. □

Proposition 11 (duplicated PINI composition)**.** *The parallel, sequential and extended sequential composition of duplicated t-PINI gadgets is duplicated t-PINI.*

Proof. The proof is very similar to the proof of Theorem 1, except for the following changes in the sizes of some sets. For parallel composition, the sets of probes are such that $|A| + 2|P| \leq 2t$ and the inputs required by the simulators satisfy $|B| \leq |B_1 \cup B_2| \leq 2|P_1 \cup P_2|$. For the sequential composition, we have $|B_2| \leq 2|P_2|$ and $|B_1| \leq 2|P_1|$, giving $|B| \leq 2|P|$. Finally, for extended sequential composition, the identity gadget is duplicated t-PINI thanks to Proposition 10. □

Proposition 12. *Any duplicated t-PINI gadget with at least $2t+1$ shares is t-probing secure.*

Proof. Any set of at most t probes can be simulated with $2t$ shares of each input sharing and is therefore safe, by Proposition 1. □

Table 3.2: Performance and security characteristics of various multiplication and refresh gadgets.

	Gadget	Security	Randomness	Add. & Sub.	Mul.
Mul.	ISW (Algorithm 2)	SNI	$\frac{d(d-1)}{2}$	$2d(d-1)$	d^2
	Belaid et al. [Bel+16]	LPINI*	$\left\lfloor \frac{d^2+2d-3}{4} \right\rfloor$	$\left\lfloor \frac{3d^2-3}{2} \right\rfloor$	d^2
	PINI1 (Algorithm 4)	PINI	$\frac{d(d-1)}{2}$	$4d(d-1)$	$2d^2$
	PINI2	PINI	$\left\lfloor \frac{d^2+6d-3}{4} \right\rfloor$	$\left\lfloor \frac{4d^2-3d-3}{2} \right\rfloor$	$2d^2$
Refresh	RefreshISW	SNI	$\frac{d(d-1)}{2}$	$d(d-1)$	0
	Bat.	SNI	$d\left(\log_2 d - \frac{1}{2}\right)^\dagger$	$d\,(2\log_2 d - 1)^\dagger$	0
	New	SNI	$\frac{d}{2}\log_2 d^\dagger$	$d\log_2 d^\dagger$	0

* It is only proven NI in [Bel+16], but the simulator built in that proof satisfies the LPINI condition with respect to any set containing a single input sharing.

† Assuming that d is a power of 2 and $d \neq 1$.

3.2.6 Performance of trivial composition strategies

Summarizing, we presented four trivial composition strategies (excluding the one based only on sharewise gadgets, which does not enable universal computations).

In chronological order, the ISW strategy came first, and introduce a very simple (and widely used) multiplication gadget, but the composition does not achieve optimal security order: the number of shares is twice the minimum required, which causes a roughly 4x cost increase (since the multiplication cost is quadratic in the number of shares).

Next, the composition based on single-output NI gadgets and SNI refresh (refreshed copy) has optimal order (i.e., $d = t+1$) but requires the addition of refresh gadgets. While recent SNI refresh gadget have a small cost compared to the multiplication when d becomes large ($\mathcal{O}(d \log d)$ versus $\mathcal{O}\left(d^2\right)$), they incur in practice significant cost overheads (see Table 3.2), especially in computations composed mainly of linear operations, which have only $\mathcal{O}(d)$ cost.

Finally, the refreshed multiplication and PINI strategies avoid this issue by allowing the use of plain copy gadgets. This requires a PINI multiplication gadget, which can either be built from scratch (PINI1 and PINI2), or from a LPINI multiplication and a SNI refresh gadget. Thanks to their specialization, the PINI1 and PINI2 gadgets have a lower randomness requirement than the refreshed multiplication constructions. This comes at the expense of a larger number of multiplications, but these are usually less costly than randomness, especially in small fields [JS17]. Regarding the comparison of the PINI1 and PINI2 gadgets, the former has the advantage of simplicity, and it is more efficient for low number

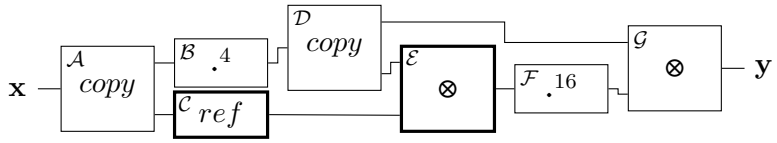


Figure 3.3: Part of AES S-box multiplication chain of [RP10] in $\mathbb{F}_{256}$. Thick boxes represent SNI gadgets while the other ones are NI, and furthermore the gadgets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{D}$ and $\mathcal{F}$ are sharewise.

of shares (e.g., it uses less randomness for $d < 8$), but becomes worse as the number of shares grows.

3.3 Generic simulation-based composition

Various trivial composition strategies based on simulatability-based security notions have been presented in the previous section. Each strategy has specific advantages and limitations, and one may wonder if combining these strategies can improve performance. A generic simulation-based strategy can achieve this, and furthermore it gives freedom to exploit the properties of all available gadgets. For example, the ISW multiplication is SNI, and this was not exploited by the refreshed-copy strategy.

Intuitively, this generic strategy amounts to direct proofs in the probe propagation framework: generically, we consider that there is any number of probes t_i in each gadget G_i such that $\sum_i t_i \leq t$, then propagate these probes. This generic propagation allows to write the set of probes on each sharing as the union of some share indexes sets whose cardinality is bounded by a sum of t_i's. One can then check whether the description of the probes on the input sharings guarantees that the composite circuit is NI, SNI or PINI.

As an example, we consider the analysis of the masked AES S-box in $\mathbb{F}_{256}$ by Belaïd et al. [Bel+16]. More precisely, we discuss a sub-part of it that performs the masked computation of $y = x^{84}$, as shown in Figure 3.3. Let us assume that all gadgets have d shares and are $d-1$-NI or SNI, we want to prove that the composition is $d-1$-NI. Let us denote by $p_{\mathcal{A}}$ the number of probes in the composing gadget $\mathcal{A}$, and similarly for the gadgets $\mathcal{B}, \ldots, \mathcal{G}$. Therefore, the probes in the NI multiplication gadget $\mathcal{G}$ can be simulated with $p_{\mathcal{G}}$ shares of each of its input sharings. Using probe propagation, we see that simulation of the probes on the gadgets $\mathcal{F}$ and $\mathcal{G}$ require at most $p_{\mathcal{F}} + p_{\mathcal{G}}$ output shares of $\mathcal{E}$. Let us assume that all gadgets have d shares and are $d-1$-NI or SNI. Since we want to prove that the composition is $d-1$-NI, we know that $p_{\mathcal{E}} + p_{\mathcal{F}} + p_{\mathcal{G}} \leq d-1$, and thus, since $\mathcal{E}$ is SNI, at most $p_{\mathcal{E}}$ shares of each input of $\mathcal{E}$ are needed for

the simulation of the probes on $\mathcal{E}$, $\mathcal{F}$ and $\mathcal{G}$ (as well as up to $p_\mathcal{G}$ shares of the upper output of $\mathcal{D}$). As a result, up to $p_\mathcal{D} + p_\mathcal{E} + p_\mathcal{F} + p_\mathcal{G}$ input shares of $\mathcal{D}$ are required for the simulation. We here see the need for $\mathcal{E}$ to be SNI, if it were only NI, $p_\mathcal{D} + p_\mathcal{E} + p_\mathcal{F} + 2p_\mathcal{G}$ input shares would be needed, which could exceed $d - 1$ and break the security. Continuing the probe propagation, we conclude that at most $p_\mathcal{A} + p_\mathcal{B} + \cdots + p_\mathcal{G}$ input shares of $\mathcal{A}$ are needed to simulate the whole gadget, therefore the composition is NI. Moreover, the SNI refresh $\mathcal{C}$ is required, as without it the previous sum would include twice the term $p_\mathcal{E}$.

This proof technique was formalized as a type system in [Bar+16] and implemented in the tool maskComp which automatically and greedily inserts SNI refresh gadgets to ensure that the composition is NI. The resulting composition can then be optimized by observing that a NI multiplication with a SNI refresh at its output can be replaced with a SNI multiplication.

We next briefly discuss another formalization of this proof technique introduced in [CS20]. There, the composition is represented as a graph whose nodes are the gadgets, and whose edges are their input/output sharings. Matching the intuition of SNI gadgets blocking the probe propagation, the output edge of each SNI gadget is removed in this graph. The NI or SNI security of the composition is then inferred from the structure of the graph. Namely, a composition is NI if there is at most one directed path between any two gadgets. Moreover, if there is no path between any input and output sharing, the gadget is SNI. Similarly to maskComp, a tool [CS20] can insert refresh gadgets needed to satisfy a given security notion. However, unlike maskComp, this insertion of refresh gadgets is done optimally, minimizing their number compared to the greedy strategy (and taking into account the relative cost of a SNI refresh w.r.t. a SNI multiplication).

The composition strategies presented in this section can significantly reduce the number of refresh gadgets required, compared to the refreshed-copy strategy: the optimization uses only 41 SNI gadgets instead of 113 for the bit-level representation of the AES S-box of Boyar and Peralta [BP12] that uses 32 AND gates. However, the PINI-based techniques have better performance (on this use-case), requiring 32 refresh gadgets for the refreshed multiplication strategy, or no refresh but a bit more arithmetic operations for PINI_1.

3.4 Tight private circuits

The tight private circuits (TPC) [BGR18] strategy improves on the previous composition strategy by moving beyond simulation-based notions: it builds composite circuits which are probing secure, but not necessarily NI. The security proof can therefore not be explained within the probe propagation framework.

A TPC is a composition of ISW multiplications (actually, any SNI multiplication could be used), SNI refresh, addition and copy gadgets (in $\mathbb{F}_{2^k}$), and these circuits are tight in the sense that removing any refresh gadgets breaks the probing security. The computation of the locations where a refresh gadget should be inserted is done by the tightPROVE tool. For the bit-level AES S-box from [BP12], tightPROVE shows that no refresh gadget is needed.

For secure composition of large gadgets, more general composition theorems are introduced in [BGR18]: the sequential composition of two probing-secure circuits is probing-secure if either the first circuit is linear, sharewise, and furthermore implements a surjective function, or if every output of the first circuit is the output of a single-output SNI gadget.

Together, these results provide a strategy to implement block ciphers such as the AES: build a TPC for the non-linear part of S-box whose outputs are all outputs of SNI gadgets, then compose them in parallel (this trivially preserves probing security). If the linear layer of the block cipher is surjective, it can be composed with the S-box layer to form a round, which can then be composed since their outputs are outputs of SNI gadgets.

3.5 Automated verification of gadgets

Since the previous composition strategies rely on the probing-security, NI, SNI or PINI security properties, there is a need for methodologies to verify that gadgets satisfy these properties. We already used two techniques: composition theorems and hand-made proofs for small and structured gadgets (e.g., ISW and PINI1 multiplication). For gadgets that are not compositions, or that do not follow composition strategies, hand-made proof are often tedious or even infeasible if the gadget is large and not well-structured. Many tools have therefore been introduced to automate these proofs, such as checkMasks [Cor18], Rebecca [Blo+18], maskVerif [Bar+19], SILVER [KSM20] and IronMask [Bel+21].

Before detailing the principle behind these tools, let us remark that they have been used for two mostly distinct purposes. One is to prove the

probing security of a full circuit. As already discussed in Section 2.4.2, this can be done only in limited cases due to computational complexity. Practically, first-order security can be verified for large circuits, while for relatively small circuits (typically an S-box), verification up to order 3 or 4 [Bar+19] can be achieved. The other use-case is to design small gadgets (e.g., SNI refresh or multiplication) which, for a specific number of shares, are more optimized than the generic constructions. In particular, SNI refresh gadgets have been designed with this technique for up to 16 shares, and with significant improvement over the best order-generic constructions [Cas+21b].

All the automated verification tools are based on the same general principle: they explore all possible probe sets (including outputs to simulate), and for each of them, they test if it falsifies the security property. This test can actually be implemented from a function that computes the simulatability set of a set of probes (see Definition 10). For simulatability-based notions, the simulatability set can be directly compared the requirements (a bound on the number of shares from each input sharings for NI/SNI, and a constraint on the set of input share indexes for PINI). For probing security, the gadget to evaluate is first sequentially composed with encoder circuits (Enc) to generate all the input sharings independently. It is then enough to compute the simulatability set on this composed circuit and test whether it is empty (if not, it contains a sensitive input variable).

The differences between the tools are therefore the algorithm to compute the simulatability sets, and the way the space of probe sets is explored. Let us now detail these for maskVerif.

The simulatability set computation of maskVerif is based on the representation of the probes as arithmetic expressions in the inputs and the randomness of the circuit. Given a tuple of such expressions, we know that the simulatability set of the corresponding probe set is a subset of all the inputs that appear in the expressions. The idea behind maskVerif is to transform these expressions to reduce the set of such inputs, while preserving the statistical distributions (conditioned on the inputs) of the values represented by the expressions. The basic transformation of maskVerif operates from a randomness variable r (the output of a random gate), and an expression e in which r does not appear. Then, since the value of r is a fresh uniform random, the distributions of the values associated to the expressions r and $r \oplus e$[4] are identical, and the substitution of every occurrence of r with $r \oplus e$ in the expressions does not

[4] maskVerif currently works only in characteristic-two fields $\mathbb{F}_{2^k}$, but the same principles could be applied to any field.

change the distribution of the corresponding probes. This substitution may remove some inputs from the probe expressions, which proves that they do not belong to the simulatability set.

maskVerif uses heuristics to choose which substitutions and which expression simplifications (based on arithmetic laws) to apply, and in which order. These heuristics must guarantee termination while being efficient and eliminating as many lexical input dependencies as possible. Due to this usage of heuristics and to the limitations of lexical analysis of expression (instead of full distribution computation), the maskVerif simulatability set algorithm lacks precision: it may return a superset of the simulatability set. However, when all the expressions are linear in the randomness (e.g., for refresh gadgets), maskVerif is precise: it computes exactly the simulatability set.

Regarding the exploration of probe sets, maskVerif uses an optimization based on the following observation: in a gadget G, let I be the dependency set of the set of probes P, then for any $P' \subseteq P$, the dependency set of P' is a subset of I. Let us take the example of checking that G is t-NI: if the dependency set of P contains at most t shares of each input sharing for some P such that $|P| > t$, then maskVerif can skip the computation of the dependency set of the $\binom{|P|}{t}$ subsets of size t of P. If $|P|$ is sufficiently large, this greatly reduces the number of sets to check [Bar+15].

Let us finally compare maskVerif to two recently introduced tools. First, IronMask is very similar to maskVerif, but its algorithm for the selection of expression substitution offers better precision. Indeed, for some gadgets[5], it always finds a sequence of substitution that leads to finding the exact (i.e., minimal) simulatability set, if it exists. On the other hand, SILVER can only analyze gadgets in $\mathbb{F}_2$, but it is always fully precise, at the expense a higher computational cost (it is therefore limited to smaller gadgets than maskVerif). Finally, SILVER can check all of probing security, NI, SNI and PINI, while maskVerif and IronMask do not implement PINI verification (but that could be done without significant algorithmic changes).

[5]The quadratic gadgets in which each random element either appears as a linear term in the expressions of the output shares, or appears only in linear expressions whose variables are the shares of one input sharing and random elements.

3.6 Case study: conversion between arithmetic and Boolean masking

Boolean masking is very convenient thanks to the universality of Boolean circuits: any computation can be implemented with Boolean masking. Moreover, Boolean masking is quite efficient when the required computations have simple Boolean representations (e.g., symmetric cryptography primitives). However, implementing any computation with Boolean masking can be very inefficient compared to alternatives, such as arithmetic masking. For example, a linear computation with variables in $\mathbb{Z}_p$ can be implemented with sharewise gadgets when using arithmetic masking modulo p (hence $\mathcal{O}(d)$ overhead), while it would require AND gadgets (with $\mathcal{O}\left(d^2\right)$ overhead) when Boolean masking is used. Therefore, if a mix of linear operations modulo p and other less structured operations (e.g., symmetric cryptography primitives) have to be applied, there is a clear need for a gadget that converts an arithmetic sharing into a tuple of Boolean masked bits (and vice-versa). Lattice-based schemes such as Kyber [Ava+19] and Saber [Bas+19] are good examples of computations that mix linear operations modulo p and Boolean operations.

In this section, we rely on the trivial composition of PINI gadgets to build new composable conversion gadgets that improve performance over the state of the art when implemented in software by exploiting the bitslicing technique. In short, bitslicing leverages the intrinsic parallelism of bitwise operations within processors. E.g., a processor that manipulates 32-bit integers can perform 32 bitwise operations with a single instruction. Therefore, bitslicing only applies to algorithms whose operations are bitwise, such as [Gro+14], but sometimes an algorithm can be re-written to use bit-level operations (while preserving efficiency) [BMP13]. In particular, Boolean masking is very well suited to bitslicing since most Boolean masking gadgets only use bit-level operations, whereas arithmetic masking gadgets use additions and multiplications (whose equivalent bitwise circuits are large) and therefore do not benefit from bitslicing.

Notations To make the distinction between the arithmetic and Boolean sharings clear, we use the following notations in this section. An arithmetic sharing of a variable $x \in [\![0, p[\![$ for some integer p is denoted

$$\mathbf{x}^{A_p} = \left(\mathbf{x}^i_{A_p}\right)_{i=0,\dots,d-1} \in [\![0, p[\![^d \text{ where } x = \mathbf{x}^0_{A_p} + \cdots + \mathbf{x}^{d-1}_{A_p} \mod p\,,$$

and for a k-bit Boolean sharing of $x \in \left[\!\left[0, 2^k\right[\!\right[$:

$$\mathbf{x}^{B,k} = \left(\mathbf{x}^i_{B,k}\right)_{i=0,\dots,d-1} \in \left[\!\left[0, 2^k\right[\!\right[^d \text{ where } x = \mathbf{x}^0_{B,k} \oplus \ldots \oplus \mathbf{x}^{d-1}_{B,k}\,.$$

Moreover, $\mathbf{x}^{B,k}[i]$ is single-bit sharing obtained by taking the ith bit of each share of $\mathbf{x}^{B,k}$ (with $i = 0$ being the least significant bit).

The sharewise gadget that implements the addition modulo p (resp. bitwise XOR) of two shared variables is denoted as $+^{\mathtt{A}}$ (resp. $\oplus^{\mathtt{B}}$). Furthermore, we denote by `SecAnd` any gadget that computes bitwise AND of Boolean-shared values (it can be instantiated by the ISW multiplication, or by the PINI_1 gadget when the PINI property is needed).

3.6.1 Addition in Boolean masking

Beyond its intrinsic usefulness (e.g., to implement ARX ciphers [JPS18]), modular addition of Boolean-masked variables is at the core of many conversion gadgets.

We first consider the addition modulo 2^k of two k-bit Boolean sharings, and denote this gadget as `SecAdd`. It can be implemented by taking the Boolean circuit of a k-bit binary adder, rewriting it to only use AND and XOR gates, and finally implementing this circuit with 1-bit `SecAnd` and $\oplus^{\mathtt{B}}$ gadgets. The 1-bit inputs of this circuit are obtained by selecting single bit sharings in the k-bit input sharings. Using a chain of full-adders (i.e., the ripple-carry architecture), this technique yields a complexity of $\mathcal{O}\left(kd^2\right)$ operations (each on single-bit words). Our implementation in Algorithm 9 is bitslice (i.e., one execution of it on a 32-bit processor computes 32 independent additions) and minimizes the number of `SecAnd` gadgets. The full-adders (Algorithm 8) compute the addition of the bits x, y and c as $a := x \oplus y$ and $(a \oplus c, x \oplus a \cdot (x \oplus c))$. This requires only one `SecAnd` per full-adder, hence $k - 1$ in total (since the carry-out does not have to be computed for the addition of the most significant bits).

Proposition 13. *Algorithm 9 and Algorithm 8 are PINI.*

Proof. These two gadgets are the composition of PINI gadgets, therefore they are PINI. □

Algorithm 8 $\mathtt{SecFullAdder}^d$ New (PINI).

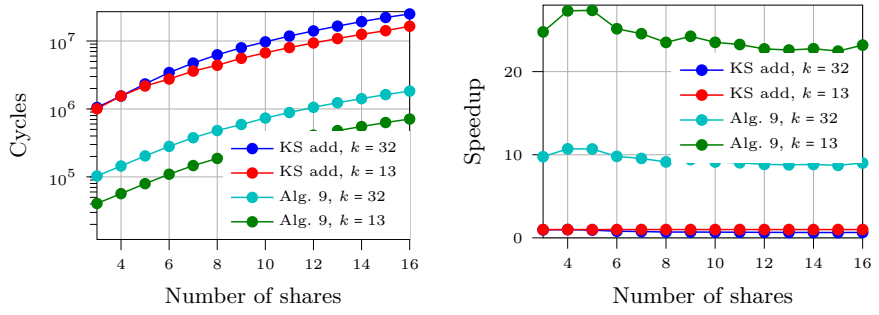
Input: Boolean sharings $\mathbf{x}^{B,1}$, $\mathbf{y}^{B,1}$ and $\mathbf{z}^{B,1}$.
Output: Boolean sharing $\mathbf{w}^{B,2}$ such that $w = x + y + z$.

1: $\mathbf{a}^{B,1} \leftarrow \mathbf{x}^{B,1} \oplus^{\mathtt{B}} \mathbf{y}^{B,1}$
2: $\mathbf{w}^{B,2}[0] \leftarrow \mathbf{z}^{B,1} \oplus^{\mathtt{B}} \mathbf{a}^{B,1}$
3: $\mathbf{w}^{B,2}[1] \leftarrow \mathbf{x}^{B,1} \oplus^{\mathtt{B}} \mathtt{SecAnd}_1^d\left(\mathbf{a}^{B,1}, \mathbf{x}^{B,1} \oplus^{\mathtt{B}} \mathbf{z}^{B,1}\right)$ ▷ PINI `SecAnd`

Algorithm 9 $\texttt{SecAdd}_k^d$ New (PINI).

Input: Boolean sharings $\mathbf{x}^{B,k}$ and $\mathbf{y}^{B,k}$, such that $x, y \in [\![0, 2^k[\![$.
Output: Boolean sharing $\mathbf{z}^{B,k}$ such that $z = x + y \mod 2^k$.

1: $\mathbf{c}^{B,1} \leftarrow (0, 0, \ldots, 0)$
2: **for** $i = 0$ to $k - 2$ **do**
3: $\quad \mathbf{t}^{B,2} \leftarrow \texttt{SecFullAdder}^d\left(\mathbf{x}^{B,k}[i], \mathbf{y}^{B,k}[i], \mathbf{c}^{B,1}\right)$ ▷ Algorithm 8
4: $\quad \left(\mathbf{z}^{B,k}[i], \mathbf{c}^{B,1}\right) \leftarrow \left(\mathbf{t}^{B,2}[0], \mathbf{t}^{B,2}[1]\right)$
5: $\mathbf{z}^{B,k}[k-1] \leftarrow \mathbf{x}^{B,k}[k-1] \oplus^{\mathtt{B}} \mathbf{y}^{B,k}[k-1] \oplus^{\mathtt{B}} \mathbf{c}^{B,1}$



(a) Cycle count. **(b) Speedup w.r.t. KS adder, k = 13.**

Figure 3.4: Performance comparison of `SecAdd` implementations.

The performance on a 32-bit microcontroller[6] of new addition gadget is compared to the state of the art in Figure 3.4. We compare to the addition gadget introduced in [Cor+15], which is not fully bitslice, but uses an optimized adder architecture (the Kogge-Stone (KS) adder), and allows performing some Boolean operations in parallel, that is, with multiple-bit $\texttt{SecAnd}$ and $\oplus^{\texttt{B}}$ gadgets. Figure 3.4 shows that Algorithm 9 provides a significant speedup over the state of the art.[7] This speedup varies with the number of bits in the operands[8], since our gadget has a linear complexity in k while the one of [Cor+15] has a logarithmic complexity in k.[9]

3.6.2 Modular addition in Boolean masking

Algorithm 10 $\texttt{SecAddModp}_k^d$ from [Bar+18] (NI).

Input: Boolean sharings $\mathbf{x}^{B,k}$ and $\mathbf{y}^{B,k}$, integer p such that $p < 2^k$ and $x, y \in [\![0, p[\![$.
Output: Boolean sharing $\mathbf{z}^{B,k}$ such that $z = x + y \mod p$.

1: $\mathbf{p}^{B,k+1} \leftarrow \left(2^k - p, 0, \ldots, 0\right)$
2: $\mathbf{s}^{B,k+1} \leftarrow \texttt{SecAdd}_{k+1}^d\left(\mathbf{x}^{B,k}, \mathbf{y}^{B,k}\right)$ ▷ Left 0-extend the input sharings by 1 bit.
3: $\mathbf{s'}^{B,k+1} \leftarrow \texttt{SecAdd}_{k+1}^d\left(\mathbf{s}^{B,k+1}, \mathbf{p}^{B,k+1}\right)$
4: $\mathbf{b}^{B,1} \leftarrow \mathbf{s'}^{B,k+1}[k]$
5: $\mathbf{c}^{B,1} \leftarrow \texttt{RefreshSNI}_1^d\left(\mathbf{b}^{B,1}\right)$
6: $\mathbf{c'}^{B,1} \leftarrow \neg\texttt{RefreshSNI}_1^d\left(\mathbf{b}^{B,1}\right)$
7: $\mathbf{c}^{B,k} \leftarrow \texttt{BitCopyMask}_k^d\left(\mathbf{c}^{B,1}, 2^k - 1\right)$ ▷ Copy input sharing where bitmask $(2^k - 1)$ is set.
8: $\mathbf{c'}^{B,k} \leftarrow \texttt{BitCopyMask}_k^d\left(\mathbf{c'}^{B,1}, 2^k - 1\right)$
9: $\mathbf{z}^{B,k} \leftarrow \texttt{SecAnd}_k^d\left(\mathbf{s}^{B,k+1}[[\![0, k[\![], \mathbf{c}^{B,k}\right) \oplus^{\texttt{B}} \texttt{SecAnd}_k^d\left(\mathbf{s'}^{B,k+1}[[\![0, k[\![], \mathbf{c'}^{B,1}\right)$ ▷ MUX

Next, we consider the $\texttt{SecAddModp}$ gadget which performs the addition modulo p. The construction of Algorithm 10 (from [Bar+18]) is based

[6] The measurements are taken on an ARM Cortex-M4 (from the NUCLEO-L4R5ZI development board). The randomness used in the gadgets is generated by an on-chip true random number generator (TRNG) that outputs a fresh 32-bit word every 53 cycles.

[7] These numbers assume that bitslicing is fully exploited, that is, it measures the time needed for 32 k-bit additions.

[8] We analyze the cases $k = 32$ and $k = 13$ as one the full width of the processor, and the other corresponds to the size of the number manipulated in the Kyber KEM [Bos+18].

[9] The logarithmic complexity is valid only when k is lower than the bus width of the processor, i.e., 32 in our case.

on the SecAdd gadget. Namely, it first computes the sum s of the inputs x and y on $k+1$ (to avoid overflow and thus modulo 2^k reduction), then adds $2^k - p$ to obtain s'. The most significant bit of s' indicates whether $x + y \geq p$. Based on this bit, either s or s' is selected as the output, using a MUX implemented with SecAnd and $\oplus^{\mathsf{B}}$ gadgets. Finally, the most significant bit is dropped to get the result on k bits.

We improve this construction by remarking that, on top of using our optimized addition (Algorithm 9), the MUX in Algorithm 10 costs $2k$ 1-bit SecAnd gadgets, and that we can replace it with the computation of $s' + p \cdot b \mod 2^k$, which costs one $\mathtt{SecAdd}_k^d$ (i.e., $k-1$ single-bit SecAnd). This replacement is correct: if $b = 0$, the result is s', and if $b = 1$ the result is $s' + p \mod 2^k = s$. Overall, our new addition modulo p requires two $k+1$-bit adders and one k-bit adder, totaling to $3k - 1$ 1-bit PINI SecAnd, hence $\mathcal{O}\left(kd^2\right)$ bit operations and randomness.

Proposition 14. *Algorithm 11 is PINI.*

Proof. All the sub-gadgets are PINI (BitCopyMask only replicates a sharing, hence it is share-isolating, which implies that it is PINI). □

Algorithm 11 $\mathtt{SecAddModp}_k^d$ New (PINI).

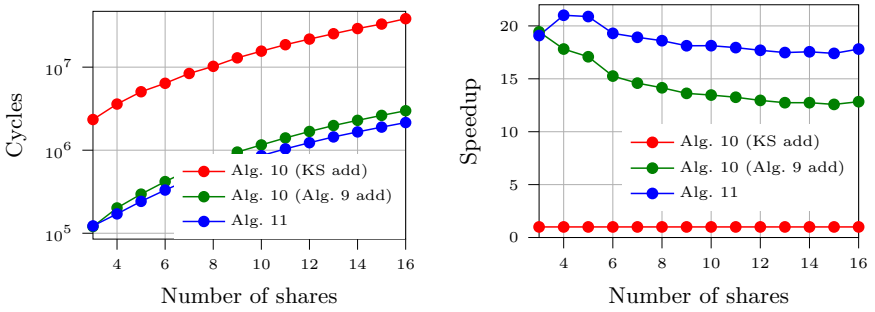
Input: Boolean sharings $\mathbf{x}^{B,k}$ and $\mathbf{y}^{B,k}$, integer p such that $p < 2^k$ and $x, y \in [\![0, p[\![$.
Output: Boolean sharing $\mathbf{z}^{B,k}$ such that $z = x + y \mod p$.

1: $\mathbf{p}^{B,k+1} \leftarrow \left(2^{k+1} - p, 0, \ldots, 0\right)$
2: $\mathbf{s}^{B,k+1} \leftarrow \mathtt{SecAdd}_{k+1}^d\left(\mathbf{x}^{B,k}, \mathbf{y}^{B,k}\right)$ ▷ Use Algorithm 9.
3: $\mathbf{s'}^{B,k+1} \leftarrow \mathtt{SecAdd}_{k+1}^d\left(\mathbf{s}^{B,k+1}, \mathbf{p}^{B,k+1}\right)$ ▷ Use Algorithm 9.
4: $\mathbf{b}^{B,1} \leftarrow \mathbf{s'}^{B,k+1}[k]$
5: $\mathbf{a}^{B,k} \leftarrow \mathtt{BitCopyMask}_k^d\left(\mathbf{b}^{B,1}, p\right)$ ▷ Copy sharing b where bitmask p is set (computes $a = p \cdot b$).
6: $\mathbf{z}^{B,k} \leftarrow \mathtt{SecAdd}_k^d\left(\mathbf{a}^{B,k}, \mathbf{s'}^{B,k+1}\right)$ ▷ Use Algorithm 9.

In Figure 3.5, the new $\mathtt{SecAddModp}_d^k$ gadget Algorithm 11 (using Algorithm 9 as base adder) is compared to Algorithm 10, instantiated with the KS adder or our optimized adder (Algorithm 9). Overall, the new gadgets bring a performance improvement of up to 21x over the state of the art, and the MUX removal optimization of Algorithm 11 alone brings a speedup of about 1.3x.

3.6.3 Arithmetic to Boolean masking conversion

Coron et al. [CGV14] introduced a simple way to convert from arithmetic to Boolean masking (SecA2BModp): mask with Boolean masking each



(a) Cycle count. **(b) Speedup w.r.t. Alg. 10 (KS adder).**

Figure 3.5: Performance comparison of $\texttt{SecAddModp}_{12}^{d}$ implementations.

arithmetic share, then perform the addition modulo p of the resulting sharings. Remarking that the addition of d' arithmetic shares can be securely masked using d'-shares Boolean masking instead of d, and organizing the $d-1$ additions to perform in a binary tree (halving the number of shares at each layer) leads to Algorithm 12 (from [Sch+19]), which has a complexity of $\mathcal{O}\left(\log(k)d^2\right)$ operations on up-to k-bit words. As an alternative, a table-based `SecA2BModp` implementation with the same complexity was recently introduced in [Cor+22].

Algorithm 12 $\texttt{SecA2BModp}_k^d$ from [Sch+19] (SNI)

Input: d shares arithmetic sharing $\mathbf{x}^{A_p}$, integer p such that $p < 2^k$ and $x \in [\![0, p[\![$.
Output: d shares Boolean sharing $\mathbf{z}^{B,k}$ such that $z = x$.

1: **if** $d = 1$ **then**
2: $\quad \mathbf{z}^{B,k} \leftarrow \mathbf{x}^{A_p}$
3: **else**
4: $\quad \mathbf{y}^{B,k} \leftarrow \texttt{SecA2BModp}_k^{\lfloor d/2 \rfloor}\left(\mathbf{x}_{A_k}^{[\![0,\lfloor d/2 \rfloor[\![}\right)$
5: $\quad \mathbf{y'}^{B,k} \leftarrow \texttt{SecA2BModp}_k^{d-\lfloor d/2 \rfloor}\left(\mathbf{x}_{A_k}^{[\![\lfloor d/2 \rfloor,d[\![}\right)$
6: $\quad \mathbf{y}^{B,k} \leftarrow \texttt{RefreshSNI}_k^d\left(\left(\mathbf{y}_{B,k}^0, \mathbf{y}_{B,k}^1, \ldots, \mathbf{y}_{B,k}^{\lfloor d/2 \rfloor - 1}, 0, \ldots, 0\right)\right)$ ▷ Expand to d shares.
7: $\quad \mathbf{y'}^{B,k} \leftarrow \texttt{RefreshSNI}_k^d\left(\left(0, \ldots, 0, \mathbf{y}_{B,k}^{\lfloor d/2 \rfloor}, \ldots, \mathbf{y}_{B,k}^{d-1}\right)\right)$ ▷ Expand to d shares.
8: $\quad \mathbf{z}^{B,k} \leftarrow \texttt{SecAddModp}_k^d\left(\mathbf{y}^{B,k}, \mathbf{y'}^{B,k}\right)$

Using our new `SecAdd` (Algorithm 9) in Algorithm 12 allows removing the refresh gadgets thanks to PINI composition, giving Algorithm 13 whose complexity is $\mathcal{O}\left(kd^2\right)$ random bits and single-bit operations. Due to the use of `SecAdd` for the addition, this gadget only works when the arithmetic sharing modulus is 2^k.

To prove that Algorithm 13 is PINI, we will use the PINI composition

Algorithm 13 $\texttt{SecA2B}_k^d$ New (PINI)

Input: d shares arithmetic sharing $\mathbf{x}^{A_{2^k}}$, such that $x \in [\![0, 2^k[\![$.
Output: d shares Boolean sharing $\mathbf{z}^{B,k}$ such that $z = x$.

1: **if** $d = 1$ **then**
2: $\quad \mathbf{z}^{B,k} \leftarrow \mathbf{x}^{A_{2^k}}$
3: **else**
4: $\quad \mathbf{y}^{B,k} \leftarrow \texttt{SecA2B}_k^{\lfloor d/2 \rfloor}\left(\mathbf{x}^{A_{2^k}}[\![0, \lfloor d/2 \rfloor[\![\right)$ ▷ $\lfloor d/2 \rfloor$ sharing.
5: $\quad \mathbf{y}'^{B,k} \leftarrow \texttt{SecA2B}_k^{d-\lfloor d/2 \rfloor}\left(\mathbf{x}^{A_{2^k}}[\![\lfloor d/2 \rfloor, d[\![\right)$ ▷ $d - \lfloor d/2 \rfloor$ sharing.
6: $\quad \mathbf{s}^{B,k} \leftarrow \left(\mathbf{y}_{B,k}^0, \mathbf{y}_{B,k}^1, \ldots, \mathbf{y}_{B,k}^{\lfloor d/2 \rfloor - 1}, 0, \ldots, 0\right)$ ▷ Expand to d shares.
7: $\quad \mathbf{s}'^{B,k} \leftarrow \left(0, \ldots, 0, \mathbf{y}'^{\lfloor d/2 \rfloor}_{B,k}, \ldots, \mathbf{y}'^{d-1}_{B,k},\right)$ ▷ Expand to d shares.
8: $\quad \mathbf{z}^{B,k} \leftarrow \texttt{SecAdd}_k^d\left(\mathbf{s}^{B,k}, \mathbf{s}'^{B,k}\right)$ ▷ Use Algorithm 9.

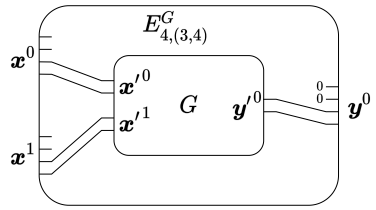


Figure 3.6: Example of 2-share to 4-share gadget embedding.

theorem, and introduce a new technique to deal with the composition of PINI gadget with various number of shares. The core idea is to embed gadgets that use a lower number of shares into "virtual gadgets" that use more shares, with a mapping from the share indexes of the embedded gadgets to the indexes of the embedding gadgets. The embedding gadget discards the input shares that are not used, and sets to 0 the output shares that are not generated by the embedded gadgets, as illustrated in Figure 3.6.

Definition 16 (Gadget embedding)**.** *Let G be a d'-share gadget with n (resp. n') input (resp. output) sharings, and let $m \in [\![0, d[\![^{d'}$ (with $d \geq d'$) have unique components ($m_i \neq m_j$ for all i, j). The d-share embedding of G with mapping m is the d-share gadget denoted $E^G_{d,m}$ described in Algorithm 14.*

Lemma 4 (PINI embedding)**.** *If G is a PINI gadget, its embedding $E^G_{d,m}$ is PINI for any d and m.*

Proof. We describe the $(d-1)$-PINI simulator for $E^G_{d,m}$ that has to simulate a set of internal probes P and the output shares with index in

Algorithm 14 $E^G_{d,m}$: embedding of the d'-shares gadget G to d shares with mapping m.

Input: d shares input sharings $\mathbf{x}0, \ldots, \mathbf{x}n-1$
Input: d shares output sharings $\mathbf{y}0, \ldots, \mathbf{y}n'-1$

1: **for** $j = 0, \ldots, n-1$ **do**
2: **for** $i = 0, \ldots, d'-1$ **do**
3: $\mathbf{x}'_j i \leftarrow \mathbf{x}_j m_i$
4: $\left(\mathbf{y}'0, \ldots, \mathbf{y}'n'\right) \leftarrow G\left(\mathbf{x}'0, \ldots, \mathbf{x}'n\right)$
5: **for** $j = 0, \ldots, n'-1$ **do**
6: **for** $i = 0, \ldots, d-1$ **do**
7: $\mathbf{y}_j i \leftarrow 0$ ▷ Initialize all shares to 0.
8: **for** $i = 0, \ldots, d'-1$ **do**
9: $\mathbf{y}_j m_i \leftarrow \mathbf{y}'_j i$ ▷ Override some output shares with outputs of G.

B. First, P can be partitioned in a set P_G of probes in G and a set P_i of probes on the input shares. Next, B is partitioned as B_0 (the elements of B that appear in m), and B_1 (the remaining elements).

Let $B'_0 = \left\{i \in [\![0, d']\![\text{ s.t. } m_i \in B_0\right\}$, we have $\left|B'_0\right| = |B_0|$. We use the PINI simulator of G to simulate the probes P_G and its output shares with index in B'_0 (which are the outputs of $E^G_{d,m}$ with index in B_0). This simulator requires knowledge of its input shares with index in $A' \cup B'$, for some A'_0 such that $\left|A'_0\right| \leq |P_G|$. Let us define $A_0 = \left\{m_i \text{ for all } i \in A'_0\right\}$, such that knowing the input shares of $E^G_{d,m}$ with index in $A_0 \cup B_0$ allows sending the inputs required to the simulator of G, hence to simulate the probes P_G and the output shares with index in B_0.

Finally, the probes in P_i can be simulated with the input shares with index in A_1, for some A_1 such that $|A_1| \leq |P_i|$, and all the output shares with index in B_1 can be trivially simulated (their value is always 0). As a result, all the required values can be simulated with the input shares of $E^G_{d,m}$ with index in $(A_0 \cup A_1) \cup B$, and $|A_0 \cup A_1| \leq |P|$. □

Proposition 15. *Algorithm 13 is PINI.*

Proof. In the case $d = 1$, this is trivial. In the other cases, we decompose the gadget in three sub-gadgets: $E^{\texttt{SecA2B}_k^{\lfloor d/2 \rfloor}}_{d,(0,\ldots,\lfloor d/2 \rfloor - 1)}$ (which computes $\mathbf{s}^{B,k}$ from $\mathbf{x}^{A_{2^k}}$), $E^{\texttt{SecA2B}_k^{d-\lfloor d/2 \rfloor}}_{d,(\lfloor d/2 \rfloor,\ldots,d-1)}$ (which computes $\mathbf{s'}^{B,k}$ from $\mathbf{x}^{A_{2^k}}$) and $\texttt{SecAdd}^d_k$ (which computes $\mathbf{z}^{B,k}$ from $\mathbf{s}^{B,k}$ and $\mathbf{s'}^{B,k}$). Since $\texttt{SecA2B}_k^{\lfloor d/2 \rfloor}$ and $\texttt{SecA2B}_k^{d-\lfloor d/2 \rfloor}$ are PINI (by induction on d), their embeddings are PINI (by Lemma 4). Furthermore, $\texttt{SecAdd}^d_k$ is PINI (Proposition 13).

Therefore, Algorithm 13 is a composition of PINI gadgets. □

A simple way to implement arithmetic modulo p to Boolean masking conversion is to adapt Algorithm 13 (`SecA2B`) to use addition modulo p (`SecAddModp`, Algorithm 11) instead of addition modulo 2^k (`SecAdd`, Algorithm 9). On top of this adaptation, we can perform a small optimization inspired by the first-order A2B conversion from [Fri+22]: the first operation of our addition modulo p (Algorithm 11) is to subtract p from one of the two operands which can be done before double the number of shares in the A2B algorithm. This has no impact on the final result, but the cost of this subtraction is divided by about 4 (since this operation is in $\mathcal{O}\left(kd^2\right)$).

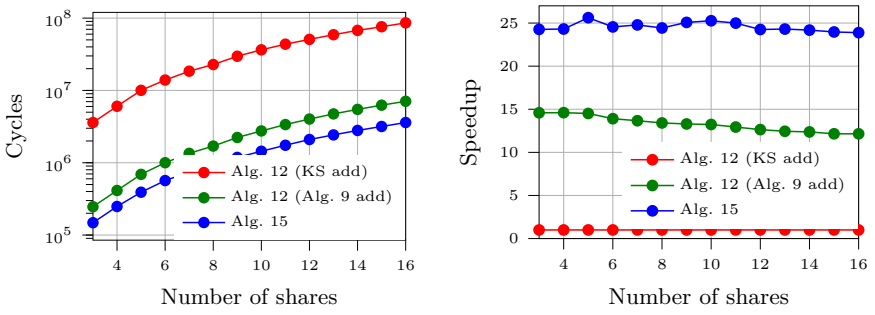
Algorithm 15 $\texttt{SecA2BModp}_k^d$ New (PINI)

Input: d shares arithmetic sharing $\mathbf{x}^{A_p}$, integer p such that $p < 2^k$ and $x \in [\![0, p[\![$.
Output: d shares Boolean sharing $\mathbf{z}^{B,k}$ such that $z = x$.

1: **if** $d = 1$ **then**
2: $\quad \mathbf{z}^{B,k} \leftarrow \mathbf{x}^{A_p}$
3: **else**
4: $\quad \mathbf{y}^{B,k} \leftarrow \texttt{SecA2BModp}_k^{\lfloor d/2 \rfloor}\left(\mathbf{x}^{A_p}[[\![0, \lfloor d/2 \rfloor[\![]\right)$ ▷ $\lfloor d/2 \rfloor$ sharing.
5: $\quad \mathbf{y}'^{B,k} \leftarrow \texttt{SecA2BModp}_k^{d-\lfloor d/2 \rfloor}\left(\mathbf{x}^{A_p}[[\![\lfloor d/2 \rfloor, d[\![]\right)$ ▷ $d - \lfloor d/2 \rfloor$ sharing.
6: $\quad \mathbf{p}^{B,k+1} \leftarrow \left(2^k - p, 0, \ldots, 0\right)$ ▷ $\lfloor d/2 \rfloor$ sharing.
7: $\quad \mathbf{s}^{B,k+1} \leftarrow \texttt{SecAdd}_{k+1}^{\lfloor d/2 \rfloor}\left(\mathbf{p}^{B,k+1}, \mathbf{y}^{B,k}\right)$ ▷ Use Algorithm 9.
8: $\quad \mathbf{s}^{B,k+1} \leftarrow \left(\mathbf{y}_{B,k+1}^0, \mathbf{y}_{B,k+1}^1, \ldots, \mathbf{y}_{B,k+1}^{\lfloor d/2 \rfloor - 1}, 0, \ldots, 0\right)$ ▷ Expand to d shares.
9: $\quad \mathbf{s}'^{B,k} \leftarrow \left(0, \ldots, 0, \mathbf{y}'^{\lfloor d/2 \rfloor}_{B,k}, \ldots, \mathbf{y}'^{d-1}_{B,k},\right)$ ▷ Expand to d shares.
10: $\quad \mathbf{u}^{B,k+1} \leftarrow \texttt{SecAdd}_{k+1}^{d}\left(\mathbf{s}^{B,k+1}, \mathbf{s}'^{B,k}\right)$ ▷ Use Algorithm 9.
11: $\quad \mathbf{b}^{B,1} \leftarrow \mathbf{u}^{B,k+1}[k]$
12: $\quad \mathbf{a}^{B,k} \leftarrow \texttt{BitCopyMask}_k^d\left(\mathbf{b}^{B,1}, p\right)$ ▷ Copy sharing b where bitmask p is set ($a := p \cdot b$).
13: $\quad \mathbf{z}^{B,k} \leftarrow \texttt{SecAdd}_k^d\left(\mathbf{a}^{B,k}, \mathbf{u}^{B,k+1}\right)$ ▷ Use Algorithm 9.

Proposition 16. *Algorithm 15 is PINI.*

Proof. The proof is almost identical to the proof of Algorithm 15. The case $d = 1$ is trivial, and in the other cases, we exhibit a decomposition into PINI sub-gadgets. We first consider the d-share embedding of the $\lfloor d/2 \rfloor$-share composite gadget whose input is $\mathbf{x}^{A_p}[[\![0, \lfloor d/2 \rfloor[\![]$ and whose output is $\mathbf{s}^{B,k+1}$. This gadget is the composition of two PINI gadgets ($\texttt{SecA2BModp}_k^{\lfloor d/2 \rfloor}$ and $\texttt{SecAdd}_{k+1}^{\lfloor d/2 \rfloor}$), hence it is PINI, and the embedding is PINI. Next, the d-share embedding of $\texttt{SecA2BModp}_k^{d-\lfloor d/2 \rfloor}$ is PINI, as well as the other d-share sub-gadgets (`SecAdd`, `BitCopyMask`). □



(a) Cycle count. **(b) Speedup w.r.t. Alg. 12 (KS adder).**

Figure 3.7: Performance comparison of $\texttt{SecA2BModp}^{d}_{12}$ implementations.

As previously, we compare the performance of $\texttt{SecA2BModp}^{d}_{k}$ implementations[10]. In Figure 3.7, we can see that the use of the new adder continues to provide a large performance gain, and the removal of the refresh gadgets and optimization of number of shares for the subtraction of p in Algorithm 13 brings almost a 2x additional speedup.

3.6.4 Boolean to arithmetic masking conversion

Similarly to arithmetic-to-Boolean, there are multiple efficient techniques for Boolean-to-arithmetic conversion.

First, one may generate $d-1$ random arithmetic shares, generate a d-share Boolean masking of the opposite of their sum (using `SecA2BModp`), add this to the input sharing (with `SecAddModp`), and finally unmask (that is, XOR the shares together) the result to get the last arithmetic share. This idea, originally introduced in [Cor+15], has been adapted to the modulo p setting in [Bar+18] (see Algorithm 16). This gadget is $(d-1)$-SNI.[11]

Second, Schneider et al. [Sch+19] introduced a conversion based on the observation that if $x, y \in \{0,1\}$, $x \oplus y = x + y - 2xy$. The gist of the conversion algorithm is to start from a 1-bit Boolean sharing $\mathbf{x}^{B,1}$, then arithmetically mask each share, and finally use the previous equation to compute the XOR of these arithmetic sharings. This single-bit conversion algorithm may then be applied to each of a multi-bit input, and the results can be recombined sharewise (with sums and multiplications by 2). Thanks to various optimizations of the algorithm [Sch+19], the

[10] We skip the discussion of the performance of $\texttt{SecA2B}^{d}_{k}$ since it follows the same trend.

[11] The proof that `SecB2AModp` is SNI is not given explicitly, in [Bar+18], but it can be deduced from the proof of Lemma 5, if `SecA2BModp` is SNI.

Algorithm 16 $\texttt{SecB2AModp}_k^d$ from [Bar+18] (SNI)

Input: d shares Boolean sharing $\mathbf{x}^{B,k}$, integer p such that $p < 2^k$ and $x \in [\![0, p[\![$.
Output: d shares arithmetic sharing $\mathbf{z}^{A_p}$ such that $z = x$.

1: **for** $i = 0$ to $d - 2$ **do**
2: $\quad \mathbf{z}^i_{A_k} \xleftarrow{\$} \mathbb{Z}_p$
3: $\quad \mathbf{z'}^i_{A_k} \leftarrow p - \mathbf{z}^i_{A_k}$
4: $\mathbf{z'}^{d-1}_{A_k} \leftarrow 0$
5: $\mathbf{a}^{B,k} \leftarrow \texttt{SecA2BModp}_k^d\left(\mathbf{z'}^{A_p}\right)$
6: $\mathbf{b}^{B,k} \leftarrow \texttt{SecAddModp}_k^d\left(\mathbf{a}^{B,k}, \mathbf{x}^{B,k}\right)$
7: $\mathbf{z}^{d-1}_{A_k} \leftarrow \texttt{UnMask}_k^{d-1}\left(\texttt{FullRefresh}_k^{d-1}\left(\mathbf{b}^{B,k}\right)\right)$

complexity of this technique is $\mathcal{O}\left(kd^2\right)$ operations on k-bit words.

Finally, Coron et al. [Cor+22] introduced recently another conversion algorithm. This algorithm also performs k single-bit conversions, but the single-bit conversion is a table-based gadget.

Algorithm 17 adapts the `SecB2AModp` from [Bar+18] (Algorithm 16) to use our new `SecA2BModp` and `SecAddModp` algorithms.[12] Furthermore, we replace the refresh gadget to reduce its cost (from $\mathcal{O}\left(d^2\right)$ to $\mathcal{O}\left(d \log d\right)$), by using the input-output separative (IOS) refresh gadget `RefreshIOS`introduced in [Gou+21] (or its generalization to any number of shares not only power of two in [BC22]).

Definition 17 (Uniformity ([Gou+21], adapted))**.** *A refresh gadget G is uniform if its output is a uniformly distributed sharing of x for any fixed input sharing $\mathbf{x}$.*[13]

Definition 18 (IOS ([Gou+21], adapted))**.** *A refresh gadget G is t-IOS if it is uniform and if for every pair of sharings $(\mathbf{x}, \mathbf{y})$ that represent the same value (i.e., such that $x = y$) and for every set of probes P with $|P| \leq t$, there exists a simulator which can perfectly simulate the probes by knowing only $|P|$ input shares and $|P|$ output shares. A refresh gadget with d shares is said to be IOS if it is $(d-1)$-IOS.*

Proposition 17. *Algorithm 17 is PINI.*

[12] The conversion modulo 2^k $\texttt{SecB2A}_k^d$ can be implemented following Algorithm 17, using the new `SecA2B` and `SecAdd` instead of `SecA2BModp` and `SecAddModp`. The security proof is not changed.

[13] This is not the same notion as the uniformity used in threshold implementations [NRR06], where the sharing $\mathbf{x}$ is assumed to be uniform. Here, the distribution of the output sharing $\mathbf{y}$ must be independent of $\mathbf{x}$, conditioned on x.

Algorithm 17 $\texttt{SecB2AModp}_k^d$ New (PINI)

Input: d shares Boolean sharing $\mathbf{x}^{B,k}$, integer p such that $p < 2^k$ and $x \in [\![0, p[\![$.
Output: d shares arithmetic sharing $\mathbf{z}^{A_p}$ such that $z = x$.

1: **for** $i = 0$ to $d-2$ **do**
2: $\quad \mathbf{z}^i_{A_p} \xleftarrow{\$} \mathbb{Z}_p$
3: $\quad \mathbf{z'}^i_{A_p} \leftarrow p - \mathbf{z}^i_{A_p}$
4: $\mathbf{z'}^{d-1}_{A_p} \leftarrow 0$
5: $\mathbf{a}^{B,k} \leftarrow \texttt{SecA2BModp}_k^d\left(\mathbf{z'}^{A_p}\right)$ ▷ Use Algorithm 15.
6: $\mathbf{b}^{B,k} \leftarrow \texttt{SecAddModp}_k^d\left(\mathbf{a}^{B,k}, \mathbf{x}^{B,k}\right)$ ▷ Use Algorithm 11.
7: $\mathbf{c}^{B,k} \leftarrow \texttt{RefreshIOS}_k^d\left(\mathbf{b}^{B,k}\right)$
8: $\mathbf{z}^{d-1}_{A_p} \leftarrow \texttt{UnMask}_k^d\left(\mathbf{c}^{B,k}\right)$ ▷ XOR all shares together.

Proof. We build a PINI simulator: given a set of probes P and share indexes B. We distinguish two cases: either (i) $d-1 \in B$ or there is a probe of P in the `UnMask` gadget, or (ii) there is no such probe.

In case (ii), we remark that the gadgets `SecA2BModp` and `SecAddModp` are PINI, as well as `RefreshIOS` (its is sharewise after application of the random-zero transform of [Cor18]). The probes in these gadgets can thus be simulated by knowing at most $|P|$ shares of $\mathbf{x}^{B,k}$ and some $\mathbf{z}^i_{A_p}$ for $i \in [\![0, d-1[\![$. Such $\mathbf{z}^i_{A_p}$, which also are the possible output shares to simulate, can be perfectly simulated since they are randomly generated by the gadget.

In case (i), we consider the $(d-1)$-PINI simulator that has to simulate the output shares with index in B and the internal probes P. Let (P_0, P_r, P_u) be a partition of P such that the probes of P_0 are in `SecA2BModp` and `SecAddModp`, the ones of P_r are in `RefreshIOS`, and the ones of P_u are in `UnMask`. We first describe the simulator, then prove that it is correct.

The PINI simulator for `SecB2AModp` first selects randomly $\mathbf{z}^{d-1}_{A_p}$, then it generates a uniformly random sharing $\mathbf{c}^{B,k}$ of $\mathbf{z}^{d-1}_{A_p}$, from which it can simulate any probe in P_u. Next, using the IOS simulator, it determines the set of share indexes B_r of $\mathbf{b}^{B,k}$ required to simulate P_r, with $|B_r| \leq |P_r|$ (some shares from $\mathbf{c}^{B,k}$ are also needed for this simulation, but they are already simulated). We then consider the PINI simulation of the composition of `SecA2BModp` and `SecAddModp` (since these two gadgets are PINI): the shares of $\mathbf{b}^{B,k}$ with index in B_r and the probes P_0 can be simulated with the shares of $\mathbf{x}^{B,k}$ and $\mathbf{z'}^{A_p}$ whose index belongs to $B_r \cup B_0$, for some B_0 such that $|B_0| \leq |P_0|$. Finally, the simulator completes the

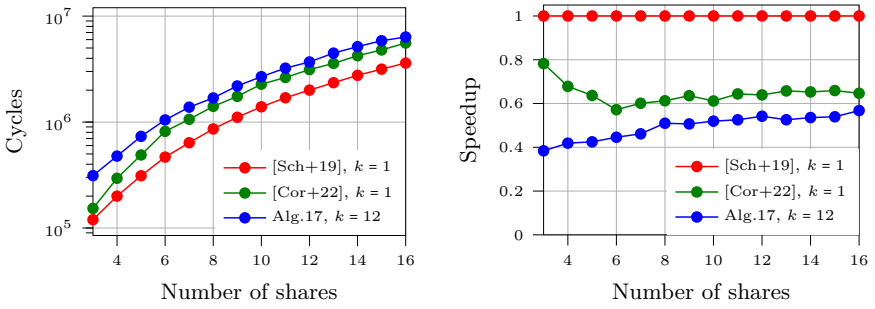
simulation by requesting the shares of $\mathbf{x}^{B,k}$ with index in $B_r \cup B_0$ and draws randomly all shares $\mathbf{z}^i_{A_p}$ with $i \in (B_r \cup B_0 \cup B) \setminus \{d-1\}$, which enables the simulation of the required $\mathbf{z'}^i_{A_p}$.

Let us first observe that the number of inputs required for the simulation is admissible: $|B_r \cup B_0| \leq |P|$. Further, let us denote by $B^* \subseteq [\![0, d-1[\![$ the set of i such that $\mathbf{z}^i_{A_p}$ is used in the simulation (we exclude $\mathbf{z}^{d-1}_{A_p}$ for now). We remark $B^* = B_r \cup B_0 \cup (B \setminus \{d-1\})$, and therefore that $|B^*| \leq |P_r \cup P_0| + |B \setminus \{d-1\}| \leq d-2$ where the latter inequality comes from the hypothesis that either $|P_u| \geq 1$ (hence $|P_0 \cup P_r| + |B| \leq d-2$), or $d-1 \in B$ (hence $|P| + |B \setminus \{d-1\}| \leq d-2$). As a result $|[\![0, d-1[\![\setminus B^*| \geq 1$, and, taking $i^* \in [\![0, d-1[\![\setminus B^*$, we observe that $\mathbf{z}^{i^*}_{A_p}$ is never used in the simulation.

We now show that the simulation is correct: for each value that is simulated, we show that its distribution matches the true distribution, and furthermore we prove that the simulation is consistent with (i.e., the simulated joint distribution is equal to the true distribution) the simulation of the values for which we already proved the correctness. First, the simulated shares $\mathbf{z}^i_{A_p}$ (except $\mathbf{z}^{d-1}_{A_p}$) and $\mathbf{z'}^i_{A_p}$ follow the same distribution as in Algorithm 17. Next, since $\mathbf{z}^{d-1}_{A_p} = z - \sum_{i=0}^{d-2} \mathbf{z}^i_{A_p} \mod p$ and since one of the terms of the sum ($\mathbf{z}^{i^*}_{A_p}$) is not used in the simulation and is uniformly distributed, $\mathbf{z}^{d-1}_{A_p}$ appears to the adversary as a fresh uniform value, and its simulation is correct. We continue with the correct simulation of the probes in P_0 and the shares $\mathbf{b}^i_{B,k}$: it follows from the PINI simulators of `SecA2BModp` and `SecAddModp`. Since `RefreshIOS` is uniform, its output sharing $\mathbf{c}^{B,k}$ is a uniform sharing of $\mathbf{z}^{d-1}_{A_p}$ which is independent of $\mathbf{b}^{B,k}$. The simulation of the probes in P_r by the `RefreshIOS` simulator ensures that the simulation of these probes and of $\mathbf{c}^{B,k}$ are correct. Finally, the simulation of the probes P_u is trivially correct. □

Let us now compare in Figure 3.8 the performance of various implementations of `SecB2AModp`. We compare Algorithm 17 to the two most efficient state of the art the algorithms (the ones from [Sch+19] and [Cor+22]), which both implement $\texttt{SecB2AModp}^d_k$ from single-bit conversions. As a result, their computational cost is proportional to k, and we observe that they have comparable cost, with a small advantage for [Sch+19] (which agrees with the results on Intel x86 processors of [Cor+22, Table 4]).

Our bitsliced conversion gadget (Algorithm 17) always operates on



(a) Cycle count. **(b) Speedup w.r.t. [Sch+19], $k = 1$.**

Figure 3.8: Performance comparison of `SecB2AModp` implementations.

$\lceil \log_2(p) \rceil$ bits (here, 12). Concretely, for 16 shares, the bitsliced conversion of any $x \in \mathbb{Z}_p$ is only twice as slow as the state of the art single-bit conversions, and is therefore on par with state of the art 2-bit conversions. For larger k-bit conversions, the advantage of Algorithm 17 grows linearly with k.

4 Composition in the robust probing model

Let us now discuss the security in more practical models, that is, taking into account some characteristics of the leakage from CMOS electronic circuits (so-called *hardware implementations*): glitches and transitions. While many previous works introduce glitch-robust masking schemes (see Table 4.1 for an overview), none of them analyzes the security against transitions, and the only arbitrary-order composable glitch-robust scheme is [Fau+18], which suffers from poor efficiency (e.g., it uses $d(d-1)$ random elements for each multiplication and has a latency of 4 clock cycles).

In this chapter, we describe the hardware private circuit (HPC) masking scheme which is trivially composable and robust against both glitches and transitions. This scheme is based on the PINI composition introduced in the previous chapter, with adaptations to ensure that the masked circuit is robust against glitches and transitions (even when then combine), both at the level of single gadgets and for gadget compositions. HPC is a trivially composable scheme, for which we propose two multiplication gadgets: one follows the "refreshed multiplication" strategy, while the other one is adapted from the PINI1 gadget. These two gadgets are more efficient than the multiplication of [Fau+18], having a latency of 2 cycles and requiring less randomness. Finally, our work on HPC has been recently extended by Knichel et al. [KSM22; KM22]: they introduced HPC multiplication gadgets with even lower latency, at the cost of increased randomness usage.

We begin the chapter by introducing the new circuit and leakage models we work with. Next, we discuss glitch-robust security: we introduce the HPC schemes and compare them to previous glitch-robust composable masking schemes. We then show how HPCs can be robust in presence of transitions, and in presence of both glitches and transitions. Finally, we describe how the security in such models can be automatically verified. We conclude this chapter with a discussion of the leakage modeling of software implementations, linking it to the models for hardware implementations.

The contributions of this chapter have been published in [Cas+21b]

Table 4.1: Robust masking schemes: state of the art overview.

Scheme	Order	Cost*	Glitch-roust	Transition-robust	Composable
TI [NRR06]	$t = 1$	✓	✓	✗	✓
HO-TI [DN22]	$t = 2$	✓	✓	✗	✓
CMS [Rep+15]	$t \leq 2$	✓	✓	✗	✗
DOM, UMA, GLM [GMK16b; GM18; GIB18]	any	✓	✓	✗	✗
Faust et al. [Fau+18]	any	✗	✓	✗	Trivially†
HPC (new)	any	✓	✓	Glitch+Transition	Trivially‡§
GHPC [KSM22]	$t = 1$	✓	✓	Glitch+Transition	Trivially‡
HPC3 [KM22]	any	✓	✓	Glitch+Transition	Trivially‡

* Pareto-optimal w.r.t. latency, area, randomness (for a given set of security properties).
† Conjectured in [Fau+18].
‡ Only glitch-robustness is achieved with trivial composition, transition-robust composition requires additional circuit structure.
§ Trivial transition-robust composition can also be achieved with O-PINI gadgets.

(glitch-robustness only) and [CS21a] (taking transitions into account), exception Section 4.3.3 which has not been published.

4.1 Circuit model

The arithmetic circuit model of Section 2.1 is not a satisfying model to analyze the side-channel leakage of electronic circuits. Indeed, as introduced in Section 1.1, such circuits contain stateful elements (the *registers*) and usually contain loops which feed back computation results from one execution cycle to the next one [MCS22b; MCS22a]. The execution of an electronic circuit (i.e., all the evaluations of logic gates over multiple clock cycles) can be treated as an arithmetic circuit by *unrolling* it: for each clock cycle, each logic gate is mapped to a new arithmetic gate, and the wires in the arithmetic circuit connect the gates according to the data flow. On top of this representation, one need to retain the relationship between the arithmetic gates and the corresponding logic gate, which is needed to model the leakage of the circuit (see Section 4.2).

We formalize this unrolling with a new *structural model* for electronic circuits, where the *execution* of such circuits is equivalent to the randomized arithmetic circuits of Section 2.1. By construction, real-world electronic circuits operate in $\mathbb{F}_2$. However, when considering busses and sets of gates instead of single wires and logic gates, one may view an electronic circuit as operating in a larger field $\mathbb{F}_q$ [GMK16a; Duv+21], with the conservative hypothesis that a probe on a wire reveals fully a field element.

Definition 19 (Structural gate)**.** *A structural gate is a tuple* (I, P, f, ℓ)*, where*

- I *is the set of inputs of the gate,*
- P *is the set of parameters partitioned into* P^P *(public parameters) and* P^S *(secret parameters),*
- $f : (I \to \mathbb{F}_q) \times (P \to S) \to \mathbb{F}_q$ *is the evaluation function (where* S *some arbitrary set for the values of the parameters),*
- $\ell : I \to \mathbb{N}$ *is the latency of the inputs (a latency of one means the input is required one cycle before the output is produced).*

Any structural gate has one output whose value is given by the evaluation function.

We note that the latency is defined "backwards" by taking the output as reference (while it is usually defined forwards as the number of cycles needed to produce the output after the input is available). The reason for this choice is that our gates always have a single output while they may have any number of inputs. This convention makes our following treatments lighter. The parameters serve two purposes: the public parameters model the *control logic*: the part of the circuit whose values are not sensitive (as opposed to the *datapath* that we model here), while the secret parameters model secret values: inputs and random tape of the circuit.

The gates we use for building our gadgets are:

- The combinational arithmetic gates which have latency 0, e.g., the 2-input XOR (in $\mathbb{F}_2$):

  $$\mathsf{XOR2} = (\{\alpha, \beta\}, \varnothing, (f_i, f_p) \mapsto f_i(\alpha) \oplus f_i(\beta), x \mapsto 0),$$

  where α and β are unique symbols representing the input wires. The f_i and f_p functions are the valuations of the inputs and the parameters.

- The 2-input MUX where the select bit is a public value:

  $$\mathsf{MUX2} = (\{\alpha, \beta\}, \{\gamma\}, (f_i, f_p) \mapsto f_i(f_p(\gamma)), x \mapsto 0),$$

  where the public parameter γ takes values in $I = \{\alpha, \beta\}$

- The registers that delay their input by one clock cycle:

  $$\mathsf{Reg} = (\{\alpha\}, \varnothing, (f_i, f_p) \mapsto f_i(\alpha), x \mapsto 1)\ .$$

- The randomness generating gates (random gate):

$$\mathsf{Rnd} = (\varnothing, \{\alpha\}, (f_i, f_p) \mapsto f_p(\alpha), \varnothing) \ ,$$

  where $\alpha \in P^S$ represents the randomness, which is encoded as a random tape in the f_p function.

- The input gates whose representation is identical to the random gate, where the secret parameter represents the input value instead of the random tape.

- The constant gates whose value is publicly parameterized. The formalization is again identical to the random gates, except that the parameter is public: $\alpha \in P^P$.

We also define wires that connect the output of a gate to the input(s) of a gate:

Definition 20 (Structural wire)**.** *A structural wire is a pair* $(g, (g', i))$ *where* g *and* g' *are gates and* i *is an input of* g'*. We say that the wire connects its source* g *to the input* i *of* g' *(its destination).*

Based on these definitions, we define structural circuits as follows.

Definition 21 (Structural circuit)**.** *A structural circuit is a directed graph whose nodes are structural gates and whose edges are structural wires. A wire connects its source to its destination. In a structural circuit, there must be no combinational loop, that is, no cycle for which all the wires have a destination with latency 0. The outputs of a circuit is a subset of its gates.*

Next, we consider the execution of a circuit over time (that is, over multiple cycles).

Definition 22 (Circuit execution)**.** *An execution of a structural circuit* $C = (G, W)$ *for the set of cycles* $T = \{t_0, \ldots, t_{n-1}\}$ *is a directed graph whose set of nodes is* $G \times T$ *(the (executed) gates) and whose set of edges is* $W \times T$ *(the (executed) wires). Executed wires connect executed gates according to their latency: let* $\ell_g(i)$ *be the latency of the destination of the structural wire* $w = (g, (g', i))$*, then the executed wire* (w, t) *connects its source* $(g, t - \ell_g(i))$ *to its destination* (g', t)*. If the source does not exist, then the wire is connected to a fresh "initial state" source executed gate (no-input gate having as output a public parameter). Each executed gate may be annotated with a parametrization function, mapping the set of all (resp. only public) parameters of the underlying structural gate to values, resulting in a complete (resp. partial) execution.*

Definition 23 (Circuit evaluation)**.** *The evaluation of a complete circuit execution is a function that maps every gate and wire to an element $\mathbb{F}_q$. The values are computed by applying the evaluation function for the gates in post-topological order, evaluating the parametrization function for the parameters and using the already computed values of the circuit evaluation function for the inputs.*

An evaluation of a partial circuit execution is parameterized by the values of the secret parameters of the input gates (i.e., the values of the inputs of the circuit). The evaluation of the partial execution is the evaluation of the full execution by using these parameters and using uniform randomnes for the evaluation functions of the random gates parameters.

Let us remark that the post-topological order always exists since time ordering and absence of combinational loop guarantee that a circuit execution is a directed acyclic graph (DAG).

In the following, we work with partial circuit executions which (with our evaluation notion) are equivalent to the randomized arithmetic circuits of Section 2.1. The circuit executions however contain additional information needed for the GT-robust probing model.

We now clarify the notion of gadget in our new framework. First, we define the gadget execution, which is similar to the gadget in the ISW circuit model.

Definition 24 (Gadget execution)**.** *A gadget execution with d shares in a circuit execution C is a subset of gates G and wires W of C, such that W is the set of wires in C whose destination is in G. The inputs (also named input shares) of the gadget are the set of its wires whose source is not in G, and its outputs (or output shares) are a subset of the gates of the gadget. The inputs (resp. outputs) are partitioned in n_i (resp. n_o) tuples of d elements designated as input (resp. output) sharings.*

The composition of gadgets forms a new gadget, with the constraint that no input of a gadget can depend on one of its outputs.

Definition 25 (Gadget execution composition)**.** *A gadget execution G is a composition of a set of disjoint gadget executions $(G_i)_{i=0,\dots,n}$ if G is the union of all G_i's (the composing gadgets). The connections between gadgets must connect the input sharings to the output sharings of the gadgets, respecting the order of shares. Moreover, each input (respectively output) sharing of the composite gadget G must be an input (resp. output) sharing of a composing gadget G_i. The composing gadgets graph (directed graph where the nodes are the gadgets and the edges are the wires connecting them) must be a DAG.*

Building on the link between partial circuit executions and randomized arithmetic circuits, we observe that a gadget execution corresponds to the definition of a gadget (Definition 1), and that the arithmetic circuit corresponding to any gadget execution composition can be rewritten as an extended sequential composition (Definition 4). Besides, parallel and sequential gadget execution composition are defined as particular cases of the composition of two gadget executions such that the corresponding arithmetic circuit is a parallel (respectively sequential) composition of the gadgets.

In the "structural" world, defining a gadget as a set of structural gates and wires would lead to a loss of information on the gadget (e.g., whether some gadget is a straight pipeline that uses each gate once, or it has a serial implementation that uses multiple times the same gate). We therefore define structural gadgets by the fact that all its executions are identical gadget executions, except for a translation in time.

Definition 26 (Translation)**.** *Let $t \in \mathbb{Z}$ and C be a circuit execution. The translation by t of a set of gates $\{(g_i, t_i)\}_{i\in[\![0,n[\![}$ and wires $\{(w_i, t_i)\}_{i\in[\![0,n'[\![}$ of C are the sets of gates $\{(g_i, t_i + t)\}_{i\in[\![0,n[\![}$ and wires $\{(w_i, t_i + t)\}_{i\in[\![0,n'[\![}$. Two sub-sets C_1 and C_2 of C are translation-equivalent if there exists an integer t such that the translation of C_1 by t is equal to C_2.*

Definition 27 (Structural gadget)**.** *A structural gadget is a non-empty set of gadget executions that are all disjoint and translation-equivalent to each other. For each structural gadget we choose a canonical execution, then each gadget execution is associated to a canonicalization which is a translation by the canonical offset t that, when applied to the execution, gives the canonical execution.*

In classical hardware implementations, each structural gadget is implemented by a distinct set of gates. Our definition is however flexible to cover more use-cases, such as implementation on a processor where the structural gates of the datapath are used by all gadgets (at different cycles).

Definition 28 (Structural gadget composition)**.** *The structural gadget S is a composition of structural gadgets $\{S_i\}_{i\in[\![0,n[\![}$ if each of its executions is the composition of some S_i executions (possibly multiple executions of each S_i).*

4.2 Robust probing model

The robust probing model of [Fau+18] can now be formalized with our new circuit model: the adversary can place up to t *extended probes* in

a gadget execution. Each extended probe is then *expanded* to model glitches, transitions and/or couplings, giving a set of *expanded probes*. We then check the security of the gadget in the threshold probing model for arithmetic circuits (see Section 2.3.1) with the expanded probes using the equivalence between partial circuit execution and randomized arithmetic circuits.

Definition 29 (Probe expansion)**.** *A probe expansion scheme defines a set of extended probes for any gadget execution. The expansion of an extended probe is a set of (executed) wires in the gadget, named the expanded probes. The expansion of a set of extended probes is the union of the expansion of the extended probes.*

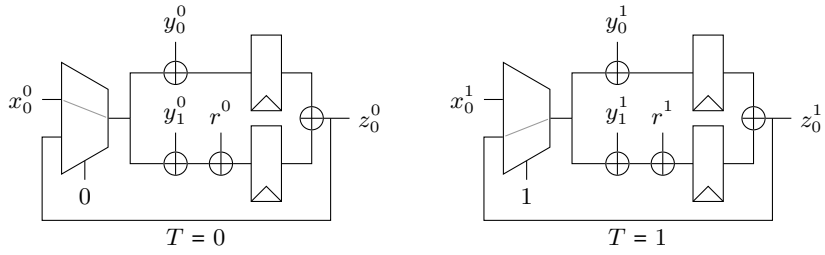
Definition 30 (Robust probing security)**.** *A gadget execution is t-robust probing secure with respect to a probe expansion scheme if the expansion of any set of at most t extended probes is safe for the gadget (when viewed as a computation graph, see Definition 6).*

We are interested in three particular cases of the robust probing model: glitches, transitions and finally the combination of glitches and transitions.[1] Those are defined by their probe expansion scheme that we describe next and illustrate in Figure 4.1.

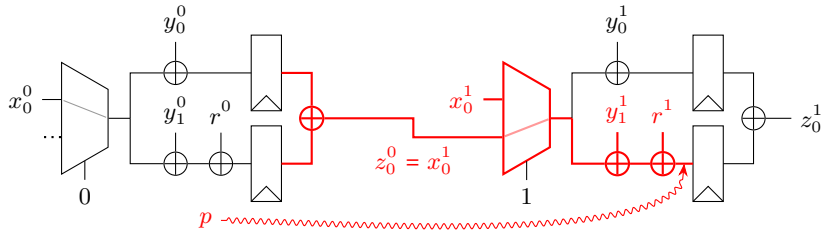
Glitch-robust In CMOS circuits, glitches propagate through combinational logic and are stopped by registers. Therefore, the glitches on a wire may depend on all the inputs of the combinational circuit that generates the value carried by the wire. The notion of "combinational circuit" can be characterized by gates with latency 0, while sequential logic such as registers have a non-zero latency. In the glitch-robust probing model, we care not only about glitches, which are ephemeral incorrect computations, but also data-dependent computation delay (i.e., data-dependency in the precise timing for the change in a wire value). Both phenomenons have broken the security of masked circuits, hence they should be modeled [MPG05; FG05].[2]

[1] We do not discuss couplings in this work, but they could be included in our framework with the help of additional information about structural gates and wires (e.g., power supply domains or electromagnetically coupled wires).

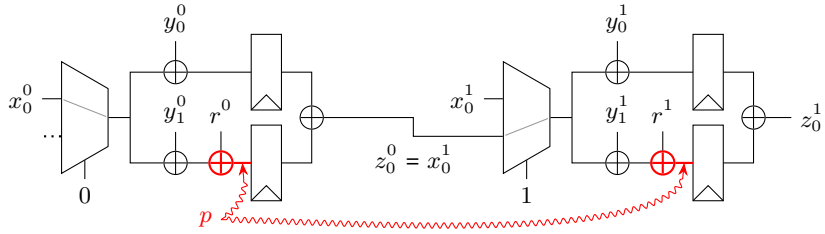
[2] Our assumption is known to be too strong in some cases, such as when one of the input of an AND gate stays at 0 over multiple cycles. The output of the gate is then stuck at 0 independently of the other input, if the capacitive coupling between the input and output of the gate is negligible. This does typically not occur for shares since they are randomized, but it can happen for control signals (i.e., public parameters), which can be exploited in glitch-robust security proofs by adapting the glitch-robust probing model to expand probes depending on the characteristics and the public parameters of each gate [Gig+21].



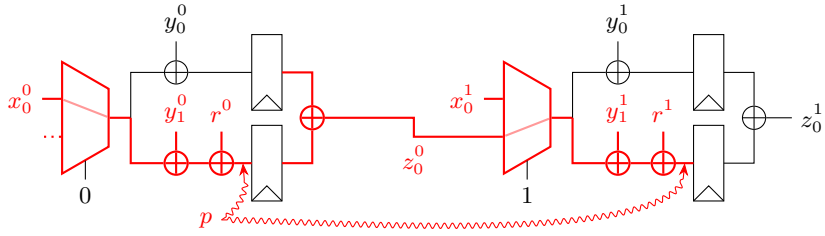
(a) Sequential circuit execution.



(b) Unrolled execution with a glitch-extended probe.



(c) Unrolled execution with a transition-extended probe.



(d) Unrolled execution with a glitch+transition-extended probe.

Figure 4.1: Example of glitch-, transition- and glitch+transition-extended probes in a circuit executed over two cycles, computing $z_0^T = \mathsf{Reg}(x_0^T \oplus y_0^T) \oplus \mathsf{Reg}(x_0^T \oplus y_1^T \oplus r^T)$ for $T = 0, 1$. The MUX is controlled such that x_0^0 is a circuit input, and $x_0^1 = z_0^0$, as shown in Figure 4.1a. There is a single extended probe p on the wire $x_0^1 \oplus y_1^1 \oplus r^1$, and the probe expansion is highlighted in red.

More formally, in the glitch-robust probing model, there is an extended probe for each gate and each wire. For input wires, the extended probe is expanded to a single probe on the wires, while for the other wires, the expansion is equal to the expansion of the extended probe on the source gate of the wire. For gates, the expansion is the union of the expansion of extended probes on the wires connected to the input with latency 0.

The design of higher-order masking schemes that are secure against glitches has been extensively studied in the literature. The first proposal was a higher-order TI (see Section 4.3.4 for a description of TIs) generalization of Bilgin et al. [Bil+14], which was rapidly shown insecure against multivariate attacks [Rep15]. Various follow-up papers then proposed efficient and innovative ways to implement higher-order masking in hardware. We mention for example the Consolidated Masking Scheme (CMS) in [Cnu+16], the Domain-Oriented Masking (DOM) in [GMK16b; GMK17], the Unified Masking Approach (UMA) in [GM18] and the Generic Low Latency Masking (GLM) in [GIB18]. Yet, and in contrast with the literature in threshold probing security, none of these proposals came with a security proof. Furthermore, this lack of proof has been shown to not only be a theoretical concern, and that the higher-order versions of all these schemes are flawed [Moo+19]. The only scheme of the literature coming with a security proof in the glitch-robust probing model is the one of Faust et al. [Fau+18] (although that proof did not consider glitch-robust composition).

Transition-robust Next, the current consumption of a CMOS gate is due (as a first approximation) to the charge or discharge of its output, and therefore depends on both its new and previous value. We name this phenomenon "transitions".

Formally, the transition-extended probes in a gadget are all its wires (w, t) and gates (g, t). The expansion of a wire (w, t) (resp. (g, t)) is $\{(w, t-1), (w, t)\}$ (resp. $\{(g, t-1), (g, t)\}$) if $(w, t-1)$ (resp. $(g, t-1)$) belongs to the gadget execution, otherwise it is simply (w, t) (resp. (g, t)).

Glitch+transition-robust Finally, in the transitions and glitches combined robust (transition+glitch-robust, or GT-robust for short) probing model, the set of extended probes is the set of transition-extended probes. The expansion of a probe is obtained by first computing its transition-expansion, giving a set of transition-expanded probes. Then, the GT-robust expansion of the probe is the glitch-expansion of this

set.[3] We note that the set of expanded probes obtained in this model is larger than the union of the expansion in the glitch- and transition-robust models, as shown in Figure 4.1.

Physically, this models the fact that the delay in a logic gate depends on both the new and the previous values on the wire. This delay then propagates through the wires in the form of glitches or data-dependent signal timing, as empirically shown in [Mül+22].

The Glitch+transition-robust-probing model is therefore better suited than the two previous models for analyzing the security of masked CMOS circuits. We first introduce techniques to guarantee security in these two simpler models, then combine them to achieve security against glitches and transitions combined.

4.3 Glitch-robust composition

Let us first consider the glitch-robust probing model. There, the expansion of the probes depends on the connection between the gates in the circuit and on the latency of the gates (i.e., whether the latency of an input is zero or not). Since this information can be extracted from circuit executions (with latency annotations to know where propagation of glitches should be stopped), we do not need to work explicitly with structural circuits in this section.

The simulatability definition can be adapted to the glitch-robust probing model, and shown to enjoy similar properties as its threshold probing counterpart.

Definition 31 (Glitch-robust simulatability)**.** *A set of extended adversarial probes P in a gadget G can be glitch-robustly simulated with a set of input shares $\mathcal{I}'$ if the glitch-expansion of P can be simulated with the inputs shares $\mathcal{I}'$, assuming that there are no glitches on the inputs (i.e., the expansion of probes on the input shares contains only the corresponding input wire).*

The notions of glitch-robust NI, glitch-robust SNI and glitch-robust PINI naturally follow from this definition. Moreover, since there is no glitch to simulate in the sharing encoder (by Definition 30), glitch-robust NI implies glitch-robust probing security, and the proof is the same as the one of Proposition 4: it is a direct application of Proposition 1.

Let us now turn to composition. Due to the localized nature of glitches, which propagates from a gadget to another only through input and output

[3] Expanding first the glitches, then the transitions gives the same set of expanded probes.

sharings, the simple composition properties of simulatability without glitches are retained.

Proposition 18 (Glitch-robust simulatability of parallel gadget composition)**.** *Let the gadget* $\mathsf{G} = \mathsf{G}_1 || \mathsf{G}_2$ *be a parallel composition of* G_1 *and* G_2*, whose sets of input shares are respectively* $\mathbf{x}$ *and* $\mathbf{y}$*. Let* P_1 *(resp.* P_2*) be a set of probes in* G_1 *(resp.* G_2*). If there exists a glitch-robust simulator* $\mathcal{S}_1$ *(resp.* $\mathcal{S}_2$*) for* G_1 *(resp.* G_2*) that can simulate the glitch-extended probes* P_1 *(resp.* P_2*) using the input shares* $\mathbf{x}_{|\mathcal{I}_1}$ *(resp.* $\mathbf{y}_{|\mathcal{I}_2}$*), then there exists a glitch-robust simulator* $\mathcal{S}$ *for the gadget* G *that can simulate the set of glitch-extended probes* $P_1 \cup P_2$ *using the input shares* $(\mathbf{x}_{|\mathcal{I}_1}, \mathbf{y}_{|\mathcal{I}_2})$*.*

Proof. The proof is identical to the proof of Proposition 2. □

For sequential composition, we first need the following lemma that formalizes the intuition that glitches on the inputs of a gadget do not change the simulation, except for the simulator needing to know the values causing these glitches.

Lemma 5. *Let* G *be a gadget and* P *a set of extended probes that can be glitch-robustly simulated using a set of inputs* $\mathcal{I}'$ *by a simulator* $\mathcal{S}$*. In presence of glitches on the inputs of* G*, there exists a simulator* $\mathcal{S}^{gl}$ *that can simulate* P *using extended probes on inputs* $\mathcal{I}'$*.*

Proof. Let $p \in P$ be an extended probe, and $\bar{p}$ (respectively $\tilde{p}$) be it expansion when there is are not (resp. when there are) glitches on the inputs of G. Then, any wire w belonging to $\tilde{p}$ either belongs to $\bar{p}$ and can be simulated by $\mathcal{S}^{gl}$ (by simply using S), or is due to a glitch on an input. Let i be that input wire, then i belongs to $\tilde{p}$, and therefore to $\bar{p}$ (as i is in G). This implies that $i \in \mathcal{I}'$ since $\mathcal{S}$ must have knowledge of i to glitch-robustly simulate p. Therefore, the simulator $\mathcal{S}^{gl}$ has access to w through the expansion of the input probe i. □

We can now prove sequential composition.

Proposition 19 (Glitch-robust simulatability of sequential gadget composition)**.** *Let the gadget* $\mathsf{G} = \mathsf{G}_1 || \mathsf{G}_2$ *be a sequential composition of* G_1 *and* G_2*, where* $\mathbf{x}$ *is set of input shares of* G_1 *and* $\mathbf{y}$ *is the set of output shares of* G_1 *and input shares of* G_2*. Let* P_1 *(resp.* P_2*) be a set of glitch-extended probes in* G_1 *(resp.* G_2*). If there exists a glitch-robust simulator* $\mathcal{S}_2$ *for* G_2 *that can simulate* P_2 *using the input shares* $\mathbf{y}_{|\mathcal{I}_2}$*) and a glitch-robust simulator* $\mathcal{S}_1$ *for* G_1 *that can simulate* P_1 *and* $\mathbf{y}_{|\mathcal{I}_2}$ *using the input shares* $\mathbf{x}_{|\mathcal{I}_1}$*, then there exists a simulator* $\mathcal{S}$ *for the gadget* G *that can simulate the set of probes* $P_1 \cup P_2$ *using the input shares* $\mathbf{x}_{|\mathcal{I}_1}$*.*

Proof. The proof is similar to the one of Proposition 3, except that the simulator for G_1 simulates glitch-extended probes on $\mathbf{y}_{|\mathcal{I}_2}$, and therefore, using Lemma 5, the glitch-robust simulator $\mathcal{S}_2$ for G_2 can be turned into $\mathcal{S}_2^{gl}$ that simulates the glitch-extended probes in P_2, taking into account the glitches on the inputs of G_2. □

Since Proposition 18 and Proposition 19 provide the same composition guarantees for glitch-robust probing security as Proposition 2 and Proposition 3, the composition strategies in the threshold probing model can be used for the glitch-robust probing model.

4.3.1 Hardware private circuits

The adaptation of the trivial PINI composition strategy to the glitch-robust probing model has been named hardware private circuits (HPC), and has been the first provably-secure glitch-robust higher-order masking scheme. The composability has already been proven in the previous section, and we focus here on introducing the gadgets, starting with sharewise gadgets.

Proposition 20. *Any sharewise gadget is glitch-robust PINI.*

Proof. The proof is identical to the proof of Proposition 6 since glitches do not propagate across separate gadget shares. □

Corollary 2. *The composition of glitch-robust t-PINI gadget execution is glitch-robust t-PINI.*

Proof. The proof is the same as the proof of Theorem 1, where Propositions 18, 19 and 20 are used instead of Propositions 2, 3 and 6. □

We next introduce in Algorithm 18 a first glitch-robust PINI AND gadget. This gadget is based on the PINI1 multiplication (Algorithm 4) but inserts registers to prevent glitches from breaking the security. This gadget however only works in $\mathbb{F}_2$: it is not glitch-robust probing secure in larger fields, as discussed next.

Proposition 21. *The HPC2 AND gadget (Algorithm 18) is glitch-robust PINI.*

Proof. Let us build a glitch-robust PINI simulator. We assume wlog that only the input wires of registers and the outputs of the gadgets are probed, (since the other extended probes are less powerful). Namely, these probes can be $\mathbf{z}_i$, $\mathbf{x}_i\mathbf{y}_i$, $u_{ij} := (\mathbf{x}_i \oplus 1)\, r_{ij}$, $v_{ij} := \mathbf{y}_j \oplus r_{ij}$ and $\mathbf{x}_i v_{ij}$. Given a set of probes adversarial extended probes P and probed output

Algorithm 18 HPC2 AND gadget with d shares (synchronization registers are omitted).

Input: Sharings $\mathbf{x}$, $\mathbf{y}$
Output: Sharing $\mathbf{z}$ such that $z = x \cdot y$.

1: **for** $i = 0$ to $d-1$ **do**
2: **for** $j = i+1$ to $d-1$ **do**
3: $r_{ij} \xleftarrow{\$} \mathbb{F}_2$; $r_{ji} \leftarrow r_{ij}$
4: **for** $i = 0$ to $d-1$ **do**
5: $\mathbf{z}_i \leftarrow \mathbf{x}_i\mathbf{y}_i$
6: **for** $j = 0$ to $d-1, j \neq i$ **do**
7: $\mathbf{z}_i \leftarrow \mathbf{z}_i \oplus \mathsf{Reg}\left((\mathbf{x}_i \oplus 1)\, r_{ij}\right) \oplus \mathsf{Reg}\left(\mathbf{x}_i \mathsf{Reg}\left(\mathbf{y}_j \oplus r_{ij}\right)\right)$

shares A, the set of required input shares X is computed as follows: for each probed $\mathbf{z}_i$ or $\mathbf{x}_i\mathbf{y}_i$, add i to X. Then, for each $i \neq j$ pair, if two out of u_{ij}, v_{ij} and $\mathbf{x}_i v_{ij}$ are probed, or if i of j belongs to X: add i and j to X. Otherwise, if u_{ij} or $\mathbf{x}_i \times v_{ij}$ is probed, add i to X, and if v_{ij} is probed, add j to X. The set B is computed as $X \setminus A$.

We observe that the set B satisfies the PINI definition: $|B| \leq |P|$ by construction. All the values to be simulated that depend only on input shares with index in X and on randomness are computed as specified by Algorithm 18 (the required randomness is generated by the simulator). The allows to simulate all the extended probes on $\mathbf{x}_i\mathbf{y}_i$, u_{ij} and v_{ij}, by construction of X. Then, for all remaining extended probes ($\mathbf{z}_i$ (for which $i \in A$) and $\mathbf{x}_i v_{ij}$), we observe that $i \in X$. They can therefore be computed as it is done by the gadget, except when the simulation of $v_{ij} = \mathbf{y}_j \oplus r_{ij}$ is needed and $j \notin X$. In this case, the simulator simulates v_{ij} by sampling a fresh random r'_{ij} (we say that the simulator *cheats* for ij).

Let us show that this algorithm is indistinguishable from the true gadget. The behavior of the simulator is identical to the behavior of the gadget, except when it cheats for ij. We next prove that if the simulator cheats for ij, then r_{ij} is not observed in the set of probes, except through v_{ij}, therefore v_{ij} is indistinguishable from a fresh r'_{ij} and simulation is correct.

The simulator cheats for ij only if $j \notin X$ and a value depending on v_{ij} is probed. The first condition implies that none of $\mathbf{z}_j$, u_{ji}, $\mathbf{x}_j v_{ij}$ and v_{ij} are probed, and at most one of $\mathbf{z}_i$, $\mathbf{x}_i v_{ij}$, u_{ij} and v_{ji} can be probed. The second condition implies that $\mathbf{z}_i$, or $\mathbf{x}_i v_{ij}$ is probed (v_{ij} cannot be probed due to the previous observation). Therefore, the only values depending on

r_{ij} that can be probed are $\mathbf{z}_i$ or $\mathbf{x}_i v_{ij}$, and exactly one of those is probed. If $\mathbf{x}_i v_{ij}$ is probed, then the simulation is correct: the extended probe expands to $\{\mathbf{x}_i, v_{ij}, \mathbf{x}_i v_{ij}\}$, which are the only observations depending on r_{ij}. If $\mathbf{z}_i$ is probed, then observations depending on r_{ij} are $(\mathbf{x}_i \oplus 1) r_{ij}$ and $\mathbf{x}_i v_{ij}$, and functions of these values. If $\mathbf{x}_i = 0$, then $\mathbf{x}_i r_{ij} = 0$ does not depend on r_{ij}, which is thus only observed through v_{ij}, hence the simulation is correct. Otherwise, we have $\bar{\mathbf{x}}_i = 0$, which implies that $(\mathbf{x}_i \oplus 1) v_{ij} = 0$, thus v_{ij} is not observed and no cheating is observed.[4] □

Combining the HPC2 and sharewise gadgets allows to efficiently mask any Boolean circuit glitch-robustly. The latency cost of such a circuit is reasonable, since sharewise gadgets do not require registers for security, and the AND gadget has a latency of one cycle with respect to one of its input sharings, and two cycles with respect to the other one. We show in [Cas+21b; MCS22a] that representations and implementations of Boolean circuits can be optimized to benefit from this asymmetric latency, resulting in a lower latency that the one of a circuit based on a two-cycle multiplication.

4.3.2 HPC in arbitrary field

We next focus on the design of a glitch-robust PINI multiplication larger fields. Similarly to the refreshed multiplication of Section 3.2.4, it can be built by composing a glitch-robust SNI multiplication gadget and a glitch-robust SNI refresh gadget.

Algorithm 19 Glitch-robust NI (DOM-indep) and SNI (FaustMUL) multiplication gadgets with d shares (synchronization registers are omitted).

Input: Sharings $\mathbf{x}$, $\mathbf{y}$
Output: Sharing $\mathbf{z}$ such that $z = x \cdot y$.

1: **for** $i = 0$ to $d - 1$ **do**
2: **for** $j = i + 1$ to $d - 1$ **do**
3: $r_{ij} \xleftarrow{\$} \mathbb{F}_2$; $r_{ji} \leftarrow -r_{ij}$
4: **for** $i = 0$ to $d - 1$ **do**
5: $\mathbf{z}_i \leftarrow \mathbf{x}_i \mathbf{y}_i$
6: **for** $j = 0$ to $d - 1, j \neq i$ **do**
7: $\mathbf{z}_i \leftarrow \mathbf{z}_i + \mathsf{Reg}\left(\mathbf{x}_i \mathbf{y}_j + r_{ij}\right)$
8: $\mathbf{z}_i \leftarrow \mathsf{Reg}\left(\mathbf{z}_i\right)$ ▷ Only for FaustMUL, not for DOM-indep.

[4] This argument does not work in larger fields, in which the HPC2 multiplication gadget is therefore not glitch-robust PINI.

Algorithm 20 Faust et al. glitch-robust PINI multiplication gadget.

Input: Sharings $\mathbf{x}$, $\mathbf{y}$
Output: Sharing $\mathbf{z}$ such that $z = x \cdot y$.
1: $\mathbf{t} \leftarrow \mathsf{FaustMUL}(\mathbf{x}, 1)$
2: $\mathbf{z} \leftarrow \mathsf{FaustMUL}(\mathbf{t}, \mathbf{y})$

In [Fau+18], Faust et al. build a simple glitch-robust SNI multiplication gadget (Algorithm 19) by inserting registers in the ISW multiplication gadget at well-chosen locations. As for HPC2, they require registers before the compression of the refreshed products, but they also added registers on the output shares of the gadget to ensure that the glitch-expansion of output shares does not reveal too much information. This leads to a latency of 2 cycles. Faust et al. used the refreshed multiplication strategy of Section 3.2.4 (Algorithm 7) with this glitch-robust SNI gadget instead of the ISW gadget originally used by Goudarzi and Rivain, giving Algorithm 20. As for the original construction, the same gadget is used for both the refresh and the actual multiplication, giving a PINI gadget with an asymmetric input latency of 2 and 4 cycles, respectively.

Let us first formally prove the security of this construction in the glitch-robust probing model (this was assumed by Faust et al., but not formally proven). The glitch-robustness of the refreshed multiplication of Faust et al. is easy to prove: since they already prove that Algorithm 19 is glitch-robust SNI, it remains to prove that the composition of the refresh and the multiplication is still PINI.

Proposition 22. *Algorithm 20 is glitch-robust PINI.*

Proof. The proof is the same as in Section 3.2.4. Indeed, Lemma 3 and Corollary 1 can be straightforwardly adapted to the glitch-robust setting without modifying their proofs, except for the use of Proposition 18 and Proposition 19 instead of Proposition 2 and Proposition 3. Moreover, Proposition 8 also holds in the glitch-robust setting since its proof does not depend on the specific definition of simulatability used. Finally, the proof that Algorithm 19 is glitch-robust SNI is given in [Fau+18]. □

In the remaining of this section, we present two optimizations to the previous construction, which leads to our HPC1 gadget. First, we remove the registers on the outputs of the multiplication, which reduces the latency by 1 cycle. Second, we optimize the refresh gadget to require only one cycle and less randomness.

For the first optimization, we remark that the multiplication without registers on the outputs (Algorithm 19) is actually identical to the DOM-indep multiplication gadget (19, from [GMK16a]). While this gadget is

not glitch-robust SNI (the expansion of the output $\mathbf{z}_i$ trivially leaks $\mathbf{x}_i$ and $\mathbf{y}_i$), it is actually LPINI with respect to the set containing the input sharing $\mathbf{x}$.

Proposition 23. *The DOM-indep multiplication gadget (Algorithm 19) with d shares is glitch-robust $(d-1)$-LPINI with respect to input set $\{\mathbf{x}\}$.*

Proof. Let A and P_1 follow the LPINI definition. The simulator initializes S_x (respectively S_y) to the set of shares of $\mathbf{x}$ (resp. $\mathbf{y}$) whose index belong to A. Wlog, let use assume that the only extended probes in P_1 are of the form $u_{ij} = \mathbf{x}_i\mathbf{y}_j + r_{ij}$ since any other extended probe is less powerful (i.e.,, its expansion is a subset of the expansion of one of these probes or of a probe on an output share).

For each u_{ij} probe in P, if $i \notin S_x$, the simulator updates S_x as $S_x \leftarrow S_x \cup \{i\}$, otherwise it sets $S_x \leftarrow S_x \cup \{j\}$; and, if $j \notin S_y$, it sets $S_y \leftarrow S_y \cup \{j\}$, otherwise it sets $S_y \leftarrow S_y \cup \{i\}$. We remark that the sets S_x and S_y, which are the indexes of the shares of $\mathbf{x}$ (and $\mathbf{y}$, respectively) needed for the simulation, satisfy the LPINI definition: $|S_y| \leq |P_1| + |A|$, and $|B| \leq |P_1|$ if $B = S_x \setminus A$.

The simulation of the probes proceeds as follows: for each pair (i, j) such that either there is a probe u_{ij} in P_1 or $i \in A$: if $i \in S_x$ and $j \in S_y$, compute $\mathbf{x}_i\mathbf{y}_j$ and $u_{ij} = \mathbf{x}_i\mathbf{y}_j + r_{ij}$ using the provided inputs (and set $r_{ji} = -r_{ij}$ to a fresh random if it is not yet set), otherwise set u_{ij} to a fresh random. Simulation is completed by computing $\mathbf{z}_i$ for all $i \in A$ as it is done by the true gadget.

We conclude the proof by showing that the simulation is indistinguishable from the gadget. The simulator behaves in the same way as the circuit, except when it needs to simulate u_{ij} where $i \notin S_x$ or $j \notin S_y$. In this case, u_{ij} is not probed but appears in a probe, therefore $\mathbf{z}_i$ is probed. This implies that $i \in S_x$, thus $j \notin S_y$, which implies that neither c_j nor u_{ji} are probed. Since u_{ij} contains the random r_{ij}, which itself does not appear in any probe except $\mathbf{z}_i$ (through u_{ij}), u_{ij} behaves as a fresh random from the point of view of the adversary, which is what the simulator generates. □

Remark. Similarly, it can be shown that DOM-indep is also glitch-robust LPINI with respect to $\{\mathbf{y}\}$. (This does not imply that it is LPINI w.r.t. $\varnothing$, i.e., PINI).

We finally introduce the HPC1 gadget in Algorithm 21, which can be instantiated with any glitch-robust SNI refresh gadget, such as the ones we introduce in the next section.

Corollary 3. *The HPC1 multiplication gadget (Algorithm 21) is glitch-robust PINI.*

Algorithm 21 HPC1: Glitch-robust PINI multiplication gadget with d shares (synchronization registers are omitted).

Input: Sharings $\mathbf{x}$, $\mathbf{y}$
Output: Sharing $\mathbf{z}$ such that $z = x \cdot y$.

1: $\mathbf{t} \leftarrow \mathsf{RefreshSNI}(\mathbf{x})$
2: $\mathbf{z} \leftarrow \mathsf{DOM\text{-}indep}(\mathbf{t}, \mathbf{y})$ ▷ Algorithm 19

Proof. This is a consequence of Lemma 3 (adapted to the glitch-robust settings, as discussed in the proof of Proposition 22) and of Proposition 23. □

4.3.3 Optimized glitch-robust SNI refresh

A simple glitch-robust SNI refresh gadget can be obtained from the SNI multiplication by 1 (Algorithm 19) [Fau+18]. This gadget has a latency of two clock cycles and uses $d(d-1)/2$ random elements. We introduce new gadget that improve these two metrics.

Off-path refreshing First, we optimize the latency by introducing a generic technique to reduce the latency of any glitch-robust SNI refresh gadget to one cycle (which is the minimum possible). This new construction is described in Algorithm 22. It works by using the original gadget while providing it with an all-zero input sharing, then summing its output with the sharing to be refreshed and adding a register layer after that. In this way, the latency of the original gadget does not matter: since it is not on the main datapath, it can be computed in advance.

Algorithm 22 OP-Refresh: Off-path refresh with d shares.

Input: Sharing $\mathbf{x}$.
Output: Sharing $\mathbf{y}$ such that $y = x$.

1: $\mathbf{z} \leftarrow \mathsf{0\text{-}Gen}(d)$ ▷ $d-1$-SOI 0-sharing generation, e.g., Algorithm 23
2: **for** $i = 0, \ldots, d-1$ **do**
3: $\quad \mathbf{y}_i \leftarrow \mathsf{Reg}(\mathbf{x}_i + \mathbf{z}_i)$

Intuitively, the original refresh gadget outputs a "glitch-robust secure" uniform sharing of zero, which can then be simply added to the sharing that must be refreshed. The output register layer prevents output probes from being extended to the inputs of the refresh. Let us now proof

the security of this construction, by first formalizing the security of the "0-sharing generation" gadget.

Definition 32 (0-sharing generation gadget)**.** *A 0-sharing generation gadget is a gadget with no inputs and one output sharing of $d-1$ shares $r_0, \ldots, r_{d-1}$ such that $r_0 + \cdots + r_{d-1} = 0$.*

Definition 33 (t-SOI)**.** *Let G be a gadget with no inputs and one d-shares output sharing, P be a set of (glitch-extended) probes in G and O be set of (glitch-extended) probes on the output shares of G. G is (glitch-robust) t-Strongly Output Independent (t-SOI) if there exists a simulator $\mathcal{S}$ such that for any P and O satisfying $|P| + |O| \leq t$, the distributions of the (extended) probes for the two following games are identical.*

Real. *The output of the real game is the values corresponding to the probes (O, P) for an execution of the gadget.*

Simulated. *The simulator $\mathcal{S}$ outputs sets of probes (O_1, O_2) such that $O_1 \cup O_2 = O$ and $|O_1| \leq |P|$, then simulates the (glitch-extended) probes belonging to P and O_1. The expansion of probes in O_2 are simulated as fresh uniform randomness.*

Using this definition, we can prove the security of Algorithm 22.

Proposition 24. *If 0-Gen is a glitch-robust $d-1$-SOI 0-sharing generation gadget, the gadget OP-Refresh (Algorithm 22) is glitch-robust $d-1$-SNI.*

Proof. Let P_i be a set of internal glitch-extended probes and P_o be a set of output glitch-extended probes P_o such that $|P_i| + |P_o| \leq d-1$. Let us assume wlog that there is no probe on $\mathbf{x}_i$ or $\mathbf{z}_i$ since the glitch-expansion of these probes are subsets of the expansion of the probe $\mathbf{x}_i + \mathbf{z}_i$. Let P_{G} be the probes of P_i that belong to $\mathsf{0\text{-}Gen}$, and P_S be the other probes (which all have the form $\mathbf{x}_i + \mathbf{z}_i$). Let O be the set of shares $\mathbf{z}_i$ such that $\mathbf{y}_i \in P_o$ or $\mathbf{x}_i + \mathbf{z}_i \in P_S$, which ensures that $|O| \leq |P_S| + |P_o|$, therefore $|P_{\mathsf{G}}| + |O| \leq d-1$. The SNI simulator proceeds as follows: run the glitch-robust $d-1$-SOI simulator with internal probes P_{G} and output probes O. Let O_1, O_2 be the resulting partition of O. The SNI simulator requests the input share $\mathbf{x}_i$ for each probe $\mathbf{x}_i + \mathbf{z}_i$ or $\mathbf{x}_i$ in P_i, and for each $\mathbf{z}_i \in O_1$. Simulation of the probes in P_{G} and P_S is then trivial using the SOI simulator and the output shares. For output probes $\mathbf{y}_i \in P_o$, if the input $\mathbf{x}_i$ is requested, simulation is trivial, while if it is not requested, then $\mathbf{z}_i \in O_2$ and $\mathbf{x}_i + \mathbf{z}_i$ is not probed, hence $\mathbf{y}_i$ can be simulated as a fresh random. The number of requested inputs is at most $|P_S| + |O_1| \leq |P_S| + |P_{\mathsf{G}}|$. □

SOI 0-sharing generation The off-path transformation requires a glitch-robust SOI 0-sharing generation gadget. Such a gadget can be built from a glitch-robust SNI refresh gadget [Cas+21b], such as the one based on the glitch-robust SNI multiplication (Algorithm 19) with an input set to 1. This construction has $\mathcal{O}\left(d^2\right)$ randomness complexity. We introduce in Algorithm 23 a specialized refresh gadget inspired the by the refresh of [Bat+16] that reduces it to $\mathcal{O}\left(d \log d\right)$.

Algorithm 23 0-Gen: SOI 0-Sharing generation gadget with d shares.

Output: Sharing $\mathbf{z}$ such that $\mathbf{z}_0 + \cdots + \mathbf{z}_{d-1} = 0$.

if $d = 1$ **then**
  $\mathbf{z}_0 \leftarrow 0$
else if $d = 2$ **then**
  $r \xleftarrow{\$} \mathbb{F}_q$
  $\mathbf{z} \leftarrow (r, -r)$
else
  $\mathbf{x}^0, \mathbf{y}^0 \leftarrow \mathsf{0\text{-}Gen}\left(\lfloor d/2 \rfloor\right), \mathsf{0\text{-}Gen}\left(\lceil d/2 \rceil\right)$ ▷ Recursive calls
  for $i = 0, \ldots, \lfloor d/2 \rfloor - 1$ **do**
    $r_i \xleftarrow{\$} \mathbb{F}_q$
    $\mathbf{x}_i^1, \mathbf{y}_i^1 \leftarrow \mathbf{x}_i^0 + r_i, \mathbf{y}_i^0 - r_i$
    $\mathbf{x}_i^2, \mathbf{y}_i^2 \leftarrow \mathsf{Reg}\left(\mathbf{x}_i^1\right), \mathsf{Reg}\left(\mathbf{y}_i^1\right)$
  if $d \mod 2 = 1$ **then**
    $\mathbf{y}_{\lfloor d/2 \rfloor}^1, \mathbf{y}_{\lfloor d/2 \rfloor}^2 \leftarrow \mathbf{y}_{\lfloor d/2 \rfloor}^0, \mathsf{Reg}\left(\mathbf{y}_{\lfloor d/2 \rfloor}^0\right)$
  $\mathbf{z} \leftarrow \left(\mathbf{x}^2, \mathbf{y}^2\right)$

Proposition 25. *The 0-Gen gadget (Algorithm 23) with d is glitch-robust $d-1$-SOI.*

Proof. The case $d = 1$ is trivial, since the probe sets P and O are empty. For $d = 2$, either P contains one probe and O is empty (hence simulation is trivial), or O contains one probe and P is empty (hence the output probe is a fresh random).

We proceed by induction for the general case. Let P^x (respectively P^y) be the set of probes in the first (resp. second) recursive call. Furthermore, let Q^x (resp. Q^y) be the set of the other internal probes, which are wlog extended probes on $\mathbf{x}_i^1$ (resp. $\mathbf{y}_i^1$), and let similarly (O^x, O^y) be a partition of output probes. For each of these sets, we denote with calligraphic letters the indices of the shares in them (e.g., $\mathcal{Q}^x = \{i \text{ s.t. } \mathbf{x}_i^1 \in Q^x\}$).

Let us first assume that d is even. Let P_O^x be a set of shares $\mathbf{x}_i^0$ such that $\mathcal{Q}^x \subseteq \mathcal{P}_{\mathcal{O}^x} \subseteq \mathcal{Q}^x \cup \mathcal{O}^x$, and $\left|P_O^x\right| = \min(d/2 - 1 - \left|P^x\right|, \left|\mathcal{Q}^x \cup \mathcal{O}^x\right|)$.

Using the $(d/2-1)$-SOI simulator for the first recursive call for the sets P^x and P^x_O, we get the sets $P^x_{O_1}$ and $P^x_{O_2}$ such that $P^x_O = P^x_{O_1} \cup P^x_{O_2}$, and the probes in $P^x_{O_2}$ can be simulated as fresh randoms. Let $U^x \subseteq O^x$ be such that $\mathcal{U}^x = {\mathcal{P}_{\mathcal{O}_\in}}^x \setminus \mathcal{Q}^x$, which ensures that any probe $\mathbf{x}^2_i \in U^x$ can be simulated as a fresh random, even if r_i is observed through another probe. We have $\left|U^x\right| \geq \left|P^x_{O_2}\right| - \left|Q^x\right|$, and since $P^x_{O_2} = P^x_O \setminus P^x_{O_1}$ where $\left|P^x_{O_1}\right| \leq \left|P^x\right|$, $\left|U^x\right| \geq \left|P^x_O\right| - \left|P^x\right| - \left|Q^x\right|$. Let us discuss two cases.

First, if $\left|\mathcal{Q}^x \cup \mathcal{O}^x\right| \leq d/2-1-\left|P^x\right|$, then ${\mathcal{P}_{\mathcal{O}}}^x = \mathcal{Q}^x \cup \mathcal{O}^x$, and therefore $\left|P^x_O\right| \geq \left|O^x\right|$. This implies $\left|U^x\right| \geq \left|O^x\right| - \left|P^x\right| - \left|Q^x\right|$. If the situation is similar for the second recursive call, using the same reasoning, we get $\left|U^y\right| \geq \left|O^y\right| - \left|P^y\right| - \left|Q^y\right|$. Therefore, we can make a SOI simulation by setting $O_2 = U^x \cup U_y$ and $O_1 = O \setminus O_2$: probes in O_2 are fresh randoms, while other probes (in P and O_1) can be simulated using the SOI simulators for the recursive calls and then following the normal operation of the gadget.

In the second case, $\left|\mathcal{Q}^x \cup \mathcal{O}^x\right| \geq d/2 - \left|P^x\right|$, giving $\left|P^x_O\right| = d/2 - 1$. Therefore, $\left|U^x\right| \geq d/2 - 1 - \left|P^x\right| - \left|Q^x\right|$. If $\left|O^x\right| \leq d/2 - 1$ (first subcase), then $\left|U^x\right| \leq d/2 - 1$ and the simulator described for the first case can be used. Otherwise (second subcase), we have $\left|O^x\right| = d/2$, which means that all output shares in $\mathbf{x}^2$ are probed. Therefore, there are at most $d/2 - 1$ other probes, which gives $\left|Q^x\right| + \left|Q^y\right| + \left|O^y\right| \leq d/2 - 1$. This means the simulator of the first case can be used for the $\mathbf{y}$ part of the gadget. Furthermore, there exits $i^* \in [\![0, d/2[\![\setminus (\mathcal{Q}^x \cup \mathcal{Q}^y \cup \mathcal{O}^y)$. By construction, r_{i^*} is not observed by the adversary through any probe except $\mathbf{x}^2_{i^*} \in O^x$, which can thus be simulated as a fresh random. With this probe is removed from O and O^x, the simulator of the first subcase can be used. The case $\left|O^y\right| = d/2$ is handled symmetrically.

Let us now discuss the case where d is odd. The proof is identical to the even case (changing $d/2$ to $\lfloor d/2 \rfloor$ when discussing the $\mathbf{x}$ part of the gadget or to $\lceil d/2 \rceil$ otherwise), except for the argument in the second subcase of the second case. Indeed, if there are $\lfloor d/2 \rfloor$ probes on O^x, we get the bound $\left|Q^x\right| + \left|Q^y\right| + \left|O^y\right| \leq \lfloor d/2 \rfloor$, which allows for all the r_i to be probed in these sets: the sets $\mathcal{Q}^x$, $\mathcal{Q}^y$ and $\mathcal{O}^y$ can be a partition of $[\![0, \lfloor d/2 \rfloor[\![$ (otherwise some r_{i^*} is not observed in the $\mathbf{y}$ part). If O^y is empty, there are $\lfloor d/2 \rfloor$ internal probes in $Q^x \cup Q^y$, therefore the simulation is trivial: O_2 can be empty. Otherwise, since all the $d-1$ probes are in O, Q^x or Q^y, P^y is empty, thus $P^y_{O_2} = P^y_O$. Let $\mathbf{y}^2 i^* \in O^y$, we know that $i^* \neq \lfloor d/2 \rfloor$ (otherwise $\mathcal{Q}^x$, $\mathcal{Q}^y$ and $\mathcal{O}^y$ would not be a partition of $[\![0, \lfloor d/2 \rfloor[\![$). Furthermore, the only way r_{i^*} is observed through the probes is through $\mathbf{y}^2 i^*$ and $\mathbf{x}^2 i^*$. Since $\left|\mathcal{Q}^y \cup \mathcal{O}^y\right| \leq \lceil d/2 \rceil - 1 - \left|P^y\right|$, $\mathbf{y}^0_{i^*} \in P^y_O = P^y_{O_2}$ is simulated as a fresh random by the SOI simulator for

the second recursive call, and therefore $\mathbf{y}^2_{i^*} = \mathbf{y}^0_{i^*} + r_{i^*}$ is independent of r_{i^*}. The probe $\mathbf{x}^2_{i^*}$ can thus be simulated as a fresh random. The proof is then identical to the even case. Finally, we remark that this discussion is not needed if $\left|O^y\right| = \lceil d/2 \rceil$, since then there are then at most $\lfloor d/2 \rfloor - 1$ other probes. □

4.3.4 Threshold implementations

The HPC constructions present the drawback of using a large amount of fresh randomness. For first-order security, the *threshold implementations* (TI) [NRR06] significantly reduce (and sometimes nullify) these randomness requirements. TIs exploit a key property of first-order security: since there is only one probe, only a single gadget can be probed by the adversary. Therefore, the analysis of composition can be reduced to the analysis of individual gadgets under two conditions: (i) no glitches propagate through multiple gadgets, and (ii) every input sharing of a gadget is uniform.

Condition (i) is achieved by requiring registers on all the output shares of every gadget, while condition (ii) is satisfied by providing uniform sharings as inputs of the circuit and requiring that every gadget is *uniform*. Informally, a gadget is uniform if, for any fixed unshared value for its inputs, a jointly uniform distribution for the input sharings induces a jointly uniform output sharing distribution (both input and output distributions are conditioned on the unshared values being correct). Such uniformity does not necessarily require usage of randomness in the gadgets, a simple example is the addition gadget: if both input sharings are uniform, the sharewise addition is uniform too. Achieving uniformity without using randomness is however not always possible without using a very large number of shares, which may reduce performance.

In addition to composition conditions, the security of each gadget has to be ensured. In TIs this translates into the *non-completeness* requirement for gadgets: no wire in the circuit can be connected (through logic gates) to all the shares of an input sharing. Non-completeness implies glitch-robust probing security of a single gadget.

Threshold implementations for many Boolean functions and cryptographic primitives have been designed over the years, including (but not limited to) the Noekeon [NRS11], PRESENT [Pos+11], Keccak [Bil+13] and AES [Mor+11].

Higher-order The basic principles of TIs do not trivially extend beyond first-order security [Rep15]. However, such an extension has been recently introduced and proven secure in [DN22] for the second order. This work

replace the non-completeness condition with the natural extension: t-order non-completeness (any set of t probes may depend on at most $d-1$ shares of an input sharing). Moreover, the uniformity is replaced with the notion of resilient uniformity, which is reached by refreshing the output of the non-linear gadgets. Despite the randomness usage due to this refreshing, [DN22] introduces a glitch-robust second order PRESENT implementation with lower randomness requirements than HPC constructions.

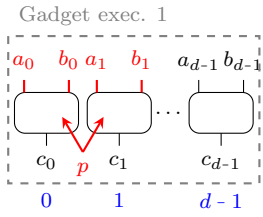
4.4 Transition-robust composition

Let us now consider the transition-robust probing model. We discuss informally the challenges and solutions for composition in this model before giving the formal results.

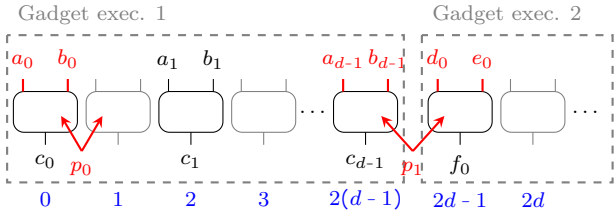
4.4.1 Transitions: problems and solutions

At first, it may seem that the transitions can be handled similarly to glitches in the PINI composition strategy. Indeed, since we model a transition-leaking probe as giving knowledge of the value carried on a wire at two consecutive cycles, a transition-leaking probe in a share-isolating gadget gives only knowledge about one circuit share. Similarly to glitch-robust PINI, we can define transition-robust PINI gadgets as gadgets that can be simulated like PINI gadgets even when there are transition probes in the gadget, leaving the transitions that span multiple gadgets to be handled by the composition theorem. Transition-robust PINI gadgets are however not trivially composable for two reasons that we explain next.

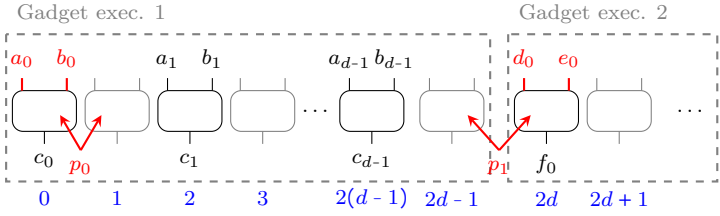
The first reason comes from the difference between the structural (i.e., physical) gates/wires in a gadget, and their algorithmic (i.e., logical) position in the computations performed by the gadget. Indeed, the same physical gadget (a set of physical gates and wires) can be *executed* multiple times with different inputs. Let us assume that a single physical gate appears in different logical positions for two executions of a gadget and there is a transition probe that touches this gate over the two executions. If this can happen without restriction, the two probes are at somewhat independent positions, leading to an effective doubling of the number of probes. An example for this (illustrated in Figure 4.2) would be an implementation of a linear gadget that intends to minimize resource utilization by applying the operation iteratively for each share with a purely combinational circuit. If the processing of each share



(a) Transition vulnerable

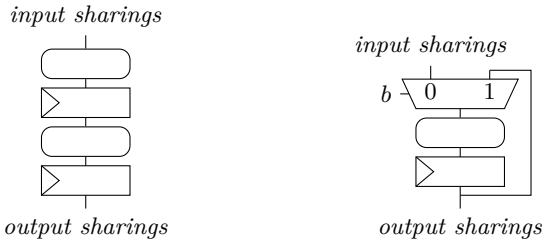


(b) Transition-robust, not iterated transition-robust



(c) Iterated transition-robust

Figure 4.2: Serial implementation (clock cycle is indicated in blue) of a 2-input linear gadget: the same gates are used sequentially for all the shares. Each double arrow indicates a transition-extended probe and red inputs are required for simulation.

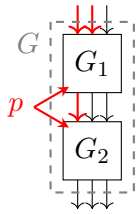


(a) Pipeline gadget **(b) Non-pipeline gadget**

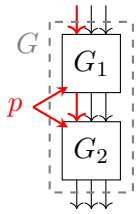
Figure 4.3: Gadgets implementing the same function in two clock cycles. Rounded boxes represent combinational logic, separated by registers. Gadget (a) is pipeline gadget while gadget (b) is not pipeline ($b = 0$ at the first execution cycle and $b = 1$ at the second cycle).

takes one cycle (Figure 4.2a), the gadget is not transition-robust: a transition-extended probe on an input wire leaks two shares from the same input sharing, reducing the security order. However, if a cycle with a non-sensitive (e.g., constant) input is interspersed between the two cycles processing those shares (Figure 4.2b), then the gadget is transition-robust PINI. Under this property, it is however still allowed to start a second execution immediately after the first one, even when this leads to transition leakage (Figure 4.2c). This transition leakage (across two different shares) may break the security if the input sharings for the two executions of the gadget are not independent (and, in order to achieve generic composition, we cannot assume that the input sharings are independent). A solution to this issue is the *iterated transition-robust* property, which cares about multiple executions of a gadget and requires that when a transition across two executions exists, it appears to probe twice the same logical gate (from an external simulation point of view). This constraint is satisfied by many gadgets, such as the ones built as a pipeline, that is, when each structural gate is used once per execution, as shown in Figure 4.3. Furthermore, structural gates and wires can be used in more than one structural gadget. Therefore, a single transition-extended probe could cover executions of two distinct structural gadgets. Such a probe is difficult to analyze with composition theorems, hence we prevent it from existing by mandating a "clearing" sensitive state in-between the use of a structural gate or wire by two distinct structural gadgets.

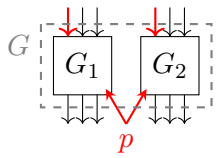
The second reason is an issue that appears when there is a transition between two executions of an iterated transition-robust PINI gadget and additionally an output of the first execution is an input of the second one (Figure 4.4a). Assuming that such a transition-extended probe is the



(a) Serial composition of two iterated transition-robust PINI gadget executions.



(b) Serial composition of two iterated transition-robust O-PINI gadget executions.



(c) Parallel composition of iterated transition-robust PINI gadget executions.

Figure 4.4: Composition of iterated transition-robust PINI and O-PINI gadget executions with $d = 3$ shares and one transition-extended probe (inputs required for simulations in red). Composition 4.4a is insecure, while 4.4b and 4.4c are secure.

only probe, simulating the second execution requires one share of each of its input sharings. For the first execution, an output share has thus to be simulated in addition to the internal probe. Hence, the simulator may require two shares of each input sharing. This shows that such a composition is not transition-robust PINI (which could then be exploited to break its probing security).

The intuition to fix this problem comes again from share-isolating gadgets, for which the previous example does not have a problem: we have the guarantee that for the first execution, the internal probe and the output shares to simulate are in the same circuit share. Only one share of each input is therefore required for simulation. The PINI property mimics the share isolation property at the input and output levels, which is why it composes smoothly. However, transitions do not only touch the inputs and outputs, but also the internals of the gadget. In the internals, PINI is counting probes, and not circuit shares, which is insufficient to capture all transitions. We therefore need a property that is even closer to share isolation than PINI, while still allowing to mix circuit shares internally (otherwise some gadgets are impossible to implement). A key observation is that for share isolating gadgets, if the simulator knows the inputs in a given circuit share, then it is able to simulate all outputs in that circuit share in addition to the probes (see Figure 4.5 for illustration). This property is not necessarily satisfied by PINI gadgets, and we name O-PINI (output-probe isolating non-interferent) the PINI gadgets that, given a set of probes, and knowing a set of input shares that satisfies the PINI constraints, can simulate jointly both the probes and their output shares for which they know the input shares with the same index. An (iterated transition-robust) O-PINI gadget solves our

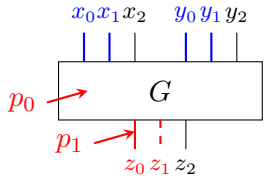


Figure 4.5: O-PINI gadget. There is one output probe on z_0 and one internal probe p (in red). The simulator has thus access to input shares with two distinct indexes, one of them being index 0 (in blue). Since the simulator has access to input shares with index 1, it must simulate the output share with index 1 (in dashed red).

example (Figure 4.4b): the simulator is asked to simulate the internal probes in both executions, hence requires the inputs with the same share index for both executions. Since it also simulates the outputs with that share index for the first execution, the second gadget can be simulated, and all of this requires only knowledge of inputs in one circuit share.

Another solution, that allows using more efficient gadgets, is to prevent cases similar to the example from happening by constraining how gadgets are composed. The constraint is that adjacent gadgets (i.e., gadgets that a transition-extended probe may cover) can be in a parallel composition (e.g., Figure 4.4c), but not in a sequential composition (e.g., Figure 4.4a). This strategy is more efficient as it allows using PINI gadgets instead of O-PINI ones, at the cost of not being a trivial composition strategy. The parallelism requirements results in a more complex security analysis (it must be verified), and restricts the structure of the circuit. However, we will show that a common way of implementing substitution-permutation networks (SPNs), which are arguably the most commonly masked algorithms, satisfies the parallelism assumption.

4.4.2 Transition-robust simulation

Let us now formalize the discussion of the previous section. We first define simulatability with transitions, following the pattern of simulatability for glitches: transitions that happen inside a gadget must be simulated, but those that happen outside the gadget (i.e., in other executions) are not considered: they will be taken into account in the composition theorems.

Definition 34 (Transition-robust simulatability.)**.** *A set of extended probes P in a gadget execution G can be transition-robustly simulated with a set of input shares $\mathcal{I}'$ if P' can be simulated with the inputs shares $\mathcal{I}'$, where P' is the restriction of the transition-expansion of P to probes in G (i.e., the expanded probes outside G are discarded).*

The notions of transition-robust NI, SNI, PINI... follow from this definition. The next step towards a composition theorem with transitions is analyzing the behavior of transitions that cross gadgets. However, as discussed in Section 4.4.1, transition-robust simulatability is not enough for secure composition, due to transition-extended probes that cover multiple gadgets.

This could be solved by forcing the two expanded probes from a single transition-extended probe to be "at the same place" in their respective gadget executions. Instead of imposing such a constraint on the structure of the gadget, we opt for a simulation-based definition: given a set of probes in executions of a structural gadget, we simulate each execution and take the unions of all the sets of shares required for the simulations.

Definition 35 (Iterated transition-robust simulatability.)**.** *Let* G *be a structural gadget whose executions are* $(\mathsf{G}_i)_{i=0,\dots,n-1}$*, P be a set of transition-extended probes in* G *and P_i be the set of probes of the expansion of P that belong to* G_i*. Moreover, let $\mathcal{I}_i$ be a set of input shares of* G_i*, and $\mathcal{I}_i'$ be its translation to the canonical execution* G_{i^*}*. P is iterated transition-robust simulatable by* $\bigcup_{i=0}^{n} \mathcal{I}_i'$ *in* G *if P_i can be simulated in* G_i *by $\mathcal{I}_i$ for all i.*

As previously, the all non-interference definitions can be adapted to the iterated transition-robust settings. Let us now look at the particular case of gadgets that are a simple pipeline, where iterated transition-robust simulatability reduces to simulatability of a single execution since there are no transitions inside a gadget and probes are always at the same position across gadgets. This requires that all executions are functionally identical.

Definition 36 (Pipeline)**.** *A structural gadget is pipeline if its canonical execution* G *uses each of its structural wires and gates only once: for a gate g (resp. wire w), there exists no $t_1 \neq t_2$ such that* $(g, t_1), (g, t_2) \in \mathsf{G}$ *(resp.* $(w, t_1), (w, t_2) \in \mathsf{G}$*).*

Lemma 6. *Let* G *be a pipeline structural gadget whose n executions* $(\mathsf{G}_i)_{i=0,\dots,n-1}$ *all have the same values for the public parameters and let* G_{i^*} *be its canonical execution. Let P be a set of transition-extended probes in the executions of* G*, and P' be the translation to* G_{i^*} *of its expansion. If P' can be simulated in* G_{i^*} *using its input shares $\mathcal{I}'$, then P is iterated transition-robust simulatable in* G *with $\mathcal{I}'$.*

Proof. We first observe that the translation G_{i^*} of the expansion of P is non-ambiguous since each structural gate and wire used in any G_i appears exactly once in G_{i^*}. Next, since all G_i have the same values for

the public parameters, they are functionally identical: the translation of P' to G_i can be simulated using the translation of $\mathcal{I}'$ to G_i. We conclude the proof by remarking that every probe of the expansion of P that belongs to G_i also belongs to the translation of P' to G_i. □

This lemma implies that if an execution of a pipeline structural gadget without public parameters satisfy a simulation-based property (such as NI, SNI, PINI), then the corresponding structural gadget satisfies the iterated transition-robust variant of the property.

Finally, we deal with transitions that cover multiple structural gadgets with the adjacency definition. Under this definition, if two structural gadgets have no adjacent execution, then the expansion of a single transition-extended probe cannot probe both gadgets.

Definition 37 (Adjacent gadgets)**.** *Two gadget executions are adjacent if there exists a wire or gate (w, t) in one of them such that $(w, t+1)$ is in the other gadget. Two structural gadgets have adjacent executions if at least one execution of the first structural gadget is adjacent to one execution of the other structural gadgets.*

4.4.3 Trivial transition-robust composition

The following O-PINI property captures the requirement explained in Section 4.4.1 to serially compose circuits with transitions trivially: when the simulator has access to an input share index, it should simulate the outputs with the same share index. O-PINI implies PINI, but not the other way around, as shown on page 86.

Definition 38 (Output Probe-Isolating Non-Interference)**.** *Given a d-shares gadget G, let P be a set of at most t probes on its internal wires and $A \subseteq [\![0, d[\![$. The gadget G is t-O-PINI iff for all P and A such that $|P| + |A| \leq t$ there exist a set $B \subseteq [\![0, d[\![$, $|B| \leq t$ such that the probes P and all the output shares of G with index in $A \cup B$ can be simulated by the input shares of G with index in $A \cup B$. We say that a gadget is O-PINI if it is t-O-PINI for any t.*[5]

First, we observe that share-isolating gadgets are O-PINI.

Proposition 26. *Share-isolating structural gadgets are iterated transition-robust O-PINI.*

[5] Similarly to PINI, if a gadget with d shares is $d-1$-OPINI, then it is t-OPINI for any t.

Proof. Let G be a share-isolating structural gadget. Let P a set of probes and $A \subseteq [\![0, d-1]\!]$. Considering a partition of the set of structural gates and wires in G according to share index, $(P_i)_{i=0,\dots,d-1}$ be a partition of P according to the circuit share they are in. At most $|P|$ sets in the partition are non-empty, let B be the set of indexes of these sets. All probes in P and outputs with share index in $A \cup B$ can then be simulated using all the inputs with share index in $A \cup B$ since there is no path between the other inputs and the values to simulate. □

We next prove that any composite gadget made of iterated transition-robust t-O-PINI gadgets is iterated transition-robust t-O-PINI. Since we have to consider all executions of a gadget at once, we provide a general all-in-one composition theorem instead of building it from parallel and sequential composition of two gadgets as in Section 3.2.2 or Section 4.3.

Theorem 2 (O-PINI composability)**.** *Let $(S_i)_{i=0,\dots,n-1}$ be a set of iterated transition-robust t-O-PINI structural gadgets that have no adjacent executions. If the structural gadget S is a structural composition of the gadgets S_i, then it is iterated transition robust t-O-PINI.*

Proof. We build an iterated transition-robust t-O-PINI simulator that takes as input a set of internal probes P, and a set of share indices of probed output shares A. Wlog, we assume that there is no probe on wires connecting gadgets: these can be considered as part of a gadget connected to the wire.

Let $(P_i)_i$ be a partition of P according the structural gadget S_i to which they belong (it is well-defined thanks to the non-adjacency hypothesis). Let $A_i^0 = A$ and $B_i^0 = \varnothing$ for each gadget S_i. Then, the following algorithm is run for $k = 1, 2, \dots, k^*$, until it reaches a fixed point (i.e., $A_i^{k^*} = A_i^{k^*-1}$ for all i): using the iterated transition-robust O-PINI simulator for each S_i (for output shares A_i^k and internal probes P_i), the set of input shares B_i^k is computed, and each A_i^{k+1} is computed as follows: $A_i^{k+1} \leftarrow A \cup \bigcup_{j \neq i} B_j^k$. Let $A^* = A \cup \bigcup_i B_i^{k^*}$.

Let us observe that at the final iteration, each simulator takes input shares with index in A^* as input, simulates its internal probes that are in P, and simulates all its output shares with index in A^* for every gadget execution. The set of input shares required for simulation is A and $B = \bigcup_i B_i^{k^*} \supset A^* \setminus A$. The overall simulation can therefore be achieved by using simulation of parallel and sequential gadget composition (Propositions 2 and 3) on the extended sequential composition corresponding to any execution of S: all expanded probes and input/output shares with index in A^* can be simulated.

We first prove that the set construction algorithm reaches a fixed point: if for all i $A_i^k \subseteq A_i^{k+1}$, then $B_i^{k+1} \subseteq B_i^{k+2}$ (if the simulator of S_i requires the minimal input set, which can be assumed wlog), which implies that $A_i^{k+1} \subseteq A_i^{k+2}$ for all i. Since by construction $A = A_i^0 \subseteq A_i^1$, it follows by induction that for all i and k, $A_i^k \subseteq A_i^{k+1}$. Since all A_i^k have cardinality at most d, the algorithm reaches a fixed point.

Next, we prove that the simulation algorithm is correct, that is, all the values it generates have the same distribution as in the true circuit. This is guaranteed by using a correct simulator for each sub-gadget, and feeding it with correct inputs: by induction (starting with the inputs it receives), the simulator knows for each gadget the input shares with index in A^* before simulating the gadget. Therefore, the simulator for the gadget can simulate correctly the probes inside the gadget and the output shares with index in A^*.

Finally, we observe that the simulator simulates all required values: it simulates the transition-expansion of P, thanks to the iterated transition-robust simulators of the structural gadgets S_i that can simulate the expansion of P_i (and of the outputs with index in $A_i^{k^*}$) in all the executions with the input shares with index A^*. Furthermore, the simulator respects the cardinality constraint $|B| \leq |P|$: $B = \bigcup_i B_i^{k^*}$ and $P = \bigcup_i P_i$, while $\left|B_i^*\right| \leq |P_i|$ for all i (thanks to each gadget O-PINI simulator). □

Multiplication gadget. To instantiate the previous general construction, we build an iterated transition-robust O-PINI multiplication gadget in $\mathbb{F}_q$. We build the O-PINI1 multiplication gadget (Algorithm 24) from the PINI1 multiplication gadget[6] and the observation that the ISW multiplication gadget (on which it is based) is SNI (in the probing model). Intuitively, the SNI property is relevant since it guarantees that outputs are somewhat "easy" to simulate. However, it is not sufficient and we have to add a 0-sharing to the output shares.

As an example of why the PINI1 gadget (Algorithm 4) is not O-PINI, we take $d = 3$ and consider probes $(\mathbf{x}_0 + 1)r_{01}$ and $(\mathbf{x}_2 + 1)r_{20}$, the simulator has to know $\mathbf{x}_0$, $\mathbf{x}_2$, r_{01} and r_{20} (at least sometimes). Hence, it needs the input shares with index 0 and 2. It can however not simulate the output $\mathbf{z}_0$, whose value is $\mathbf{x}_0(\mathbf{y}_0 + \mathbf{y}_1 + \mathbf{y}_2) + r_{01} + r_{02}$: it does not know $\mathbf{y}_0$, and it outputs r_{01} and $r_{20} = -r_{02}$, thus those are known by the distinguisher.

Remark. We describe the gadget in an algorithmic way for the sake of

[6] We may also insert any register in the gadget, e.g., giving HPC2, since they only impact the propagation of glitches, which we don't analyze here.

readability, whereas formally we describe patterns for structural gadgets: a set of gates and wires, and a way to generate gadget executions out of them. The algorithmic description is not ambiguous: it can be mapped to the structural gates and wires by unrolling the loops in the algorithm and adding a gate for each operation (and connecting the wires accordingly). Then, since the gadgets described in this way are all pipeline, there is only one way (up to a translation) of executing them such that the set of inputs matches the algorithmic description.

Algorithm 24 O-PINI1 multiplication gadget with d shares.

Input: Sharings $\mathbf{x}$ and $\mathbf{y}$.
Output: Sharings $\mathbf{z}$, such that $z = xy$.

1: $\mathbf{w} \leftarrow \mathsf{PINIMul}(\mathbf{x}, \mathbf{y})$ ▷ PINI1, HPC1 or HPC2 multiplication.
2: **for** $i = 0$ to $d - 2$ **do**
3: $\quad \mathbf{s}_i \xleftarrow{\$} \mathbb{F}_q$
4: $s_{d-1} \leftarrow -\sum_{i=0}^{d-2} s_i$
5: $\mathbf{z} \leftarrow \mathbf{w} + \mathbf{s}$

Proposition 27. *The gadget O-PINI1 is iterated transition-robust t-O-PINI for $t < d$.*

Proof. The proof relies on the following intuition: if there is no probe in the sum of the s_i variables, then simply using the PINI simulator for the multiplication, then using a fresh random to simulate any missing output works. Otherwise, there is at most $d - 2$ other probes, and, since $d - 1$ randoms are used to compute each share $\mathbf{w}_i$, not all of them can be probed. For the sake of brevity, we do not give the full proof and remark that the proof that the O-PINI2 gadget is glitch+transition-robust O-PINI applies here, if glitches and registers are ignored. □

4.4.4 Optimized transition-robust composition

In this section, we analyze the particular case where the executions $\mathsf{G}_0, \ldots, \mathsf{G}_{n-1}$ of a structural gadget can be written as a parallel composition $\mathsf{G}_0 \| \ldots \mathsf{G}_{n-1}$ (forming a *parallel gadget*). We show that when each group of executions of an instance across which there are transitions forms a parallel gadget, O-PINI gadgets are not required and PINI is actually enough. This particular case is of interest in practical implementations of block ciphers, when the same structural gadget is used multiple times, for example to compute sequentially multiple parallel S-boxes. Such an architecture is known as "S-box serial", where *serial* refers to sequential

executions, and should not be confused with the distinct notion of serial composition.

Proposition 28 (Parallel PINI executions)**.** *Let S be a structural gadget whose executions from a parallel gadget. If S is iterated transition-robust t-PINI, then the structural gadget whose only execution is the parallel gadget is iterated transition-robust t-PINI.*

Proof. First, since there is only one execution, iterated transition-robustness is equivalent to transition-robustness, that we prove next. Let A be the set of shares indexes for which simulation outputs are required and let P be the set of internal probes. There exists a set of share indexes B of size at most $|P|$ such that, for each execution G_i of S, the iterated transition-robust t-PINI simulator of S can simulate a set made of the output shares of G_i with index in A and the elements of the transition-expansion of P that are in G_i. □

We are interested in a full block cipher, meaning that there are multiple rounds, and therefore not all the S-boxes are parallel executions. We solve this by treating the S-boxes of each round as a separate structural gadget, which intuitively solves the problem of serial composition. This does not come without constraint: since the structural gadgets must be non-adjacent, we must leave an "empty" cycle in the S-box between two rounds.

This leads to the next composition theorem which allows PINI gadgets in a transition-robust composition.

Theorem 3. *Let S be a structural composition of iterated transition-robust t-O-PINI gadgets and transition-robust t-PINI gadgets. Assuming that S has a single execution G and none of the structural composing gadgets have adjacent executions, if every PINI gadget in S has a single execution, then G is transition-robust t-PINI.*

Proof. We build a transition-robust t-PINI simulator for a set of internal probes P and a set of share indices of probed output shares A.

Let $(\mathsf{G}_i)_{i=0,\dots,n-1}$ be the gadgets executions is S, in reverse topological order in the gadget composition graph (i.e., from output to input). Let $(S_j)_{j=0,\dots,n'-1}$ be the set of structural composing gadgets, and let $s(i) = j$ if G_i is an execution of S_j. Let us partition the set of probes according to the structural composing gadgets P_j is the set of probes in S_j (this can be done thanks to non-adjacency), and let P_i' be the probes in the transition-expansion of $P_{s(i)}$ that belong to G_i.

Let $A^0 = \mathrm{A}, A_j^0 = A$ and $B_j^0 \leftarrow \varnothing$ for all $j \in [\![0, n'[\![$. Then, for every i from 0 to $n-1$, the simulator proceeds as follows. Using the PINI or

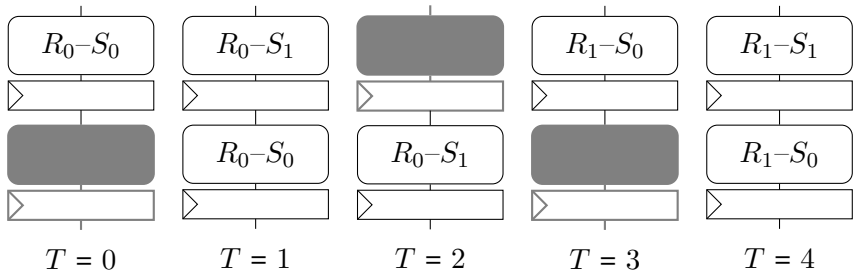


Figure 4.6: 2-cycle pipeline S-box in a dummy SPN with 2 S-boxes per round (assuming the linear layer is combinational, i.e., takes zero cycles). At the first three cycles, the computations for round 0 (R_0) are executed: the two S-boxes (R_0–S_0 and R_0–S_1) go through the pipeline. In two out of three cycles, one of the pipeline stages is not used (i.e., does not contain sensitive data). The same goes on for the three next clock cycles computing the second round (R_1). The computation of the S-box R_1–S_0 cannot start at cycle $T = 2$ since the output of the first round is not yet computed. The S-boxes of distinct rounds are non-adjacent thanks to this "bubble" (colored in gray) in the pipeline, hence transitions only touch parallel S-boxes.

O-PINI simulator for $S_{s(i)}$ (for output shares with index in $A_{s(i)}$ and internal probes P_i'), the set of input shares $B^{i+1}_{s(i)}$ is computed. For all $j \neq s(i)$, let $B^{i+1}_j = B^i_j$, and for all j, let $A^{i+1}_j = A \cup \bigcup_{j'=j} B^{i+1}_{j'}$, and let $A^{i+1} = A \cup \bigcup_j B^{i+1}_j$. Finally, the simulator requires the input set of shares $B = \bigcup_j B^n_j$.

Let us remark that for all i, the output shares of G_i with index in $A^i = A^i_{s(i)} \cup B^i_{s(i)}$ and the probes in P_i can be simulated using the input shares with index in $A^{i+1} = A^{i+1}_{s(i)} \cup B^{i+1}_{s(i)}$. For O-PINI gadgets, this is a direct consequence of the iterated transition-robust O-PINI definition, and for PINI gadgets, $B^i_{s(i)} = \varnothing$ since there is a single execution, hence the claim follows from the transition-robust PINI definition. As a result, the overall simulation can be achieved by using simulation of parallel and sequential gadget composition (Propositions 2 and 3) on the extended sequential composition corresponding to the G.

Finally, we remark that by construction, $\left|B^i_j\right| \leq \left|P_j\right|$, therefore the simulator respects the cardinality constraint $\left|B\right| \leq \left|P\right|$. □

SPN implementation

We conclude this section by discussing the transition-robustness of the common *S-box serial* substitution-permutation network (SPN) implementation. This architecture iterates the executions of a round, where one

round is made of some hardware implementing the permutation (linear) layer, and some S-boxes (which may be used sequentially multiple times per round). If the S-box is pipeline and PINI, Proposition 28 implies that the structural gadget made of all its usages for one round is iterated transition-robust PINI. If furthermore all the other gadgets (the linear layer and the key addition) are share-isolating and there is one cycle where no sensitive data is input to the S-boxes (e.g., there is a bubble in the S-box pipeline, see Figure 4.6), the S-boxes of each round can be analyzed as a distinct structural gadget that is not adjacent to any other gadget. Theorem 3 therefore implies that the whole SPN implementation is transition-robust PINI (hence probing secure in the transition-robust probing model).

The block cipher implementations of [Cas+21b] are based on a trivial glitch-robust PINI composition (using HPC1 or HPC2 multiplications and share-isolating gadgets), have pipeline S-boxes and correspond to the above architecture description. These implementations are thus both transition-robust and glitch-robust probing secure.

4.5 Glitch+transition-robust composition

Let us now adapt our constructions to the GT-robust probing model. Since the transition-robust composition theorems are based on parallel and sequential gadget composition simulation (Propositions 2 and 3), they can easily be adapted to consider glitches, using gadget composition glitch-robust simulation instead (Propositions 18 and 19).

We first formally define (iterated) glitch+transition simulatability.

Definition 39 (Glitch+transition-robust simulatability.)**.** *A set of extended probes P in a gadget G can be glitch+transition-robustly simulated by a set of input shares $\mathcal{I}'$ if P' can be glitch-robustly simulated with the input shares $\mathcal{I}'$, where P' is the restriction of the transition-expansion of P to probes in G.*

Definition 40 (Iterated GT-robust simulatability.)**.** *Let G be a structural gadget whose executions are $(\mathsf{G}_i)_{i=0,\dots,n-1}$, P be a set of glitch+transition-extended probes in G and P_i be the set of probes of the transition-expansion of P that belong to G_i. Moreover, let $\mathcal{I}_i$ be a set of input shares of G_i, and $\mathcal{I}_i'$ be its translation to the canonical execution G_{i^*}. P is iterated glitch+transition-robust simulatable by $\bigcup_{i=0}^{n} \mathcal{I}_i'$ in G if P_i can be glitch-robustly simulated in G_i by $\mathcal{I}_i$ for all i.*

Since these definitions are very close to (iterated) transition-robust simulatability, the composition theorems can be straightforwardly adapted.

Let us start with sharewise gadgets.

Proposition 29. *Share-isolating structural gadgets are iterated GT-robust O-PINI.*

Proof. The proof is identical to the proof of Proposition 26, since the glitches cannot propagate if there is no connection between circuit shares. □

4.5.1 Trivial composition

The adaptation of Theorem 2 to add security against glitches is straightforward.

Corollary 4. *Let $(S_i)_{i=0\ldots,n-1}$ be a set of iterated GT-robust t-O-PINI structural gadgets that have no adjacent executions. If the structural gadget S is a structural composition of the gadgets S_i, then it is iterated GT-robust t-O-PINI.*

Proof. The proof is the same as the one of Theorem 2, except that simulatability is replaced with glitch-robust simulatability, hence the proof relies on Propositions 18, 19 and 29. □

Let us now introduce the multiplication gadget O-PINI2 in Algorithm 25, and prove that it is iterated GT-robust O-PINI.

Algorithm 25 O-PINI2 multiplication gadget with d shares.

Input: Sharings $\mathbf{x}$ and $\mathbf{y}$.
Output: Sharings $\mathbf{z}$, such that $z = xy$.

1: $\mathbf{w} \leftarrow \mathsf{HPCMul}(\mathbf{x}, \mathbf{y})$ ▷ HPC1 or HPC2 multiplication.
2: **for** $i = 0$ to $d - 2$ **do**
3: $\quad \mathbf{s}_i \xleftarrow{\$} \mathbb{F}_q$
4: $s_{d-1} \leftarrow -\sum_{i=0}^{d-2} s_i$
5: $\mathbf{z} \leftarrow \mathsf{Reg}(\mathbf{w} + \mathsf{Reg}(\mathbf{s}))$

Proposition 30. *The gadget O-PINI2 (Algorithm 25) is iterated GT-robust t-O-PINI for $t < d$, when HPCMul is HPC1 or HPC2.*

Proof. First, we observe that O-PINI2 is pipeline, and it has no public parameter, hence using a variant of Lemma 6 for glitch-robust simulatability (which does not change its proof), we only have to prove that one execution of it is glitch-robust O-PINI.

We build a simulator based on the glitch-robust PINI simulator of HPCmul. The simulation of probes is delegated to the HPCmul simulator,

except $\mathbf{w}_i + \mathsf{Reg}(\mathbf{s}_i)$ probes which are given as $\mathbf{w}_i$ probes to the HPC2 simulator, and the $\mathbf{s}_{d-1}$ probe which is generated as specified by the gadget. Other probes are strictly less powerful and are thus ignored wlog. Finally, simulation of outputs $\mathbf{z}_i$ where i belongs to the set $\mathcal{I}'$ of input share indexes required is done as follows: if all intermediate variables in $\mathbf{z}_i$ have already been simulated by the HPCmul simulator, perform as specified by the gadget. Otherwise, generate a fresh random variable.

We now discuss correctness of the simulation of $\mathbf{z}_i$'s (correctness for s_i's is trivial and for other variables it is a direct consequence of the correctness of the HPCmul simulator).

If there is a probe on $\mathbf{s}_{d-1}$, then there are at most $d-2$ other probes. In that case, in the sum computing $\mathbf{z}_i$, either $\mathbf{w}_i$ or $\mathbf{w}_i + \mathsf{Reg}(\mathbf{s}_i)$ is probed (and simulation of $\mathbf{z}_i$ is trivially correct), or at most $d-2$ random r_{ij} (from the execution of HPCmul) are observed, since no probe except $\mathbf{w}_i$ and $\mathbf{w}_i + \mathsf{Reg}(\mathbf{s}_i)$ can observe more than one r_{ij} at once. Therefore, at least one r_{ij} appearing as a term in the $\mathbf{z}_i$ expression is fresh (i.e., not observed except through $\mathbf{z}_i$), thus simulation is correct. Otherwise, if $\mathbf{w}_i$ nor $\mathbf{w}_i + \mathsf{Reg}(\mathbf{s}_i)$ is probed, $\mathbf{s}_i$ is fresh and therefore simulation is correct. □

Based on the O-PINI2 multiplication and sharewise gadgets, any arithmetic circuit can be securely masked w.r.t. the GT-robust probing model. This however comes at the cost of a multiplication gadget with increased latency (by one cycle) and randomness usage (by $d-1$ random elements) over the HPC1/HPC2 masking schemes.

4.5.2 Optimized composition

Theorem 3 can be adapted to the GT-robust setting.

Corollary 5. *Let S be a structural gadget composition of iterated GT-robust t-O-PINI gadgets and GT-robust t-PINI gadgets. Assuming that S has a single execution G and none of the structural composing gadgets have adjacent executions, if every PINI gadget in S has a single execution, then G is GT-robust t-PINI.*

Proof. The proof is the same as the one of Theorem 3, except that simulatability is replaced with glitch-robust simulatability, hence the proof relies on Propositions 18, 19 and 29. □

Furthermore, the HPC1 and HPC2 gadgets are GT-robust iterated PINI since they are glitch-robust PINI and pipeline. We can adapt Proposition 28 to the GT-robust setting.

Table 4.2: Performance and security characteristics of various glitch-robust multiplication gadgets.

Gadget	Security	Randomness	Add. & Sub.	Mul.	Reg.*	Latency
Alg. 19 (DOM) [GMK16a]	NI	$\frac{d(d-1)}{2}$	$2d(d-1)$	d^2	d^2	1 & 1
Alg. 19 [Fau+18]	SNI	$\frac{d(d-1)}{2}$	$2d(d-1)$	d^2	d^2+d	2 & 2
Alg. 20 [Fau+18]	PINI	$d(d-1)$	$3d(d-1)$	d^2	d^2+6d-1	2 & 4
Alg. 21 (HPC1)†	PINI	$\frac{d}{2}(d-1+\log_2 d)$	$2d(d-1)+d\log_2 d$	d^2	$d^2+d\log_2 d$	1 & 2
Alg. 18 (HPC2)	PINI	$\frac{d(d-1)}{2}$	$4d(d-1)$	$2d^2$	$\frac{7}{2}d^2-\frac{3}{2}d$	1 & 2
Alg. 25 (O-PINI2)	O-PINI	$+(d-1)$‡	$+(2d-2)$‡	$+0$‡	$+d$‡	2 & 3

* Including synchronization registers.
† Cost formulas are given for power-of-2 d (and $d \neq 1$).
‡ Additional cost on top of the HPC1 or HPC2 instance.

Proposition 31 (Parallel PINI executions)**.** *Let S be a structural gadget whose executions from a parallel gadget. If S is iterated GT-robust t-PINI, then the structural gadget whose only execution is the parallel gadget is iterated GT-robust t-PINI.*

Proof. The proof is identical to the proof of Proposition 31, where glitch-robust simulation is used instead of simulation in the threshold probing model. □

4.5.3 Block cipher implementations

Thanks to the similarity between the optimized transition-robust composition results in the GT-robust ones, the SPN implementation discussed on page 89 can be used to build GT-robust probing-secure implementations of block ciphers, provided that pipeline glitch-robust S-boxes are used.

From the cost of the various glitch-robust multiplication gadgets (see Table 4.2), we can see that the HPC1 and HPC2 gadgets enjoy better latency than the other arbitrarily composable gadgets. The difference in cost between these gadgets and the O-PINI2 gadget shows that using the optimized composition strategy of Section 4.5.2 over the trivial composition strategy of Section 4.5.1 can bring significant performance gains. Regarding the comparison of HPC1 and HPC2 when working in $\mathbb{F}_2$ (otherwise HPC2 cannot be used), HPC2 has a lower randomness usage but higher logic gate utilization than HPC1. Besides, HPC2 has the advantage of simplicity, since its structure is the same for all values of d, while the refresh in HPC1 has a varying number of pipeline stages, and therefore complex requirements on the input randomness timing.

Based on these considerations, we built multiple block cipher imple-

mentations: PRESENT-128 (one architecture with various serialization levels) [Cas+21b], Clyde-128 (one architecture with multiple serialization levels) [MCS22b], and AES-128 (three architectures: serialized 8-bit and 32-bit implementations, and a round-based one) [MCS22a]. All implementations are generic with respect to the number of shares and are based on the HPC2 gadget (the PRESENT implementation can also use the HPC1 gadget, and the overall logic overhead of using HPC2 over HPC1 is at most 10 % on the full block cipher for $d \leq 8$). All the implementations have been verified by the fullVerif tool (see Section 4.7).

4.6 Software implementations

In addition to hardware implementations introduced in Section 4.1, masking is also applied to software implementations of cryptographic algorithms. These implementations are executed on processors whose side-channel leakage is quite far from the idealized leakage of the threshold probing model. Indeed, the processors are typically CMOS circuits, and therefore exhibit glitch and transition leakage. In this section, we briefly summarize the various state-of-the-art approaches to model this leakage and prove the security of a software masked implementation.

A first approach is to view the processor running the implementation as a hardware masked implementation, where the program executed only impacts the control signals of the datapath of the processor. The main advantage of this approach is it accuracy, since it can analyze the true logic gates leakage in the GT-robust probing model. The corresponding downside is that it requires knowledge of the netlist of the processor, which is rarely made public. Moreover, the most advanced tool in this direction, CoCo [Gig+21; GPM21] only makes direct robust probing security checks, and cannot verify the security of compositions. Consequently, it does not scale to (even moderately) large computations.

Another approach introduced in [Bar+21] uses a similar technique, but relies on a high-level description of micro-architectural state of the processor, along with the possible transitions. This is the basis for the automated verification of dedicated security notions (variants of NI and SNI that take into account the data stored in the (micro)architectural state of the processor across the execution of multiple gadgets). The main limitation of this approach is the need for an accurate description of the processor, which would ideally be based on the netlist of the processor. If this netlist is not available, the description of the processor can be built by reverse-engineering of the processor [MOW17], with the help of leakage assessment tests on the processor [MPW22], but this comes with

the risk of missing some leakage.

Finally, the Tornado tool [Bel+20b] works in the *register probing model*, which is a variant of the robust probing model where the expansion of a probe on a register of a CPU contains all the bits of that register. This model can be viewed as a specialized case of the glitch-robust probing model, since parts of the ALU of a microprocessor (e.g., an adder) combine all bits from a register. The register probing model however does not take into account transitions, and it does not necessarily cover all glitches either. On the other hand, Tornado can automatically generate register probing secure implementations of full ciphers, thanks to the use of the tight private circuits composable masking scheme [BGR18].

4.7 Automated verification of implementations

Let us conclude this chapter by introducing the fullVerif tool, which allows verifying the GT-robust security of hardware implementations based on gadget compositions. Despite the masking schemes introduced in Section 4.3 and Section 4.5 being relatively simple and their composability guarantees being strong, implementing them still requires a skilled hardware designer. Besides implementing the correct functionality, which can be tested through standard techniques such as test vectors, all the assumptions of the underlying security proofs must also be fulfilled to ensure security. Even for trivially composable schemes, which have arguably the least complex requirements, constraints such as non-adjacency are not typical in non-masked implementations, hence might not be taken into account by the designer everywhere in the design.

Furthermore, while it appears that masking composition proofs are only concerned with high-level assumptions, such as the kind of gadgets and the structure of the circuit, they in fact make other assumptions (implicit or not) which can be falsified by hardware implementations, have no impact on the functionality of the implementation, and are thus hard to verify by classical testing. Examples of such assumptions include:

- each gadget uses fresh, independent randomness;
- no more computation on shares is performed than specified by the algorithms (e.g., in parallel with or after useful computations are finished);
- no more computation on randomness is performed than specified by the algorithms;
- the order of the shares in a sharing is never shuffled.

We next describe a tool (named fullVerif) that allows verifying all these assumptions. The tool is based on a library of elementary gadgets (e.g.,

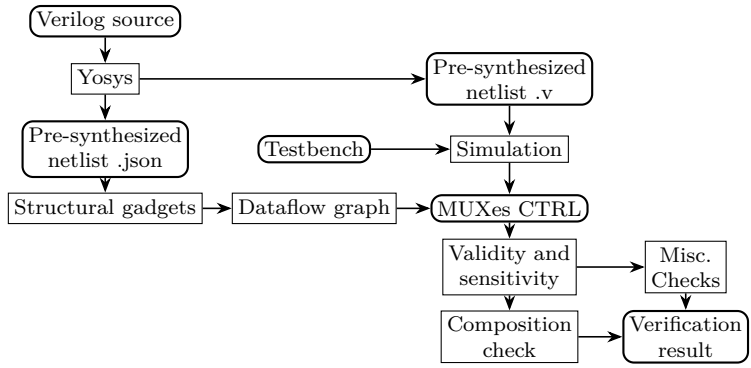


Figure 4.7: Process of the verification tool.

XOR or HPC2) which are annotated with their security properties.[7] At a high level, the tool takes a Verilog implementation as an input and outputs another, pre-synthesized implementation (that is equivalent to the input implementation), along with the result of the verification (error or success). It works by following the steps illustrated in Figure 4.7 that we detail next.

First, the open-source synthesizer Yosys [WGK13] is used to produce a netlist of all the gadgets, while preserving the hierarchy of the gadgets. Second, this netlist is simulated using a user-provided testbench (using IcarusVerilog). Third, the netlist is analyzed to build a graph of structural gadgets, which is then unrolled over all execution cycles, leading to a dataflow graph that represents the gadget execution composition (Definition 25). This graph is close to the gadget composition graphs that we analyze in composition proofs, but with two major differences: it contains so-called "MUX gadgets" (MUXing two sharings according to a non-sensitive control signal, modeled as a public parameter) which are usually overlooked in high-level masking proofs, and it contains gadgets for which the inputs are invalid (i.e., do not carry a meaningful value) but which are nevertheless executed. The next stage is extract the value of the control signal of every MUX gadget from the simulation (this is the single step where the result of the simulation is used). Then, we annotate each sharing with validity ("Does it contain a meaningful value?") and sensitivity ("Does it depend on the input sharings?") information.

[7] These annotations are trusted by the tool, and are therefore possibly wrong. The library is however quite small and simple, and is therefore fairly easy to audit. The main vulnerability is probably security-damaging optimizations by synthesis tools, which is imperfectly mitigated by the use of optimization-inhibiting attributes such as `keep` or `preserve`. These are however non-standard across toolchains, therefore finding a generic, practical and reliable solution is still an open problem.

Finally, the composition strategy is verified on the dataflow graph, from which all the gadgets that have no sensitive input are removed. In the case of trivial composition, it simply checks that all the gadgets satisfy the security property (e.g., PINI or O-PINI). For our optimized GT-robust composition, the adjacent executions of PINI gadgets are automatically grouped, and an error is produced if the all the gadgets of a group do not form a parallel composition. Other composition strategies (such as SNI-based ones) are not implemented in the tool, the implementing them would require minor additions: only the "Composition check" in Figure 4.7 is specific to our composition strategies.

The tool supports verification of sub-gadgets, only if they are pipeline. Otherwise, such gadgets may be inlined by the synthesizer in the top-level gadget. This has no adverse impact of the quality of the security verification. It however makes the possible errors harder to investigate, as they are now produced in the context of a much larger gadget.

The following pipeline gadgets are included in the library:

- affine gadgets: XOR, NOT;
- AND gates: DOM, HPC1, HPC2;
- register;
- SNI refresh;
- MUX (with public control input);
- sharing generation from a public value (it is trivial since it sets $d-1$ shares to 0, but it is needed to pass tool verification).

Verification that sharewise gadgets are actually sharewise is done automatically[8], while the other (i.e., SNI and PINI) gadgets are checked manually. Automatically verifying them using a tool such as maskVerif of SILVER is left to future work.

During the processing stages, many security checks are performed in addition to the check of composition properties already mentioned. We list the main ones along with the stage where they are performed.[9]

Structural gadgets:

- Any input sharing of a gadget must be connected to exactly one source gadget of which it is an output sharing;[10]

[8] Except for the MUX gadget, which is special-cased in the tool since it is at the core of its working principle.

[9] Other errors can be detected, ranging from syntax errors in the annotations to invalid outputs while they should be valid. The tool can therefore also detect some correctness issues, not only security ones.

[10] We currently allow implicit copy gadgets, that is, that an output sharing is used as input sharings more than once.

- The number of shares in any sharing is the number of shares of the gadget;
- Each wire belonging to a sharing is only used as part of that sharing and not elsewhere;
- Each output sharing of a composite gadget is generated as an output sharing of one of its composing gadgets;
- All the gadgets are connected to the same clock signal, otherwise cycle-based analysis of dataflow over time cannot be done properly.

Dataflow graph:

- No combinational loop exists in the circuit;
- Control signals for MUX gadgets do not depend on shares or randomness;
- No computation is performed on randomness, except storing it in registers selecting it with MUXes and feeding it to gadgets.

Misc. checks:

- All outputs of the composite gadget (at the specified cycle) should be connected to valid sharings;
- Each random input of a gadget is connected to a wire carrying randomness;
- Each random input of a gadget having at least one sensitive input sharing should be connected to a fresh random bit. For this check, a dataflow graph of the sub-circuit handling randomness (i.e., wires, registers, MUXes) is built;
- At the cycle after all the outputs have been produced, there should be no sensitive sharing remaining in the gadget (otherwise non-verified computations might happen, since we stop analysis at that cycle).

The usage of fullVerif at various stages in the development of the implementations of three block ciphers (see Section 4.5.3) has led to the detection of many design and implementations mistakes. After investigation, most design issues identified by fullVerif were shown to break the GT-robust probing security.

Tool performance The computational complexity of the tool is quasi-linear in the number of gadgets and quadratic in the number of shares (like a simulation tool). Practically, fullVerif is fast enough to be used "interactively" when developing or debugging an implementation: its overhead on top of the behavioral simulation of one encryption is at most 2 seconds for a full AES with up to 16 shares.

Table 4.3: Masking formal verification tools' overview.

	Abstract	Concrete
Direct	maskVerif [Bar+15]*	REBECCA [Blo+18]
	IronMask [Bel+21]*	CoCo [Gig+21]
		maskVerif [Bar+19]*
		SILVER [KSM20]*
		scVerif [Bar+21]*
Composition-based	maskComp [Bar+16]	Tornado [Bel+20b]
	TightProve [BGR18]	fullVerif

* The tool can verify composable security properties such as NI, SNI or PINI, but it does not check the correct composition of the gadgets.

Related works We next provide a brief account of the state-of-the-art tools aimed at verifying masked algorithms in the robust or threshold probing model to situate fullVerif. The main tools to which our proposal compares are listed in Table 4.3. Such tools can verify abstract implementations or concrete ones (i.e., actual code, including physical non-idealities); they can also aim at direct verification (which is limited to small circuits and security orders) or composition-based verification.

Direct verification in the glitch-robust, transition-robust and GT-robust probing models is very similar to direct verification in the threshold probing model discussed in Section 3.5. Indeed, a verification tool for the threshold probing model can be turned into a robust probing model verification tool by adding a probe expansion step between the selection of a set of probes and the computation of the simulatability set.

For composition, the fullVerif tool is the first one that can verify the composability of concrete hardware implementations including transitions and glitches. The most similar tool is Tornado [Bel+20b], which works for software implementations on microcontrollers (see Section 4.6).

5 Security in the random probing model

Proving the (robust) probing security of a masking scheme is a good first step towards ensuring that an implementation that uses it will be secure in practice. However, (robust) probing security is oblivious to parameters that can strongly impact the practical security, such as the amount of leakage produced by a single variable (e.g., if it is used many times) or the number of distinct variables that can leak similar information (e.g., bijectively related variables). These limitations lead to fairly loose bounds when the security in more practical leakage models (such as noisy leakage or random probing) is proven by reduction to the threshold probing model [DDF19].

We propose direct security proofs in the random probing model (see Section 2.3.3) to work around the aforementioned limitations: in this model, every wire in the circuit leaks, and if a variable is used more than once, copy gates must be inserted, increasing the number of wires that carry this variable. In this chapter, we try to have random probing security proofs that are as tight as possible. In particular, if a circuit is t-threshold probing secure circuit, its security in the p-random probing model should scale in $\mathcal{O}\left(p^t\right)$ (previous works scale in $\mathcal{O}\left(p^s\right)$ with $s < t$). For this purpose, we use two complementary tools: first, we introduce the probe distribution tables (PDT), from which random probing security can be proven for compositions of small gadgets (i.e., gadgets with few inputs and few shares). Second, we use the local random probing model (LRPM) heuristic to analyze and optimize multiplication gadgets regarding their security against horizontal attacks. The LRPM and PDT analysis of some small gadget compositions are also analyzed, showing the strengths and weaknesses of the LRPM.

This chapter is organized as follows: we first discuss in Section 5.1 the previous approaches to prove security in the random probing model. Next, we introduce the PDTs for random probing security analysis, and we compare the method to the state of the art (Section 5.2, published in [Cas+21a]). Then, PDT-based bounds are compared to the LRPM in Section 5.3.4 (this is an unpublished contribution). Finally, the LRPM is used for noise rate analysis and optimization of multiplication gadgets in

the remaining part of Section 5.3 (this is the main contribution of [CS19]).

5.1 Generic approaches and open problems

Before introducing the contributions of this thesis on random probing security, let us describe two state-of-the-art masking approaches building random probing secure circuits.[1]

A first approach is based on region probing security: a region probing secure circuit is partitioned into regions such that any set of probes with at most t probes in each region is safe [ISW03; Bar+16]. Region probing security is therefore strictly stronger that threshold probing security. If all the regions in a circuit have at most n wires, then the expected number of wires observed by a p-random probing adversary is at most np in each region. Therefore, if t is sufficiently large, the whole circuit has a good security level in the p-random probing model [Bar+16; DDF19].

Concretely, the composition of $2t$-SNI gadgets is t region probing secure [Bar+16], and such a circuit can be built as a trivial composition of $d = 2t + 1$-shares ISW multiplication gadgets and d-shares sharewise gadgets whose output is SNI-refreshed. In this construction, each gadget is a region and contains at most $n = 7d^2 + 2d$ wires, therefore the full circuit is $ke^{-\frac{pn}{3}}$-secure in the p-random probing model for any $p \leq \frac{t+1}{2n}$, where k is the number of gadgets in the circuit [DDF19]. Concretely, to reach 2^{-16}-security, this means more than 100 shares (for $p < 10^{-3}$), which is much larger than current practical implementations.

The other approach is based on expansion [AIS18], which can be viewed as recursive masking. More precisely, a circuit C^0 is first masked with a low number of shares (e.g., $d = 3$ or $d = 5$), which gives a new circuit C^1. Then, the same masking scheme is applied to C^1, giving C^2, etc. Recent developments of this technique have improved its efficiency [Bel+20a; BRT21] (see also Figure 5.9). This approach allows to build circuits with a constant noise rate and in practice the required noise level ($p = 2^{-7.5}$) is better than the one required with the region-probing based approach.

While the expansion-based techniques are promising, their proof is not tight: the security level does not scale as p^{t+1} for asymptotically small p, as one can expect (and achieve with the best known attacks). In this chapter, we adopt another approach: we first analyze (in Section 5.2) simple masked circuits (e.g., based on the ISW multiplication), starting with small circuits and a low number of shares and trying to remain as

[1] There exists other approaches such as the wire shuffling countermeasure [ISW03; CS21b], but these have worse performance vs security trade-off for low number of shares.

tight as possible. This method is however computationally expensive, and therefore limited to gadgets with a low number of shares and to compositions of a few gadgets. We therefore introduce in Section 5.3 a heuristic method to investigate these circuits with larger number of shares, and in particular to analyze how the noise rate of the ISW multiplication can be improved.

5.2 Tighter bound

We introduced in [Cas+21a] a methodology for proving tighter security bounds in the random probing model for the most practical (low noise, low number of shares) region of the design space. A first component is STRAPS, a tool for the Sampled Testing of the RAndom Probing Security of small circuits, which uses the Monte-Carlo technique for probability bounding. Since this tool is limited to the analysis of small circuits and/or small security orders due to computational reasons, we next combine it with a new compositional strategy that exploits a new security property for masked gadgets, the Probe Distribution Table (PDT), which gives tighter security bounds for composed circuits and is integrated in the STRAPS tool. This combination of tool and compositional strategy allows us to analyze significantly larger circuits and security orders than an exhaustive approach.

5.2.1 Sampled testing of small circuits

In this section, we show how to efficiently compute an upper bound on the random probing security level of relatively small gadgets. We first derive a way to compute the security level of a gadget for various values of p, using some computationally heavy pre-processing. Next, we explain a way to use statistical confidence intervals to reduce the cost of the pre-processing. Finally, we detail how these techniques are implemented in a practical algorithm, implemented in the STRAPS tool that we describe in Section 5.2.4.

A simple bound We can obtain the security level of a small circuit by computing first the statistical distribution of $\mathcal{L}_p(\mathsf{G})$ (i.e., $\Pr[\mathcal{L}_p(\mathcal{A}) = \mathcal{A}']$ for each subset $\mathcal{A}' \subseteq \mathcal{A}$). Then, for each possible set of probes $\mathcal{A}'$, we compute the simulatability set in order to determine if the set is a successful attack, denoted as $\delta_{\mathcal{A}'} = 1$, while $\delta_{\mathcal{A}'} = 0$ otherwise [Bel+20a]. A first naive algorithm to compute the security level ϵ is thus given by

the equation

$$\epsilon = \sum_{\substack{\mathcal{A}' \subseteq \mathcal{A} \\ \text{s.t. } \delta_{\mathcal{A}'}=1}} \Pr[\mathcal{L}_p(\mathcal{A}) = \mathcal{A}']. \tag{5.1}$$

The computational cost of iterating over all possible probe sets grows exponentially with $|\mathcal{A}|$: for a circuit with $|\mathcal{A}|$ internal wires, one has to do $2^{|\mathcal{A}|}$ simulatability set computations for each value of p (e.g., $|\mathcal{A}| = 57$ for the ISW multiplication with three shares). To efficiently cover multiple values of p, we introduce a first improvement to the naive algorithm given by Equation 5.1. For each $i \in \{0, \ldots, |\mathcal{A}|\}$, we compute the number c_i of sets of probes of size i that are successful attacks $c_i = \left|\left\{\mathcal{A}' \in \mathcal{A}^{(i)} \text{ s.t. } \delta_{\mathcal{A}'} = 1\right\}\right|$. Then, we can compute

$$\epsilon = \sum_{i=0}^{|\mathcal{A}|} p^i (1-p)^{|\mathcal{A}|-i} c_i, \tag{5.2}$$

which gives us a more efficient algorithm to compute random probing security, since it re-uses the costly computation of c_i for multiple values of p.

The VRAPS tool [Bel+20a] computes c_i for small values of i by computing $\delta_{\mathcal{A}'}$ for all $\mathcal{A}' \in \mathcal{A}^{(i)}$. This is however computationally intractable for larger i values, hence they use the bound $c_i \leq \binom{|\mathcal{A}|}{i}$ in such cases.

A statistical bound Let us now show how to improve the bound $c_i \leq \binom{|\mathcal{A}|}{i}$ while keeping a practical computational cost. At a high level, we achieve this by using a Monte-Carlo method whose idea is as follows: instead of computing directly ϵ, we run a randomized computation that gives us information about ϵ (but not its exact value). More precisely, the result of our Monte-Carlo method is a random variable ϵ^U that satisfies $\epsilon^U \geq \epsilon$ with probability at least $1 - \alpha$ (the confidence level), where α is a parameter of the computation. That is, $\Pr_{\text{MC}}\left[\epsilon^U \geq \epsilon\right] \geq 1 - \alpha$, where $\Pr_{\text{MC}}$ means the probability over the randomness used in the Monte-Carlo method.[2] In the rest of this work, we use $\alpha = 10^{-6}$ since we consider that it corresponds to a sufficient confidence level.[3]

Let us now detail the method. First, let $r_i = c_i / \left|\mathcal{A}^{(i)}\right|$. We remark that r_i can be interpreted as a probability: $r_i = \Pr_{\mathcal{A}' \xleftarrow{\$} \mathcal{A}^{(i)}}[\delta_{\mathcal{A}'} = 1]$.

[2] In other words, $[0, \epsilon^U]$ is a conservative confidence interval for ϵ with nominal coverage probability of $1 - \alpha$.

[3] This parameter is not critical: we can obtain a similar value for ϵ^U with higher confidence level by increasing the amount of computation: requiring $\alpha = 10^{-12}$ would roughly double the computational cost of the Monte-Carlo method.

The Monte-Carlo method actually computes r_i^U such that $r_i^U \geq r_i$ with probability at least $1 - \alpha/(|\mathcal{A}| + 1)$. Once the r_i^U are computed, the result is

$$\epsilon^U = \sum_{i=0}^{|\mathcal{A}|} p^i (1-p)^{|\mathcal{A}|-i} \binom{|\mathcal{A}|}{i} r_i^U, \tag{5.3}$$

which ensures that $\epsilon^U \geq \epsilon$ for any p with confidence level $1-\alpha$, thanks to the union bound. Next, r_i^U is computed by running the following experiment: take t_i samples $\mathcal{A}' \xleftarrow{\$} \mathcal{A}^{(i)}$ uniformly at random (this sampling is the random part of the Monte-Carlo method) and compute the number s_i of samples for which $\delta_{\mathcal{A}'} = 1$. By definition, s_i is a random variable that follows a binomial distribution $B(t_i, r_i)$: the total number of samples is t_i and the "success" probability is r_i. We can thus use the bound derived in [Sch08]. If r_i^U satisfies $\mathrm{CDF}_{\mathrm{binom}}(s_i; t_i, r_i^U) = \alpha/(|\mathcal{A}| + 1)$, then $\Pr[r_i^U \geq r_i] = 1 - \alpha/(|\mathcal{A}| + 1)$, which gives

$$r_i^U = \begin{cases} 1 & \text{if } s_i = t_i, \\ x \quad \text{s.t.} \quad I_x(s_i + 1, t_i - s_i) = 1 - \alpha/(|\mathcal{A}| + 1) & \text{otherwise,} \end{cases} \tag{5.4}$$

where $I_x(a, b)$ is the regularized incomplete beta function. We can similarly compute a lower bound ϵ^L such that $\epsilon^L \leq \epsilon$ with confidence coefficient $1-\alpha$, which we compute by replacing r_i^U with r_i^L in Equation 5.3, where:

$$r_i^L = \begin{cases} 0 & \text{if } s_i = 0, \\ x \quad \text{s.t.} \quad I_x(s_i, t_i - s_i + 1) = \alpha/(|\mathcal{A}| + 1) & \text{otherwise.} \end{cases} \tag{5.5}$$

A hybrid algorithm Our Monte-Carlo method has a main limitation: when $r_i = 0$ the bound r_i^U will not be null (it will be proportional to $1/t_i$). This means that we cannot prove tightly the security of interesting gadgets when p is small. For instance, let us take a fourth-order secure gadget (that is, $r_0 = r_1 = r_2 = r_3 = r_4 = 0$). If $r_1^U \neq 1$, then ϵ^U scales like $r_1^U p$ as p becomes small (other, higher degree, terms become negligible). A solution to this problem would be to set t_i to a large number, such that, in our example, r_1^U would be small enough to guarantee that $r_1^U p \ll r_5 p^5$ for all considered values of p. If we care about $p = 10^{-3}$, this means $r_1^U \ll 10^{-12} \cdot r_5 \leq 10^{-12}$. This is however practically infeasible since the number of samples t_1 is of the order of magnitude $1/r_1^U > 10^{12}$.

There exists another solution, which we call the hybrid algorithm: perform a full exploration of $\mathcal{A}^{(i)}$ (i.e., use the algorithm based on Equation 5.2) when it is not computationally too expensive (i.e., when

$\left|\mathcal{A}^{(i)}\right|$ is below some limit N_{max}), and otherwise use the Monte-Carlo method. The goal of this hybrid algorithm is to perform a full exploration when $r_i = 0$ (in order to avoid the limitation discussed above), which can be achieved for gadgets with a small number d of shares. Indeed, r_i can be null only for $i < d$ (otherwise there can be probes on all the shares of the considered input sharing), and the number of cases for the full exploration is therefore $\left|\mathcal{A}^{(i)}\right| = \binom{|\mathcal{A}|}{i} \leq \binom{|\mathcal{A}|}{d-1}$, which is smaller than N_{max} if d and $|\mathcal{A}|$ are sufficiently small. The latter inequality holds if $|\mathcal{A}| \geq 2(d-1)$, which holds for all non-trivial gadgets.

Algorithm 26 Random probing security algorithm: compute r_i^U, r_i^L for a given $\mathcal{A}$ and i. The parameters are N_{max} and N_t.

Require $N_t \leq N_{max}$
$N_{sets} = \binom{|\mathcal{A}|}{i}$
$t_i \leftarrow 1$, $s_i \leftarrow 0$ ▷ t_i: total number of samples, s_i: successful attacks
while $t_i \leq N_{max} \wedge s_i < N_t$ **do** ▷ First Monte-Carlo sampling loop
  $\mathcal{A}' \xleftarrow{\$} \mathcal{A}^{(i)}$
  if $\delta_{\mathcal{A}'} = 1$ **then**
    $s_i \leftarrow s_i + 1$.
  $t_i \leftarrow t_i + 1$
if $N_{sets} \leq t_i$ **then** ▷ Enumerate $\mathcal{A}^{(i)}$ if it is cheaper than Monte-Carlo.
  $s_i \leftarrow 0$
  for all $\mathcal{A}' \in \mathcal{A}^{(i)}$ **do**
    if $\delta_{\mathcal{A}'} = 1$ **then**
      $s_i \leftarrow s_i + 1$
  $r_i^U \leftarrow s_i / N_{sets}$, $r_i^L \leftarrow s_i / N_{sets}$
else ▷ Re-run Monte-Carlo to avoid bias due to N_t early stopping.
  $s_i \leftarrow 0$
  Repeat t_i **times**
    $\mathcal{A}' \xleftarrow{\$} \mathcal{A}^{(i)}$
    if $\delta_{\mathcal{A}'} = 1$ **then**
      $s_i \leftarrow s_i + 1$
  Compute r_i^U and r_i^L using Equations 5.4 and 5.5.

Algorithm 26 describes how we choose between full enumeration and Monte-Carlo sampling, which is the basis of our STRAPS tool (see Section 5.2.4 for more details). The algorithm adds a refinement on top of the above explanation: if we can cheaply show that r_i is far from zero, we do not perform full exploration even if it would not be too expensive. It accelerates the tool, while keeping a good bound. This optimization is

implemented by always starting with a Monte-Carlo sampling loop that takes at most N_{max} samples, with an early stop if s_i goes above the value of a parameter N_t (we typically use parameters such that $N_{max} \gg N_t$). The parameter N_t determines the relative accuracy of the bound we achieve when we do the early stop: in the final sampling, we will have $s_i \approx N_t$, which means that the uncertainty on r_i decreases as N_t increases. The parameter N_{max} has an impact when r_i is small and we do not reach N_t successful attacks: it limits both the maximum size of $\mathcal{A}^{(i)}$ for which full exploration is performed, and the number of samples used for the Monte-Carlo method.

Remark. The Monte-Carlo method is limited to the random probing model and cannot be used to prove security in the threshold probing model since proving security in this model means proving that $r_i = 0$, which it cannot do. Our hybrid algorithm, however, can prove threshold probing security for the numbers of probes i where it does full enumeration of $\mathcal{A}^{(j)}$ for all $j \in \{0, \ldots, i\}$.

Simulatability set We use the simulatability set computation algorithm from maskVerif [Bar+19] since it is more efficient than the alternatives (see Section 3.5). One drawback of this algorithm is that in some cases, it wrongly reports that the adversary succeeds, which implies that the statistical lower bound is not anymore a lower bound for the security level, and the statistical upper bound is not completely tight (but it is still an upper bound for the true security level). In this case, we refer to the statistical lower bound as the *stat-only* lower bound. While the stat-only lower bound is not indicative of the security level, it remains useful to quantify the statistical uncertainty and therefore to assess whether one could improve the tightness of the upper bound by increasing the number of samples in the Monte Carlo method.

5.2.2 Security of some simple gadgets

We now present the results of random probing security evaluations using the previously described algorithm. First, we discuss the sharewise XOR gadget and the ISW multiplication gadget with d shares. Next, we discuss the impact of the two parameters of our algorithm (N_{max} and N_t) on the tightness of the results and on the computational complexity (i.e., the execution time) of the tool.

In Figure 5.1 (left), we show the security level (with respect to one of the inputs) of the addition gadget for $d = 1, \ldots, 6$ shares. We can see that the security level of the gadget is proportional to p^d, which

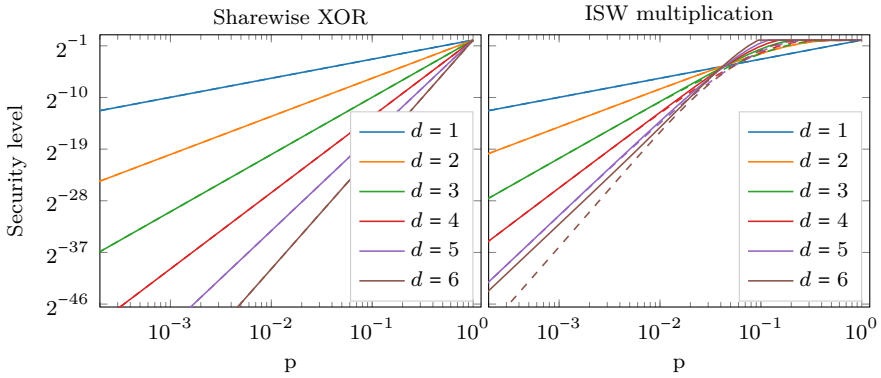


Figure 5.1: Security of masked gadgets (with respect to the input sharing x, assuming the input sharing y is public). The continuous line is an upper bound, while the dashed line is the stat-only lower bound. $N_{max} = 10^7$, $N_t = 1000$.

is expected. Indeed, this gadget is sharewise, hence and the adversary needs at least one probe in each circuit share to succeed. This trend can also be linked with the security order in the threshold probing model. Since the gadget is $d-1$-threshold probing secure, a successful attack contains at least d probes, hence has probability proportional to p^d.

We can observe a similar trend for the ISW multiplication gadget (Figure 5.1, right). Since the gadget is $d-1$-threshold probing secure, the security level scales proportionally to p^d for small values of p. For larger values of p, the security level of this gadget is worse than p^d, which is due to the larger number of wires, and the increased connectivity compared to the addition gadgets. It implies that there are many sets of probes of sizes $d+1$, $d+2$, ... that are successful attacks (which is usually referred to as horizontal attacks in the practical side-channel literature [Bat+16]). These sets make up for a large part of the success probability when $p > 0.05$ due to their large number, even though they individually have a lower probability of occurring than a set of size d (for $p < 0.5$).

Next, we discuss the impact of parameters N_{max} and N_t in Algorithm 26 on the tightness of the bounds we can compute. We first focus on the impact of N_t, which is shown on Figure 5.2. For $N_t = 10$, we have a significant distance between the statistical upper and lower bounds, while the gap becomes small for $N_t = 100$ and $N_t = 1000$. This gap appears as a bounded factor between the upper and lower bounds which, as discussed previously, is related to the accuracy of the estimate of a proportion when we have about N_t positive samples.

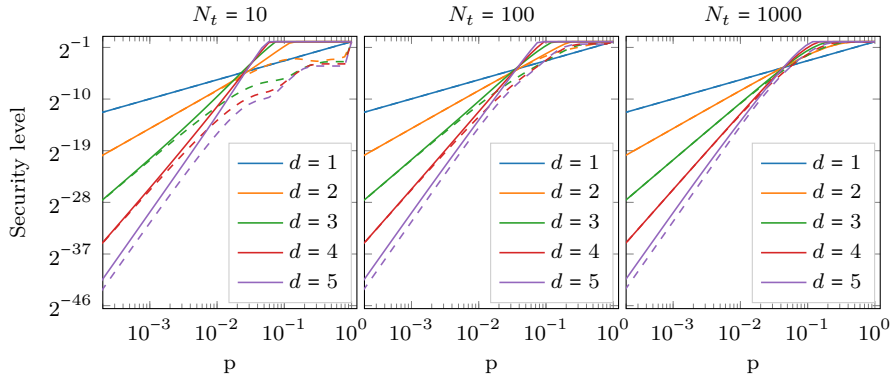


Figure 5.2: Impact of the parameter N_t of Algorithm 26 on the security bounds of masked ISW multiplication gadgets (w.r.t. the input sharing x). $N_{max} = 10^7$.

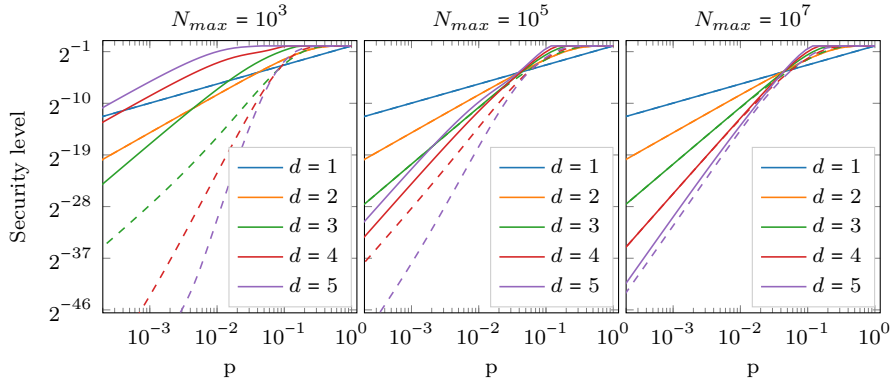


Figure 5.3: Impact of the parameter N_{max} of Algorithm 26 on the security bounds of masked ISW multiplication gadgets (w.r.t. the input sharing x). $N_t = 1000$.

We also look at the impact of N_{max} on Figure 5.3. We observe a gap between the bounds for too low N_{max} values, which gets worse as the number of shares increases. Indeed, when N_{max} is too small, we cannot do an enumeration of all the sets of $d-1$ probes, hence we cannot prove that the security order of the gadget is at least $d-1$, which means that the upper bound is asymptotically proportional to $p^{d-1'}$, with $d' < d-1$.

We finally observed that the computational cost is primarily dependent on N_{max} and the circuit size, while N_t has a lower impact (for the values considered). For instance, the execution time of the tool for the ISW multiplication with $n = 6$, $N_{max} = 10^8$ and $N_t = 100$ is about 33 h on a 24-core computer.

Table 5.1: Notations for composite gadgets.

Gadget	Input wires	Output gates	Internal wires
G_0	$\mathcal{I}_0$	$\hat{\mathcal{O}}_0$	$\mathcal{W}_0$
G_1	$\mathcal{I}_1$	$\hat{\mathcal{O}}_1$	$\mathcal{W}_1$
$\mathsf{G}_1 \circ \mathsf{G}_0$	$\mathcal{I}_0$	$\hat{\mathcal{O}}_1$	$\mathcal{W}_0 \parallel_{(\lvert\mathcal{W}_0\rvert)} \mathcal{I}_1 \parallel_{(\lvert\mathcal{I}_1\rvert)} \mathcal{W}_1$
$\mathsf{G}_0 \Vert \mathsf{G}_1$	$\mathcal{I}_0 \parallel_{(\lvert\mathcal{I}_0\rvert)} \mathcal{I}_1$	$\hat{\mathcal{O}}_0 \parallel_{(\lvert\hat{\mathcal{O}}_0\rvert)} \hat{\mathcal{O}}_1$	$\mathcal{W}_0 \parallel_{(\lvert\mathcal{W}_0\rvert)} \mathcal{W}_1$

5.2.3 Probe distribution tables

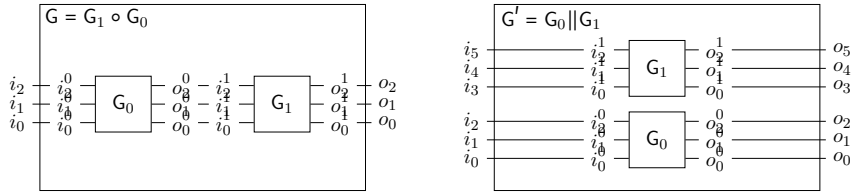
In the previous section, it became clear that the tool is limited if it directly computes the security of complex circuits. This leads to the need to investigate composition properties. The existing definitions of random probing composability and random probing expandability in [Bel+20a] are based on counting probes at the inputs and outputs of gadgets which are needed to simulate the leakage. We have recognized that ignoring the concrete random distribution over the needed input/output wires, and only counting the wires leads to a significant loss of tightness. Therefore we introduce our new security notion, the PDT. Rather than considering only whether a set of probe is a successful attack, the full information of the simulatability set is included in the PDT. Moreover, in contrast to the previous section, we take into account the output gates, which is needed for composition.

Gadget composition and sets of wires Let us first introduce in Table 5.1 some notations to keep track of input wires, output gates and internal wires in gadget compositions. We remark that in this section, the input and output sharings of parallel gadget compositions are not allowed to be freely re-ordered.[4]

For these definitions, we work with ordered finite sets. That is, given a finite set A (e.g., one of the sets $\mathcal{W}$, $\mathcal{I}$ or $\hat{\mathcal{O}}$ of a gadget G), we assign to each element of A a unique index in $[\![0, \lvert A\rvert[\![$. Then, given disjoint finite sets A and B, we denote by $C = A \parallel_{(k)} B$ the union of A and B ordered such that a wire with index i in A has index i in C, and a wire with index i in B has index $k + i$ in B. The $\parallel_{(\cdot)}$ operator is right-associative, which means that $A_2 \parallel_{(k_1)} A_1 \parallel_{(k_0)} A_0 = A_2 \parallel_{(k_1)} \left(A_1 \parallel_{(k_0)} A_0\right)$.

Figure 5.4 illustrates how to renumber the input wires and output

[4] Re-ordering can still be performed using dedicated re-ordering gadgets which contain only identity gates but, unlike identity gadgets, may shuffle wires. The PDT of these gadgets (which is the identity matrix, up to a permutation of the rows) has to be taken into account in the composition.



(a) Sequential composition. **(b) Parallel composition.**

Figure 5.4: Examples of sequential composition (5.4a) and parallel composition (5.4b).

gates in the case of gadgets with three inputs wires and three output gates. For example, the input wire set of the parallel composition G' is $\mathcal{I} = \{i_0, i_1, \ldots, i_5\}$ which is the wire union $\mathcal{I} = \mathcal{I}_0 \,\|_{(|\mathcal{I}_0|)}\, \mathcal{I}_1$ of the input wires $\mathcal{I}_0 = \{i_0^0, i_1^0, i_2^0\}$ and $\mathcal{I}_1 = \{i_0^1, i_1^1, i_2^1\}$ of the gadgets G_0 and G_1.

Probe distributions

Next, we introduce our new security properties, the PD (Probe Distribution) and the PDT (Probe Distribution Table). Intuitively, given a set of wires $\mathcal{W}$ and a leakage process $\mathcal{L}$ (hence $\mathcal{L}(\mathcal{W}) \subseteq \mathcal{W}$), the PD of $\mathcal{L}(\mathcal{W})$ is a vector of size $2^{|\mathcal{W}|}$ that represents the statistical distribution of $\mathcal{L}(\mathcal{W})$. In more detail, for each subset $\mathcal{W}' \subseteq \mathcal{W}$, there is a corresponding element of the PD with value $\Pr\left[\mathcal{L}(\mathcal{W}) = \mathcal{W}'\right]$. The PDT notion extends the idea in a way that makes it useful for analyzing gadget compositions: it links the set of output probes on the gadget to the distribution of the simulatability set of the gadget (i.e., to the inputs needed to simulate the leakage). More precisely, for a gadget G, the PDT is a matrix in $[0,1]^{|\mathcal{I}|\times|\mathcal{O}|}$, such that each column is associated to a subset of the outputs $\mathcal{O}' \subseteq \mathcal{O}$. Each column is a PD that represents the distribution of $\mathcal{S}^{\mathsf{G}}(\mathcal{L}(\mathsf{G}) \cup \mathcal{O}')$ (viewed as a subset of the set of inputs $\mathcal{I}$). The two main results (Theorem 4 and Theorem 5) of the next section relate the PDT of a sequential (resp., parallel) gadget composition to the matrix (resp., tensor) product of the PDTs of the composing gadgets. We first formalize the mapping between subsets of wires and indices in vectors/matrices.

Definition 41 (Index representation of subsets of wires)**.** *For any set of wires $\mathcal{W}$ of which each element has a unique index in $[\![0, |\mathcal{W}|[\![$, we associate to each subset $\mathcal{W}'$ of $\mathcal{W}$ the index*

$$\tilde{\mathcal{W}'} = \sum_{i=}^{|\mathcal{W}|} b_i 2^i \quad \textit{with} \quad \begin{cases} b_i = 1 & \textit{if element } i \textit{ of } \mathcal{W} \textit{ belongs to } \mathcal{W}', \\ b_i = 0 & \textit{otherwise.} \end{cases}$$

For example, the wire set $\mathcal{W} = \{\omega_0, \omega_1\}$ has 4 subsets $\mathcal{W}'$, that we represent with their index below:

$\mathcal{W}'$	$\varnothing$	$\{\omega_0\}$	$\{\omega_1\}$	$\{\omega_0, \omega_1\}$
$\tilde{\mathcal{W}}'$	0	1	2	3

Let us now give the formal definition of the PD.

Definition 42 (Probe Distribution (PD))**.** *Let $\mathcal{L}$ be a leakage process for a set of wires $\mathcal{W}$. The probe distribution (PD) of $\mathcal{L}$ with respect to $\mathcal{W}$ is $\mathbf{p} \in [0,1]^{2^{|\mathcal{W}|}}$ such that for all $\mathcal{W}' \subseteq \mathcal{W}$, $\mathbf{p}_{\tilde{\mathcal{W}}'} = \Pr\left[\mathcal{L}(\mathcal{W}) = \mathcal{W}'\right]$.*

The PD of $\mathcal{L}_p(\mathcal{W})$ in the previous example is

$$\mathbf{p} = \left((1-p)^2, p(1-p), p(1-p), p^2\right) .$$

We next give the definition of the PDT.

Definition 43 (Probe Distribution Table (PDT))**.** *Let G be a gadget with input wires $\mathcal{I}$ and output wires $\mathcal{O}$. For any $\mathcal{O}' \subseteq \mathcal{O}$, let $\mathbf{p}_{\tilde{\mathcal{O}}'}$ be the PD of $\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}) \cup \mathcal{O}')$. The PDT of G ($\boldsymbol{PDT}_{\mathsf{G}}$) is a $[0,1]^{2^{|\mathcal{I}|} \times 2^{|\mathcal{O}|}}$ matrix with all the $\mathbf{p}_{\tilde{\mathcal{O}}'}$ as columns, that is*

$$\boldsymbol{PDT}_{\mathsf{G}} = \left(\mathbf{p}_j\right)_{j \in [2^{|\mathcal{O}|}]},$$

with $j = \tilde{\mathcal{O}}'$ for all subsets $\mathcal{O}' \subseteq \mathcal{O}$. The notation $\boldsymbol{PDT}_{\mathsf{G}}(\tilde{\mathcal{I}}', \tilde{\mathcal{O}}')$ refers to the element of $\mathbf{p}_{\tilde{\mathcal{O}}'}$ associated to $\mathcal{I}'$.

The definition implies that $\mathbf{PDT}_{\mathsf{G}}(\tilde{\mathcal{I}}', \tilde{\mathcal{O}}') = \Pr\left[\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}) \cup \mathcal{O}') = \mathcal{I}'\right]$. Furthermore, the PDT of a gadget is independent of its environment (i.e., of the PD of its output wires).

A first example of PDT is the one of the addition and multiplication gates (when viewed as gadgets with one share). In the first column, no output has to be simulated, and thus the only leakage comes from the two input wires. For the second column, knowledge of both inputs is needed to simulate the output. This gives:

$$\mathbf{PDT}_{+} = \mathbf{PDT}_{\times} = \begin{array}{c|cc} \mathbf{PDT} & \mathcal{O}' = \varnothing & \mathcal{O}' = \{0\} \\ \hline \mathcal{I}' = \varnothing & (1-p^2) & 0 \\ \mathcal{I}' = \{0\} & p(1-p) & 0 \\ \mathcal{I}' = \{1\} & p(1-p) & 0 \\ \mathcal{I}' = \{0,1\} & p^2 & 1 \end{array} \tag{5.6}$$

The second example is the simple refresh gadget G_r with two shares where a random value is added to two different wires. The random value

leaks three times with probability p (one time in the copy gate and two times in the addition gate). The leakage probability of the random value is therefore $q = 1 - (1-p)^3$, and we get:

$\mathbf{PDT}_{\mathsf{G}_r} =$

PDT	$\mathcal{O}' = \varnothing$	$\mathcal{O}' = \{0\}$	$\mathcal{O}' = \{1\}$	$\mathcal{O}' = \{1,0\}$
$\mathcal{I}' = \varnothing$	$(1-p)^2$	$(1-q)(1-p)^2$	$(1-q)(1-p)^2$	0
$\mathcal{I}' = \{0\}$	$p(1-p)$	$(q+qp)(1-p)$	$(1-q)p(1-p)$	0
$\mathcal{I}' = \{1\}$	$p(1-p)$	$(1-q)p(1-p)$	$(q+(1-q)p)(1-p)$	0
$\mathcal{I}' = \{0,1\}$	p^2	$qp+(1-q)p^2$	$qp+(1-q)p^2$	1

The PDT is related to the security level in the random probing model.

Proposition 32 (Security level from PDT)**.** *Let* G *be a gadget and* $\boldsymbol{PDT}_{\mathsf{G}}$ *its Probe Distribution Table. Let* s *be the security level of* G *with respect to an input sharing. If the set of shares of the considered input sharing is* $\mathcal{I}'$*, then*

$$\mathbf{e}^T \cdot \boldsymbol{PDT}_{\mathsf{G}} \cdot \mathbf{p}_{\varnothing} = \sum_{\mathcal{I}'' \supseteq \mathcal{I}'} \boldsymbol{PDT}_{\mathsf{G}}(\tilde{\mathcal{I}''}, 0) \geq s,$$

where $\mathbf{p}_{\varnothing} = (1, 0, \ldots, 0)$ *is the PD corresponding to no output leakage and* $\mathbf{e}_i = 1$ *for all* $i = \tilde{\mathcal{I}''}$ *with* $\mathcal{I}'' \supseteq \mathcal{I}'$*, while* $\mathbf{e}_i = 0$ *otherwise.*

Proof. Let $\mathcal{A}'$ be a set of wires that is an attack, that is, that depends on the considered unshared value. Simulating $\mathcal{A}'$ therefore requires at least all the shares in $\mathcal{I}'$, hence

$$s \leq \Pr_{\mathcal{A}' \leftarrow \mathcal{L}_p(\mathsf{G})} \left[\mathcal{S}^{\mathsf{G}}(\mathcal{A}') \supset \mathcal{I}' \right] .$$

Then, by definition of $\mathcal{L}_p(\mathsf{G})$ and of the PDT,

$$s \leq \sum_{\mathcal{I}'' \supset \mathcal{I}'} \Pr\left[\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G})) = \mathcal{I}'' \right] = \sum_{\mathcal{I}'' \supset \mathcal{I}'} \mathbf{PDT}_{\mathsf{G}}(\tilde{\mathcal{I}''}, 0) .$$

This proves the inequality. The equality in the proposition holds by construction of $\mathbf{e}$. □

We now give a few results that constitute the basis for the PDT composition theorems. A first result links the PD of the input wires needed to simulate the leakage of the gadget and some of its outputs to the PDT of the gadget and the PD of its outputs. This is the foundation for the analysis of sequential gadget composition.

Proposition 33 (PDT and PD)**.** *Let* G *be a gadget with outputs* $\hat{\mathcal{O}}$ *and inputs* $\mathcal{I}$*. If a leakage process* $\mathcal{L}'(\hat{\mathcal{O}})$ *has a PD* $\mathbf{p}$ *with respect to* $\hat{\mathcal{O}}$*, then* $\boldsymbol{PDT}_{\mathsf{G}} \cdot \mathbf{p}$ *is the PD of* $\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}) \cup \mathcal{L}'(\hat{\mathcal{O}}))$ *with respect to input wires* $\mathcal{I}$*.*

Proof. The solution can be directly derived from the definitions: Let $(v_i)_{i\in 2^{|\mathcal{I}|}} = \mathbf{PDT}_{\mathsf{G}} \cdot \mathbf{p}$. For any $\mathcal{I}' \subseteq \mathcal{I}$, it holds that

$$\begin{aligned} v_{\tilde{\mathcal{I}}'} &= \sum_{\hat{\mathcal{O}}' \subseteq \hat{\mathcal{O}}} \mathbf{PDT}_{\mathsf{G}}(\tilde{\mathcal{I}}', \tilde{\hat{\mathcal{O}}}') \cdot p_{\tilde{\hat{\mathcal{O}}}'} \\ &= \sum_{\hat{\mathcal{O}}' \subseteq \hat{\mathcal{O}}} \Pr\left[\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}) \cup \hat{\mathcal{O}}') = \mathcal{I}'\right] \cdot \Pr\left[\mathcal{L}'(\hat{\mathcal{O}}) = \hat{\mathcal{O}}'\right] \\ &= \sum_{\hat{\mathcal{O}}' \subseteq \hat{\mathcal{O}}} \Pr\left[\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}) \cup \hat{\mathcal{O}}') = \mathcal{I}', \mathcal{L}'(\hat{\mathcal{O}}) = \hat{\mathcal{O}}'\right] \\ &= \Pr\left[\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}) \cup \mathcal{L}'(\hat{\mathcal{O}})) = \mathcal{I}'\right]. \end{aligned}$$

The final equation is exactly the i^{th} entry of the PD of $\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}) \cup \mathcal{L}'(\hat{\mathcal{O}}))$ with $i = \tilde{\mathcal{I}}'$. □

Probe distribution ordering

Before giving PDT composition results, we have to describe a partial order "$\dot{\leq}$" for probe distributions and probe distribution tables, and give a few properties of these orders. The high-level idea is that $\mathbf{p}$ is "larger" than $\mathbf{p}'$ (denoted $\mathbf{p} \dot{\geq} \mathbf{p}'$) if $\mathcal{L}$ gives more information than $\mathcal{L}'$. In other words, $\mathbf{p} \dot{\geq} \mathbf{p}'$ if we can simulate $\mathcal{L}'(\mathcal{W})$ with $\mathcal{L}(\mathcal{W})$, where $\mathcal{L}$ (resp., $\mathcal{L}'$) is the leakage process associated to $\mathbf{p}$ (resp., $\mathbf{p}'$).

Definition 44 (Partial order for distributions)**.** *For a set of wires $\mathcal{W}$, let $\mathcal{L}$ and $\mathcal{L}'$ be leakage processes with PDs $\mathbf{p}$ and $\mathbf{p}'$. We say that $\mathbf{p}$ is larger than $\mathbf{p}'$ and write $\mathbf{p} \dot{\geq} \mathbf{p}'$ iff $\mathcal{L}'$ is simulatable by $\mathcal{L}$, that is, if there exists a probabilistic algorithm $\mathcal{S}$ that satisfies $\mathcal{S}(\mathcal{X}) \subseteq \mathcal{X}$ such that the distribution of $\mathcal{L}'(\mathcal{W})$ and $\mathcal{S}(\mathcal{L}(\mathcal{W}))$ are equal.*

On the one hand, this relation is reflexive, antisymmetric, and transitive. Let $\mathbf{p}$, $\mathbf{p}'$ and $\mathbf{p}''$ be three PDs, it holds:

- $\mathbf{p} \dot{\geq} \mathbf{p}$, since we can always use the identity as simulator.
- If we know $\mathbf{p} \dot{\geq} \mathbf{p}'$ and $\mathbf{p} \dot{\leq} \mathbf{p}'$, there exists $\mathcal{S}$ and $\mathcal{S}'$ such that the distributions of $\mathcal{L}(\mathcal{W})$ and $\mathcal{S}'(\mathcal{S}(\mathcal{L}(\mathcal{W})))$ are equal. Since $\mathcal{S}'(\mathcal{S}(\mathcal{X})) \subseteq \mathcal{X}$ for any $\mathcal{X} \subseteq \mathcal{W}$, the equality of distributions implies $\mathcal{S}'(\mathcal{S}(\mathcal{X})) = \mathcal{X}$, and therefore $\mathcal{S}'(\mathcal{X}) = \mathcal{S}(\mathcal{X}) = \mathcal{X}$. As a result, $\mathcal{L}' = \mathcal{L}$, and therefore $\mathbf{p} = \mathbf{p}'$.
- If it holds that $\mathbf{p} \dot{\geq} \mathbf{p}'$ and $\mathbf{p}' \dot{\geq} \mathbf{p}''$, there exists a simulator $\mathcal{S}'$ that simulates the process defined by $\mathbf{p}'$ with the process defined by $\mathbf{p}$, and a simulator $\mathcal{S}''$ that does the same for $\mathbf{p}''$ and $\mathbf{p}'$. Hence,

$\mathcal{S} := \mathcal{S}' \circ \mathcal{S}''$ simulates the process defined by $\mathbf{p}''$ with the process of $\mathbf{p}$, and it follows $\mathbf{p} \dot{\geq} \mathbf{p}''$.

On the other hand, the order is only partial since it can happen that we have two leakage processes such that for both processes there exists no simulator to simulate the other.

The partial order for PDs is respected by linear combinations:

Proposition 34. *Let* $(\mathbf{p}_i)_{i=0,\dots,k-1}$, $\left(\mathbf{p}_i'\right)_{i=0,\dots,k-1}$ *be PDs such that* $\mathbf{p}_i \dot{\geq} \mathbf{p}_i'$ *for all* i. *Let* $(\alpha_i)_{i=0,\dots,k-1}$ *be such that* $0 \leq \alpha_i \leq 1$ *for all* i *and* $\sum_{i=0}^{k-1} \alpha_i = 1$. *If we denote* $\mathbf{p} = \sum_{i=0}^{k-1} \alpha_i \mathbf{p}_i$ *and* $\mathbf{p}' = \sum_{i=0}^{k-1} \alpha_i \mathbf{p}_i'$, *then* $\mathbf{p}$ *and* $\mathbf{p}'$ *are PDs and furthermore,* $\mathbf{p} \dot{\geq} \mathbf{p}'$.

Proof. Let $\mathcal{W}$ be a set of wires whose associated random processes $(\mathcal{L}_i)_{i=0,\dots,k-1}$ and $\left(\mathcal{L}_i'\right)_{i=0,\dots,k-1}$ have $(\mathbf{p}_i)_{i=0,\dots,k-1}$ and $\left(\mathbf{p}_i'\right)_{i=0,\dots,k-1}$ as PDs. Further, let $\mathcal{S}^i$ be such that $\mathcal{S}^i(\mathcal{L}_i(\mathcal{W}))$ has the same distribution as $\mathcal{L}_i'$. Let $\mathcal{L}$ be such that

$$\Pr\left[\mathcal{L}(\mathcal{W}) = \mathcal{W}'\right] = \sum_{i=0}^{k-1} \alpha_i \Pr\left[\mathcal{L}_i(\mathcal{W}) = \mathcal{W}'\right],$$

and similarly for $\mathcal{L}'$. Firstly, $\mathcal{L}$ and $\mathcal{L}'$ are well-defined: the probabilities given above are non-negative and sum to 1. Next, the PD of $\mathcal{L}$ (resp. $\mathcal{L}'$) is $\mathbf{p}$ (resp. $\mathbf{p}'$). Finally, we build the simulator $\mathcal{S}$. Let $\mathcal{L}''$ be a random process that, on input $\mathcal{W}$, selects randomly $i \in [\![0, k[\![$ (such that the probability of taking the value i is α_i), and outputs $\mathcal{S}^i(\mathcal{L}_i(\mathcal{W}))$. Then, let $\mathcal{S}$ be a random process such that $\Pr[\mathcal{S}(\mathcal{W}'') = \mathcal{W}'] = \Pr[\mathcal{L}'' = \mathcal{W}' | \mathcal{L} = \mathcal{W}'']$ for all $\mathcal{W}', \mathcal{W}'' \subseteq \mathcal{W}$. We observe that for all $\mathcal{W}' \subseteq \mathcal{W}$,

$$\begin{aligned}
\Pr[\mathcal{S}(\mathcal{L}) = \mathcal{W}'] &= \sum_{\mathcal{W}'' \subseteq \mathcal{W}} \Pr[\mathcal{S}(\mathcal{W}'') = \mathcal{W}'] \cdot \Pr[\mathcal{L} = \mathcal{W}''] \\
&= \sum_{\mathcal{W}'' \subseteq \mathcal{W}} \Pr[\mathcal{L}'' = \mathcal{W}' | \mathcal{L} = \mathcal{W}''] \cdot \Pr[\mathcal{L} = \mathcal{W}''] \\
&= \Pr[\mathcal{L}'' = \mathcal{W}'] \,.
\end{aligned}$$

Since $\mathcal{L}''$ has the same distribution as $\mathcal{L}'$, this means that $\Pr[\mathcal{S}(\mathcal{L}) = \mathcal{W}'] = \Pr[\mathcal{L}' = \mathcal{W}']$. □

Let us now extend the partial order of PDs to the whole PDT, with the same meaning: if $\mathbf{PDT}_{\mathsf{G}_0} \dot{\leq} \mathbf{PDT}_{\mathsf{G}_1}$, the amount of information leaked in G_0 is less than the information leaked in G_1.

Definition 45 (Partial order for PDT's)**.** *Let* $\mathbf{A}, \mathbf{B} \in [0,1]^{2^{|\mathcal{I}|} \times 2^{|\mathcal{O}|}}$ *be two PDTs, we write*

$$\mathbf{A} \dot{\leq} \mathbf{B}$$

if for any PD $\mathbf{p} \in [0,1]^{2^{|\mathcal{O}|}}$ *it holds* $\mathbf{A} \cdot \mathbf{p} \dot{\leq} \mathbf{B} \cdot \mathbf{p}$.

As shown in Proposition 33, $\mathbf{A} \cdot \mathbf{p}$ and $\mathbf{B} \cdot \mathbf{p}$ are PDs, therefore the partial order of PDTs is well-defined. The order of PDs and PDTs compose, but we need a small lemma to prove it.

Lemma 7. *Let* $\mathcal{W}$ *be a set of wires and let* f *and* g *be functions such that* $f(\mathcal{X}) \subseteq \mathcal{X}$ *and* $g(\mathcal{X}) \subseteq \mathcal{X}$ *for any* $\mathcal{X} \subseteq \mathcal{W}$*. Let* $\mathcal{L}$ *be a probabilistic process for* $\mathcal{W}$*, and let* $\mathbf{p}$ *(respectively* $\mathbf{p}'$*) be the PD of* $f(\mathcal{L}(\mathcal{W}))$ *(resp.* $f(\mathcal{L}(\mathcal{W}))$*). If* $f(\mathcal{X}) \subseteq g(\mathcal{X})$ *for any* $\mathcal{X} \subseteq \mathcal{W}$*, then* $\mathbf{p} \dot{\leq} \mathbf{p}'$.

Proof. Let us build a simulator $\mathcal{S}$ such that for any $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{W}$, $\Pr[\mathcal{S}(\mathcal{Y}) = \mathcal{X}] = \Pr[f(\mathcal{L}(\mathcal{W})) = \mathcal{X} | g(\mathcal{L}(\mathcal{W})) = \mathcal{Y}]$ (and $\mathcal{S}$ is independent of $\mathcal{L}$). Therefore, for any $\mathcal{X} \subseteq \mathcal{W}$, applying the law of total probability gives

$$\Pr[f(\mathcal{L}(\mathcal{W})) = \mathcal{X}] = \sum_{\mathcal{Y} \subseteq \mathcal{W}} \Pr[\mathcal{S}(\mathcal{Y}) = \mathcal{X}] \cdot \Pr[g(\mathcal{L}(\mathcal{W})) = \mathcal{Y}] \,.$$

Then, since $\mathcal{S}$ is independent of $\mathcal{L}$,

$$\Pr[\mathcal{S}(\mathcal{Y}) = \mathcal{X}] = \Pr[\mathcal{S}(\mathcal{Y}) = \mathcal{X} | g(\mathcal{L}(\mathcal{W})) = \mathcal{Y}]$$

and therefore

$$\sum_{\mathcal{Y} \subseteq \mathcal{W}} \Pr[\mathcal{S}(\mathcal{Y}) = \mathcal{X}] \cdot \Pr[g(\mathcal{L}(\mathcal{W})) = \mathcal{Y}] = \Pr[\mathcal{S}(g(\mathcal{L}(\mathcal{W}))) = \mathcal{X}] \,,$$

which shows that $\mathcal{S}$ satisfies Definition 44. □

Proposition 35. *Let* $\mathbf{A}, \mathbf{B} \in [0,1]^{2^{|\mathcal{I}|} \times 2^{|\mathcal{O}|}}$ *be two PDTs, and* $\mathbf{p}, \mathbf{p}' \in [0,1]^{2^{|\mathcal{O}|}}$ *be two PDs. If* $\mathbf{A} \dot{\leq} \mathbf{B}$ *and* $\mathbf{p} \dot{\leq} \mathbf{p}'$*, then* $\mathbf{A} \cdot \mathbf{p} \dot{\leq} \mathbf{B} \cdot \mathbf{p}'$.

Proof. We first show that $\mathbf{A} \cdot \mathbf{p} \dot{\leq} \mathbf{A} \cdot \mathbf{p}'$ By Definition 44, there exists a set of probes $\mathcal{W}$, a leakage process $\mathcal{L}'$ and a probabilistic algorithm $\mathcal{S}$ such that $\mathbf{p}'$ is the PD of $\mathcal{L}'(\mathcal{W})$ and $\mathbf{p}$ is the PD of $\mathcal{S}(\mathcal{L}'(\mathcal{W}))$. Using Proposition 33, we get that $\mathbf{A} \cdot \mathbf{p}'$ is the PD of $\mathcal{S}^{\mathsf{G}}(\mathcal{A}' \cup \mathcal{X})$ and $\mathbf{A} \cdot \mathbf{p}$ is the PD of $\mathcal{S}^{\mathsf{G}}(\mathcal{A}' \cup \mathcal{S}(\mathcal{X}))$ where $\mathcal{A}' \xleftarrow{\$} \mathcal{L}_p(\mathcal{W})$ and $\mathcal{X} \xleftarrow{\$} \mathcal{L}'(\mathcal{W})$.

The input set required by a simulator can only grow (or stay constant) if it has to simulate additional (output) leakage, hence $\mathcal{S}^{\mathsf{G}}(\mathcal{A}' \cup \mathcal{S}(\mathcal{X})) \subseteq \mathcal{S}^{\mathsf{G}}(\mathcal{A}' \cup \mathcal{X})$. Therefore, we can apply Lemma 7 to get $\mathbf{A} \cdot \mathbf{p} \dot{\leq} \mathbf{A} \cdot \mathbf{p}'$.

Next, by definition of $\mathbf{A} \dot{\leq} \mathbf{B}$, we have $\mathbf{A} \cdot \mathbf{p}' \dot{\leq} \mathbf{B} \cdot \mathbf{p}'$, which concludes the proof. □

Corollary 6 (PDT order is column-wise)**.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be PDTs, with columns* $(\mathbf{a}_i)_{i=0,\dots,|\mathcal{O}|-1}$ *and* $\left(\mathbf{b}'_i\right)_{i=0,\dots,|\mathcal{O}|-1}$ *respectively. Then,* $\mathbf{A} \dot{\geq} \mathbf{B}$ *iff* $\mathbf{a}_i \dot{\geq} \mathbf{b}_i$ *for all* $i \in [\![0, |\mathcal{O}|[\![$.

Proof. If $\mathbf{A} \dot{\geq} \mathbf{B}$, then for any $i \in [\![0, |\mathcal{O}|[\![$, let $\mathbf{e}$ be such that $e_j = 1$ if $i = j$ and $e_j = 0$ otherwise. Since $\mathbf{e}$ is a PD, Proposition 35 implies $\mathbf{p}_i = \mathbf{A} \cdot \mathbf{e} \dot{\geq} \mathbf{B} \cdot \mathbf{e} = \mathbf{p}'_i$.

In the other direction, let use assume that $\mathbf{a}_i \dot{\geq} \mathbf{b}_i$, for all i. Then for any PD α (whose elements are denoted α_i), $\mathbf{A} \cdot \alpha$ and $\mathbf{B} \cdot \alpha$ are linear combinations of $\mathbf{a}_i$ and $\mathbf{b}_i$ with coefficients α_i, for which Proposition 34 applies. Therefore, $\mathbf{A} \cdot \alpha \dot{\geq} \mathbf{B} \cdot \alpha$. □

This implies that PDT ordering is preserved through matrix product.

Corollary 7. *Let* $\mathbf{A}$*,* $\mathbf{B}$*,* $\mathbf{C}$*,* $\mathbf{D}$ *be PDTs. If* $\mathbf{A} \dot{\leq} \mathbf{B}$ *and* $\mathbf{C} \dot{\leq} \mathbf{D}$*, then* $\mathbf{A} \cdot \mathbf{C} \dot{\leq} \mathbf{B} \cdot \mathbf{D}$.

Proof. Let us denote by $\mathbf{X}_{*,i}$ the $(i+1)$-th column of a matrix $\mathbf{X}$. Then, for all $i \in [\![0, |\mathcal{O}|[\![$, $(\mathbf{A} \cdot \mathbf{C})_{*,i} = \mathbf{A} \cdot \mathbf{C}_{*,i}$ and $(\mathbf{B} \cdot \mathbf{D})_{*,i} = \mathbf{B} \cdot \mathbf{D}_{*,i}$. By Corollary 6, $\mathbf{C} \dot{\leq} \mathbf{D}$ implies $\mathbf{C}_{*,i} \dot{\leq} \mathbf{D}_{*,i}$ for all i. Therefore, from Proposition 35, we get $\mathbf{A} \cdot \mathbf{C}_{*,i} \dot{\leq} \mathbf{B} \cdot \mathbf{D}_{*,i}$. Using Corollary 6 again gives $\mathbf{A} \cdot \mathbf{C} \dot{\leq} \mathbf{B} \cdot \mathbf{D}$. □

Finally, we relate the partial order for PDs and PDTs to the security level.

Proposition 36 (Security level bound from PDT bound)**.** *Let* s *be the security level of a gadget* G *with respect to a set of input shares* $\mathcal{I}'$*. Let* $\boldsymbol{PDT}$ *be the PDT of* G *and let* $\boldsymbol{PDT}'$ *be a PDT. If* $\boldsymbol{PDT}' \dot{\geq} \boldsymbol{PDT}$*, then* $\mathbf{e}^T \cdot \boldsymbol{PDT}' \cdot \mathbf{p}_\varnothing \geq s$*, where* $\mathbf{e}$ *is defined as in Proposition 32.*

Proof. Using Proposition 32, we know that $\mathbf{e}^T \cdot \mathbf{PDT} \cdot \mathbf{p}_\varnothing \geq s$. With Proposition 35, we know that $\mathbf{PDT}' \cdot \mathbf{p}_\varnothing \dot{\geq} \mathbf{PDT} \cdot \mathbf{p}_\varnothing$. Let $\mathcal{L}$ (respectively $\mathcal{L}'$) be the random process associated to $\mathbf{PDT} \cdot \mathbf{p}_\varnothing$ (resp. $\mathbf{PDT}' \cdot \mathbf{p}_\varnothing$), and let $\mathcal{S}$ be the simulator that simulates $\mathcal{L}$ from $\mathcal{L}'$. By definition of $\mathcal{S}$, $\Pr\left[\mathcal{I}' \subseteq \mathcal{S}(\mathcal{L}'(\mathcal{I}))\right] \leq \Pr\left[\mathcal{I}' \subseteq \mathcal{L}'(\mathcal{I})\right]$. Since $\mathcal{S}$ simulates $\mathcal{L}(\mathcal{I})$, $\Pr\left[\mathcal{I}' \subseteq \mathcal{S}(\mathcal{L}'(\mathcal{I}))\right] = \Pr\left[\mathcal{I}' \subseteq \mathcal{L}(\mathcal{I})\right]$, which leads to $\mathbf{e}^T \cdot \mathbf{PDT} \cdot \mathbf{p}_\varnothing = \Pr\left[\mathcal{I}' \subseteq \mathcal{L}(\mathcal{I})\right] \leq \Pr\left[\mathcal{I}' \subseteq \mathcal{L}'(\mathcal{I})\right] = \mathbf{e}^T \cdot \mathbf{PDT}' \cdot \mathbf{p}_\varnothing$. □

Composition rules

In this section, we give the two main composition theorems for the PDT of parallel and sequential gadget compositions. Next, we show how the compositions theorems can be used to compute PDTs for larger composite gadgets and illustrate our results on the AES S-box example.

Theorem 4 (PDT parallel composition)**.** *Let* G_0 *and* G_1 *be two gadgets with* $\boldsymbol{PDT}_{\mathsf{G}_0}$ *and* $\boldsymbol{PDT}_{\mathsf{G}_1}$ *as PDTs. Further, let* $\mathsf{G} = \mathsf{G}_0 \| \mathsf{G}_1$ *with* $\boldsymbol{PDT}_{\mathsf{G}}$*. It holds that*

$$\boldsymbol{PDT}_{\mathsf{G}} = \boldsymbol{PDT}_{\mathsf{G}_0} \otimes \boldsymbol{PDT}_{\mathsf{G}_1} \ ,$$

where $\otimes$ *denotes the matrix outer product.*

Proof. Let $\mathcal{I}_0$, $\mathcal{I}_1$, $\hat{\mathcal{O}}_0$, and $\hat{\mathcal{O}}_1$ the input wires and output gates of G_0 and G_1, respectively. Hence, $\mathcal{I} = \mathcal{I}_0 \|_{(n)} \mathcal{I}_1$, $\hat{\mathcal{O}} = \hat{\mathcal{O}}_0 \|_{(m)} \hat{\mathcal{O}}_1$ are the input wires and output gates of G with $n = |\mathcal{I}_0|$ and $m = \left|\hat{\mathcal{O}}_0\right|$.

For any $\mathcal{I}' = \mathcal{I}'_0 \|_{(n)} \mathcal{I}'_1 \subseteq \mathcal{I}$ and $\hat{\mathcal{O}}' = \hat{\mathcal{O}}'_0 \|_{(m)} \hat{\mathcal{O}}'_1 \subseteq \hat{\mathcal{O}}$, we want to evaluate $\mathbf{PDT}_{\mathsf{G}}(\tilde{\mathcal{I}'}, \tilde{\hat{\mathcal{O}}'}) = \Pr\left[\mathcal{S}(\mathcal{L}_p(\mathsf{G}) \cup \hat{\mathcal{O}}') = \mathcal{I}'\right]$. Since $\mathcal{L}_p(\mathsf{G}) = \mathcal{L}_p(\mathsf{G}_0) \cup \mathcal{L}_p(\mathsf{G}_1)$, using Proposition 2 we get

$$\begin{aligned} &\Pr\left[\mathcal{S}(\mathcal{L}_p(\mathsf{G}) \cup \hat{\mathcal{O}}') = \mathcal{I}'\right] \\ &= \Pr\left[\mathcal{S}^{\mathsf{G}_0}\left(\mathcal{L}_p(\mathsf{G}_0) \cup \hat{\mathcal{O}}'_0\right) = \mathcal{I}'_0, \mathcal{S}^{\mathsf{G}_1}\left(\mathcal{L}_p(\mathsf{G}_1) \cup \hat{\mathcal{O}}'_1\right) = \mathcal{I}'_1\right] . \end{aligned}$$

Since $\mathcal{L}_p(\mathsf{G}_0)$ and $\mathcal{L}_p(\mathsf{G}_1)$ are independent, as well as $\mathcal{S}^{\mathsf{G}_0}$ and $\mathcal{S}^{\mathsf{G}_1}$, we get

$$\begin{aligned} &\Pr\left[\mathcal{S}(\mathcal{L}_p(\mathsf{G}) \cup \hat{\mathcal{O}}') = \mathcal{I}'\right] \\ &= \Pr\left[\mathcal{S}^{\mathsf{G}_0}\left(\mathcal{L}_p(\mathsf{G}_0) \cup \hat{\mathcal{O}}'_0\right) = \mathcal{I}'_0\right] \cdot \Pr\left[\mathcal{S}^{\mathsf{G}_1}\left(\mathcal{L}_p(\mathsf{G}_1) \cup \hat{\mathcal{O}}'_1\right) = \mathcal{I}'_1\right] \\ &= \mathbf{PDT}_{\mathsf{G}_0}(\tilde{\mathcal{I}'_0}, \tilde{\mathcal{O}'_0}) \cdot \mathbf{PDT}_{\mathsf{G}_1}(\tilde{\mathcal{I}'_1}, \tilde{\mathcal{O}'_1}) \ . \end{aligned}$$

Considering all entries $(\tilde{\mathcal{I}'}, \tilde{\mathcal{O}'})$ of $\mathbf{PDT}_{\mathsf{G}}$, we get $\mathbf{PDT}_{\mathsf{G}} = \mathbf{PDT}_{\mathsf{G}_1} \otimes \mathbf{PDT}_{\mathsf{G}_0}$. □

Remark. Theorem 4 can be generalized to any parallel composition of sub-circuits, even if those sub-circuits are not gadgets. For instance, a share-wise gadget with d shares is the parallel composition of d identical sub-circuits (a single addition gate for the addition gadget). The PDT of the addition gate $\mathbf{PDT}_{+}$ is given in Equation 5.6, therefore $\mathbf{PDT}_{\mathsf{G}_{+,d}}$, the PDT of the addition gadget, can be computed as

$$\mathbf{PDT}_{\mathsf{G}_{+,d}} = P\left(\bigotimes_{i=0}^{n-1} \mathbf{PDT}_{+}\right) ,$$

where P is a permutation that reorders the index of the input shares from $(\mathbf{x}_0, \mathbf{y}_0 \ldots, \mathbf{x}_{d-1}, \mathbf{y}_{d-1})$ to $(\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}, \mathbf{y}_0, \ldots, \mathbf{y}_{d-1})$. Such a reordering amounts to a shuffling of the rows of the PDT.

Theorem 5 (PDT sequential composition)**.** *Let* G_0 *(respectively* G_1*) be a gadget whose PDT is* $\boldsymbol{PDT}_{\mathsf{G}_0}$ *(resp.* $\boldsymbol{PDT}_{\mathsf{G}_1}$*), with* n_0 *(resp.* n_1*) input wires and* m_0 *(resp.* m_1*) output wires, such that* $m_0 = n_1$*. Further, let* $\mathsf{G} = \mathsf{G}_1 \circ \mathsf{G}_0$ *whose* $\boldsymbol{PDT}$ *is* $\boldsymbol{PDT}_{\mathsf{G}}$*. It holds that*

$$\boldsymbol{PDT}_{\mathsf{G}} \dot{\leq} \boldsymbol{PDT}_{\mathsf{G}_0} \cdot \boldsymbol{PDT}_{\mathsf{G}_1} \ .$$

Proof. Let $\mathcal{I}_0$, $\mathcal{I}_1$, $\hat{\mathcal{O}}_0$, $\hat{\mathcal{O}}_1$ the set of input wires and output gates of G_0 and G_1, respectively. This means that $\mathcal{I}_0$ and $\hat{\mathcal{O}}_1$ are the set of input wires and output gates of G. Let $\overline{\mathbf{PDT}} = \mathbf{PDT}_{\mathsf{G}_0} \cdot \mathbf{PDT}_{\mathsf{G}_1}$, we get for any $\mathcal{I}' \subseteq \mathcal{I}_0$ and $\hat{\mathcal{O}}' \subseteq \hat{\mathcal{O}}_1$

$$\begin{aligned}
&\overline{\mathbf{PDT}}(\tilde{\mathcal{I}}'', \tilde{\hat{\mathcal{O}}}') \\
&= \sum_{\hat{\mathcal{O}}'' \subseteq \hat{\mathcal{O}}_0} \Pr\left[\mathcal{S}^{\mathsf{G}_0}(\mathcal{L}_p(\mathsf{G}_0) \cup \hat{\mathcal{O}}'') = \mathcal{I}'\right] \cdot \Pr\left[\mathcal{S}^{\mathsf{G}_1}(\mathcal{L}_p(\mathsf{G}_1) \cup \hat{\mathcal{O}}') = \hat{\mathcal{O}}''\right] \\
&= \sum_{\hat{\mathcal{O}}'' \subseteq \hat{\mathcal{O}}_0} \Pr\left[\mathcal{S}^{\mathsf{G}_0}(\mathcal{L}_p(\mathsf{G}_0) \cup \hat{\mathcal{O}}'') = \mathcal{I}', \mathcal{S}^{\mathsf{G}_1}(\mathcal{L}_p(\mathsf{G}_1) \cup \hat{\mathcal{O}}') = \hat{\mathcal{O}}''\right] \\
&= \Pr\left[\mathcal{S}^{\mathsf{G}_0}\left(\mathcal{L}_p(\mathsf{G}_0) \cup \mathcal{S}^{\mathsf{G}_1}(\mathcal{L}_p(\mathsf{G}_1), \hat{\mathcal{O}}')\right) = \mathcal{I}'\right] \ ,
\end{aligned}$$

where the second equality is due to the independence of $\mathcal{L}_p(\mathsf{G}_0)$ and $\mathcal{L}_p(\mathsf{G}_1)$, as well as $\mathcal{S}^{\mathsf{G}_0}$ and $\mathcal{S}^{\mathsf{G}_1}$.

Further, for any $\hat{\mathcal{O}}' \subseteq \hat{\mathcal{O}}_1$, $\mathbf{PDT}_{\mathsf{G}}(\tilde{\hat{\mathcal{O}}}')$ is the PD of

$$\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}) \cup \hat{\mathcal{O}}') = \mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G}_0) \cup \mathcal{L}_p(\mathsf{G}_1) \cup \hat{\mathcal{O}}')$$

while $\overline{\mathbf{PDT}}(\tilde{\hat{\mathcal{O}}}')$ is the PD of

$$\mathcal{S}^{\mathsf{G}_0}\left(\mathcal{L}_p(\mathsf{G}_0) \cup \mathcal{S}^{\mathsf{G}_1}(\mathcal{L}_p(\mathsf{G}_1) \cup \hat{\mathcal{O}}')\right) \ .$$

Let $\mathcal{A}_0' \xleftarrow{\$} \mathcal{L}_p(\mathsf{G}_0)$ and $\mathcal{A}_1' \xleftarrow{\$} \mathcal{L}_p(\mathsf{G}_1)$, Proposition 3 gives

$$\mathcal{S}^{\mathsf{G}}\left(\mathcal{A}_0' \cup \mathcal{A}_1' \cup \hat{\mathcal{O}}'\right) \subseteq \mathcal{S}^{\mathsf{G}_0}\left(\mathcal{A}_0' \cup \mathcal{S}^{\mathsf{G}_1}\left(\mathcal{A}_1' \cup \hat{\mathcal{O}}'\right)\right) \ .$$

Therefore, using Lemma 7, $\mathbf{PDT}_{\mathsf{G}}(\tilde{\hat{\mathcal{O}}}') \dot{\leq} \overline{\mathbf{PDT}}(\tilde{\hat{\mathcal{O}}}')$. Finally, Corollary 6 implies $\mathbf{PDT}_{\mathsf{G}} \dot{\leq} \overline{\mathbf{PDT}}$. □

Corollary 8. *Let* $(\mathsf{G}_i)_{i \in [k]}$ *be gadgets that can be sequentially composed to form* $\mathsf{G} = \mathsf{G}_{k-1} \circ \cdots \circ \mathsf{G}_0$*. It holds that*

$$\boldsymbol{PDT}_{\mathsf{G}} \dot{\leq} \boldsymbol{PDT}_{\mathsf{G}_0} \cdot \ldots \cdot \boldsymbol{PDT}_{\mathsf{G}_{k-1}} .$$

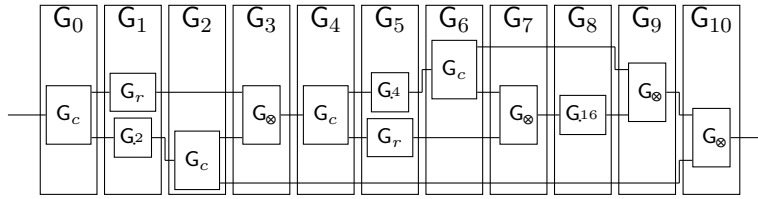


Figure 5.5: AES S-box circuit (using the implementation from [RP10]) as a serial composition of gadgets. The symbols G_c, G_r, $\mathsf{G}_\otimes$ and $\mathsf{G}_{.x}$ are respectively copy, refresh, multiplication and exponentiation to the power of x gadgets.

G_0	G_c	$\mathbf{PDT}_{\mathsf{G}_0} = \mathbf{PDT}_{\mathsf{G}_c}$
G_1	$\mathsf{G}_r \Vert \mathsf{G}_{.2}$	$\mathbf{PDT}_{\mathsf{G}_1} = \mathbf{PDT}_{\mathsf{G}_r} \otimes \mathbf{PDT}_{\mathsf{G}_{.2}}$
G_2	$\mathsf{ID} \Vert \mathsf{G}_c$	$\mathbf{PDT}_{\mathsf{G}_2} = \mathbf{PDT}_{\mathsf{ID}} \otimes \mathbf{PDT}_{\mathsf{G}_c}$
G_3	$\mathsf{G}_\otimes \Vert \mathsf{ID}$	$\mathbf{PDT}_{\mathsf{G}_3} = \mathbf{PDT}_{\mathsf{G}_\otimes} \otimes \mathbf{PDT}_{\mathsf{ID}}$
G_4	$\mathsf{G}_c \Vert \mathsf{ID}$	$\mathbf{PDT}_{\mathsf{G}_4} = \mathbf{PDT}_{\mathsf{G}_c} \otimes \mathbf{PDT}_{\mathsf{ID}}$
G_5	$\mathsf{G}_{.4} \Vert \mathsf{G}_r \Vert \mathsf{ID}$	$\mathbf{PDT}_{\mathsf{G}_5} = \mathbf{PDT}_{\mathsf{G}_{.4}} \otimes \mathbf{PDT}_{\mathsf{G}_r} \otimes \mathbf{PDT}_{\mathsf{ID}}$
G_6	$\mathsf{G}_c \Vert \mathsf{ID} \Vert \mathsf{ID}$	$\mathbf{PDT}_{\mathsf{G}_6} = \mathbf{PDT}_{\mathsf{G}_c} \otimes \mathbf{PDT}_{\mathsf{ID}} \otimes \mathbf{PDT}_{\mathsf{ID}}$
G_7	$\mathsf{ID} \Vert \mathsf{G}_\otimes \Vert \mathsf{ID}$	$\mathbf{PDT}_{\mathsf{G}_7} = \mathbf{PDT}_{\mathsf{ID}} \otimes \mathbf{PDT}_{\mathsf{G}_\otimes} \otimes \mathbf{PDT}_{\mathsf{ID}}$
G_8	$\mathsf{ID} \Vert \mathsf{G}_{.16} \Vert \mathsf{ID}$	$\mathbf{PDT}_{\mathsf{G}_8} = \mathbf{PDT}_{\mathsf{ID}} \otimes \mathbf{PDT}_{\mathsf{G}_{.16}} \otimes \mathbf{PDT}_{\mathsf{ID}}$
G_9	$\mathsf{G}_\otimes \Vert \mathsf{ID}$	$\mathbf{PDT}_{\mathsf{G}_9} = \mathbf{PDT}_{\mathsf{G}_\otimes} \otimes \mathbf{PDT}_{\mathsf{ID}}$
G_{10}	$\mathsf{G}_\otimes$	$\mathbf{PDT}_{\mathsf{G}_{10}} = \mathbf{PDT}_{\mathsf{G}_\otimes}$
$\mathsf{G}_{\text{S-box}}$	$\mathsf{G}_{10} \circ \mathsf{G}_9 \circ \ldots \circ \mathsf{G}_0$	$\mathbf{PDT}_{\text{S-box}} \dot{\leq} \mathbf{PDT}_{\mathsf{G}_0} \cdot \mathbf{PDT}_{\mathsf{G}_1} \cdot \ldots \cdot \mathbf{PDT}_{\mathsf{G}_{10}}$

Table 5.2: Composition of the AES S-box and its approximated PDT.

Proof. This is a direct consequence of Theorem 5 and Corollary 7. □

As an example for the usage of the PDT composition theorems, we consider the PDT of the AES S-box depicted in Figure 5.5, which is decomposed into a representation close to an extended sequential composition.[5] Its PDT is bounded by $\mathbf{PDT}_{\text{S-box}}$ defined in Table 5.2. We compute the S-box with the gadgets $\mathsf{G}_{.2}$, $\mathsf{G}_\otimes$, G_r, and G_c. In addition, we also use the identity gadget ID, and assume that it produces no leakage, hence its PDT is the identity matrix. As described in Table 5.2, the gadgets G_0-G_{10} are parallel compositions of the gadgets $\mathsf{G}_{.2}$, $\mathsf{G}_{.4}$, $\mathsf{G}_{.16}$, $\mathsf{G}_\otimes$, G_r, G_c, and ID (we can compute their PDTs using Theorem 4). Thus, $\mathsf{G}_{\text{S-box}}$ is a sequential composition of G_0-G_{10}. We can compute its PDT using Corollary 8, as shown in Table 5.2.

We conclude by noting that some well-known matrix product and tensor product distributive and associative properties mirror the properties of the gadget compositions (when the operations are well-defined):

[5] We do not use the full extended sequential composition for the sake of concision.

$$(\mathbf{A}\cdot\mathbf{B})\cdot\mathbf{C}=\mathbf{A}\cdot(\mathbf{B}\cdot\mathbf{C}) \qquad (\mathsf{G}_0\circ\mathsf{G}_1)\circ\mathsf{G}_2=\mathsf{G}_0\circ(\mathsf{G}_1\circ\mathsf{G}_2)$$
$$(\mathbf{A}\otimes\mathbf{B})\otimes\mathbf{C}=\mathbf{A}\otimes(\mathbf{B}\otimes\mathbf{C}) \qquad (\mathsf{G}_0\|\mathsf{G}_1)\|\mathsf{G}_2=\mathsf{G}_0\|(\mathsf{G}_1\|\mathsf{G}_2)$$
$$(\mathbf{A}\cdot\mathbf{B})\otimes(\mathbf{C}\cdot\mathbf{D})=(\mathbf{A}\otimes\mathbf{C})\cdot(\mathbf{B}\otimes\mathbf{D}) \qquad (\mathsf{G}_0\circ\mathsf{G}_1)\|(\mathsf{G}_2\circ\mathsf{G}_3)=(\mathsf{G}_0\|\mathsf{G}_2)\circ(\mathsf{G}_1\|\mathsf{G}_3)$$

This means that our composition theorems give the same result independently of the way we decompose a composite gadget. This gives us freedom to choose, e.g.,, the most efficient way when we deal with relatively large computations.

5.2.4 Practical PDT-based security bounds

In this section, we adapt the method of Section 5.2.1 to compute bounds for PDTs. We then show how to turn those bounds into gadget security levels using the PDT properties and composition theorems. We finally describe the tool that implements our methodology.

Bounding PDTs

Let us describe how to adapt the method of Section 5.2.1 to bound PDTs. That is, given a gadget G and its PDT $\mathbf{PDT}$, we want to generate an upper bound $\mathbf{PDT}^U$ such that $\mathbf{PDT}^U \geq \mathbf{PDT}$ with probability at least $1-\alpha$ (e.g., $1-10^{-6}$), and the $\geq$ operator defined for matrices and vectors as element-wise. We note that $\mathbf{PDT}^U$ is not a PDT: the sum of the elements in anyone of its columns may be larger than 1.

There are two main differences with the bound of Section 5.2.1: (1) we have to handle all possible cases for the probes on the output shares of the gadgets (i.e., all the columns of the PDT), and (2) we care about the full distribution of the input probes, not only the probability of successful attack.

The upper bound $\mathbf{PDT}^U$ can be computed by grouping probe sets by size (similarly to Equation (5.3)):

$$\mathbf{PDT}^U(\tilde{\mathcal{I}}',\tilde{\hat{\mathcal{O}}}') = \sum_{i=0}^{|\mathcal{W}|} p^i(1-p)^{|\mathcal{W}|-i}\cdot\left|\mathcal{W}^{(i)}\right|\cdot\mathbf{R}_i^U(\tilde{\mathcal{I}}',\tilde{\mathcal{O}}')$$

satisfies $\mathbf{PDT}^U(\tilde{\mathcal{I}}',\tilde{\hat{\mathcal{O}}}') \geq \mathbf{PDT}(\tilde{\mathcal{I}}',\tilde{\hat{\mathcal{O}}}')$ if

$$\mathbf{R}_i^U(\tilde{\mathcal{I}}',\tilde{\hat{\mathcal{O}}}') \geq \frac{\left|\left\{\mathcal{W}'\subseteq\mathcal{W}^{(i)}\text{ s.t. }\mathcal{S}^{\mathsf{G}}(\mathcal{L}_p(\mathsf{G})\cup\hat{\mathcal{O}}')=\mathcal{I}'\right\}\right|}{\left|\mathcal{W}^{(i)}\right|} \tag{5.7}$$

for all $i \in [\![0, |\mathcal{W}|[\![$. Therefore, if Equation 5.7 is satisfied for each $(\mathcal{I}', \hat{\mathcal{O}}', i)$ tuple with probability at least $1 - \alpha / \left((|\mathcal{W}| + 1)\, 2^{|\mathcal{I}| \cdot |\hat{\mathcal{O}}|} \right)$, then $\mathbf{PDT}^U \geq \mathbf{PDT}$ with probability at least $1 - \alpha$ (by the union bound).

The computation of all the elements $P_i^U(\tilde{\mathcal{I}}', \tilde{\hat{\mathcal{O}}}')$ can be performed identically to the computation of r_i^U in Section 5.2.1, except for changing the criterion for a Monte-Carlo sample $\mathcal{W}'$ to be counted as positive (i.e., be counted in s_i): $\mathcal{S}(\mathcal{W}' \cup \hat{\mathcal{O}}') = \mathcal{I}'$ (instead of $\delta_{\mathcal{W}'} = 1$). Furthermore, the algorithm can be optimized by running only one sampling for each $(i, \hat{\mathcal{O}}')$ pair: we take $t_{i,\hat{\mathcal{O}}'}$ samples, and we classify each sample $\mathcal{W}'$ according to $\mathcal{S}(\mathcal{W}' \cup \hat{\mathcal{O}}')$. This gives sample counts $s_{i,\hat{\mathcal{O}}',\mathcal{I}'}$ for all $\mathcal{I}' \subseteq \mathcal{I}$, and from there we can use Equation (5.4).[6]

Finally, we use the hybrid strategy of Algorithm 26, with the aforementioned modifications.[7] The computation of a statistical-only lower bound $\mathbf{PDT}^L$ is done in the same way, except that Equation (5.5) is used instead of Equation (5.4).

From PDT bound to security level bound.

Let us take positive matrices $A^U \geq A$ and $B^U \geq B$. It always holds that $A^U \otimes B^U \geq A \otimes B$ and $A^U \cdot B^U \geq A \cdot B$. Therefore, if we use PDT bounds in Corollary 8 (respectively, Theorem 4), we get as a result – denoted $\overline{\mathbf{PDT}}^U$ and computed as $A^U \cdot B^U$ (resp., $A^U \otimes B^U$) – a corresponding bound for the composite PDT – denoted $\overline{\mathbf{PDT}}$ and computed as $A \cdot B$ (resp., $A \otimes B$): $\overline{\mathbf{PDT}}^U \geq \overline{\mathbf{PDT}} \dot{\geq} \mathbf{PDT}$. Then, if we use $\overline{\mathbf{PDT}}^U$ in the formula for the computation of the security level (Proposition 36) instead of $\overline{\mathbf{PDT}}$, we get

$$s^U = \mathbf{e}^T \cdot \overline{\mathbf{PDT}}^U \cdot \mathbf{p}_\varnothing \geq \mathbf{e}^T \cdot \overline{\mathbf{PDT}} \cdot \mathbf{p}_\varnothing \geq s.$$

We compute the statistical-only lower bound s^L similarly. One should however keep in mind that $s^L \leq s$ does not hold in general, since Proposition 36 and the sequential composition theorem only guarantee an upper bound (in addition to the non-tightness coming from the maskVerif algorithm). Again, the statistical-only lower bound is however

[6] The random variables $s_{i,\hat{\mathcal{O}}',\mathcal{I}'}$ for all $\mathcal{I}' \subseteq \mathcal{I}$ are not mutually independent, hence the derived bounds are not independent of each other, but this is not an issue since the union bound does not require independent variables.

[7] And additionally the change of the condition $s_i < N_t$ by $s_{i,\hat{\mathcal{O}}'\mathcal{I}} < N_t$. The rationale for this condition is that, intuitively, if we have many "worst-case" samples, then we should have a sufficient knowledge of the distribution $\left(P_i(\tilde{\mathcal{I}}', \tilde{\hat{\mathcal{O}}}') \right)_{\mathcal{I}' \subseteq \mathcal{I}}$.

useful for estimating the uncertainty on the security level that comes from the Monte-Carlo method: if there is a large gap between s^L and s^U, increasing the number of samples in the Monte-Carlo sampling can result in a better s^U (on the other hand, s^L gives a limit on how much we can hope to reduce s^U by increasing the number of samples).

Tool

We implemented the computation of the above bounds in the open-source tool STRAPS (Sampled Testing of the RAndom Probing Security). This tool contains a few additional algorithmic optimizations that do not change the results but significantly reduce the execution time (e.g., we exploit the fact that, in some circuits, many wires carry the same value, and we avoid to explicitly compute PDTs of large composite gadgets to reduce memory usage). Regarding performance, for the computation of the security of the AES S-box (see Figure 5.9), almost all the execution time goes into computing the PDT of the ISW multiplication gadgets. Computing the PDTs of the other gadgets is much faster as they are smaller, and computing the composition takes a negligible amount of time (less than 1 %). The total running time for the AES S-box is less than 5 s for 1, 2 and 3 shares, 30 s for 4 shares, 3 min for 5 shares, and 33 h for 6 shares on a 24-core computer (dual 2.3 GHz Intel(R) Xeon(R) CPU E5-2670 v3).

STRAPS presents a few similarities with VRAPS [Bel+20a]. While STRAPS mainly computes PDT bounds and VRAPS computes random probing expandability bounds, both metrics relate to the random probing security of a gadget, and both tools include their own implementation of the maskVerif simulatability set algorithm. (The implementations use different heuristics, but the core principle explained in Section 3.5 remain the same.) The main differences between these tools are twofold. First, STRAPS uses a mix of Monte-Carlo sampling and full exploration of the sets of probes, whereas VRAPS does only full exploration. Second, STRAPS computes and uses the simulatability set for a given set of internal and output probes, while VRAPS only stores whether the size of the simulatability set exceeds a given threshold. Thanks to this weaker requirement, VRAPS is able to exploit the set exploration algorithm of maskVerif, which accelerates the full exploration of the sets of probes by avoiding an exhaustive enumeration of all subsets [Bar+19].

5.2.5 Experiments & SOTA comparison

We finally illustrate how to use our PDT bounding tool and the PDT composition theorems in order to bound the security of larger circuits, and to extract useful intuitions about the trade-off between the number of shares and level of noise required to reach a given security level. We also compare our results with previous works by Dziembowski et al. [DFZ19] and Belaïd et al. [Bel+20a; BRT21].

We begin by evaluating the impact of using composition theorems instead of a direct security evaluation. In Section 5.2.2, we concluded that directly analyzing the security of even a single multiplication gadget in the random probing model tightly is computationally intensive. On Figure 5.6, we show the security of a slightly more complex gadget $\mathsf{ISW}(\mathbf{x}, \mathsf{SNI\text{-}Ref}(\mathbf{x}^2))$ evaluated as either the composition of four gadgets[8] (a copy gadget, a squaring, an SNI refresh and an ISW multiplication), or as a single gadget (we call it integrated evaluation). We can see that when the gadget becomes large ($d = 5$) and for a similar computational complexity, the results for the PDT composition are statistically tighter thanks to the lower size of its sub-gadgets. We also observe that, when upper and lower bounds converge, the security level computed from PDT composition is close to the one computed by the integrated evaluation, although the latter one is slightly better. We conclude that the PDT composition technique can provide useful results in practically relevant contexts where we build gadget compositions for which the integrated evaluation is not satisfying.

Next, we investigate different refreshing strategies when computing the $\mathbf{x}^3$ operation with an ISW multiplication gadget. Namely, we compare the situation with no refreshing which is known to be insecure in the threshold probing model [Cor+13], the simple refreshing with linear randomness complexity which does not offer strong composability guarantees, and an SNI refresh gadget from [Cas+21b]. The results are illustrated in Figure 5.7. In the first case (with no refreshing), we observe the well-known division by two of the statistical security order

[8] For each gadget, we use the $\alpha = 10^{-6}$ as confidence level. The overall confidence level for each bound is then $1-(1-\alpha)^{n_G} \leq n_G\alpha$, where n_G is the number of distict gadget PDTs bounded with confidence α (n_G does not depend on the number of times the gadgets are used, hence it does not scale with the size of the circuit). If ever n_G is large (that is, many types of gadgets are used), $n_G\alpha$ can be kept small by reducing α (which is easy, see Footnote 3 page 104). In all our experiments, $n_G \leq 2$ since we only bound the PDTs of the ISW multiplication and SNI refresh gadgets, while the PDTs we use for the copy and squaring gadgets are exact, and not bounds.

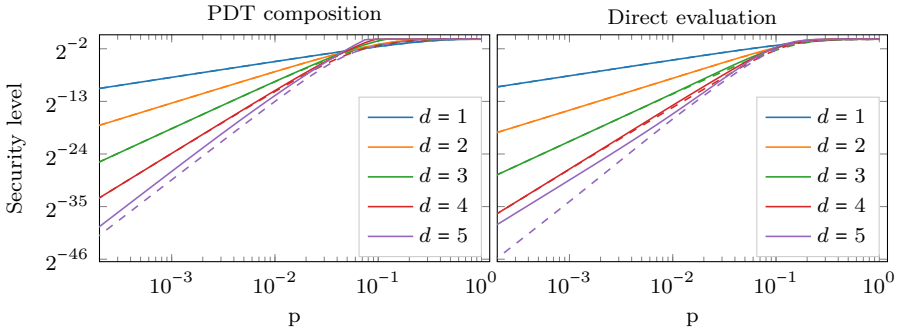


Figure 5.6: Security of a cubing gadget $\mathsf{ISW}(\mathrm{x}, \mathsf{SNI\text{-}Ref}(\mathrm{x}^2))$. The left plot comes from PDT composition while the right plot is a direct security evaluation of the full circuit as a single gadget. The continuous line is an upper bound, while the dashed line is the stat-only lower bound. $N_{max} = 2 \times 10^6$, $N_t = 1000$.

(reflected by the slope of the security curves in the asymptotic region where the noise is sufficient and curves become linear): the security level is asymptotically proportional to $p^{\lceil (d-1)/2 \rceil}$. On the other side of the spectrum, the composition with an SNI refresh guarantees a statistical security order of $d - 1$. Finally, the most interesting case is the one of the simple refresh gadget, for which we observe a statistical security order reduction for $d \geq 3$, of which the impact may remain small for low noise levels. For instance, we can see that for $p \geq 2 \times 10^{-3}$, the curves for the simple and the SNI refresh gadgets are almost the same, with the security order reduction becoming more and more apparent only for lower values of p. So this analysis provides us with a formal quantitative understanding of a gadget's security level which, for example, suggests that depending on the noise levels, using SNI gadgets may not always be needed.

We extend this analysis of a simple gadget to the case of a complete AES S-box in Figure 5.8. All the previous observations remain valid in this case as well. Furthermore, this figure confirms that our results get close to the ones reported for concrete worst-case attacks in [DFS19]. Namely, already for the (low) number of shares and (practical) levels of noise we consider, we observe a statistical security order of $d - 1$ for a practically relevant (AES S-box) circuit.[9]

Eventually, we compare our bounds with state-of-the-art results for the

[9] To make the results more easily comparable, one can just assume connect the leakage probability with the mutual information of [DFS19] by just assuming that the mutual information per bit (i.e., when the unit is the field element) equals p.

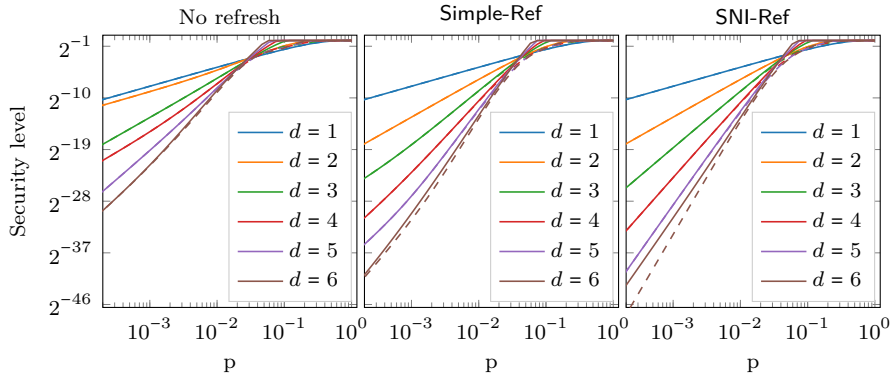


Figure 5.7: Security of the cubing ISW(x, Ref(x^2)), where Ref is identity (no refreshing), Simple-Ref, or SNI-Ref gadget. The continuous line is an upper bound, while the dashed line is the stat-only lower bound. $N_{max} = 10^8$, $N_t = 100$.

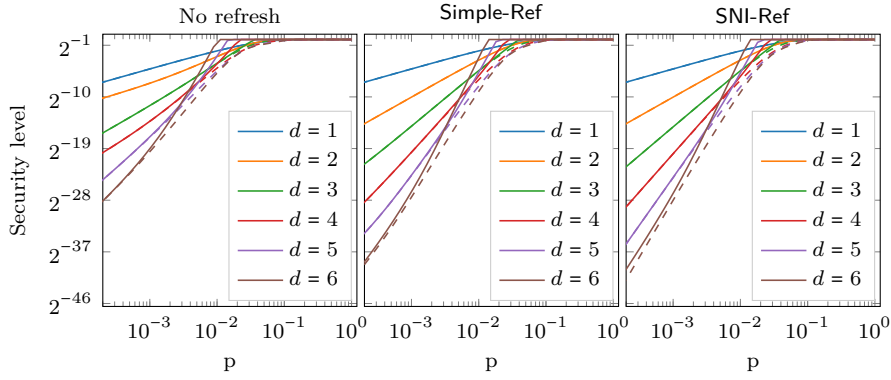


Figure 5.8: Security of the non-linear part of an AES S-box in $\mathbb{F}_{256}$, where Ref is either an identity (no refreshing), the Simple-Ref gadget, or the SNI-Ref gadget. The continuous line is an upper bound, while the dashed line is the stat-only lower bound. $N_{max} = 10^8$, $N_t = 100$.

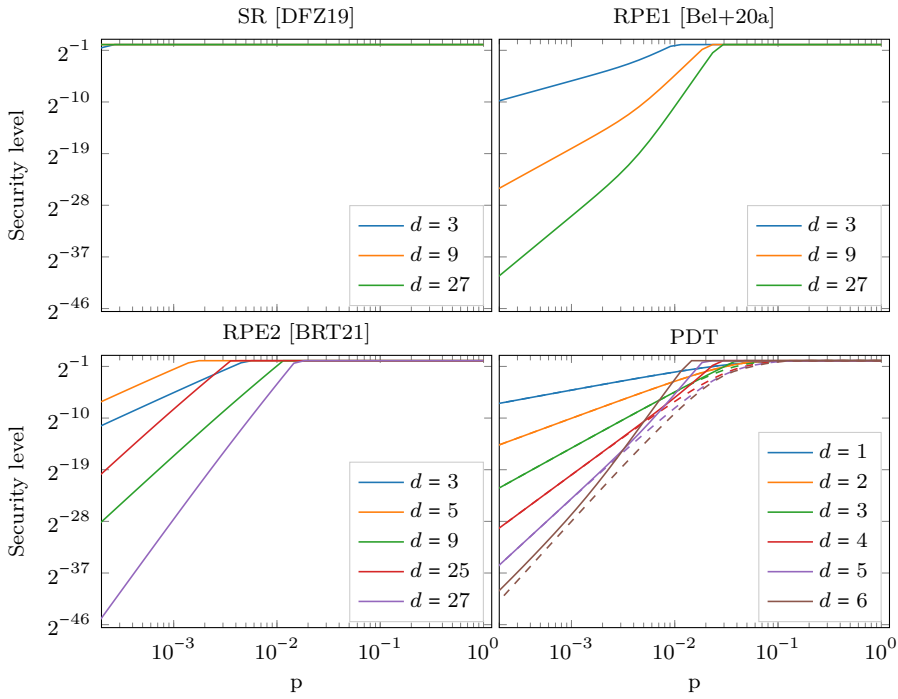


Figure 5.9: Security of the non-linear part of an AES S-box in $\mathbb{F}_{256}$, based on the best result of each paper. For the PDT, we use a SNI refresh gadget to ensure $d-1$-threshold probing security. All the circuits have a size $\mathcal{O}(n^2)$ with similar constant factor.

non-linear part of the AES S-box in Figure 5.9, in order to highlight that such tight results were not available with existing solutions. Precisely, we compare our results with the works that provide the best bounds in the low-noise region that we consider: the Simple Refreshing (SR) strategy [DFZ19], and the first (RPE1) [Bel+20a] and second (RPE2) [BRT21] sets of gadgets from the Random Probing Expansion strategy of Belaïd et al. We see that amongst the previous works we consider here, RPE2 with 27 shares achieves the best maximum tolerated leakage probability and statistical security order. Our PDT-based analysis of the SNI-refreshed AES S-box with the ISW multiplication achieves a similar security level with only 6 shares. In this last experiment, the number of shares d is an indicator for the circuit size since all schemes have a circuit size in $\mathcal{O}(d^2)$. So we conclude that our results enable a significant improvement of the provable security claims of practical masked circuits in the random probing model.

5.3 Local random probing model

While the evaluation technique introduced in Section 5.2 provides a tighter bound compared to the state of the art, it suffers from two main limitations. First, the computation of the PDT of a gadget is computationally expensive, which limits it to small gadgets (e.g., up to $d = 6$ for the ISW multiplication). It is therefore necessary to use composition theorems to analyze larger gadgets, which leads to the second limitation: the sequential composition of PDT is not tight (Theorem 5 provides only an inequality). In this section, we mitigate these limitations by building a more efficient evaluation technique for computing the security of a circuit in the random probing model. This efficiency gain comes at the expense of changing the leakage model: we evaluate security in the so-called local random probing model (LRPM) [Guo+20], which can be viewed as a heuristic version of the random probing model that takes into account computationally bounded adversaries.

Security analysis LRPM relies on bounding the mutual information that can be extracted by an adversary with a soft analytical side-channel attack (SASCA), using the belief propagation (BP) algorithm. This methodology is motivated by the state of the art in side-channel attacks: SASCA is the most powerful attack technique known against masked implementations [BS21; BCS21].

5.3.1 SASCA and BP

Given some side-channel leakage L, the optimal attack would be to compute $\Pr[K = k|L]$ for all k's, where K is the sensitive variable. Assuming knowledge of a perfect leakage model $\Pr[L|K]$ (and a uniform prior on the key distribution), one can compute these probabilities through the Bayes rule. This however has a prohibitive (computational) cost.[10] The SASCA approach [VGS14] expresses the target implementation as a code (in the meaning of coding theory) in which information on every intermediate variable extracted from the leakage, then the Belief Propagation (BP) algorithm is run to decode it, recovering the key.

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 and differences two elements:

[10] Even if K is restricted to an enumerable value, when masking is applied, the leakage model becomes $\Pr[L|(\mathbf{K}_0, \ldots, \mathbf{K}_{d-1})]$, therefore the computational cost grows exponentially in d [GS18].

$x = x_1 \pm x_2$, products of two elements: $x = x_1 \cdot x_2$ and bijections: $y = \mathsf{g}(x)$ 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. It 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.[11] 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 information at each node is combined to get the final estimated PDF. We refer to [VGS14] for the details.

5.3.2 Information propagation

SASCA has been analyzed in [Guo+20], which introduces a technique to bound its data complexity denoted as the Local Random Probing Model (LRPM). It is based on analyzing the BP algorithm from the viewpoint of the mutual information. Guo et al. showed that a slightly modified version of the BP algorithm can be used to get an upper bound MI_t on the information that can be extracted from a leaking computation thanks to a SASCA. The idea is that rather than propagating probability densities for performing an attack (This is computationally expensive and gives the success/failure for one attack, which makes it impractical to evaluate high security levels.[12]), the information leakages are propagated.

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

[12] It also avoids convergence issues when the factor graph has cycles.

The bound can then be translated into a bound on the success rate of the adversary (and data complexity of the attack) [DFS19].

The LRPM is therefore a specialization of the RPM where the way the leakages are exploited is restricted to some information propagation rules similar to those of the BP algorithm, as we explain next.

The LRPM models SASCA by using the factor graph and the BP algorithm, with a slight modification: the messages are no longer PDF estimates, but MI 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 amount of leakage observed on the variable. The second consequence is a modification of the rules to compose messages [Guo+20]:

- For messages of the first step (from variables to functions): the message sent is the sum of the intrinsic and the extrinsic information, upper bounded to $\mathsf{MI} = 1$.[13] This heuristic follows from the fact that combining independent estimates for a variable at most sums the information of the estimates.

- For messages of the second step (from functions to variables): using the random probing model, it is shown in [Guo+20] that the information on the result of a sum/difference or on the operand of any operation (sum/difference 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 [Guo+20] (since they do not use the result of multiplications in other operations): the information on the result of a multiplication is estimated as [CS19]

$$\mathsf{MI}_{\times \to Z} = \frac{\mathsf{MI}_{X \to \times} + \mathsf{MI}_{Y \to \times}}{|\mathbb{F}|} + \mathsf{MI}_{X \to \times} \mathsf{MI}_{Y \to \times} \left(1 - \frac{2}{|\mathbb{F}|}\right)$$

  where the variables X and Y are the factors and Z is the product.

- The case of bijection relationships is special: since the MI on one of the variables related by a bijection translates into the same MI on the other variable, the two nodes can be merged and the bijection function node 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. The information on the sensitive variable node, denoted MI_t, is the final result of this algorithm.

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

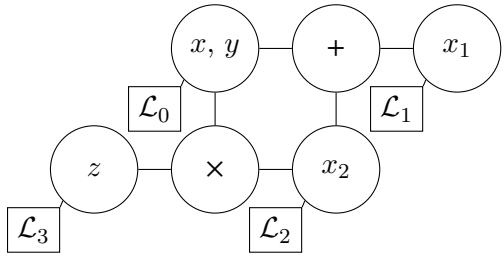


Figure 5.10: Factor graph with six nodes. $\mathcal{L}$ labels indicate intrinsic information.

For illustration, we next give an example of factor graph in Figure 5.10. The corresponding equations for the 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:

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

and those sent at the second step are:

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

5.3.3 Factor graph design

Our goal is to use the LRPM of [Guo+20] in order to measure the security of various (existing and new) masked gadgets. More precisely, 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). Additionally, we analyze larger gadgets based on the compositions of gadgets, such as the AES S-box shown in Figure 5.5.

Using 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 relationships on the variables that appear in the gadget first include all 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 which only appears through it shares $\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}$ in the gadget. We include them as well in the factor graph with the sharing relationships $x = \mathbf{x}_0 + \cdots + \mathbf{x}_{d-1}$, and we include the relationships between the unmasked variables that can be deduced from the functionality of the circuit. Other implicit relationships appear when a gadget uses refresh gadgets internally: the sum of the inputs of a refresh is equal to the sum of its outputs (which, as will be seen next, may be relevant to improve security against horizontal attacks). We illustrate it with a simple circuit: $\mathbf{y} = \mathsf{R}(\mathbf{x})$, where $\mathsf{R}(\cdot)$ is a refresh gadget, $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{d-1})$ an input sharing and $\mathbf{y} = (\mathbf{y}_0, \ldots, \mathbf{y}_{d-1})$ an output sharing. Let us assume that there are many manipulations of the shares $\mathbf{y}_i$. In this case, the BP algorithm can propagate this information to the shares $\mathbf{x}_i$, but (especially if the random elements in the refresh and the shares $\mathbf{x}_i$ are manipulated few times and the refresh has a high logic depth – for example by iterating many times a simple refresh algorithm) the information extracted on the shares $\mathbf{x}_i$ can be much smaller than the information available on $\mathbf{y}_i$. A SASCA targeting the input sensitive variable based on the relationship $x = \mathbf{x}_0 + \cdots + \mathbf{x}_{d-1}$ will therefore lead to a much lower information than actually available, by exploiting the relationship $x = y = \mathbf{y}_0 + \cdots + \mathbf{y}_{d-1}$. For this example, the solution is simple: insert in the factor graph not only the equation $x = \mathbf{x}_0 + \cdots + \mathbf{x}_{d-1}$ but also $x = y = \mathbf{y}_0 + \cdots + \mathbf{y}_{d-1}$. We generally apply this strategy for all the (possibly more complex) gadgets we analyze. That is, 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, the information leakage is derived from the p-random probing model: a single wire leaks the value it carries with probability p, therefore the information leakage is $\mathsf{MI}_o = p$ (the information unit is the number of field elements). As a graph, when n wires carry the same value, we assign it a single variable node (instead of n nodes with bijection relations) whose information leakage is $\mathsf{MI}_o = 1 - (1 - p)^n$.

5.3.4 LRPM versus PDT

This section compares the result of an LRPM security evaluation with the bound obtained by PDT composition and with the success rate of a

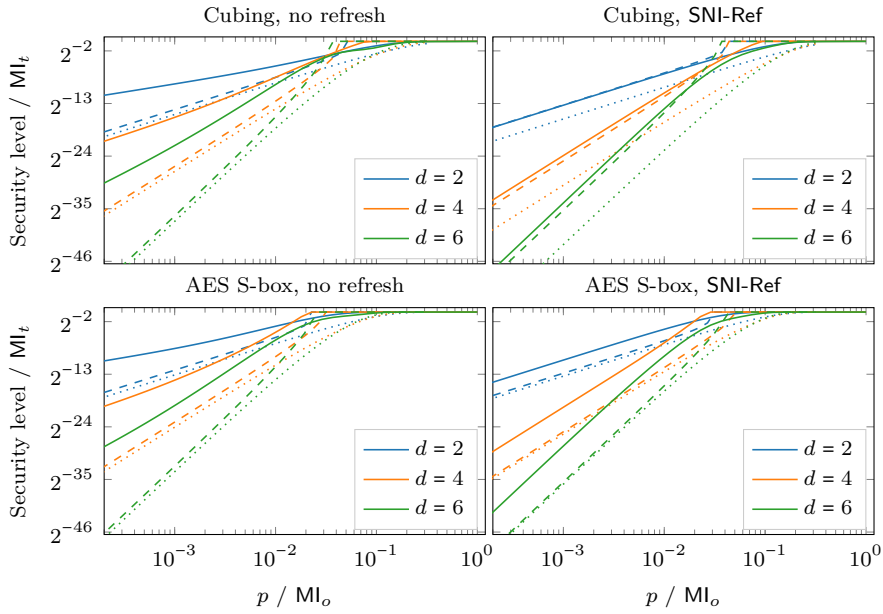


Figure 5.11: Security of the cubing ($\mathsf{ISW}(\mathbf{x}, \mathsf{Ref}(\mathbf{x}^2))$) and inversion (non-linear part of the AES S-box, see Figure 5.5) in $\mathbb{F}_{256}$, with the refresh gadget set either to the optimized SNI refresh of Battistello et al. [Bat+16] or to the identity gadget. The solid line is the bound based on PDT composition (see Figure 5.7 and Figure 5.8), the dashed curve is the LRPM bound, and the dotted curve is the "shares only" attack.

simple attack. This will show the usefulness and the limitations of the LRPM, before it is used in the next sections for gadgets where the PDT approach cannot be used due to its computational complexity.

The models are compared on two circuits: the cubing on $\mathbb{F}_{256}$ (implemented with an ISW multiplication and a sharewise squaring), and an AES S-box in $\mathbb{F}_{256}$ (see Figure 5.5). For both circuit, two variants are shown: with and without SNI refresh gadgets (used where needed to ensure $d-1$-probing security). The result of the comparison is shown in Figure 5.11, where the PDT bound is the one described in Section 5.2.5 and the factor graph design for the LRPM analysis follows the methodology of Section 5.3.3. The simple attack follows a "shares only" strategy: the adversary exploits only the leakage of wires whose value is equal to an input share of a gadget in the composition. Then, if the set of shares it recovers contains all the shares of one sharing, the attack succeeds.

From Figure 5.11, we observe first that both the "shares only" attack and the LRPM evaluation fail to detect the lower-order flaws: when no refreshing is used, these curves predict a much better security level than

the one based on PDTs (e.g., for $d = 2$, the slope of the curve correspond to a second-order attack, while a first-order attack is possible). This failure is by design: both the "shares only" and LRPM model a dth-order attack.

When the security order is $d - 1$ (the compositions with SNI-Ref), the three curves are more similar. For the cubing, the LRPM and the PDT bound are very close, and there is a gap between these and the "shares only" attack, while for the AES S-box, the LRPM is very close to the "shares only" attack (except at low noise level), while the PDT bound gives a worse bound. A possible explanation for this is the limited ability of the LRPM to propagate information from far apart locations in the gadget graph (except for shares leakage), while the PDT approach can do this, at the expense possibly under-estimating the security level.

In the following, we analyze single multiplication gadgets with the LRPM. Informally, these evaluations should not suffer too much from the limitations shown above since these gadgets are $(d - 1)$-probing secure and their factor graphs have small diameter: most of the leakage is closely related to an input or output share of the gadget.

5.3.5 Analysis of ISW and PINI1 gadgets

We first apply our methodology to the ISW multiplication gadget in $\mathbb{F}_2$, for $d = 4$ and $d = 16$ shares. The results are reported in Figure 5.12, leading to the following conclusions.[14]

In general, for low observation MI (MI_o) the trend of the target MI (i.e., MI_t) is an asymptote of slope $t = d - 1$ (as expected for t-th order security). For larger MI_o, we see that the curves leave the asymptote until they reach $\mathsf{MI}_t = 1$. There are thus two regions in the graph: the low MI_o and high MI_o regions. The boundary between these regions varies depending on the curves (see for example ISW for $d = 4$ and $d = 16$ in Figure 5.12), 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 the slope of the asymptote is always equal to $t = d - 1$ (we only consider t-probing secure gadgets), we will focus on the minimum noise level, captured by the location of the boundary – concretely, we use the point where $\mathsf{MI}_t = 1$ for this purpose. The noise rate is then captured by the evolution of this minimum noise level with the number of shares.

More specifically, we see that the location of the boundary strongly depends on the number of shares for the ISW gadget. This comes from

[14] The source code for the LRPM tool and the plots is available at `https://github.com/cassiersg/lrpm-bounder`.

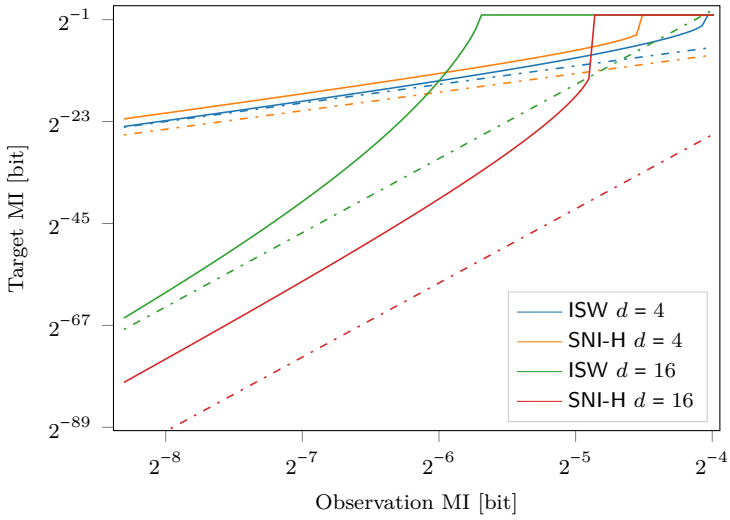


Figure 5.12: Target MI as a function of observation MI, for the ISW multiplication gadget [ISW03] and the SNI-H multiplication gadget [Bat+16]. The continuous line is the bound on the target MI that can be extracted by the BP algorithm and the dashed line is a lower bound for the target MI computed assuming that only the leakage from the input shares is exploited.

the fact that this gadget manipulates each input share d times, which explains why the minimum noise level with 4 shares is lower (larger MI_o) than with 16 shares. We remark that the ratio of 4 between these values correspond to the noise rate of $\mathcal{O}(1/t)$ that is expected for the ISW multiplication.[15]

By contrast, for the gadget of Battistello et al. [Bat+16] 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 $\mathcal{O}(\log t)$ times, the SNI-H gadget gains interest over the ISW one as t increases, thanks to its better noise rate (e.g., for $d = 16$). At lower orders (e.g., $d = 4$), the SNI-H gadget remains worse than the 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

[15] Formally, the proofs in [DFS19] require a noise rate of $\mathcal{O}\left(1/t^2\right)$, but as discussed in [DFS19] the latter is assumed to be due to the proof that is not completely tight.

(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 $\mathsf{MI}_t = \min\left(1, (n_o \mathsf{MI}_o)^d\right)$, 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 ISW at low MI_o, the BP algorithm is not able to extract much information from the internal leakages (i.e., leakage from the intermediate variables) 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 elements in the multiplication. At larger MI_o, the information leaked on the internal variables becomes sufficient to (partially) recover the random elements, hence this information can be propagated to the input shares. Interestingly, and while the lower bound is tight for the ISW gadget in the high noise region, it is not for the SNI-H gadget (even though the slope is still t). 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 analyses confirm quantitatively the qualitative claim of [Bat+16] that the repeated manipulation of some shares is the main weakness exploited by horizontal attacks. They cause an horizontal shift of the information theoretic curves in Figure 5.12, and impose a noise level that increases with the order of the implementation. If the noise level does not increase with the order, this shift directly translates into a security loss on MI_t of a factor $\mathcal{O}\left(t^t\right)$ for the ISW gadget (which is reduced to $\mathcal{O}\left(\log(t)^t\right)$ if the countermeasure of Battistello et al. is applied). In other words, this experiment highlights that the use of the ISW (resp., SNI-H) multiplication gadget at high order requires an amount of noise such that $\mathsf{MI}_o \propto 1/t$ (resp., $\mathsf{MI}_o \propto 1/\log(t)$).

The application of our methodology to the PINI_1 multiplication gadget is reported in Figure 5.13. We observe that the global trend is very close to the one of the ISW multiplication, except that there is a 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 $(\mathbf{x}_i + 1)r_{ij} + \mathbf{x}_i(\mathbf{y}_j - r_{ij})$ (instead of $\mathbf{x}_i\mathbf{y}_j + r_{ij}$ for ISW). Since the PINI_1 gadget has $\mathcal{O}(t)$ manipulations of the input shares, it still has a noise rate in $\mathcal{O}(1/t)$. Therefore, its use at high security order faces the same problem as the ISW gadget: it requires a large amount of noise (i.e., $\mathsf{MI}_o \propto 1/t$).

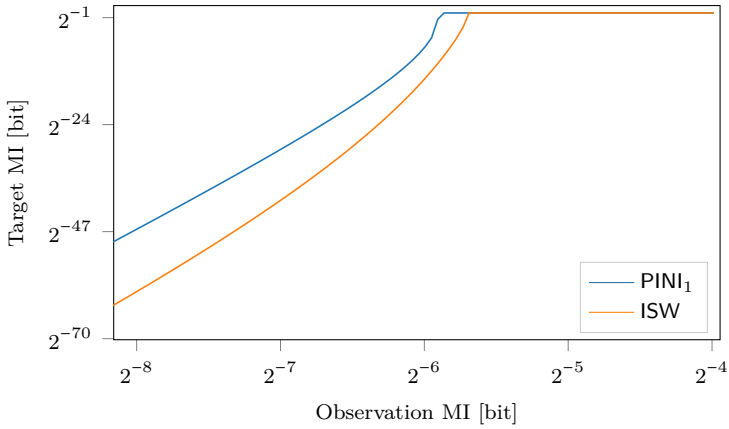


Figure 5.13: Target MI as a function of observation MI, for the ISW multiplication, and the PINI$_1$ multiplication with 16 shares. The continuous line is the result of the belief propagation algorithm for the MI and the dashed line is a lower bound for the target MI computed assuming only the leakage on input shares are exploited.

5.3.6 Optimizing gadgets for noise rate

In this section, we introduce a framework for analyzing multiplication gadgets (from the literature and our new constructions) by splitting them into three stages: the `MatGen` stage produces a $d \times d$ matrix of (possibly refreshed) pairs of shares of x and y, the `Product` stage computes the partial products of those shares, which gives a d^2 sharing of the product, and the `Compression` stage compresses this sharing into a d-share output. This process is illustrated by the following process (for $d = 4$):

$$\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix}, \begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \end{bmatrix} \xrightarrow{\texttt{MatGen}} \begin{bmatrix} (x_{0,0}, y_{0,0}) & (x_{0,1}, y_{1,0}) & (x_{0,2}, y_{2,0}) & (x_{0,3}, y_{3,0}) \\ (x_{1,0}, y_{0,1}) & (x_{1,1}, y_{1,1}) & (x_{1,2}, y_{2,1}) & (x_{1,3}, y_{3,1}) \\ (x_{2,0}, y_{0,2}) & (x_{2,1}, y_{1,2}) & (x_{2,2}, y_{2,2}) & (x_{2,3}, y_{3,2}) \\ (x_{3,0}, y_{0,3}) & (x_{3,1}, y_{1,3}) & (x_{3,2}, y_{2,3}) & (x_{3,3}, y_{3,3}) \end{bmatrix} \dots$$

$$\dots \xrightarrow{\texttt{Product}} \begin{bmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \alpha_{0,3} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \alpha_{3,0} & \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{bmatrix} \xrightarrow{\texttt{Compression}} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix}. \quad (5.8)$$

We next show how this general construction applies in a few particular cases of interest.

- ISW multiplication. This is the simplest case, where the `MatGen` stage is close to the identity: it assigns $(x_{i,j}, y_{i,j}) = (\mathbf{x}_i, \mathbf{y}_j)$, the `Product` stage is an element-wise product: $\alpha_{i,j} = x_{i,j} y_{i,j}$, and the `Compression` stage sums products and uniform randomness in a

way that ensures correctness and security: $\mathbf{z}_i = \alpha_{i,i} + \sum_{j=0,j\neq i}^{d-1}(r_{ij} + \alpha_{i,j})$, where $r_{ij} = -r_{ji}$ are random elements.

- PINI_1 multiplication. This algorithm uses the same `MatGen` stage as the ISW multiplication. Its `Product` stage uses the "masked shares multiplication trick" to achieve PINI, computing $\alpha_{i,j} = r_{ij}(x_{i,j} + 1) + x_{i,j}(y_{i,j} - r_{ij})$ (and. $\alpha_{i,i} = x_{i,i}y_{i,i}$). Since this stage already includes the refreshing of the share, the `Compression` is very simple: $\mathbf{z}_i = \alpha_{i,i} + \sum_{j=0,j\neq i}^{d-1} \alpha_{i,j}$.
- Multiplication of Battistello et al. [Bat+16]. This algorithm uses the same `Product` and `Compression` stages as the ISW multiplication, but its `MatGen` stage is modified in order to resist horizontal attacks. This is discussed in detail in next.

It should be noted that each of the stages may use some random bits. In the `MatGen` stage, the randomness cost comes from the use of additional "internal" refresh gadgets (which primarily impact security against horizontal attacks), such as for the multiplication of Battistello et al. A good example of randomness use in the `Product` stage is given by the PINI_1 gadget. Finally, the randomness in the `Compression` stage is primarily needed for probing security (it is the only stage where randomness is used in the ISW multiplication).

Refreshing in multiplication gadgets Let us now describe the `MatGen` stage of the multiplication of Battistello et al. (SNI-H) and compare it with the ISW multiplication. We then investigate how the SNI-H scheme can be improved against horizontal attacks, and how these ideas can be adapted to the PINI_1 gadget. Finally, we build new gadgets that trade a bit of resistance to horizontal attacks for reduced randomness complexity.

We introduce in Algorithm 27 a generic construction for the `MatGen` stage of Equation 5.8. This algorithm is parameterized on two gadgets, $\mathsf{Refresh}_0$ and $\mathsf{Refresh}_1$. In the simplest case of the ISW multiplication, `MatGen` is directly obtained by using the identity gadget as both $\mathsf{Refresh}_0$ and $\mathsf{Refresh}_1$. The countermeasure of Battistello et al. (SNI-H) uses an SNI refresh gadget for $\mathsf{Refresh}_1$ and the identity gadget as $\mathsf{Refresh}_0$. Our formalization in Algorithm 27 and the representation of the SNI-H tree (Figure 5.14b) compared to the tree of ISW (Figure 5.14a) suggest a natural improvement (from the security against horizontal attacks' viewpoint) where more refresh gadgets are added, that we denote as SNI-H+ (Figure 5.14c). This gadget is based on the ISW multiplication, but it uses Algorithm 27 as `MatGen` with both $\mathsf{Refresh}_0$ and $\mathsf{Refresh}_1$ as SNI refresh gadgets, as shown in Table 5.3. Whereas the SNI-H gadget

Algorithm 27 `MatGen`

Input: Refresh algorithms $\mathsf{Refresh}_0$ and $\mathsf{Refresh}_1$.
Input: Sharings $\mathbf{x}, \mathbf{y}$.
Output: $M \in \left(\mathbb{F}_q^2\right)^{d \times d}$ such that $\sum_{i,j} x_{i,j} y_{i,j} = xy$, where $(x_{i,j}, y_{i,j}) = M_{i,j}$.

1: **if** $d = 1$ **then**
2: $\quad M \leftarrow [(x_0, y_0)]$
3: **else**
4: $\quad$ **for** $i = 0, 1$ **do**
5: $\quad\quad X^{(0,i)} \leftarrow \mathsf{Refresh}_i\left((\mathbf{x}_0, \ldots, \mathbf{x}_{\lfloor d/2 \rfloor - 1})\right)$;
6: $\quad\quad X^{(1,i)} \leftarrow \mathsf{Refresh}_i\left((\mathbf{x}_{\lfloor d/2 \rfloor}, \ldots, \mathbf{x}_{d-1})\right)$;
7: $\quad\quad Y^{(0,i)} \leftarrow \mathsf{Refresh}_i\left((\mathbf{y}_0, \ldots, \mathbf{y}_{\lfloor d/2 \rfloor - 1})\right)$;
8: $\quad\quad Y^{(1,i)} \leftarrow \mathsf{Refresh}_i\left((\mathbf{y}_{\lfloor d/2 \rfloor}, \ldots, \mathbf{y}_{d-1})\right)$;
9: $\quad\quad$ **for** $j = 0, 1$ **do**
10: $\quad\quad\quad M^{(j,i)} = \mathrm{MatRef}(X^{(j,i)}, Y^{(j,i)})$
11: $\quad M \leftarrow \begin{bmatrix} M^{(0,0)} & M^{(0,1)} \\ M^{(1,0)} & M^{(1,1)} \end{bmatrix}$

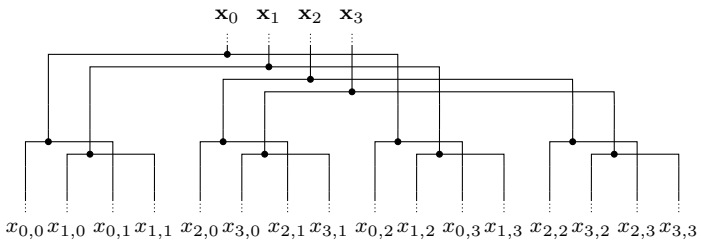
manipulates each input shares $\mathcal{O}(\log d)$ times, SNI-H+ manipulates every share only twice. The security of this gadget is shown in Figure 5.15. As expected, the curves for SNI-H+ are shifted to the right compared to the SNI-H curves. Furthermore, the minimum noise level of SNI-H shifts to the left as the order increases (suggesting $\mathcal{O}(\log t)$ noise rate), the minimum noise level of SNI-H+ is constant for $d \geq 8$. This confirms the intuition that having a constant number of share manipulations translates into having a constant MI_o noise requirement and leads us to conjecture that this gadget has a constant noise rate. This is similar to the gadgets based on the expansion technique [AIS18] (see Section 5.1), which have a proven constant noise rate (albeit their required noise level is higher than for SNI-H+).

Improved PINI gadgets The idea of "internal refreshes" used for SNI-H+ can be combined with the "masked shares' multiplication" trick of the PINI_1 gadget. Moreover, the computation $r_{ij}(\mathbf{x}_i + 1)$ can be replaced with the computation $r_{ij}\mathbf{x}_i + r_{ij}$, which reduces the amount of leakage that depends only on $\mathbf{x}_i$. The result is the PINI_3-H+ gadget shown in Algorithm 28.

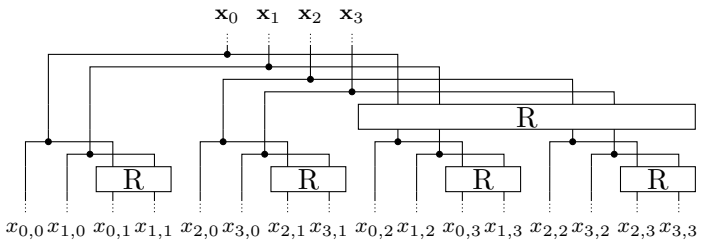
In Figure 5.16, we observe that the evolutions of PINI_1 and PINI_2 into

Multiplication	$\mathsf{Refresh}_0$	$\mathsf{Refresh}_1$
ISW	Identity	Identity
SNI-H	Identity	BatRef
SNI-H+	BatRef	BatRef
SNI-H*	SimpleRef	SimpleRef

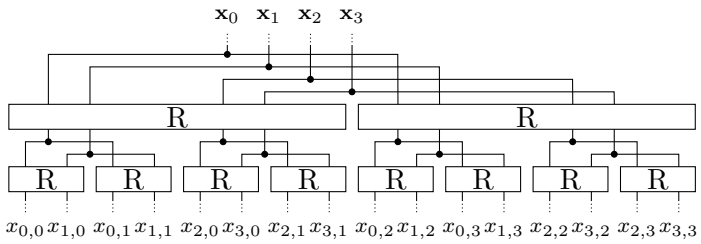
Table 5.3: Refreshing gadgets used for various multiplication gadgets. BatRef is the SNI refresh gadget of Battistello et al. ([Bat+16], Algorithm 6). SimpleRef is a NI refresh using t random bits (Gadget 1 of [Bar+16]). The Identity gadget simply connects its inputs to its outputs.



(a) No refreshing — SNI.



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



(c) Double refreshing — SNI-H+.

Figure 5.14: Exemplary instantiations of the generalized `MatGen` algorithm (Algorithm 27) for $d = 4$. The algorithm basically corresponds to two trees of internal refresh gadgets (one for sharings of x, one for those of y). The input shares are $\mathbf{x}_0, \ldots, \mathbf{x}_3$ and $x_{i,j}$ are the outputs of the `MatGen` stage (see Equation 5.8).

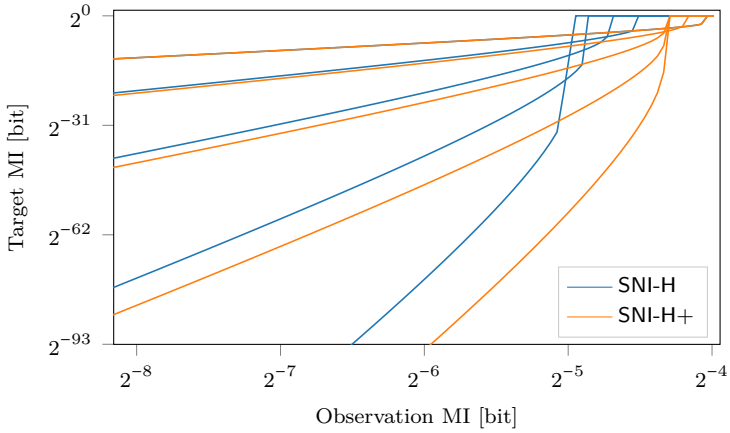


Figure 5.15: Target MI as a function of observation MI, for the SNI-H and SNI-H+ multiplication gadgets with $d = 2, 4, 8, 16$ shares.

Algorithm 28 Generalized PINI_3 multiplication gadget.

Input: Sharings $\mathbf{x}$, $\mathbf{y}$
Output: Sharing $\mathbf{z}$ such that $z = x \cdot y$.

1: $M \leftarrow \mathtt{MatGen}((\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}), (\mathbf{y}_0, \ldots, \mathbf{y}_{d-1}))$ ▷ See Algorithm 27.
2: **for** $i = 0$ to $d - 1$ **do**
3: **for** $j = i + 1$ to $d - 1$ **do**
4: $r_{ij} \xleftarrow{\$} \mathbb{F}_q$; $r_{ji} \leftarrow -r_{ij}$
5: **for** $i = 0$ to $d - 1$ **do**
6: $(x_{i,i}, y_{i,i}) \leftarrow (M)_{i,i}$
7: $\mathbf{z}_i \leftarrow x_{i,i} y_{i,i}$
8: **for** $j = 0$ to $d - 1, j \neq i$ **do**
9: $(x_{i,j}, y_{i,j}) \leftarrow (M)_{i,j}$
10: $\mathbf{z}_i \leftarrow \mathbf{z}_i + \big((x_{i,j} r_{ij} + r_{ij}) + x_{i,j}(y_{i,j} - r_{i,j})\big)$

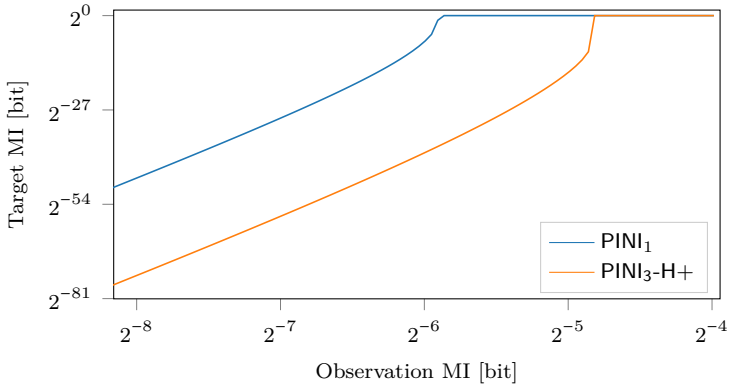


Figure 5.16: Target MI as a function of observation MI, for the PINI$_1$ and PINI$_3$-H+ multiplication gadgets with $d = 16$ shares.

PINI$_3$-H+ and PINI$_2$-H+ follow the same trend of shifting information curves to the right (as it happens for the evolution from ISW into SNI-H+), while preserving their other distinguishing features. In particular, the PINI$_2$-H+ curve still has the "staircase of asymptotes" aspect which reduces the security level in the medium/large MI$_o$ region.

Figure 5.17, shows that the asymptotic behavior of the PINI$_3$-H+ gadget is identical to the one of SNI-H+: the location of the boundary between regions is also almost independent of the order. The main difference between 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 (in the masked shares' multiplication trick), similarly to PINI$_1$ versus ISW.

We finally mention a last construction for a PINI multiplication with constant noise rate: use the SNI-H+ gadget in the refreshed multiplication construction of Section 3.2.4. The security level of the resulting RefMult-H+ gadget is displayed in Figure 5.17: it is very close to SNI-H+.

Security vs randomness cost trade-off Battistello et al. use a SNI refresh with randomness complexity $\mathcal{O}(t \log t)$ in the `MatGen` stage. However, since this refresh is not used to prove composability in the t-probing model, the SNI property is not strictly required. We hence analyze the case where this SNI refresh (Algorithm 6 of [Bat+16]) is replaced with a simple refresh using t random bits (Gadget 1 of [Bar+16]). The latter leads to new gadgets SNI-H* and PINI$_3$-H*, which are adapted versions of SNI-H+ and PINI$_3$-H+, (as per Table 5.3). We observe in Figure 5.18 that the security level of the these new gadgets is almost the same as the one of the gadgets based on the refresh of Battistello et al.

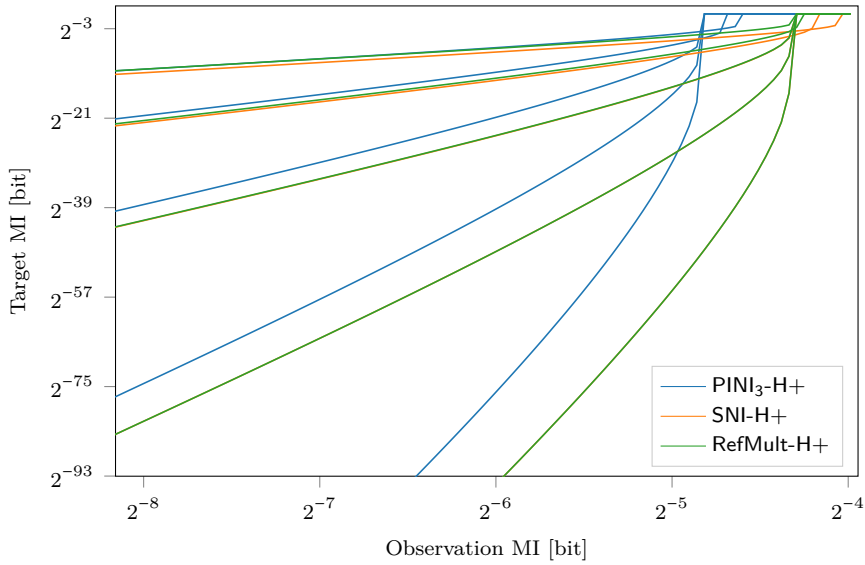


Figure 5.17: Target MI as a function of observation MI, for the SNI-H+, PINI$_3$-H+ and RefMult-H+ multiplication gadgets with $d = 2, 4, 8, 16$ shares. The curves for SNI-H+ and RefMult-H+ are superimposed for $d \geq 8$.

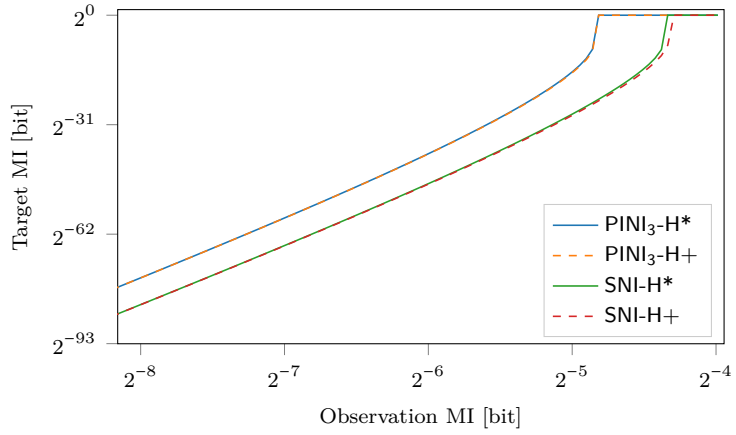


Figure 5.18: Target MI as a function of observation MI, for the SNI-H+, SNI-H*, PINI$_3$-H+ and PINI$_3$-H* multiplication gadgets with $d = 16$ shares.

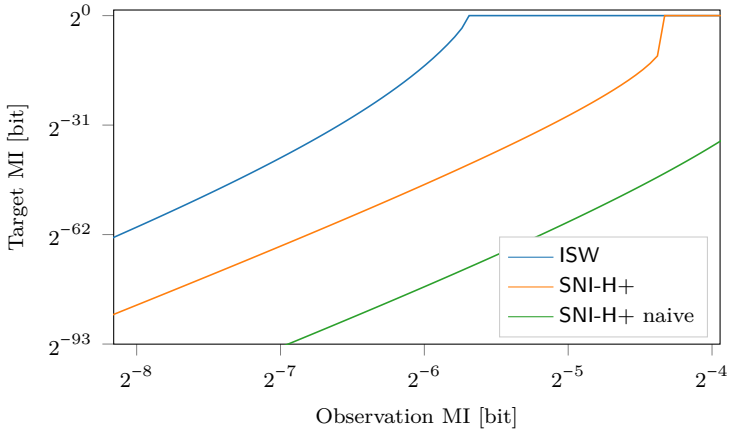


Figure 5.19: Target MI as a function of observation MI, for the SNI-H+ multiplication gadget with $d = 16$ shares. For the SNI-H+ naive curve, the target MI is extracted through the sums of the input and output shares only (naive attack). For the SNI-H+ curve, the improved attack is considered (equations about sums of refresh input/outputs are added. The ISW curve (which is the same for both the naive and the non-naive attack) is shown as a reference point.

Regarding the randomness cost of the gadgets, they all have a complexity in $\mathcal{O}\left(t^2\right)$ (this is immediately visible for the `Compression` stage, and proven in [CS19] for the H+ `MatGen`), but the use of the H* `MatGen` versus the H+ one results in a reduced randomness cost of about 50 %. More broadly, the H* `MatGen` uses roughly two times the amount of randomness used by the ISW multiplication [CS19].

5.3.7 A note on SASCA and factor graphs

Finally, we empirically validate the argument of Section 5.3.3 that adding to the factor graph equations about all the input/output relationships of refresh gadgets is actually needed to perform worst-case SASCA. As illustrated in Figure 5.19, removing those relationships from the factor graph gives a less informative "SNI-H+ naive" curve. We observe that the naive attack causes a significant loss to the adversary, exhibited by a shift of the curve to the left, by a factor that depends on the order and the type of protection against horizontal attacks considered.

We conclude that this technique should be used when analyzing SASCA in the LRPM, since it can lead to significant reductions of the worst-case complexity and does not add stronger hypotheses on the capabilities of the adversary. The only potential disadvantage of this technique is that it adds cycles in the factor graph, which might harm the convergence of

the BP algorithm in practice (the analysis of which is a scope for further research).

Conclusion and open problems

In this thesis, we analyzed the security of masking schemes in three leakage models: threshold probing, robust probing and random probing.

First, we introduced the new probe-isolating non-interference (PINI) definition that can serve as the basis for trivially composable masking schemes in the t-threshold probing model. These arithmetic masking schemes are simple and efficient: they require a minimal number of shares ($t + 1$), use sharewise gadgets for affine operations and the PINI1 or PINI2 gadget for multiplications, and do not need additional refreshing thanks to trivial composition. Moreover, the PINI1 gadget uses the same amount of randomness as the ISW multiplication ($d(d-1)/2$ random elements for d shares), and the PINI2 gadget has even better asymptotic complexity (it uses $d^2/4 + \mathcal{O}(d)$ random elements).

Next, we have shown how the PINI definition and the PINI gadgets can be extended to reach security in the robust probing model. This led to the hardware private circuit (HPC) masking scheme that is the first arbitrary-order masking scheme secure against both hardware glitches and transitions at arbitrary order. The HPC gadgets are based the PINI ones: the affine gadgets are sharewise combinatorial circuits, while the multiplication gadgets are based on PINI multiplication algorithms (PINI1 or refreshed multiplication) with the use of registers to prevent glitch propagation. Similarly to PINI masking schemes, no additional refresh gadgets are needed. The HPC gadgets inherit the performance characteristics of the PINI gadgets and have low latency: affine gadgets are purely combinatorial (0 cycles), and multiplication gadgets have a latency of 2 (respectively 1) cycles with respect to one (resp. the other) input sharing. We also introduced the fullVerif tool that verifies the security of HPC implementation: it automatically detects implementation errors that can break robust probing security.

Finally, we tackled the question of noise level with analyzes in the random probing model. Indeed, robust probing circuits are known to be practically secure if there is enough noise (and if leakage from extended probes is independent), but this noise level cannot be quantified from the robust probing model itself: a more quantitative model such as the random probing model is needed. We introduced the probe distribution tables (PDTs) as a composable way to characterize the random probing

security of masked gadgets. Compared to the state of the art, PDT-based security analysis is tighter: it requires fewer shares to reach a given security level for a given amount of noise. However, computing the PDT of a gadget with many input shares is computationally expensive, limiting its use to small sharings (we limited ourselves to $d \leq 6$) and to composition of few gadgets in parallel (an S-box is feasible, not a full block cipher). We circumvent this limitation by using the local random probing model (LRPM) heuristic to analyze the noise rate of our gadgets and to design new ones with improved noise rate, resulting in lower noise requirements when the number of shares is large.

Practically, the contributions of this thesis can be used as the basis of a methodology for side-channel security evaluation. Thanks to its high level of abstraction (it does not involve any physical measurement), this method can be used as a preliminary evaluation (i.e., one that can be used to inform the design process before any costly circuit fabrication is done). This method is made of two steps, relying on the complementarity between the robust probing model and the random probing model, while working around their limitations (no noise level analysis in the robust probing model, limited scalability in the random probing model).

The first step is a composition-based proof of security in the robust probing model. This guarantees the absence of composition flaws, ensuring that, given a sufficient amount of noise, the real-world security of the implementation scales exponentially with the number of shares, even in presence of glitches and transitions. Moreover, this analysis can be done with fullVerif to directly verify the implementation, eliminating the possibility of mismatch between the algorithmic description used for the verification and the actual implementation.

The second step is the analysis of small parts of the circuit (e.g., an S-box or a part of a linear layer) in the random probing model, using PDTs or the LRPM (if there are too many shares for PDTs). Then, we assume that, for a given noise level, the random probing of the full circuit is similar to the one of these small parts (more precisely, that the security level is inversely proportional to the number of gates in the masked circuit). The rationale behind this extrapolation is the observation that horizontal attacks do not scale well to parts of circuit that handle independent (i.e., refreshed) sharings. Therefore, the extrapolation requires that sharings are often refreshed, and will not hold if a circuit is a long sequence of sharewise gadgets that operate on few sharings, in which case the noise level required to counter horizontal attacks will grow proportionally to the number of gadgets.

Let us now discuss some problems left open in this thesis. Solving them would improve the evaluation method explained above, by strengthening its guarantees (e.g., removing heuristics and assumptions) or by extending its applicability.

First, the random probing analysis is heuristically extrapolated to the full circuits, due to the infeasibility of computing PDT compositions for a large circuit. This scalability challenge comes from the definition of the PDT: it describes distributions of the set of input wire needed to simulate a circuit, which is a distribution over a space of size $2^{|\mathcal{I}|}$, where $|\mathcal{I}|$ is the number of input wires of the circuit. For a d-share AES round, this means that the PDT is a matrix with 2^{128d} rows (excluding the key schedule). To make the PDT idea scale to large circuits, one must find a compressed representation for this probability distribution, on which computation is efficient. The compressed representation can be a bound for the actual PDT, but it should be tight enough to ensure that the final security bound is close to the actual security level of the implementation.

Second, the random probing model does not include glitches nor transitions, and as a result the PDT security evaluations to not take them into account. While it seems fairly straightforward to introduce a robust random probing model where the probes are extended and to adapt STRAPS to it, this opens multiple questions. For example, should there be a glitch-extended probe for every wire? Is the leakage probability the same for every kind of probe, or do glitch-extended probes become less likely when they include more wires? How to handle transition-extended probes that cross multiple gadgets?

Next, one of our goals is to minimize the discrepancies between the verification stage and the final circuit that is fabricated. Our main effort in this direction is the introduction of the fullVerif tool. This effort is however not complete: fullVerif currently only checks properties of gadget compositions and not the properties (e.g., PINI) of the gadgets themselves, which could be done by integrating maskVerif or SILVER with fullVerif. Moreover, fullVerif (as well as maskVerif and SILVER) verify high-level netlists, which are typically further optimized in the course of the circuit design flow. To ensure that the latter optimizations do not break the security of the masking scheme, fullVerif could be modified to additionally verify the security of the final circuit.

Lastly, this thesis is focused on arithmetic masking, which is only one of many styles of masking. However, some contributions such as the PINI definition or the PDTs are not intrinsically dependent on arithmetic masking. It would therefore be interesting to investigate if they can be useful for the analysis and design of other styles of masking. This is not

trivial since, for example, code-based masking can improve the security against fault attacks by introducing redundancy but, by doing so, breaks to property that d-PINI implies $d - 1$ probing security.

Bibliography

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. "The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol". In: *EUROCRYPT (1)*. Vol. 11476. Lecture Notes in Computer Science. Springer, 2019, pp. 129–158.

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. "Private Circuits: A Modular Approach". In: *CRYPTO (3)*. Vol. 10993. Lecture Notes in Computer Science. Springer, 2018, pp. 427–455.

[Ava+19] Roberto Avanzi et al. "CRYSTALS-Kyber algorithm specifications and supporting documentation". In: *NIST PQC Round* 3 (2019), p. 4.

[Azo+22] Melissa Azouaoui, Olivier Bronchain, Vincent Grosso, and Kostas Papagiannopoulos. "Bitslice Masking and Improved Shuffling: How and When to Mix Them in Software?" In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.2 (2022), pp. 140–165.

[Bar+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. "Verified Proofs of Higher-Order Masking". In: *EUROCRYPT (1)*. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 457–485.

[Bar+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. "Strong Non-Interference and Type-Directed Higher-Order Masking". In: *CCS*. ACM, 2016, pp. 116–129.

[Bar+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. "Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model". In: *EUROCRYPT (1)*. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 535–566.

[Bar+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. "Masking the GLP Lattice-Based Signature Scheme at Any Order". In: *EUROCRYPT (2)*. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 354–384.

[Bar+19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. "maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults". In: *ESORICS (1)*. Vol. 11735. Lecture Notes in Computer Science. Springer, 2019, pp. 300–318.

[Bar+20] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. "Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations". In: *J. Cryptogr. Eng.* 10.1 (2020), pp. 17–26.

[Bar+21] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. "Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.2 (2021), pp. 189–228.

[Bas+19] Andrea Basso, Jose Maria Bermudo Mera, Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Michiel Van Beirendonck, and Frederik Vercauteren. "SABER: Mod-LWR based KEM". In: *NIST PQC Round* 3 (2019).

[Bat+16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. "Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme". In: *CHES*. Vol. 9813. Lecture Notes in Computer Science. Springer, 2016, pp. 23–39.

[BC22] Olivier Bronchain and Gaëtan Cassiers. "Bitslicing Arithmetic/Boolean Masking Conversions for Fun and Profit with Application to Lattice-Based KEMs". In: *IACR Cryptol. ePrint Arch.* (2022), p. 158.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation Power Analysis with a Leakage Model". In: *CHES*. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 16–29.

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. "Give Me 5 Minutes: Attacking ASCAD with a Single Side-Channel Trace". In: *IACR Cryptol. ePrint Arch.* (2021), p. 817.

[Bel+16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. "Randomness Complexity of Private Circuits for Multiplication". In: *EUROCRYPT (2)*. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 616–648.

[Bel+20a] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. "Random Probing Security: Verification, Composition, Expansion and New Constructions". In: *CRYPTO (1)*. Vol. 12170. Lecture Notes in Computer Science. Springer, 2020, pp. 339–368.

[Bel+20b] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. "Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations". In: *EUROCRYPT (3)*. Vol. 12107. Lecture Notes in Computer Science. Springer, 2020, pp. 311–341.

[Bel+20c] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Mode-Level vs. Implementation-Level Physical Security in Symmetric Cryptography - A Practical Guide Through the Leakage-Resistance Jungle". In: *CRYPTO (1)*. Vol. 12170. Lecture Notes in Computer Science. Springer, 2020, pp. 369–400.

[Bel+20d] Davide Bellizia et al. "Spook: Sponge-Based Leakage-Resistant Authenticated Encryption with a Masked Tweakable Block Cipher". In: *IACR Trans. Symmetric Cryptol.* 2020.S1 (2020), pp. 295–349.

[Bel+21] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. "IronMask: Versatile Verification of Masking Security". In: *IACR Cryptol. ePrint Arch.* (2021), p. 1671.

[Ben+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. "Deep learning for side-channel analysis and introduction to ASCAD database". In: *J. Cryptogr. Eng.* 10.2 (2020), pp. 163–188.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. "Tight Private Circuits: Achieving Probing Security with the Least Refreshing". In: *ASIACRYPT (2)*. Vol. 11273. Lecture Notes in Computer Science. Springer, 2018, pp. 343–372.

[Bil+13] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. "Efficient and First-Order DPA Resistant Implementations of Keccak". In: *CARDIS*. Vol. 8419. Lecture Notes in Computer Science. Springer, 2013, pp. 187–199.

[Bil+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. "Higher-Order Threshold Implementations". In: *ASIACRYPT (2)*. Vol. 8874. Lecture Notes in Computer Science. Springer, 2014, pp. 326–343.

[Blo+18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. "Formal Verification of Masked Hardware Implementations in the Presence of Glitches". In: *EUROCRYPT (2)*. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 321–353.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. "Logic Minimization Techniques with Applications to Cryptology". In: *J. Cryptol.* 26.2 (2013), pp. 280–312.

[Bos+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM". In: *EuroS&P*. IEEE, 2018, pp. 353–367.

[BP12] Joan Boyar and René Peralta. "A Small Depth-16 Circuit for the AES S-Box". In: *SEC*. Vol. 376. IFIP Advances in Information and Communication Technology. Springer, 2012, pp. 287–298.

[Bro22] Olivier Bronchain. "Worst-case side-channel security : from evaluation of countermeasures to new designs". en. PhD thesis. UCL - Université Catholique de Louvain, 2022. URL: `https://dial.uclouvain.be/pr/boreal/object/boreal:258155` (visited on 04/21/2022).

[Bro+22] Olivier Bronchain, François Durvaux, Loïc Masure, and François-Xavier Standaert. "Efficient Profiled Side-Channel Analysis of Masked Implementations, Extended". In: *IEEE Trans. Inf. Forensics Secur.* 17 (2022), pp. 574–584.

[BRT21] Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. "On the Power of Expansion: More Efficient Constructions in the Random Probing Model". In: *EUROCRYPT (2)*. Vol. 12697. Lecture Notes in Computer Science. Springer, 2021, pp. 313–343.

[BS21] Olivier Bronchain and François-Xavier Standaert. "Breaking Masked Implementations with Many Shares on 32-bit Software Platforms or When the Security Order Does Not Matter". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (2021), pp. 202–234.

[BS97] Eli Biham and Adi Shamir. "Differential Fault Analysis of Secret Key Cryptosystems". In: *CRYPTO*. Vol. 1294. Lecture Notes in Computer Science. Springer, 1997, pp. 513–525.

[BUS21] Davide Bellizia, Balazs Udvarhelyi, and François-Xavier Standaert. "Towards a Better Understanding of Side-Channel Analysis Measurements Setups". In: *CARDIS*. Vol. 13173. Lecture Notes in Computer Science. Springer, 2021, pp. 64–79.

[Cas+19] Gaëtan Cassiers, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "SpookChain: Chaining a Sponge-Based AEAD with Beyond-Birthday Security". In: *SPACE*. Vol. 11947. Lecture Notes in Computer Science. Springer, 2019, pp. 67–85.

[Cas+21a] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. "Towards Tight Random Probing Security". In: *CRYPTO (3)*. Vol. 12827. Lecture Notes in Computer Science. Springer, 2021, pp. 185–214.

[Cas+21b] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. “Hardware Private Circuits: From Trivial Composition to Full Verification”. In: *IEEE Trans. Computers* 70.10 (2021), pp. 1677–1690.

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. “Secure Conversion between Boolean and Arithmetic Masking of Any Order”. In: *CHES*. Vol. 8731. Lecture Notes in Computer Science. Springer, 2014, pp. 188–205.

[Cnu+16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. “Masking AES with d+1 Shares in Hardware”. In: *CHES*. Vol. 9813. Lecture Notes in Computer Science. Springer, 2016, pp. 194–212.

[Cnu+17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. “Does Coupling Affect the Security of Masked Implementations?” In: *COSADE*. Vol. 10348. Lecture Notes in Computer Science. Springer, 2017, pp. 1–18.

[Cor+13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. “Higher-Order Side Channel Security and Mask Refreshing”. In: *FSE*. Vol. 8424. Lecture Notes in Computer Science. Springer, 2013, pp. 410–424.

[Cor14] Jean-Sébastien Coron. “Higher Order Masking of Look-Up Tables”. In: *EUROCRYPT*. Vol. 8441. Lecture Notes in Computer Science. Springer, 2014, pp. 441–458.

[Cor+15] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. “Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity”. In: *FSE*. Vol. 9054. Lecture Notes in Computer Science. Springer, 2015, pp. 130–149.

[Cor18] Jean-Sébastien Coron. “Formal Verification of Side-Channel Countermeasures via Elementary Circuit Transformations”. In: *ACNS*. Vol. 10892. Lecture Notes in Computer Science. Springer, 2018, pp. 65–82.

[Cor+22] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. “High-order Table-based Conversion Algorithms and Masking Lattice-based Encryption”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.2 (2022), pp. 1–40.

[CS19] Gaëtan Cassiers and François-Xavier Standaert. “Towards Globally Optimized Masking: From Low Randomness to Low Noise Rate or Probe Isolating Multiplications with Reduced Randomness and Security against Horizontal Attacks”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 162–198.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. "Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference". In: *IEEE Trans. Inf. Forensics Secur.* 15 (2020), pp. 2542–2555.

[CS21a] Gaëtan Cassiers and François-Xavier Standaert. "Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.2 (2021), pp. 136–158.

[CS21b] Jean-Sébastien Coron and Lorenzo Spignoli. "Secure Wire Shuffling in the Probing Model". In: *CRYPTO (3)*. Vol. 12827. Lecture Notes in Computer Science. Springer, 2021, pp. 215–244.

[DDF19] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. "Unifying Leakage Models: From Probing Attacks to Noisy Leakage". In: *J. Cryptol.* 32.1 (2019), pp. 151–177.

[DF12] Stefan Dziembowski and Sebastian Faust. "Leakage-Resilient Circuits without Computational Assumptions". In: *TCC*. Vol. 7194. Lecture Notes in Computer Science. Springer, 2012, pp. 230–247.

[DFS19] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. "Making Masking Security Proofs Concrete (Or How to Evaluate the Security of Any Leaking Device), Extended Version". In: *J. Cryptol.* 32.4 (2019), pp. 1263–1297.

[DFZ19] Stefan Dziembowski, Sebastian Faust, and Karol Zebrowski. "Simple Refreshing in the Noisy Leakage Model". In: *ASIACRYPT (3)*. Vol. 11923. Lecture Notes in Computer Science. Springer, 2019, pp. 315–344.

[DN22] Siemen Dhooghe and Svetla Nikova. "Resilient uniformity: applying resiliency in masking". In: *Cryptogr. Commun.* 14.1 (2022), pp. 41–58.

[Duv+21] Sébastien Duval, Pierrick Méaux, Charles Momin, and François-Xavier Standaert. "Exploring Crypto-Physical Dark Matter and Learning with Physical Rounding Towards Secure and Efficient Fresh Re-Keying". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.1 (2021), pp. 373–401.

[Dwo+01] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. *Advanced Encryption Standard (AES)*. en. 2001. DOI: `https://doi.org/10.6028/NIST.FIPS.197`.

[Fau+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. "Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.3 (2018), pp. 89–120.

[FG05] Wieland Fischer and Berndt M. Gammel. “Masking at Gate Level in the Presence of Glitches”. In: *CHES*. Vol. 3659. Lecture Notes in Computer Science. Springer, 2005, pp. 187–200.

[Fri+22] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. “Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 414–460.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. “Generic Low-Latency Masking in Hardware”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.2 (2018), pp. 1–21.

[Gig+21] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. “Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs”. In: *USENIX Security Symposium*. USENIX Association, 2021, pp. 1469–1468.

[GM18] Hannes Groß and Stefan Mangard. “A unified masking approach”. In: *J. Cryptogr. Eng.* 8.2 (2018), pp. 109–124.

[GMK16a] Hannes Groß, Stefan Mangard, and Thomas Korak. “Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order”. In: *TIS@CCS*. ACM, 2016, p. 3.

[GMK16b] Hannes Groß, Stefan Mangard, and Thomas Korak. “Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order”. In: *IACR Cryptol. ePrint Arch.* (2016), p. 486.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. “An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order”. In: *CT-RSA*. Vol. 10159. Lecture Notes in Computer Science. Springer, 2017, pp. 95–112.

[Gou+21] Dahmun Goudarzi, Thomas Prest, Matthieu Rivain, and Damien Vergnaud. “Probing Security through Input-Output Separation and Revisited Quasilinear Masking”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (2021), pp. 599–640.

[GPM21] Barbara Gigerl, Robert Primas, and Stefan Mangard. “Secure and Efficient Software Masking on Superscalar Pipelined Processors”. In: *ASIACRYPT (2)*. Vol. 13091. Lecture Notes in Computer Science. Springer, 2021, pp. 3–32.

[GR17] Dahmun Goudarzi and Matthieu Rivain. “How Fast Can Higher-Order Masking Be in Software?” In: *EUROCRYPT (1)*. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 567–597.

[Gro+14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. “LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations”. In: *FSE*. Vol. 8540. Lecture Notes in Computer Science. Springer, 2014, pp. 18–37.

[GS18] Vincent Grosso and François-Xavier Standaert. “Masking Proofs Are Tight and How to Exploit it in Security Evaluations”. In: *EUROCRYPT (2)*. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 385–412.

[GST17] Daniel Genkin, Adi Shamir, and Eran Tromer. “Acoustic Cryptanalysis”. In: *J. Cryptol.* 30.2 (2017), pp. 392–443.

[Guo+20] Qian Guo, Vincent Grosso, François-Xavier Standaert, and Olivier Bronchain. “Modeling Soft Analytical Side-Channel Attacks from a Coding Theory Viewpoint”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.4 (2020), pp. 209–238.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. “Private Circuits: Securing Hardware against Probing Attacks”. In: *CRYPTO*. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 463–481.

[Jan+22] Jan Jancar, Marcel Fourné, Daniel De Almeida Braga, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. ““They’re not that hard to mitigate”: What Cryptographic Library Developers Think About Timing Attacks”. eng. In: *43rd IEEE Symposium on Security and Privacy*. San Francisco: IEEE, 2022.

[JPS18] Bernhard Jungk, Richard Petri, and Marc Stöttinger. “Efficient Side-Channel Protections of ARX Ciphers”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.3 (2018), pp. 627–653.

[JS17] Anthony Journault and François-Xavier Standaert. “Very High Order Masking: Efficient Implementation and Security Evaluation”. In: *CHES*. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 623–643.

[JWI20] JWIG. *Application of Attack Potential to Smartcards and Similar Devices*. 2020. URL: `https://www.sogis.eu/documents/cc/domains/sc/JIL-Application-of-Attack-Potential-to-Smartcards-v3-1.pdf` (visited on 04/23/2022).

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis”. In: *CRYPTO*. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397.

[KM22] David Knichel and Amir Moradi. “Low-Latency Hardware Private Circuits”. In: *IACR Cryptol. ePrint Arch.* (2022), p. 507.

[Koc96] Paul C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *CRYPTO*. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. “SILVER - Statistical Independence and Leakage Verification”. In: *ASIACRYPT (1)*. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 787–816.

[KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. “Generic Hardware Private Circuits Towards Automated Generation of Composable Secure Gadgets”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 323–344.

[LBS19] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. “Reducing a Masked Implementation’s Effective Security Order with Setup Manipulations And an Explanation Based on Externally-Amplified Couplings”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 293–317.

[Mas+22] Loïc Masure, Gaëtan Cassiers, Julien Hendrickx, and François-Xavier Standaert. “Information Bounds and Convergence Rates for Side-Channel Security Evaluators”. In: *IACR Cryptol. ePrint Arch.* (2022), p. 490.

[MCS22a] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. “Handcrafting: Improving Automated Masking in Hardware with Manual Optimizations”. In: *COSADE*. Vol. 13211. Lecture Notes in Computer Science. Springer, 2022, pp. 257–275.

[MCS22b] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. “Unprotected and Masked Hardware Implementations of Spook v2”. In: *IACR Cryptol. ePrint Arch.* (2022), p. 254.

[MDS02] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. “Examining Smart-Card Security under the Threat of Power Analysis Attacks”. In: *IEEE Trans. Computers* 51.5 (2002), pp. 541–552.

[Moo+19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. “Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 256–292.

[Mor+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. “Pushing the Limits: A Very Compact and a Threshold Implementation of AES”. In: *EUROCRYPT*. Vol. 6632. Lecture Notes in Computer Science. Springer, 2011, pp. 69–88.

[Mor+16] Kathleen M. Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. “PKCS #1: RSA Cryptography Specifications Version 2.2”. In: *RFC* 8017 (2016), pp. 1–78.

[MOW17] David McCann, Elisabeth Oswald, and Carolyn Whitnall. “Towards Practical Tools for Side Channel Aware Software Engineering: ’Grey Box’ Modelling for Instruction Leakages”. In: *USENIX Security Symposium*. USENIX Association, 2017, pp. 199–216.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. “Side-Channel Leakage of Masked CMOS Gates”. In: *CT-RSA*. Vol. 3376. Lecture Notes in Computer Science. Springer, 2005, pp. 351–365.

[MPW22] Ben Marshall, Dan Page, and James Webb. "MIRACLE: MIcRo-ArChitectural Leakage Evaluation A study of micro-architectural power leakage across many devices". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 175–220.

[Mül+22] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. "Transitional Leakage in Theory and Practice Unveiling Security Flaws in Masked Circuits". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.2 (2022), pp. 266–288.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. "Threshold Implementations Against Side-Channel Attacks and Glitches". In: *ICICS*. Vol. 4307. Lecture Notes in Computer Science. Springer, 2006, pp. 529–545.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches". In: *J. Cryptol.* 24.2 (2011), pp. 292–321.

[Ouy+21] Charles-Henry Bertrand Van Ouytsel, Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. "How to fool a black box machine learning based side-channel security evaluation". In: *Cryptogr. Commun.* 13.4 (2021), pp. 573–585.

[Pos+11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. "Side-Channel Resistant Crypto for Less than 2, 300 GE". In: *J. Cryptol.* 24.2 (2011), pp. 322–345.

[Pou+17] Romain Poussier, Qian Guo, François-Xavier Standaert, Claude Carlet, and Sylvain Guilley. "Connecting and Improving Direct Sum Masking and Inner Product Masking". In: *CARDIS*. Vol. 10728. Lecture Notes in Computer Science. Springer, 2017, pp. 123–141.

[PSV15] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. "Leakage-Resilient Authentication and Encryption from Symmetric Cryptographic Primitives". In: *CCS*. ACM, 2015, pp. 96–108.

[QS01] Jean-Jacques Quisquater and David Samyde. "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards". In: *E-smart*. Vol. 2140. Lecture Notes in Computer Science. Springer, 2001, pp. 200–210.

[Rep15] Oscar Reparaz. "A note on the security of Higher-Order Threshold Implementations". In: *IACR Cryptol. ePrint Arch.* (2015), p. 1.

[Rep+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. "Consolidating Masking Schemes". In: *CRYPTO (1)*. Vol. 9215. Lecture Notes in Computer Science. Springer, 2015, pp. 764–783.

[Res18] Eric Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3". In: *RFC* 8446 (2018), pp. 1–160.

[RP10] Matthieu Rivain and Emmanuel Prouff. “Provably Secure Higher-Order Masking of AES”. In: *CHES*. Vol. 6225. Lecture Notes in Computer Science. Springer, 2010, pp. 413–427.

[Sch08] Fritz Scholz. *Confidence Bounds & Intervals for Parameters Relating to the Binomial, Negative Binomial, Poisson and Hypergeometric Distributions With Applications to Rare Events*. 2008.

[Sch+19] Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. “Efficiently Masking Binomial Sampling at Arbitrary Orders for Lattice-Based Crypto”. In: *Public Key Cryptography (2)*. Vol. 11443. Lecture Notes in Computer Science. Springer, 2019, pp. 534–564.

[SÖP04] François-Xavier Standaert, Siddika Berna Örs, and Bart Preneel. “Power Analysis of an FPGA: Implementation of Rijndael: Is Pipelining a DPA Countermeasure?” In: *CHES*. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 30–44.

[TOS10] Eran Tromer, Dag Arne Osvik, and Adi Shamir. “Efficient Cache Attacks on AES, and Countermeasures”. In: *J. Cryptol.* 23.1 (2010), pp. 37–71.

[TV04] Kris Tiri and Ingrid Verbauwhede. “A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation”. In: *DATE*. IEEE Computer Society, 2004, pp. 246–251.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. “Soft Analytical Side-Channel Attacks”. In: *ASIACRYPT (1)*. Vol. 8873. Lecture Notes in Computer Science. Springer, 2014, pp. 282–296.

[Wan+20a] Weijia Wang, Chun Guo, François-Xavier Standaert, Yu Yu, and Gaëtan Cassiers. “Packed Multiplication: How to Amortize the Cost of Side-Channel Masking?” In: *ASIACRYPT (1)*. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 851–880.

[Wan+20b] Weijia Wang, Pierrick Méaux, Gaëtan Cassiers, and François-Xavier Standaert. “Efficient and Private Computations with Code-Based Masking”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.2 (2020), pp. 128–171.

[WGK13] Clifford Wolf, Johann Glaser, and Johannes Kepler. “Yosys-a free Verilog synthesis suite”. In: *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*. 2013.

Publications

Material of the thesis

The main contributions of this thesis are published in the following works.

[Bar+19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. "maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults". In: *ESORICS (1)*. Vol. 11735. Lecture Notes in Computer Science. Springer, 2019, pp. 300–318.

[BC22] Olivier Bronchain and Gaëtan Cassiers. "Bitslicing Arithmetic/Boolean Masking Conversions for Fun and Profit with Application to Lattice-Based KEMs". In: *IACR Cryptol. ePrint Arch.* (2022), p. 158.

[Cas+21a] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. "Towards Tight Random Probing Security". In: *CRYPTO (3)*. Vol. 12827. Lecture Notes in Computer Science. Springer, 2021, pp. 185–214.

[Cas+21b] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. "Hardware Private Circuits: From Trivial Composition to Full Verification". In: *IEEE Trans. Computers* 70.10 (2021), pp. 1677–1690.

[CS19] Gaëtan Cassiers and François-Xavier Standaert. "Towards Globally Optimized Masking: From Low Randomness to Low Noise Rate or Probe Isolating Multiplications with Reduced Randomness and Security against Horizontal Attacks". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 162–198.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. "Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference". In: *IEEE Trans. Inf. Forensics Secur.* 15 (2020), pp. 2542–2555.

[CS21a] Gaëtan Cassiers and François-Xavier Standaert. "Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.2 (2021), pp. 136–158.

Other works

The following publications have been written during the work that led to this thesis.

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. "Give Me 5 Minutes: Attacking ASCAD with a Single Side-Channel Trace". In: *IACR Cryptol. ePrint Arch.* (2021), p. 817.

[Bel+20c] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Mode-Level vs. Implementation-Level Physical Security in Symmetric Cryptography - A Practical Guide Through the Leakage-Resistance Jungle". In: *CRYPTO (1)*. Vol. 12170. Lecture Notes in Computer Science. Springer, 2020, pp. 369–400.

[Bel+20d] Davide Bellizia et al. "Spook: Sponge-Based Leakage-Resistant Authenticated Encryption with a Masked Tweakable Block Cipher". In: *IACR Trans. Symmetric Cryptol.* 2020.S1 (2020), pp. 295–349.

[Cas+19] Gaëtan Cassiers, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "SpookChain: Chaining a Sponge-Based AEAD with Beyond-Birthday Security". In: *SPACE*. Vol. 11947. Lecture Notes in Computer Science. Springer, 2019, pp. 67–85.

[Mas+22] Loïc Masure, Gaëtan Cassiers, Julien Hendrickx, and François-Xavier Standaert. "Information Bounds and Convergence Rates for Side-Channel Security Evaluators". In: *IACR Cryptol. ePrint Arch.* (2022), p. 490.

[MCS22a] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. "Handcrafting: Improving Automated Masking in Hardware with Manual Optimizations". In: *COSADE*. Vol. 13211. Lecture Notes in Computer Science. Springer, 2022, pp. 257–275.

[MCS22b] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. "Unprotected and Masked Hardware Implementations of Spook v2". In: *IACR Cryptol. ePrint Arch.* (2022), p. 254.

[Ouy+21] Charles-Henry Bertrand Van Ouytsel, Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. "How to fool a black box machine learning based side-channel security evaluation". In: *Cryptogr. Commun.* 13.4 (2021), pp. 573–585.

[Wan+20a] Weijia Wang, Chun Guo, François-Xavier Standaert, Yu Yu, and Gaëtan Cassiers. "Packed Multiplication: How to Amortize the Cost of Side-Channel Masking?" In: *ASIACRYPT (1)*. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 851–880.

[Wan+20b] Weijia Wang, Pierrick Méaux, Gaëtan Cassiers, and François-Xavier Standaert. "Efficient and Private Computations with Code-Based Masking". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.2 (2020), pp. 128–171.

Software and hardware

The following open-source software components and hardware IPs have been designed for this thesis. The main ones are listed below.

- The fullVerif tool described in Section 4.7 (`https://github.com/cassiersg/fullverif`).
- The STRAPS tool described in Section 5.2.4 (`https://github.com/cassiersg/straps`).
- The SCALib library that contains highly optimized implementations of state-of-the-art tools for side-channel security evaluation (`https://github.com/simple-crypto/SCALib`). SCALib was co-authored with Olivier Bronchain.
- Implementations of the Spook cipher (`https://www.spook.dev/implementations`):
  - The reference implementation in C (co-authored with Olivier Bronchain and Charles Momin).
  - Speed-optimized implementations for `x86_64` processors (in C, with architecture-specific optimizations).
  - A simple implementation in Python.
  - A speed-optimized implementation for ARM Cortex-M processors (Olivier Bronchain was the main author).
  - Hardware (FPGA and ASIC) implementations with optimized area vs. throughput trade-off (co-authored with Charles Momin).
  - Side-channel protected hardware (FPGA and ASIC) implementation using the HPC2 (see Section 4.3.1) masking scheme (co-authored with Charles Momin).
- A masked implementation of AES (128-bit key, encryption only) using the HPC2 masking scheme: `https://github.com/uclcrypto/aes-hpc2` (Charles Momin was the main author).

- Masked implementations of the Kyber and Saber KEMs [BC22] based on PINI gadgets (Section 3.2.2), including the conversion gadgets of Section 3.6: `https://github.com/uclcrypto/pqm4_masked` (co-authored with Olivier Bronchain).