PhD- FSTM-2022-143
The Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 22/11/2022 in Esch-Sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Lorenzo SPIGNOLI
Born on 11 July 1990 in Rome, (Italy)

SIDE-CHANNEL COUNTERMEASURES IN THE PROBING MODEL

Dissertation defence committee

Dr Jean-Sébastien Coron, dissertation supervisor
Professor, Université du Luxembourg

Dr Ingrid Verbauwhede
Professor, Katholieke Universiteit Leuven

Dr Alex Biryukov, Chairman
Professor, Université du Luxembourg

Dr Francois-Xavier Standaert
Professor, Université Catholique de Louvain

Dr David Naccache, Vice Chairman
Professor, Ecole Normale Supérieure

A chi non ha mai vinto.
A chi non è mai vinto.

Contents

Abstract

Cryptography in its more general sense, is treating cryptosystem as a single-piece of software, mostly with the aim of encrypting/decrypting messages. The so-called black-box model captures the abstract security of a cryptosystem, but it does not consider the physical implementation, where the mathematical description of a function is actually performed by a device. In fact, physical implementations contradict the assumption over the intermediate computation not being accessible to the attacker. In the physical cryptanalysis model, new attacks became possible; and the Side-Channel Analysis (SCA) targets one of them: i.e. the leakage produced by the computation. In this thesis, we focused on the countermeasures required to prevent side-channel attacks.

In [ISW03], Ishai, Sahai and Wagner proposed a new formal model to protect algorithms from implementation leakage. And, since its appearance at CRYPTO 2003, the ISW model seemed to be a perfect fit to counter SCA; especially, because the authors provided the first probably secure construction in the side-channel area. At CHES 2010, Rivain and Prouff (RP) introduced an elegant masking technique to protect the AES against power analysis attacks. RP masking is provably secure in the probing model, but this solid theoretical underpinning comes at the cost of a massive increase in execution time. The ISW model, along with the RP masking implementation, provided the context where we funnel our researches.

Our first contribution describes the first improvement of the wire shuffling countermeasure described in [ISW03]. More precisely, we show how to get worst case statistical security against t probes with running time $\mathcal{O}(t)$ instead of $\mathcal{O}(t \log t)$ (and $O(t^2)$ for the classical masking countermeasure); our construction is also much simpler to implement. We also provided a practical implementation for AES that outperforms the masking countermeasure for $t \geq 6\,000$.

Our second contribution describes a software optimization methods to accelerate the low-level arithmetic in the field $\mathbb{F}_{2^8}$, which has a significant impact on the overall performance of a masked implementation of the AES. Among these optimizations is an improved technique for table-based multiplication in $\mathbb{F}_{2^8}$ that allows one to avoid the special treatment of 0-values, thereby speeding up the multiplication of masked operands. Furthermore, we introduce a novel exponentiation-based algorithm for inversion in $\mathbb{F}_{2^8}$, which reduces the overall number of table

look-ups and the amount of randomness needed for the refreshing of masks compared to the original RP inversion.

1

Introduction

Before introducing the *Side-Channel Analysis* (SCA), it might be useful to briefly go through the evolution of *Cryptography* in order to give an insight into the whole field and its terminology. We will retrace some of the milestones which drove this discipline to become as we know it today: from the historical to the modern approach, from the private-key to the public-key settings, and from the classical to the physical cryptanalysis.

1.1 Historical vs Modern Cryptography

Leafing through modern dictionaries among many, there is a common definition that captures the essence of historical cryptography: "the art of writing or solving codes". In fact, at its very beginning, every attempt to describe cryptography was referred to as an *art*. This was due to the fact that the process of constructing or breaking codes relied exclusively on the ability of the cryptographer, who, indeed, had to develop a good knowledge of the codes mechanisms, however, the creativity of the individual was more crucial. Without any upon-agreed definition of what is meant to be "secure", encryption protocols were designed according to this mixture of creativity and ingenuity, and their security would be evaluated through an iterative process of searching for vulnerabilities and patching them ad hoc. This "artistic" approach seemed to be sufficient, considering that cryptography's initial application was limited to enabling two parties to communicate secretly, even in the presence of an eavesdropper who could monitor the exchange of messages between them. Moreover, such communicating parties were constituted predominantly by military organisations and governments, which, given the nature of the scope, were intended to keep the cryptography community as secret and restricted as possible. It is interesting to see how cryptographers were closer to illusionists than mathematicians in those early stages, focused more on misleading the eyes rather than dealing with theorems and proofs. Nevertheless, looking at it in retrospect, it is already possible to notice involuntary similarities with the *scientific method*; anticipating the modern evolution of the field as a proper *science* and a mathematical discipline.

The process that spotlighted this hidden nature lasted hundreds of decades and was characterised by several attempts, all considered secure a priori until the mechanisms behind them were exposed. This process of studying cryptographic systems to look for weaknesses or leaks of information is called *Cryptanalysis*, and it plays a crucial role even nowadays. Going in order, one of the oldest piece of evidence of encryption methods dates back to Julius Caesar in the last century CE. As established in *De Vita Caesarum*, Caesar used to *cipher* (i.e. encrypt) messages intended for his generals with the purpose of hiding the contents from his enemies. Julius Caesar's intuition was to use a "cyclically shifted" alphabet, where each letter is replaced by another letter located three places further in the alphabet; in this way, a message (also referred to as *plaintext*) like beginattacknow would have been sent as EHJLQDWWDFNQRZ (also refereed as *ciphertext*), making it look like gibberish to anyone not aware of the encryption process. However, what was considered a "simple trick" leaded, centuries later, to a whole category of encryption schemes called *monoalphabetic substitution ciphers*, where each letter of the plaintext is replaced univocally with some other character according to a different logic. They were popular until a study about frequency analysis came out, breaking such cryptosystems. In fact, the study proposed by Al-Kindi showed a pattern between specific letters and their occurrence frequency in the language, making it easy to map each letter of the ciphertext with the corresponding one in the plaintext. In the 16th century, monoalphabetic ciphers were succeeded by *polyalphabetic substitution ciphers*: within the same cipher, the encryption algorithm alternated different monoalphabetic substitutions. The *Vegenère's cipher* is a famous example of this category, but it was also considered unbreakable for many years until an extension of the frequency analysis proved otherwise. In turn, in the 19th and 20th centuries, more advanced polyalphabetic ciphers appeared, which differed from the previous ones by their exploitation of mechanical devices. The Enigma machine belongs to this kind, and the Germans used it to encrypt radio communication during WWII. Enigma allowed the encoding of a message in billions of ways, making it incredibly difficult for other nations to crack German codes during the war and for a time the code seemed unbreakable. As we know from history, the Allies eventually broke Enigma, gaining the upper hand in the resolution of the conflict.

Moreover, cryptography propelled its transformation with the emergence of electronic devices and computers in the late 20th century. On one side, these new technologies speeded up complex computation in a reasonable time, allowing the design of more complex ciphers. In the 70s, a block cipher proposed by cryptographers at IBM was selected by the US National Bureau of Standards (NSA) as the first official Data Encryption Standard (DES); DES has been (and still is) widely used, analysed and improved since then. On the other side, the development of general-purpose computers extended the need for security beyond the governmental circles, capturing more and more attention, especially among mathematicians. These new large-scale consumers of cryptography pushed for a more rigorous approach, with the idealistic goal to provide the encryption scheme with a "certificate" (i.e. a *proof*) attesting its security. Consequently, the community began to see "mathematics" as a powerful mean to fill this scientific-approach gap. In fact, as we will see, Kerckoffs' law, Shannon's *information theory*, the *complexity theory* and other studies have influenced how researchers think about

the broader field of computer security. In sum, cryptography has gone from a heuristic set of tools concerned with ensuring secret communication for the military to a science that helps secure systems for ordinary people all across the globe, forcing the field to evolve into a branch of applied mathematics and computer science.

An emphasis on definitions, assumptions and proofs distinguishes modern cryptography from classical. Those three principles are the foundations for whoever wants to approach the discipline, so it is worth briefly mentioning them. Formal definitions consist of a not ambiguous description of what threats must be prevented and what security guarantees must be achieved. A priori understanding of the security goal has been recognized as more effective than going the other way around. Additionally, it is interesting to notice that there are multiple (and all correct) ways to define security and the "better" definition exclusively depends on the context in which the scheme is deployed. In turn, a clear and exhaustive definition allows us to achieve the ultimate goal of producing security proofs. Such proofs act as a guarantee ensuring that no adversary can break the construction taken into consideration. Unfortunately, in practice, most cryptographic schemes cannot be proven unconditionally; otherwise, such proofs would be used to answer harder open questions from the computational complexity that seems far from being solved. Consequently, proofs of security typically rely on assumptions: some statement that is not proven but collectively conjectured to be true. As we said for the definitions, also for assumptions is challenging to define a scale, but clearly, older and long-studied assumptions should be preferred compared to new and ad hoc ones. Alternatively, as a general guideline, assumptions with a simple statement are often chosen too, since they are easier to study (or refute).

1.2 Symmetric vs Asymmetric Cryptography

It is possible to divide modern cryptography into two main categories: *Symmetric* and *Asymmetric*. As implicitly described in the previous section, symmetric encryption involves using the same key for both encryption and decryption; the plaintext is encrypted depending on the key, and the same key is required to decrypt the ciphertext back to the original form. Hence, the key must be known in advance since it is used by both the sender and receiver. How such a key is securely exchanged remains out of the scope and is assumed to be already staged. For example, the secrecy of the "red line" used by the White House and the Kremlin during the Cold War was based on a symmetric-key encryption scheme called *One-Time Pad*, where a long key, written on paper and handily exchanged between the US and URSS emissaries, was applied. Clearly, whoever knows the key is able to encrypt or decrypt any message exchanged, so it is essential to keep it secret. Since the same key is used, symmetric cryptography is also referred to as *private-key* cryptography.

Most of the above-mentioned protocols, like the Caesar's and the Vegenère's ciphers, or the DES and all its variants, fall into the category. Despite being one of the oldest encryption techniques, symmetric schemes are still widely used today since they are faster and easier to

implement compared to their asymmetric counterpart. This is due to several aspects that are still not achieved in asymmetric encryption: the ciphertext has the same length as the plaintext, the key shared by the parties hardly exceeds 128/256-bits and in most cases, the encryption phase requires a low-usage of resources. For such reasons and more, symmetric encryption is preferable in a context where a large amount of data needs to be encrypted and transmitted. Although, a few drawbacks balance the advantages from a theoretical point of view. The first has already been mentioned and it is an a priori assumption that the key has already been shared among the involved parties. Despite the problem, this brings to a slow-rate scaling, since sharing the same key with a new party must be carefully managed. Another relies on the fact that symmetric encryption suffers from key exhaustion issues; multiple usages of the same key over time may leak information about the key itself, which may lead an adversary to a complete reconstruction. Additionally, as we will see in full detail since it is the subject of the thesis, side-channel attacks can recover keys by exploiting the hardware characteristic of the devices where the protocol is running. Consequently, without appropriate key hierarchy and rotation, the probability of a breach represents a serious issue, especially for large-scale systems.

In opposition to symmetric, asymmetric cryptography enables parties to communicate privately without assuming that some secret information has already been shared. In asymmetric settings, the receiving party computes a *pair* of keys, one for the encryption phase (called the *public key*) and one for decrypting (called the *secret key*). Specifically, the public key is used by the sender to encrypt the plaintext. In contrast, the receiver requires the secret key to decrypt the ciphertext. Note that both keys are computed by the receiver and are independent of any specific senders. In this way, the receiver may publish its public key (say on its website or business card), and whoever wants to send a message can obtain the public key and proceeds with the ciphering; clearly, differently from the private-key settings, here, the roles in the exchange are not interchangeable: the receiver cannot reply using its keys, but it needs the sender's public key. This new way of exchanging messages marked a revolution in cryptography, allowing communication between parties to take place entirely through insecure channels. At the same time, in symmetric encryption protocols, the initial stage (i.e. the agreement on the secret key) can only occur via secure channels. On the flipping side, such a public key has to be assumed to be known by any attackers as well. For evident reasons, asymmetric cryptography is also referred to as *public-key cryptography*.

Hence, this new paradigm can overcome the significant disadvantage of private-key encryption: the issues related to the key distribution (at least to some extent). As explained, no data needs to be computed and agreed on during the process. For the same reason, asymmetric encryption is considered more secure and perfect for large-scale and "open" systems. Furthermore, public-key cryptography allows *digital signatures*, the digital equivalence of a handwriting signing on a paper document. This feature incorporates properties like *message authentication* and *non-repudiation* and has become an essential requirement nowadays; the private key is used to generate a signature which is hard to forge, and it identifies non- ambiguously a user while anyone may use the public key to confirm the signature of such user,

as well as, once the user signed some data it is impossible to deny it afterwards. On the other side, these stronger security guarantees come at a price: efficiency. In fact, compared to private-key settings, public-key encryption requires very high resource utilization, mathematical time-consuming operations and large-sized keys, and it produces longer ciphertexts with respect to the initial plaintexts. In turn, if symmetric is preferable for the transmission of big chunks of data, the asymmetric counterpart is used to transfer a small amount of information.

As one last point, it may be interesting to underline the difference in how the two branches are evaluated and then considered secure. Usually, symmetric ciphers come with proofs of correctness, i.e. the decryption algorithm always returns the correct plaintext. While concerning the actual security, block-ciphers need to pass through years of unsuccessful cryptanalysis; in years, no attack can break the system has been found. Note that the attacks must be faster than an exhaustive search, which is always possible but at the rate of hundreds of years. Instead, asymmetric cryptosystems mostly rely on a hard problem from number theory, where the hardness is defined as the computational impossibility of finding a solution to the problem. Factorization of large prime numbers and discrete logarithm computation in multiplicative groups are the most used (but not the only ones) in modern cryptography. Here, the security proof is based on reduction; that is, claiming and proving that the existence of an adversary able to break the system would result in breaking the underlying hard problems.

1.3 Black-Box Model vs Physical Cryptanalysis

So far, we have dealt with cryptography in its more general sense, treating a cryptosystem as a single piece of software, mainly intending to encrypt/decrypt messages. It evolved in more than that; a cryptosystem is essentially an algorithm implementing security services like confidentiality, integrity, authenticity and more. In turn, cryptosystems use *cryptographic primitives* as building blocks to achieve these specific security services. Cryptographic primitives are designed to compute some very specific task in a very precisely defined and highly reliable fashion.

Thus, cryptographic primitives are single components of larger and more complex systems. Hence, if a primitive is breakable, the whole system has to be considered breakable. On the other side, improving the security or the efficiency of primitives benefits the entire system. Nevertheless, both cryptosystems and primitives can be investigated in the same ways. The classical way is usually called black-box: in this model, protocols are assumed to be attacked by adversaries, which cannot exploit the inner structure of the primitives nor the intermediate computations of building blocks. That is, schemes are treated as oracles (i.e. black-boxes), where the adversary may query with inputs of its choice but obtain only the final output. Moreover, the security is evaluated according to the abilities of the adversary to query the oracle (un)bounded number of times, obtaining (un)bounded number of input/output pairs and still not recovering sensitive information from them. The black-box model captures the abstract security of a cryptosystem or a primitive but it does not consider the physical imple-

mentation, where the mathematical description of a function is actually performed by a device. In fact, physical implementations contradict the assumption of the intermediate computation not being accessible to the attacker. In practice, observations like the execution time, the power consumption or the electromagnetic radiation produced during the computation may be translated into actual intermediate values, giving the adversary information not considered in the black-box model. In the *physical cryptanalysis* model, new attacks became possible. They can be divided into two main categories: *fault analysis*, where the computation is disrupted, causing improper results, and *side-channel analysis*, considered the leakage produced by the computation. We will focus on the second category, where we funnelled our research.

1.3.1 Side-Channel Analysis

As we mentioned, physical attacks take advantage of implementation-specifics characteristics to recover sensitive data. Therefore, this dependency on the specific implementation makes such attacks less general and, in turn, harder to contrast. Despite their peculiarity, the literature classified them as follows:

1. *Invasive attacks*: where a physical intervention on the device is required from the adversary. Namely, physically probe a wire of the device with a needle to observe the data passing through it.
2. *Non-Invasive attacks*: where the adversary exploits information unintentionally, but automatically, leaked by the device. Such external data are often running-time, power consumption etc.
3. *Active attacks*: where the attacker tries to intentionally jeopardise the correct functioning of the machine. As an example, fault-injections actively induce errors in the computation and observe the devise response.
4. *Passive attacks*: where the information leaked comes only from the act of observing the device's behaviour. Power analysis is the most common passive attack in which the power consumption of a device executing the algorithm is analysed to find the secret data.

However, the attacks most often considered in the literature are non-intensive and passive; i.e. timing information, power consumption and electric radiations. Moreover, the numerous range of attacks produced, joined with the exponential increase of software applications in our daily life, has turned side-channel attacks into a real and practical concern. For those reasons, we are going to summarize the state of the art of the discipline. Or at least, the state of art the that provided the humus for our research.

Hence, since their introduction in the late 90s, side-channel attacks have been under several investigations. Despite the numerous customised patches adopted against well-known physical attacks, a heavy effort has been made to develop protection strategies. Among them, one of the widely used relies on applying *secret sharing* at the implementation level; where any sensitive data x is manipulated as a set of n shares. The so-called high-order masking was first introduced in [CJRR99], but it gained popularity with the milestone publication [ISW03]

at CRYPTO in 2003. Ishai, Sahai and Wegner's strategies achieved provable security in the so-called *probing security model*. Ishai et al. proposed a high-order masking scheme for multiplication over $\mathbb{F}_2$ in quadratic complexity; in other words, they worked in the boolean context. The ISW scheme was later used by Rivain and Prouff to pass from boolean to arithmetic masking, [RP10]. This extension to $\mathbb{F}_{2^n}$ allowed to introduce an efficient masked implementation of AES. This represented a turning point since before their work all previous masking schemes were proved to be secure only for 1-order or 2-order. In fact, the randomized-table countermeasure [CJRR99] achieved security only against 1-order attacks; while the Schramm-Paar countermeasure [SP06], designed to resist any n-order attacks, has been successfully broken by a 3-order attack in [CPR07].

Thus, over the years, Rivain-Proof masking has been investigated and a lot of publications came out to improve the efficiency via different approaches; let's recall some of them. In [Cor14], Coron described a technique to mask look-up tables of block-cipher of any order. Moreover, in those years, a flaw in [RP10] has been discovered concerning the refreshing of the used randomness in the multiplication algorithm. Such flaw was spotted by the authors themself, joined with others, in [CGPZ16]; and definitively fixed in [BBD+16]. Actually, Barthe et al. have contributed with their work in [BBD+16], not only in proposing a secure way to refresh randomness but also in introducing a new stronger security definition called *strong non-interference*, now considered among the most appreciated security in literature, in the physical attacks context.

Hence, since the beginning, it has been noticed that the most challenging part of the masking process is the non-linear operations; while any linear function can be masked easily and in $\mathcal{O}(n)$, there is no way (so far) to mask non-linear functions in less the $\mathcal{O}(n^2)$. Thus, the total complexity is dependent on the number of non-linear multiplications involved in the polynomial evaluation. The work done by Coron, Roy and Vivek [CRV14] represents the current best-known generic method for this aim. Furthermore, an alternative to the ISW-based polynomial methods was proposed by Carlet, Prouff, Rivain and Roche in [CPRR15], where they introduced a so-called algebraic decomposition method that can express an S-box in terms of a polynomial of low-degree.

In conclusion, despite its popularity and the many improvements proposed by the community, high-order masking still implies a strong overhead and practical implementation limitations. We will show our new contributions in this area, both theoretical and practical, hoping to get to reduce the gap existing between academia and practical usage in the real world.

2

Preliminaries

In this chapter, we are going to recall the basic background mentioned in the rest of this thesis. We start by characterizing the notion of circuits (see Section 2.1) and introducing the masking countermeasure (see Section 2.2). Then, we describe the operations performed by the encryption scheme AES (see Section 2.3).

2.1 Circuits

As in [ISW03], we consider probing attacks mainly in the setting of circuits. A deterministic circuit C can be pictured as a directed acyclic graph, where gates are represented as vertices and the wires are expressed as edges. As a well-established practice, we also assume that every gate has fan-in and fan-out at most 2. Note that this will not imply a loss in the generality since it is always possible to transform any circuit with general fan-in/fan-out in one belonging to the above type, with a logarithmic cost in the depth. Moreover, a random-bit gate is a gate having no inputs (i.e. fan-in 0) and outputting a random bit, that is sent along its output wire; the bit is selected uniformly and independently. The size of a circuit, denoted by $|C|$, is the number of gates, while its depth is defined as the length of the longest path from an input to an output. We refer to a circuit just described as a *stateless* circuit.

In contradistinction, a *stateful* circuit is a stateless circuit but augmented with memory cells. A memory cell is a gate with fan-in and fan-out 1, that, on any invocation, outputs the previous input to the gate, and stores the current input for the next invocation. In other words, a memory cell acts as a delay element. When C denotes a stateful circuit, it is required that such circuit is initialized with some initial state s_0; usually, $C[s_0]$ is used to indicate a circuit C with memory cells initially filled with s_0. Stateful circuits can also have external input and output wires.

2.2 Masking Countermeasure

As mentioned in the previous chapter, SCA attacks are based on the observation of a set of intermediate variables (say, for example, t) computed during a cryptosystem processing. One of the most used and studied techniques is the randomization of the cryptosystem in such a way that the probability distribution of any subset of at most t intermediate computations is independent of the secret key. In order to achieve this randomization, it is applied to the data of the circuit what has been called the *masking countermeasure*, [CJRR99]. In a nutshell, this technique maps any sensitive variable x (i.e a variable that depends on some required-secret information) into n shares $x_1, x_2, \ldots, x_n$ satisfying the following equation:

$$x_1 \oplus x_2 \oplus \ldots \oplus x_n = x \tag{2.1}$$

Often the first $n-1$ shares (also called *masks*) are initially assigned uniformly at random, while x_n is computed as $x_n = x \oplus x_1 \oplus \ldots \oplus x_{n-1}$ to maintain the validity of Equation (2.1). Furthermore, the term *order* of the masking is identified as the number of shares generated at random; for instance, if we take the above masking in the examination, we would say that it is an $(n-1)$-order masking. In literature, sometimes the order has been also referred to as d; but, in this thesis, to avoid confusion, we will mostly stick to the notation "n" for the number of shares and "$n-1$" for the order. The main guarantee provided by an $(n-1)$-order masking is that every $(n-1)$-tuple of intermediate variables is statistically independent of the secret key.

When the masking is applied to some cryptosystem, the challenge is to deal with the shares of x and not with x itself; each operation must be redesigned to manipulate shares while the input-output functionality is maintained. On one side, linear operations are straightforward to mask since they can be computed independently and at the shares-level (i.e between shares with the same index). That is, for every linear function f the following equation is held:

$$f(x_1) \oplus f(x_2) \oplus \ldots \oplus f(x_n) = f(x_1 \oplus x_2 \oplus \ldots \oplus x_n) \tag{2.2}$$

On the flipping coin, properly masking non-linear functions (like boolean or arithmetic multiplication or implementation of the S-box used in AES, see Section 2.3) is a harder task. How to compute those non-linear functions has a crucial role, in fact, it will imply how fast, how efficient and how secure the masked cryptosystem is; making it more or less practical in the real world.

Concerning, again, the notation adopted in this thesis, we would like to remark to the reader that the set of shares or masks will mainly be referred to as a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ or as a set $(x_i)_{1 \leq i \leq n}$; moreover, sometimes we will refer to a specific subset of shares with the notion $x|_I := (x_i)_{i \in I}$, meaning only the shares whose index belongs to the set I must be considered.

2.3 Advanced Encryption Standard

The block cipher Rijndael was selected by NIST in 2001 to become the encryption standard called *Advanced Encryption Standard* (or AES) [DR02]. We are going to briefly recall its structure. It might be useful to start from the difference between Rijndael, proposed by Deamen and Rijmen, and the final accepted standard AES. Such disparities only involve the range of parameters in the length of both blocks and key applicable to the cryptosystem. Concretely, while Rijndael allows every multiple of 32 bits in the range $[128, 256]$ bits; AES supports only blocks of fixed length 128 bits with a key that may be 128, 192 or 256 bits long.

2.3.1 Specifications of Rijndael

The Rijndael block cipher is structured as a multiple rounds transformation. Such transformations are applied to what they have called the *state*: one-dimensional arrays of 8-bit bytes. Where the input state is the plaintext and the output state is the ciphertext. The state is usually represented as a $4 \times \mathsf{N_b}$ matrix, where $\mathsf{N_b}$ is the number of columns (computed as the block length divided by 32); symmetrically, the key is represented as a $4 \times \mathsf{N_k}$ matrix, where $\mathsf{N_k}$ is the number of columns (computed as the key length divided by 32). See Figure 21 for an illustration. Note that, even if we have displayed the state and the key in their array notation (i.e. $s_0, s_1, s_2, \ldots, s_{4\cdot\mathsf{N_b}}$ and $k_0, k_1, k_2, \ldots, s_{4\cdot\mathsf{N_k}}$), sometimes, for convenience, we will use the matrix notation of the elements (i.e. $s_{i,j}$ $0 \leq i \leq 4 \cdot \mathsf{N_b}$ and $k_{i,j}$ $0 \leq i \leq 4 \cdot \mathsf{N_k}$).

$$\begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix} \qquad \begin{pmatrix} k_0 & k_4 & k_8 & k_{12} & k_{16} & k_{20} \\ k_1 & k_5 & k_9 & k_{13} & k_{17} & k_{21} \\ k_2 & k_6 & k_{10} & k_{14} & k_{18} & k_{22} \\ k_3 & k_7 & k_{11} & k_{15} & k_{19} & k_{23} \end{pmatrix}$$

Fig. 21: State and Key layout for $\mathsf{N_b} = 4$ and $\mathsf{N_k} = 6$.

For each element (i.e byte), the authors used the polynomial representation of $\mathbb{F}_{2^8}$. Namely, in the scheme, each $s_{i,j}$ and $k_{i,j}$ is treated as a polynomial of a degree smaller the 8, with a coefficient in $\mathbb{F}_2$. Moreover, the multiplication has been chosen to work modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. As mentioned at the beginning of the Section, Rijndael consists of an iterated application of a round transformation. The number of rounds is fixed and determined according to the block and key length, we refer to it as $\mathsf{N_r}$. In practice, such values are specified in Figure 22.

$\mathsf{N_k}$ \ $\mathsf{N_b}$	4	5	6	7	8
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

Fig. 22: Number of Rounds of Rijndael. Note that for AES $\mathsf{N_b}$ is fixed to 4 and $\mathsf{N_k} \in \{4, 6, 8\}$.

In a nutshell, each round is a composition of the following operations:

1. SubBytes: is the only non-linear transformation of the cipher. This transformation consists of a specific S-box applied to the state. Such S-box is designed to achieve non-linearity (minimizing the maximum input-output correlation amplitude and the maximum difference propagation probability) and algebraic complexity. The chosen S-box is defined by the following function in $\mathbb{F}_{2^8}$:

$$\mathsf{SB}_8 : a \rightarrow \mathrm{Aff}_8(\mathrm{Inv}_8(a)) \tag{2.3}$$

Where Inv_8 is defined in Equation (2.4) and it is the inversion function in the field;

$$\mathrm{Inv}_8 : a \rightarrow a^{-1} \tag{2.4}$$

while Aff_8 is an invertible affine transformation produced from a polynomial multiplication XORed with a constant and is defined in Equation (2.5):

$$\mathrm{Aff} : a \rightarrow \left(\begin{bmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{bmatrix} \times a \right) \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{2.5}$$

2. ShiftRows: is a byte transposition that cyclically shifts the rows of the state over different offsets (respectively 0, 1, 2 and 3 for the first, second, third and fourth row); see Figure 23. Such offsets are chosen to achieve optimal diffusion to maximal resistance against truncated differential attacks.

$$\begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix} \rightarrow \begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_5 & s_9 & s_{13} & s_1 \\ s_{10} & s_{14} & s_2 & s_6 \\ s_{15} & s_3 & s_7 & s_{11} \end{pmatrix}$$

Fig. 23: The input state (right) vs ShiftRows's output state.

3. MixColumns: is a permutation of the state column by column. The columns of the state are multiplied by the polynomial $3x^3+x^2+x+2 \mod x^4+1$, see Figure 24. The polynomial has been chosen considering dimension, linearity, diffusion and performance on 8-bit processors.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} s_{4 \cdot i} \\ s_{(4 \cdot i)+1} \\ s_{(4 \cdot i)+2} \\ s_{(4 \cdot i)+3} \end{bmatrix}$$

Fig. 24: The multiplication of the polynomial $3x^3 + x^2 + x + 2 \mod x^4 + 1$ is performed per each column of the state, i.e for $1 \leq i \leq 4$.

4. AddRoundKey: is a XORing operation between the state and RoundKey, namely each round uses its own round key. The array of round keys (called ExpandedKey[]) comes from a key schedule procedure taking as input the key of the cipher and, according to the length of the key see Algorithm 2.1 and Algorithm 2.2, returns an array of $\mathsf{N_k}$ RoundKeys. The size of the ExpandedKey is equal to the block length multiplied by the number of rounds.

$$\begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix} \oplus \begin{pmatrix} k_0^{(r)} & k_4^{(r)} & k_8^{(r)} & k_{12}^{(r)} \\ k_1^{(r)} & k_5^{(r)} & k_9^{(r)} & k_{13}^{(r)} \\ k_2^{(r)} & k_6^{(r)} & k_{10}^{(r)} & k_{14}^{(r)} \\ k_3^{(r)} & k_7^{(r)} & k_{11}^{(r)} & k_{15}^{(r)} \end{pmatrix}$$

Fig. 25: The input state (right) is XORed with the key computed by ExpandedKey for the r-th round, with $1 \leq r \leq \mathsf{N_r}$.

Algorithm 2.1 KeyExpansion - for $\mathsf{N_k} \leq 6$

Input: CipherKey: K[4][$\mathsf{N_k}$]; S-box: SB[256]; RC: [1,0,0,0].
Output: ExpandedKey: W[4][$\mathsf{N_b}(\mathsf{N_r}+1)$]

for $j = 0$ **to** $\mathsf{N_r} - 1$ **do**
  for $i = 1$ **to** 3 **do**
    W[i][j] $\leftarrow$ K[i][j]
for $j = \mathsf{N_k}$ **to** $\mathsf{N_b}(\mathsf{N_k}+1) - 1$ **do**
  if $j \mod \mathsf{N_k} = 0$ **then**
    W[0][j] $\leftarrow$ W[0][$j - \mathsf{N_k}$] + SB[W[0][$j-1$]] + RC[$j/\mathsf{N_k}$]
    for $i = 1$ **to** 3 **do**
      W[i][j] $\leftarrow$ W[i][$j - \mathsf{N_k}$] + SB[W[$i+1 \mod 4$][$j-1$]]
  else
    for $i = 0$ **to** 3 **do**
      W[i][j] $\leftarrow$ W[i][$j - \mathsf{N_k}$] + W[i][$j-1$]
return W

Algorithm 2.2 KeyExpansion - for $\mathsf{N_k} > 6$

Input: CipherKey: $\mathsf{K}[4][\mathsf{N_k}]$; S-box: SB[256]; RC: [1,0,0,0].
Output: ExpandedKey: $\mathsf{W}[4][\mathsf{N_b}(\mathsf{N_r}+1)]$

for $j = 0$ **to** $\mathsf{N_r} - 1$ **do**
  for $i = 1$ **to** 3 **do**
    $\mathsf{W}[i][j] \leftarrow \mathsf{K}[i][j]$
for $j = \mathsf{N_k}$ **to** $\mathsf{N_b}(\mathsf{N_k}+1) - 1$ **do**
  if $j \mod \mathsf{N_k} = 0$ **then**
    $\mathsf{W}[0][j] \leftarrow \mathsf{W}[0][j - \mathsf{N_k}] + \mathrm{SB}[\mathsf{W}[0][j-1]] + \mathrm{RC}[j/\mathsf{N_k}]$
    for $i = 1$ **to** 3 **do**
      $\mathsf{W}[i][j] \leftarrow \mathsf{W}[i][j - \mathsf{N_k}] + \mathrm{SB}[\mathsf{W}[i+1 \mod 4][j-1]]$
  else if $j \mod \mathsf{N_k} = 4$ **then**
    for $i = 0$ **to** 3 **do**
      $\mathsf{W}[i][j] \leftarrow \mathsf{W}[i][j - \mathsf{N_k}] + \mathrm{SB}[\mathsf{W}[i][j-1]]$
return W
  else
    for $i = 0$ **to** 3 **do**
      $\mathsf{W}[i][j] \leftarrow \mathsf{W}[i][j - \mathsf{N_k}] + \mathsf{W}[i][j-1]$
return W

In conclusion, we recall the algorithm: the procedure starts with the expansion of the key and the first XOR between the initial state and the cipher key follows. Afterwards, the algorithm iterates the round procedure for $N_r - 1$ times, since the last round is composed slightly differently. Namely, for the last round, the MixColumn is removed, as it adds no security to the cipher since it can be trivially inverted. See Algorithm 2.3 for the details.

Algorithm 2.3 AES - Encryption protocol

Rijndael(State,CipherKey)
  KeyExpansion(CipherKey, ExpandedKey)
  AddRoundKey(State, ExpandedKey[0])
  for $i = 1$ **to** $N_r - 1$ **do**
    SubBytes(State)
    ShiftRows(State)
    MixColumn(State)
    AddRoundKey(State, ExpandedKey[i])
  SubBytes(State)
  ShiftRows(State)
  AddRoundKey(State, ExpandedKey[N_r])

Remark 2.1. In this section, only the encryption phase has been considered since the decryption is straightforward. In fact, note that each of the above operations is easily invertible and for

this reason they have been omitted. Nevertheless, we refer the reader to [DR02] for the inverted variant of the above functions, as well as, the decryption algorithm and the security proofs.

3

Side-Channel Analysis

In this chapter, we are going to see in detail some of the publications mentioned in the Introduction. More specifically, we face the state of the art that is at the bottom of the contributions proposed in this thesis and done in those four years. From the ISW model (see Section 3.1) to the first proven secure masked implementation of AES (see Section 3.2). Introducing the SNI security (see Section 3.3). Lastly, we describe the models considered in our works: the regional probing model (see Section 3.4) and the software probing model (see Section 3.5).

3.1 The ISW Model

In [ISW03], Ishai, Sahai and Wagner showed a new formal model to protect primitives from implementation leakage, but from a designing point of view. Their work had the aim of proposing new security definitions to guarantee an inherent resistance against a large class of side-channel attacks. The goal was to build protection for generic probing attacks. To achieve the generic characteristic, the approach was to ignore how the information leaks but to focus on predicting how much and at what rate the information is going to leak. Moreover, their model is generic in the sense that privacy is ensured for any cryptosystem or primitive. Consequently, the authors reasoned in terms of circuits; in particular, they showed how to transform any circuit into a different and larger one, where security is guaranteed even in presence of an attacker observing a bounded amount of wires. In other words, the ISW model provided definitions of security against probing attacks and constructions that are proven to achieve such definitions.

The impossibility of achieving obfuscation, presented in [BGI$^+$01], forced some limitation in the power of the attacker considered in their definitions. In fact, the result in [BGI$^+$01] can be extended to any adversary probing any internal computation. The limitation introduced in their paper was defined as a parameter: in their model, they considered attackers that can observe or probe up to t wires in the examined circuit within one clock cycle. Nevertheless, such a t-limited adversary sounded like a reasonable restriction either considering the fact that in practice almost all known side-channel attacks are able to recover just part of the

information or taking into account that a single physical probe is performed through a very expensive needle, making a bound over the number of probes that a real-world attacker can afford. Finally, it is interesting to stress that no theoretical model formulation, before [ISW03], had ever achieved *provable* security. The authors defined security in terms of *transformers* T: an efficiently computable randomized algorithm taking as input a circuit C and returning a different circuit C'. In addition, such C' needs to encounter some properties: that is C' needs to satisfy *soundness* and *privacy* to be considered t-private. The several definitions given in [ISW03] differ from each other according to how soundness and privacy are stated; in turn, they are characterized by: 1) the type of the examined circuit, i.e stateless or stateful and 2) the straight of the privacy guarantees, i.e. perfect, statistical or computational.

In the following subsections, we are going to recall definitions, constructions and proofs for the perfect privacy in stateless (Section 3.1.1) and stateful (Section 3.1.2) settings; then we will introduce the random probing model (Section 3.1.3) since it is essential to understand definitions, constructions and proofs for the statistical privacy in stateless (Section 3.1.4) and stateful (Section 3.1.5) setting.

3.1.1 Perfect Privacy for Stateless Circuits

Definition. Informally, to achieve perfect privacy for stateless circuits there must exist a transformer able to take any circuit C and compute it in order to output a new circuit C', where the input-output functionality of C and C' are identical for any distinguisher and any adversary being able to probe up to t internal wires of C' would learn nothing from those values. In addition, the original input and output of C should be kept hidden. In order to satisfy this requirement, the transformer can make use of a randomized *input encoder* I and an *output decoder* O. Both I and O cannot be accessed (or probed) by the adversary and they need to be designed to be independent of the circuit C. Note that the assumption of not being probed for the encoder/decoder can be easily obtained in practice too; since typically such circuit needs very few gates to be computed, they may be implemented by temper-resistant hardware components, where the expensive cost is balanced by the small size. Formally:

Definition 3.1 (Perfect privacy for stateless circuits [ISW03]). *Let T be an efficiently computable deterministic function mapping a stateless circuit C to a stateless circuit C', and let I, O be as above. We say that (T, I, O) is a t-private stateless transformer if it satisfies the:*

1. ***Soundness.*** *The input-output functionality of $O \circ C' \circ I$ (i.e., the iterated application of I, C', O in that order) is indistinguishable from that of C.*
2. ***Privacy.*** *It is required that the view of any t-limited adversary, which attacks $O \circ C' \circ I$ by probing at most t wires in C', can be simulated from scratch, i.e. without access to any wire in the circuit. The identity of the probed wires has to be chosen in advance by the adversary.*

Construction. In [ISW03], the authors presented their construction to achieve perfect privacy in the stateless setting. As formalized below, they used a simple secret-sharing scheme, this

technique is often referred to as *masking countermeasure*. The idea is to deal with every intermediate variable x as n shares $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ and process each x_i separately along the circuit. As a consequence, the gates have to be carefully computed in such a way that intermediate computations do not leak sensitive information. For security against t probes, the three algorithms (I, T, O) require $n = 2t + 1$ shares; and they are defined as follows:

- Encode I. Each binary input x is mapped to n binary values. First, $n - 1$ random bits $r_1, \ldots, r_{n-1}$ are independently generated. The encoding of x is composed of these $n - 1$ random values together with $r_n = x \oplus r_1 \oplus \ldots \oplus r_{n-1}$. The circuit I computes the encoding of each input bit independently.
- Decode O. The output returned by $T(C)$ has the form $y_1, \ldots, y_n$. The associated output bit of C computed by O is $y_1 \oplus \ldots \oplus y_n$.
- Transformer T. Assume without loss of generality that the original circuit C consists of only XOR and AND gates. The transformed circuit C' maintains the invariant that corresponding to each wire in C will be n wires in C' carrying an n-sharing of the value on that wire of C. More precisely, the circuit C' is obtained by transforming the gates of C as follows.
  For an XOR gate with inputs a, b and output c, let in C' be the corresponding wires $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$. From $c = a \oplus b = \bigoplus_{i=1}^{n} a_i \oplus b_i$, we let $c_i = a_i \oplus b_i$ for $1 \leq i \leq n$. Consider an AND gate in C with inputs a, b and output c; we have $c = a \wedge b = \bigoplus_{i,j} a_i b_j$. In the transformation of this gate, intermediate values $z_{i,j}$ for $i \neq j$ are computed. For each $1 \leq i < j \leq n$, $z_{i,j}$ is computed uniformly random, while $z_{j,i}$ is set to $(z_{i,j} \oplus a_i b_j) \oplus a_j b_i$. Now, the output bits $c_1, \ldots, c_n$ in C' are defined to be the sequence $c_i = a_i b_i \oplus \bigoplus_{j \neq i} z_{i,j}$.

Remark 3.2. We would like to remark that in the original paper [ISW03] the description of the transformer was considering circuits composed of only NOT and AND gates; differently from them, we preferred to work with only XORs and ANDs. Nevertheless, the two descriptions of circuits are equivalent. Intuitively, they processed the NOT gate simply by maintaining the first $n - 1$ wires as they are and putting an inverter after the n-th wire.

Since it will be helpful later, we also provided an algorithmic description of AND in Algorithm 3.1. In the transformed circuit $C' = T(C)$, every XOR gate and AND gate in C are therefore expanded to gadgets of size $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ respectively, and the gadgets in C' are connected in the same way as the gates in C. This completes the description of T.

Algorithm 3.1 AND gadget - d-order secure multiplication over $\mathbb{F}_2$

Input: shares a_i satisfying $\bigoplus a_i = a$, shares b_i satisfying $\bigoplus b_i = b$
Output: shares c_i satisfying $\bigoplus c_i = a \cdot b$

1: **for** $i = 1$ **to** n **do**
2: **for** $j = i + 1$ **to** n **do**
3: $r_{i,j} \xleftarrow{\$} \{0, 1\}$
4: $r_{j,i} \leftarrow (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$
5: **for** $i = 1$ **to** n **do**
6: $c_i \leftarrow a_i b_i$
7: **for** $(j = 1$ **to** $n) \wedge (i \neq j)$ **do**
8: $c_i \leftarrow c_i \oplus r_{i,j}$
9: **return** $(c_1, \ldots, c_n)$

Security. As said previously, one of the strongest points of the ISW model is the fact that it is provably secure. Thus, the authors could provide proof that the above construction is indeed perfectly t-private in the stateless setting as stated in the following. We will give a scratch of the proof proposed in [ISW03] but we refer the reader to the original paper for a detailed explanation.

Theorem 3.3. *The above construction is a perfectly t-private stateless transformer (T, I, O), such that T maps any stateless circuit C of depth d to a randomized stateless circuit of size $\mathcal{O}(|C| \cdot t^2)$ and depth $\mathcal{O}(d \log t)$.*

Proof (Scratch of the proof). In order to prove the theorem, the authors had to show that the construction achieves soundness and privacy. If it should be easy to verify that C and C' have the same input-output functionality, proving the privacy requires a bit more effort. Specifically, it implies generating values for the t probes chosen by the adversary and making them indistinguishable from the ones the attacker would have obtained in a real execution of C'. In addition, such t queries have to be provided without any knowledge of the input of C. The term *simulator* indicates the algorithm computing from scratch the t values for the probes.

The proof starts proving the t-privacy for a *single* gadget: the goal is to carefully define a set of input wires indices $\mathcal{I}$ such that the joint distribution of values assigned to the probed wires can be simulated from the a_is and b_is belonging to $\mathcal{I}$. This would be enough to ensure privacy since, by definition, the input shares $\{a_i\}$ and $\{b_i\}$, as well as the output shares $\{c_i\}$, have the property that any $n - 1$ subset of such shares are distributed as uniformly random, thus, assigning random values to $a_i|_{\mathcal{I}}$ and $b_i|_{\mathcal{I}}$ (and computing the probes accordingly to them) would allow an indistinguishable view as long as the cardinality of $\mathcal{I}$ is not exceeding $n - 1$ elements. Hence, first set $\mathcal{I}$ is defined: supposing the adversary has probed $w_1, w_2, \ldots, w_t$, each

w_h can carry only a few values, i.e a_i, b_i, a_ib_i, $z_{i,j}$[1] or any sum among them, a_ib_j, $z_{i,j}$[1] $\oplus\, a_ib_j$ and c_i. The procedure is the following:

1. If w_h is in the form a_i, b_i, c_i, a_ib_i, $z_{i,j}$ or any sum among them, then index i is added to $\mathcal{I}$.
2. If w_h is in the form a_ib_j or $z_{i,j} \oplus a_ib_j$, then both indices i and j are added to $\mathcal{I}$.

Note that in both cases for each probe w_h up to two indices are added to $\mathcal{I}$, so it holds that $|\mathcal{I}| \leq 2t < n$ as required. Now, after composing the set of indices, proofs state how to simulate every type of probe:

- If $w_h = a_i$, b_i, a_ib_i then $i \in \mathcal{I}$ and a uniform random assignment perfectly simulates them.
- The $w_h = z_{i,j}$ case needs a bit more attention since we can have more than one case: 1) $i \notin \mathcal{I}$, then, independently from j, $z_{i,j}$ is not part of any w_h and it can be unassigned; 2) $i \in \mathcal{I} \wedge j \notin \mathcal{I}$, then $z_{i,j}$ is assigned a random value, it perfectly simulates it since $j \notin \mathcal{I}$ guarantees that no w_h is using $z_{i,j}$ where $i > j$, and otherwise if $i < j$ C' would have assigned a random value anyway; 3) $i \in \mathcal{I} \wedge j \in \mathcal{I}$, then $z_{i,j}$ is computed as it would be in C' since a_i, b_i, a_j and b_j have been already perfectly simulated.
- If $w_h = a_ib_j$, $z_{i,j} \oplus a_ib_j$, c_i then it can be perfectly simulated from the previous two assignment, proving the t-privacy for a single gadget.

To conclude the proof, the security is extended to the entire circuit and it proceeded very similarly to the single gadget. In particular, they looked at the circuit as a subsequence of gadgets going from the inputs to the outputs of C': $G_1, G_2, \ldots, G_{|C'|}$. Then, each gadget is examined and a unique set $\mathcal{I}$ of indices is computed as described above and the probes are simulated from $a_{|\mathcal{I}}$ and $b_{|\mathcal{I}}$ in the same way aforementioned. The t-privacy still holds since the bound on the number of corrupted wires is applied to the entire circuits and they cannot exceed t; in this way the set $\mathcal{I}$ would maintain cardinality less than n. Consequently, the perfect simulation can be extended to C'.

□

3.1.2 Perfect Privacy for Stateful Circuits

Definition. As it was for the stateless setting, privacy is defined in terms of a transformer T mapping a stateful circuit C, including its initial state s_0, into a new stateful circuit C' with a new initial state s'_0. Moreover, differently from the stateless counterpart, they considered the circuit inputs and outputs to be public, i.e. only the internal state is kept private. The adversary can now access the transformed circuit and invoke it multiple times, choosing freely the new invocation inputs; the adversary may choose the next input based on what it has observed in the previous execution. The stateful case is, surely, a stronger and more realistic model to use in practice; however, as we will see, the construction will exploit the private transformer from the stateless circuits, since it will make it easy to prove stateful privacy once the stateless security has already been achieved. Formally:

[1] For some $i \neq j$.

Definition 3.4 (Perfect privacy for stateful circuits [ISW03]). *Let T be an efficiently computable randomized algorithm mapping a stateful circuit C along with an initial state s_0 to a stateful circuit C' along with an initial state s'_0. We say that T is a t-private stateful transformer if it satisfies the:*

1. ***Soundness.*** *The input-output functionality of C initialized with s_0 is indistinguishable from that of C' initialized with s'_0. This should hold for any sequence of invocations on an arbitrary sequence of inputs. In other words, $C[s_0]$ and $C'[s'_0]$ are indistinguishable to an interactive distinguisher.*
2. ***Privacy.*** *We require that C' be private against a t-limited interactive adversary. Specifically, the adversary is given access to C' initialized with s'_0 as its internal state. Then, the adversary may invoke C' multiple times, adaptively choosing the inputs based on the observed outputs. Prior to each invocation, the adversary may fix an arbitrary set of t internal wires to which it will gain access in that invocation. To define privacy against such a t-limited adversary, we require the existence of a simulator which can simulate the adversary's view using only black-box access to C', i.e., without having access to any internal wires.*

Construction. The ISW construction for perfect privacy in the stateful model proceeds as follows: they made use of the stateless transformer $T(C, t)$ from Theorem 3.3 to secure every gate of the original circuit C while ensuring privacy for the memory cell requires a different approach. This is due to the adaptivity of the adversaries in the stateful context. Namely, as said previously, the attacker can now move the probes between executions and choose adaptively the input. To overcome this power of the adversary, each memory cell in C will be encoded using I (as described in Section 3.1.1): let denote by $E_t(x)$ such encoding, where x is the input being encoded and t its probing tolerance (that is the shares of $E_t(x)$ are t-wise independent). Thus, the stateful transformer is defined as $T = (T_C, T_s)$: T_C is the stateless transformer from above and it would be applied to every gate, while T_s is an encoding $E_{2t}(s_i)$ and it would be applied to every memory cell s_i. In addition, C' is built to consider the encoding of the initial state $E_{2t}(s_0)$ as an input and every next state of the memory is always returned as an output. Note that encoding against $2t$ is crucial to guarantee the t-privacy: in this model, the adversary could have probed t input shares of the memory cell in one execution and probe t output shares of the same memory cell in the next execution; in the stateful construction it is a necessary protection against $2t$ probes. Consequently, the number of shares used in the stateful setting rise to $n = 4t + 1$ for tolerance to $2t$ corruptions.

Security. Perfect privacy in the stateful model follows very similarly to perfect privacy in the stateless setting. In particular, it is possible to look at multiple invocations of the stateful C' as an unwound larger stateless circuit Q, where the input-output functionality is maintained. In this way, the proof would go like before with the only difference that in Q inputs and outputs are hidden while in C' they are given to the adversary; nevertheless, those values can be passed to the simulator and can be easily incorporated into the simulation. The proof from Theorem 3.3 would perfectly simulate the internal probes, while the memory corruptions can

be simulated by assigning random values due to the *re-randomization* property: each of the concatenated invocations in Q has outputs that are t-wise independent if conditioned to its input values. That is enough to prove the following:

Theorem 3.5. *The above construction is a perfectly t-private stateful transformer (T, I, O), such that T maps any stateful circuit C of depth d to a randomized stateful circuit of size $\mathcal{O}(|C| \cdot t^2)$ and depth $\mathcal{O}(d \log t)$.*

3.1.3 Random Probing Model

As mentioned in Section 3.1, the authors relaxed the perfect security in order to improve the running time of the above constructions. In fact, the major disadvantage of the perfect security is their complexity grows quadratically in the number of shares (and, in turn, in the number of probes): for both stateless and stateful models the overall running time is $\mathcal{O}(|C|t^2)$. The improvement in time complexity came at a price in terms of the level of security, which now is no more perfect but *statistical*: i.e. the construction is allowed to leak, but with negligible probability. To achieve this new definition, as we will explain in detail in the next subsection, they introduced the *average-case* security, formally defined as the *random probing model*. The average-case is in contraposition with the adversarial characteristic given so far, in which the adversary is restricted to t probes but of its own choice[2]. For obvious reasons, privacy that takes into consideration that kind of adversaries is referred to as *worst-case* security. On the other side, in the average-case, the attacker has no power over the leaking wires; instead, each wire of the circuit C will be considered probed with some independent probability p. Formally:

Definition 3.6 (Random probing model [ISW03]). *A circuit transformer $T = T(C, k)$ is said to be (statistically) p-private in the average case if $C' = T(C, k)$ is statistically private against an adversary which corrupts each wire in C' with independent probability p. That is, the joint distribution of the random set of corrupted wires and the values observed by the adversary can be simulated up to a $k^{-\omega(1)}$ statistical distance.*

Consider now the ISW construction for the stateful model (see Theorem 3.5) with k number of shares. It should be easy to see that such a transformer is perfectly $(k/4)$-private. Using Chernoff's bound and a leakage probability $p = \Omega(1/k)$ the authors claimed that every probe can be perfectly simulated but with negligible probability. This led them to state the following lemma:

Lemma 3.7. *There exists a circuit transformer $T(C, k)$ producing a circuit C' of size $\mathcal{O}(|C| \cdot k^2)$, such that T is $\Omega(1/k)$-private in the average case.*

Proof. Let us first recall the Chernoff's bound. it is obtained by applying Markov's inequality to e^{tX}. For every $t > 0$:

[2] Excluding the input wires in the stateless setting.

$$\Pr[X \geq a] = \Pr[e^X \geq e^a] \leq \frac{\mathbb{E}[e^{t \cdot X}]}{e^{t \cdot a}}$$

Assume $X = X_1 + \cdots + X_n$ where $X_1, \ldots, X_n$ are independent variables. We get for any $t > 0$:

$$\Pr[X \geq a] \leq e^{-t \cdot a} \prod_{i=1}^{n} \mathbb{E}[e^{tX_i}]$$

If the variables X_i have their value in $\{0, 1\}$, letting $p_i = \Pr[X_i = 1]$, we get:

$$\mathbb{E}[e^{tX_i}] = e^0 \cdot (1 - p_i) + e^t \cdot p_i = 1 + p_i \cdot (e^t - 1) \leq e^{p_i \cdot (e^t - 1)}$$

This gives:

$$\Pr[X \geq a] \leq e^{-t \cdot a} \cdot e^{\sum_{i=1}^{n} p_i \cdot (e^t - 1)} = e^{-t \cdot a + \mu \cdot (e^t - 1)}$$

where we let $\mu = \mathbb{E}[X] = \sum_{i=1}^{n} p_i$. We take $a = (1 + \delta) \cdot \mu$ for $\delta > 0$. We take $t = \log(1 + \delta)$. This gives the Chernoff bound:

$$\Pr[X \geq (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu$$

Using the inequality $2\delta/(2 + \delta) \leq \log(1 + \delta)$, we obtain:

$$\Pr[X \geq (1 + \delta)\mu] \leq \exp\left(\frac{-\delta^2 \mu}{2 + \delta} \right)$$

For $\delta \geq 1$, we have:

$$\Pr[X \geq (1 + \delta)\mu] \leq \exp\left(\frac{-\delta \mu}{3} \right)$$

Moreover, before getting into the proof itself, we would like to remark that such proof has been made explicit by us since it was just given as a statement in [ISW03]. Thus, let us consider the ISW construction from Theorem 3.3, i.e. the perfect private scheme in the stateless probing model, but with a number of shares dependent from the security parameter k, i.e. $n = 2k + 1$, and, consequently, secure against the probing of at most k probes per gadget, where each gadget has a number of wires $m \leq c \cdot k^2$ for some constant c.

Hence, let $X_1, \ldots, X_m$ be independent Bernoulli random variables, where $X_i = 1$ if the i-th wire is probed, and $X_i = 0$ otherwise; it holds that $\Pr[X_i = 1] = p$ for all $1 \leq i \leq m$. Let $X = X_1 + \ldots + X_m$ be the total number of leaking wires in the gadget. Theorem 3.3 guarantees that it is possible to perfectly simulate the probed wires if X does not exceed k. Thus, using the Chernoff bound with $\delta \geq 1$ (see **??**), it is possible to calculate the probability that the event $X > k$ occurs:

$$\Pr[X \geq (1 + \delta)\mathbb{E}[X]] \leq \exp\left(-\frac{\delta}{3} \mathbb{E}[X] \right) \tag{3.1}$$

where $\mathbb{E}[X]$ is computable and equal to $p \cdot m$. Furthermore, we fix $p = 1/(2c \cdot k)$. In addition, since we must upper-bound $\Pr[X \geq k]$, we can fix δ such that:

$$k = (1 + \delta)\mathbb{E}[X]$$

It is possible to explicate k from the number of wires $m \leq c \cdot k^2$; this gives that:

$$1 + \delta = \frac{k}{\mathbb{E}[X]} = \frac{k}{p \cdot \ell} = \frac{2c \cdot k^2}{\ell} \geq \frac{2c \cdot k^2}{c \cdot k^2} \geq 2$$

and therefore the inequation $\delta \geq 1$ is respected, as required. Moreover, it holds that:

$$\frac{\delta}{3}\mathbb{E}[X] = \frac{\delta}{3} \cdot \frac{k}{1 + \delta} \geq \frac{k}{6}$$

Therefore, from (Equation (3.1)), it is possible to compute the probability:

$$\Pr[X \geq k] \leq \exp(-k/6)$$

Hence, from the equation above, the simulation failure probability over all the gadgets is computed multiplying the failure probability of a single gadget for the total number of gadgets, i.e. it is at most $|C| \cdot \exp(-k/6)$.

□

For the sake of clarity, we stress the fact that this model is not able to provide worst-case security. The goal to guarantee privacy when the attacker can probe a $\Omega(|C'|/k)$ of wires is still considered unachievable. As we are going to see in the next subsection, the relevance that Ishai et al. gave to the random probing model was to use it as a building block necessary to achieve statistical privacy in the worst-case setting.

3.1.4 Statistical Privacy for Stateless Circuits

Definition. To obtain a construction with complexity $\mathcal{O}(t \log t)$, the authors introduced a relaxation to the security model, in which it is tolerated a leakage of the secrets, albeit with a negligible probability; this is called the statistical model of security. But the weaker privacy guarantees will be balanced by the $\mathcal{O}(t \log t)$ complexity of the construction. However, to avoid confusion, let's remind the reader we are back to the worst-case scenario, where the adversary can choose the probes it wants up to t. Although the construction will make use of the average-case transformer, statistical privacy is defined to consider only attackers that can freely choose. The definition below is similar to the perfect privacy model, except that now the simulation can fail with negligible probability:

Definition 3.8 (Statistical privacy for stateless circuits). *Let T be an efficiently computable deterministic function mapping a stateless circuit C to a stateless circuit $\tilde{C}$, and let*

I, O be as above. We say that (T, I, O) is a statistically t-private stateless transformer if it satisfies the:

1. ***Soundness.*** *The input-output functionality of $O \circ \tilde{C} \circ I$ (i.e., the iterated application of I, $\tilde{C}$, O in that order) is indistinguishable from that of C.*
2. ***Privacy.*** *We require that the view of any t-limited adversary, which attacks $O \circ \tilde{C} \circ I$ by probing at most t wires in $\tilde{C}$, can be simulated except with negligible probability. The identity of the probed wires has to be chosen in advance by the adversary.*

Construction. Their construction proceeds in two steps. Firstly, the masking countermeasure seen in Section 3.1.3 is applied; namely, the transformer $T(C, k)$ proved to be p-private in the average-case, for $p = \Omega(1/k)$ with security parameter k. Secondly, $C' = T(C, k)$ is transformed into a larger circuit $\widetilde{C}$, where only a fraction of the computation is useful. In other words, the circuit $\widetilde{C}$ would perform the same computation as C', but only on a small random subset of its wires; the remaining wires of $\widetilde{C}$ would contain no useful information for the adversary. More specifically, the transformer $\widetilde{T}$ mapping C' in $\widetilde{C}$ takes each wire i of C' and replaces it with a new set of ℓ wires in $\widetilde{C}$; say each i is replaced by $(i, 1), \ldots, (i, \ell)$. Recall that every such wire can take a value from the set $\{0, 1, \$\}$. Thus, for every wire i in C' carrying a value $v_i \in \{0, 1\}$, the wires $(i, 1), \ldots, (i, \ell)$ in $\widetilde{C}$ carry the value v_i in a random position (independently of other ℓ-tuples), and the value \$ in the all remaining $\ell - 1$ ones; see Figure Figure 31 for an illustration.

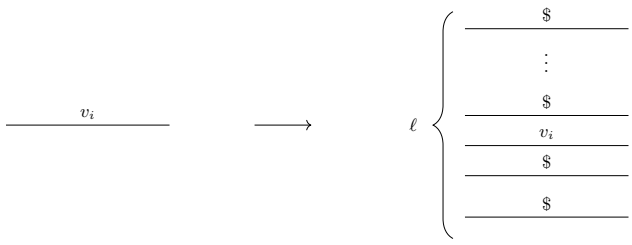

Fig. 31: A wire with signal v_i in C' (left), and the corresponding ℓ wires in $\widetilde{C}$ (right); only one of the ℓ wires contains the signal v_i, while the others contain the dummy value \$.

Consequently to the above rearrangements of wires, modifications to the encoding and decoding are required. The old algorithms no longer fit the expanded circuit logic; nevertheless, from the implementation point of view, the new Encode' and Decode' use the old Encode and Decode as a subroutine. But now, for the sake of simplicity, the following descriptions will not take into account this last remark:

- Encode'. To encode a value v, first generate at random an index $j \xleftarrow{\$} [1, \ell]$ and output an ℓ-tuple in which v will be the j-th element, while the other elements carry a dummy value \$. That is, return $(\$, \ldots, \$, v, \$, \ldots, \$)$, where v is at the j-th position.

- Decode'. Given a ℓ-tuple $(\$, \ldots, \$, v, \$, \ldots, \$)$, return v.

Once the encoding and decoding algorithms are adjusted, let's pass to the transformation of the intermediate circuit. Hence, consider a gate in C' such that input wires i_1 and i_2 are processed to obtain $v_i = v_{i_1} * v_{i_2}$; namely, the i-th wire of C' is computed by applying a boolean operation $* \in \{\mathsf{AND}, \mathsf{XOR}\}$ to the i_1-th and the i_2-th wire of C'. Informally, $\widetilde{T}$ would transform such gate in a 2ℓ-input, ℓ-output gadget in $\widetilde{C}$. The gadget, first, puts both values v_{i_1} and v_{i_2} in random but adjacent positions, and then it combines them to obtain the value $v_{i_1} * v_{i_2}$ in a randomly defined wire out of the ℓ output ones. In order to achieve the "random but adjacent position" property, the construction resorted to a *sorting network* as a building block, see [AKS83]. Technically, the gate processing is described in [ISW03] as follows:

- **Preprocessing.** Compute $\ell + 1$ uniformly and independently random integers $r, r_1, \ldots, r_\ell$ from the range $[0, 2^k]$, where k is the security parameter. For each $1 \leq j \leq \ell$, use the values $v_{i_1,j}, v_{i_2,j}$ (of wires (i_1, j) and (i_2, j)) to form a pair $(\mathsf{key}_j, \mathsf{val}_j)$ such that:
  1. key_j is set to r_j if $v_{i_1,j} = v_{i_2,j} = \$$ and to r otherwise;
  2. val_j is set to $\$$ if both $v_{i_1,j}$, $v_{i_2,j}$ are $\$$; to a bit value b if one of $v_{i_1,j}$,$v_{i_2,j}$ is b and the other is $\$$, and to $b_1 * b_2$ if $v_{i_1,j} = b_1$ and $v_{i_2,j} = b_2$.
- **Sorting.** A sorting network is applied to the above ℓ-tuple of pairs using key as the sorting key. Let $(u_1, \ldots, u_\ell)$ denote the ℓ-tuple of symbols val_j sorted according to the keys key_j.
- **Postprocessing.** The j-th output v_j is obtained by looking at u_j, u_{j+1}, u_{j+2}: if u_j, $u_{j+1} \neq \$$ then $v_j = u_j * u_{j+1}$, if $u_j = u_{j+2} = \$$ and $u_{j+1} \neq \$$ then $v_j = u_{j+1}$, and otherwise $v_j = \$$.

In other words, if the input signals v_{i_1} and v_{i_2} are initially located at positions j_1 and j_2 for some $j_1 \neq j_2$, then by definition $\mathsf{key}_{j_1} = \mathsf{key}_{j_2} = r$, and therefore after sorting by key_j the signal values v_{i_1} and v_{i_2} will be contiguous; hence, in the postprocessing phase the output signal $v_{i_1} * v_{i_2}$ will be computed, and located at some random position j_3[3]. Note that such a gadget can be implemented by a circuit of size $\mathcal{O}(k \cdot \ell \log \ell)$ and depth $\mathsf{poly}(\log \ell + \log k)$, where the $\ell \log \ell$ factor is mainly due to the cost of the sorting network. To complete the description of $\widetilde{T}$, it is missing the transformation of a random gate, but it is straightforward: the random bit gate is replaced with a gadget composed of ℓ input and ℓ output wires, and each input wire would consist of a random bit z_j (for $1 \leq j \leq \ell$); then an index $r \in [\ell]$ is generated uniformly at random, in this way the output wires of the gadget in $\widetilde{C}$ are outputting $\$$ symbol for every $j \neq r$ and z_r otherwise. This completes the description of $\widetilde{T}$.

Security. For the worst-case statistical privacy of the final circuit $\widetilde{C}$, the authors proceeded to reduce it to the p-random probing security of C'. In particular, they provided the following lemma:

Lemma 3.9. *Suppose that C' is p-private in the average case. Then the circuit $\tilde{C}$, constructed with $\ell = \mathcal{O}(t/p^7)$, is statistically t-private in the worst case.*

[3] The same holds if $j_1 = j_2$.

Proof (Proof Sketch). In a nutshell, for each compromised wire of $\widetilde{C}$, the adversary can either observe some random integer r_i, a \$ symbol, or an actual value v_i of the i-th wire of C'. Let $\mathcal{I}$ denote the set of indices i such that v_i has been probed. Ishai et al. argued that for any fixed index set $\mathcal{I}_0$ and for ℓ defined as above, then $\Pr[\mathcal{I}_0 \subseteq \mathcal{I}] \leq p^{|\mathcal{I}_0|}$. Thus, an adversary attacking any fixed set of t wires in $\widetilde{C}$ cannot learn more than an adversary corrupting each wire of C' independently with probability $p = \Omega(1/k)$. They proved that argument, picking a set $\mathcal{I}_1 \subseteq \mathcal{I}_0$ such that: 1) $|\mathcal{I}_1| \geq |\mathcal{I}_0|/7$; 2) each value in $\mathcal{I}_1$ is observed with probability (at most) p^7 and the events of observing different values in $\mathcal{I}_1$ are independent. Such $\mathcal{I}_1$ makes $\Pr[\mathcal{I}_0 \subseteq \mathcal{I}] \leq \Pr[\mathcal{I}_1 \subseteq \mathcal{I}_0] \leq (p^7)^{|\mathcal{I}_1|} \leq (p^7)^{|\mathcal{I}_0|/7} = p^{|I_0|}$.

□

Remark 3.10. We would like to point out a minor difference with respect to [ISW03]. In the original paper, Lemma 3.9 is stated the same but with ℓ defined as $\ell = \mathcal{O}(t/p^4)$. We claim that this is slightly incorrect. The fraction 1/4 is presented as the result of the maximal matching of a graph of degree 4; actually, the only assumption one can make about the cardinality of an uncharacterised graph is the matching will contain at most a 1/7 of the edges (and not 1/4 as used in the proof of [ISW03, Lemma 2], see for example [BDD+01]). Note that this technicality does not change the asymptotic behaviour with respect to the number of probes t which is still $\mathcal{O}(t \log t)$, but only the dependence with respect to the security parameter k.

From the above lemma and Lemma 3.7, should be clear to derive the statistical t-privacy in the worst-case setting for $\widetilde{C}$.

Theorem 3.11. *There exists a statistically t-private stateless transformer $(\widetilde{I}, \widetilde{T}, \widetilde{O})$, such that $\widetilde{T}(C, k)$ transforms a circuit C to a circuit $\widetilde{C}$ of size $\mathcal{O}(|C| \cdot k^{10} \cdot t \cdot (\log k + \log t))$.*

Proof. The statistical t-privacy of $\tilde{C}$ follows from Lemma 3.9. The intermediate circuit $C' = T(C, k)$ has complexity $\mathcal{O}(|C| \cdot k^2)$. Then C' is expanded by a factor $\mathcal{O}(k \cdot \ell \log \ell)$; still from Lemma 3.9 and with $p = \Omega(1/k)$, one can take $\ell = \mathcal{O}(t \cdot k^7)$; the expansion factor is therefore $\mathcal{O}(k \cdot (t \cdot k^7) \log(t \cdot k^7)) = \mathcal{O}(k^8 \cdot t \cdot (\log t + \log k))$. The final complexity is therefore $\mathcal{O}(|C| \cdot k^{10} \cdot t \cdot (\log k + \log t))$.

□

Remark 3.12. We would remark on one difference with [ISW03]: in the original paper, as claimed by the authors themself, the time complexity hid a $k^{\mathcal{O}}$ factor since it is considered small in comparison to t. We made it explicit to be comparable with our improved construction as we will see in Chapter 4.

3.1.5 Statistical Privacy for Stateful Circuits

Definition. The definition of worst-case statistical privacy in the stateful model is the same as for perfect privacy (see Section 3.1.2), except that the simulator can now fail with negligible probability ε:

Definition 3.13 (Statistical privacy for stateful circuits.). *Let T be an efficiently computable randomized algorithm mapping a stateful circuit C along with an initial state s_0 to a stateful circuit $\widetilde{C}$ along with an initial state s'_0. We say that T is a statistical t-private stateful transformer if it satisfies the:*

1. ***Soundness.*** *The input-output functionality of C initialized with s_0 is indistinguishable from that of $\widetilde{C}$ initialized with s'_0. This should hold for any sequence of invocations on an arbitrary sequence of inputs.*
2. ***Privacy.*** *We require that $\widetilde{C}$ be private against a t-limited interactive adversary. Specifically, the adversary is given access to $\widetilde{C}$ initialized with s'_0 as its internal state. Then, the adversary may invoke $\widetilde{C}$ multiple times, adaptively choosing the inputs based on the observed outputs. Prior to each invocation, the adversary may fix an arbitrary set of t internal wires to which it will gain access in that invocation. To define privacy against such a t-limited adversary, we require the existence of a simulator which can simulate the adversary's view, using only black-box access to $\widetilde{C}$, except with negligible probability.*

Construction. As for the stateless case, the transformer proceeds in two steps, with each wire in the intermediate circuit C' being expanded into ℓ wires in the final circuit $\widetilde{C}$, such that only one of the ℓ wires contains the original signal v_i from C', while the other wires contain only the dummy value \$. However, for the stateful construction some additional countermeasures needed to be addressed, because the adversary can move its probes between executions, and therefore it could accumulate knowledge about the locations of the signal in $\widetilde{C}$. The authors prevented such attacks by using a perfectly t-private encoding of a random cyclic shift for each pack of ℓ wires, for each signal v_i from C' that must be transmitted from one execution to the other. Such t-private encoding has complexity $\mathcal{O}(t^2)$, and since for every memory cell this cyclic shift requires a circuit of size $\mathcal{O}(\ell \log \ell)$, the additional complexity is $\mathcal{O}(st^2 \ell \log \ell)$, which gives a total complexity of $\tilde{\mathcal{O}}(st^3)$ for s memory cells.

Security. The statistical t-privacy in the stateful model is pretty similar to the one provided for the stateless case. The transformation of each gate is guaranteed statistically t-private by Theorem 3.11. In addition, as explained above, the memory cells are made t-private by a perfect t-privacy transformation of a cyclic shift. This was enough to state the following:

Theorem 3.14 (Worst-case statistical privacy, stateful model). *There exists a statistically t-private stateful transformer $\widetilde{T}$, such that $\widetilde{T}(C, k)$ maps a circuit C with s memory cells to a circuit $\widetilde{C}$ of size $\mathcal{O}(|C| \cdot t \log t + s \cdot t^3 \log t)$. The depth of $\widetilde{C}$ is the same as that of C, up to* polylog *factors.*

3.2 Rivain-Prouff Masking

Since its first appearance, the masking countermeasures seemed to be a perfect fit to contrast SCA. As proved by Chari et al. in [CJRR99], the power of applying a secret-sharing scheme

relies on the fact that the risk of leakage decreases exponentially with respect to the number of shares. One of the many advantages of this approach is that it has been used to ensure standard encryption cipher like DES and AES. We are going to focus on AES. Until 2010, the majority of publications produced on the topic were focused on first-order and second-order masking schemes. On the other side, since [ISW03] appeared, not much was proposed to achieve higher-order privacy; the ISW constructions were producing a significant overhead when translated into actual software. In 2010 Rivain and Prouff proposed a way to mask the implementation of AES for general order $n-1$ and practical overhead. Moreover, they decreased the total number of shares required to $n = t + 1$, instead of $n = 2t + 1$ as we saw in [ISW03].

In [RP10], the authors started by noticing that, although the ISW constructions represent an important achievement in theory, they suffer from intrinsic practical issues. On one side, the hardware implementation of the masked circuit induces a significant expansion in terms of the silicon area. As we saw in Section 3.1.1, the transformed AND gadget is expanded quadratically with respect to the original single AND gate; specifically: every AND gate in C is replaced with n^2 number of ANDs and $2n(n-1)$ number of XORs, in addition, the transformation requires $n(n-1)/2$ random bits. Moreover, masking at the hardware level is subjected to *glitches*; which can be mitigated by involving synchronizing hardware components in the circuit but at the price of expanding its size again. On the flipping side, a software implementation of the ISW constructions is immune to silicon-overhead or to glitches, but this approach, accurately as described in [ISW03], can not be considered practical due to the prohibitive time-overhead. Thus, with the aim to reduce the software time-overhead, Rivain and Prouff generalized ISW constructions to work, not only with booleans inputs (i.e $\mathbb{F}_2$) but also with elements belonging to any finite field (i.e $\mathbb{F}_{2^k}$). We are going to explain in detail the Rivain-Prouff masking for AES proposed in [RP10].

As explained in Section 2.3, the AES block cipher is structured as multiple rounds transformation, consisting of a key addition, a linear and a non-linear layer. The masking of the non-linear operation, i.e the masking of the S-box, is the trickiest part of the implementation. In turn, once the S-box is ensured to be securely masked, protecting the remaining operations comes pretty straightforward.

3.2.1 Masking of the S-box

In order to mask the S-box, the authors had to provide a secure method to mask the power-to-254 function for any order $n-1$. The exponentiation to 254, as described in [DR02], consists of 4 multiplications over $\mathbb{F}_{2^k}$ and 7 squarings. Where exponentiation to the power of 4 and 16 are considered, respectively, as 2 and 4 consequent squarings. Hence, to securely compute such exponentiation, they implemented the following methods for the squarings and the multiplications:

- *Squaring:* due to the linearity of the operation[4], the squaring is secured by simply dealing with each share separately and independently. That is, if $x_1 \oplus x_2 \oplus \cdots \oplus x_n = x$ then it holds that:
$$x_1^2 \oplus x_2^2 \oplus \cdots \oplus x_n^2 = x^2$$
Please note that the same holds for every exponentiation to the power of 2^p with $p \in \mathbb{N}$; in turn, the same "independently share-by-share" approach ensures privacy for x_i^4 and x_i^{16} too.
- *Multiplication:* the authors' proposed multiplication follows the ISW AND-gadget, see Algorithm 3.1. This should not surprise the reader, since a logical AND gate is nothing more than a multiplication over the field $\mathbb{F}_2$. Thus, replacing each binary input with elements in $\mathbb{F}_{2^k}$, each AND with a multiplication over $\mathbb{F}_{2^k}$ and each XOR with an addition over $\mathbb{F}_{2^k}$ it is not affect the completeness of Algorithm 3.1. This gives the following SecMult algorithm below.

Algorithm 3.2 SecMult - $(n-1)$-order secure multiplication over $\mathbb{F}_{2^k}$

Input: shares a_i satisfying $\bigoplus a_i = a$, shares b_i satisfying $\bigoplus b_i = b$
Output: shares c_i satisfying $\bigoplus c_i = a \cdot b$
1: **for** $i = 1$ **to** n **do**
2: **for** $j = i + 1$ **to** n **do**
3: $r_{i,j} \xleftarrow{\$} \mathbb{F}_{2^k}$
4: $r_{j,i} \leftarrow (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$
5: **for** $i = 1$ **to** n **do**
6: $c_i \leftarrow a_i b_i$
7: **for** $(j = 1$ **to** $n) \wedge (i \neq j)$ **do**
8: $c_i \leftarrow c_i \oplus r_{i,j}$
9: **return** $(c_1, \ldots, c_n)$

Unfortunately as argued in [RP10], the secure computation of squarings and the multiplications is not enough to guarantee the security for the whole exponentiation. In order to do that, the masks of the inputs given to the SecMult need to be mutually independent and this means they need to be refreshed at some point during the exponentiation. This operation is processed by the following secure refreshing algorithm:

[4] The operation operates on a field of characteristic 2.

Algorithm 3.3 RefreshMasks

Input: shares x_i satisfying $x = \bigoplus x_i$
Output: shares x_i satisfying $x = \bigoplus x_i$
1: **for** $i = 2$ **to** n **do**
2: $\quad \mathsf{tmp} \xleftarrow{\$} \mathbb{F}_{2^k}$
3: $\quad x_1 \leftarrow x_1 \oplus \mathsf{tmp}$
4: $\quad x_i \leftarrow x_i \oplus \mathsf{tmp}$
5: **return** $(x_1, \ldots, x_n)$

Remark 3.15. For the sake of consistency, we proposed the RefreshMasks algorithm as explained in [RP10], even if it has been proved to have a flaw in [BBD+16]. We will see in Section 3.3 why it is insecure and how Barthe et al. modified it to make it private.

As said at the beginning, being able to securely compute multiplications, squarings and refreshing the randomness of the shares allowed them to securely compute the exponentiation-to-254. But computing the power-to-254 in the most efficient way possible is not the straightforward. Clearly, SecMult is way more expensive in terms of time compared to squaring and refreshing: the exponentiation algorithm has to be designed to use the least number of multiplications. The exponentiation procedure proposed by Rivain and Prouff contains 4 such multiplications and 7 squarings, see Algorithm 3.4:

Algorithm 3.4 Exponentiation to the 254

Input: x
Output: $y = x^{254}$
1: $z \leftarrow x^2$ ▷ $z = x^2$
2: $y \leftarrow zx$ ▷ $y = x^2 x = x^3$
3: $w \leftarrow y^4$ ▷ $w = (x^3)^4 = x^{12}$
4: $y \leftarrow yw$ ▷ $y = x^3 x^{12} = x^{15}$
5: $y \leftarrow y^{16}$ ▷ $y = (x^{15})^{16} = x^{240}$
6: $y \leftarrow yw$ ▷ $y = x^{240} x^{12} = x^{252}$
7: $y \leftarrow yz$ ▷ $y = x^{252} x^2 = x^{254}$
8: **return** y

Given the efficient order of operations and given a secure computation of multiplications, squarings and refreshings, it should be easy to assemble them in an $(n-1)$-order secure exponentiation to 254 over $\mathbb{F}_{2^k}$ as the authors presented:

Algorithm 3.5 SecExp254 - $(n-1)$th-order secure exponentiation to the 254 over $\mathbb{F}_{2^k}$

Input: shares $x_1, x_2, \ldots, x_n$ satisfying $x = \bigoplus x_i$
Output: shares $y_1, y_2, \ldots, y_n$ satisfying $y = \bigoplus y_i = x^{254}$
1: **for** $i = 1$ **to** n **do** $z_i \leftarrow x_i^2$
2: $(x_1, x_2, \ldots, x_n) \leftarrow \mathsf{RefreshMasks}(x_1, x_2, \ldots, x_n)$
3: $(y_1, y_2, \ldots, y_n) \leftarrow \mathsf{SecMult}((z_1, z_2, \ldots, z_n), (x_1, x_2, \ldots, x_n))$
4: **for** $i = 1$ **to** n **do** $w_i \leftarrow y_i^4$
5: $(w_1, w_2, \ldots, w_n) \leftarrow \mathsf{RefreshMasks}(w_1, w_2, \ldots, w_n)$
6: $(y_1, y_2, \ldots, y_n) \leftarrow \mathsf{SecMult}((y_1, y_2, \ldots, y_n), (w_1, w_2, \ldots, w_n))$
7: **for** $i = 1$ **to** n **do** $y_i \leftarrow y_i^{16}$
8: $(y_1, y_2, \ldots, y_n) \leftarrow \mathsf{SecMult}((y_1, y_2, \ldots, y_n), (w_1, w_2, \ldots, w_n))$
9: $(y_1, y_2, \ldots, y_n) \leftarrow \mathsf{SecMult}((y_1, y_2, \ldots, y_n), (z_1, z_2, \ldots, z_n))$
10: **return** $(y_1, y_2, \ldots, y_n)$

Furthermore, for the sake of completeness, the authors pointed out that Algorithm 3.5 computes the exponentiation using $8n(n-1)+4(n-1)$ XOR gates, $4n^2$ multiplications, n squarings, n power-to-4 and n power-to-16. It needs $3n+n(n-1)/2$ bytes of memory and $2n(n-1)+2(n-1)$ random bytes. Thus, to complete the secure computation of the S-box, it is only missing to secure the affine transformation Af. This is straightforward since, like for the squaring, Af is linear and it can be computed separately and independently share by share; that is:

$$Af(x_1) \oplus Af(x_2) \oplus \cdots \oplus Af(x_n) = \begin{cases} Af(x) & \text{if } n \text{ is even,} \\ Af(x) \oplus \mathsf{0x63} & \text{if } n \text{ is odd.} \end{cases}$$

Algorithm 3.6 summarizes the algorithmic description of the Rivain-Prouff masking of the S-box:

Algorithm 3.6 SecSbox - $(n-1)$th-order secure computation of S-box

Input: shares $x_1, x_2, \ldots, x_n$ satisfying $x = \bigoplus x_i$
Output: shares $y_1, y_2, \ldots, y_n$ satisfying $y = \bigoplus y_i = S(x)$
1: $(y_1, y_2, \ldots, y_n) \leftarrow \mathsf{SecExp254}(x_1, x_2, \ldots, x_n)$
2: **for** $i = 1$ **to** n **do** $y_i \leftarrow Af(x_i)$
3: **if** $((n-1) \mod 2 = 1)$ **then** $y_1 \leftarrow y_1 \oplus \mathsf{0x63}$
4: **return** $(y_1, y_2, \ldots, y_n)$

3.2.2 Masking the whole AES

Firstly, as it has been for the ISW model, the inputs of the AES need to be encoded (or masked). Here also the secret key is considered as an input. Hence, we will assume that: the state $\mathbf{s} = (s_{a,b})_{1 \le a,b \le 4}$ is mapped as n states $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n$ such that $\mathbf{s} = \mathbf{s}_1 \oplus \mathbf{s}_2 \oplus \ldots \oplus \mathbf{s}_n$.

Intuitively, the generation of such states follows the Encoder I: first, $n-1$ arrays of size 4×4 bytes are computed randomly; and then, $\mathbf{s}_n$ is processed according to the above formula. Moreover, the secret key $\mathbf{k} = (k_{a,b})_{1 \leq a,b \leq 4}$ is mask exactly how the state above. Computing a n-shared key: $\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_n$ such that $\mathbf{k} = \mathbf{k}_1 \oplus \mathbf{k}_2 \oplus \ldots \oplus \mathbf{k}_n$. Now, once the inputs are encoded, the different AES transformations (see Section 2.3) are proposed in the following by the authors:

1. **KeyExpansion**: this operation generates a 4×4 byte key $\mathbf{k}^r$ for each round $r \in [\mathsf{N_r} + 1]$. The process explained in Section 2.3 needs to be masked: the key schedule $\mathbf{w}$, now, is split in n schedules $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n$. Consequently, each column $(\mathbf{w}_i)_{*,j}$ of the i-th schedule $\mathbf{w}_i$ is computed as: $(\mathbf{w}_i)_{*,j} = (\mathbf{w}_i)_{*,j-\mathsf{N_k}} \oplus \mathbf{t}_i$, where the $\mathbf{t}_i$s denote the 4-byte shares of $\mathbf{t}$ and it is easy to masked since SubWord is just applying the S-box; and both RotWord and Rcon are straightforward. The authors provided in [RP10] an algorithmic description of the key expansion:

Algorithm 3.7 SecKeyExp - $(n-1)$th-order secure AES key expansion

Input: key shares $\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_n$ satisfying $\mathbf{k} = \bigoplus \mathbf{k}_i$
Output: shares $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n$ satisfying $\mathbf{w} = \bigoplus \mathbf{w}_i$
1: **for** $j = 1$ **to** $\mathsf{N_k}$ **do**
2: **for** $i = 1$ **to** n **do** $(\mathbf{w}_i)_{*,j} \leftarrow (\mathbf{k}_i)_{*,j}$
3: **for** $j = \mathsf{N_k} + 1$ **to** $4(\mathsf{N_k} + 1)$ **do**
4: **for** $i = 1$ **to** n **do** $\mathbf{t}_i \leftarrow (\mathbf{w}_i)_{*,j}$
5: **if** $((j \mod \mathsf{N_k} = 0) \vee (\mathsf{N_k} = 8) \wedge (j \mod \mathsf{N_k} = 4))$ **then**
6: **for** $l = 1$ **to** n **do**
7: $((\mathbf{t}_1)_l, (\mathbf{t}_2)_l, \ldots, (\mathbf{t}_n)_l) \leftarrow \mathsf{SecSbox}((\mathbf{t}_1)_l, (\mathbf{t}_2)_l, \ldots, (\mathbf{t}_n)_l)$
8: **if** $(j \mod \mathsf{N_k} = 0)$ **then**
9: **for** $i = 1$ **to** n **do**
10: $\mathbf{t}_i \leftarrow \mathsf{RotWord}(\mathbf{t}_i)$
11: $\mathbf{t}_1 \leftarrow \mathbf{t}_1 \oplus \mathsf{Rcon}_{j/\mathsf{N_k}}$
12: **for** $i = 1$ **to** n **do** $(\mathbf{w}_i)_{*,j-1} \leftarrow (\mathbf{w}_i)_{*,j-\mathsf{N_k}} \oplus \mathbf{t}_i$
13: **return** $(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n)$

2. **AddRoundKey**: this transformation is computed by XORing the state $\mathbf{s}_r$, obtained at the round r, with the key for the r-th round $\mathbf{k}^r$. Given the linearity of the operation, this can be computed at the share level:

$$\mathbf{s} \oplus \mathbf{k} = (\mathbf{s}_1 \oplus \mathbf{k}_1^r) \oplus (\mathbf{s}_2 \oplus \mathbf{k}_2^r) \oplus \ldots \oplus (\mathbf{s}_n \oplus \mathbf{k}_n^r)$$

3. **SubBytes**: this transformation simply applies the masked S-box described before to each byte of the state, i.e to the n-tuple of byte shares:

$$\mathsf{SubBytes}(\mathbf{s}) = (\text{S-box}(s_{a,b}))_{1 \leq a,b \leq 4}$$

4. **ShiftRows**: this transformation cyclically shifts the rows of the state. Clearly, this can be performed share by share:

$$\mathsf{ShiftRows}(\mathbf{s}) = \bigoplus_{i=1}^{n} \mathsf{ShiftRows}(\mathbf{s}_i)$$

5. **MixColumns**: this transformation is considering the columns of the state, and each of them is multiplied by the polynomial $3x^3 + x^2 + x + 2 \mod x^4 + 1$. This also can be performed share by share like for ShiftRows:

$$\mathsf{MixColumns}(\mathbf{s}) = \bigoplus_{i=1}^{n} \mathsf{MixColumns}(\mathbf{s}_i)$$

This let the authors to provide an algorithmic description of the whole masked AES, which is secure against any $(n-1)$-order SCA:

Algorithm 3.8 SecAES - $(n-1)$th-order secure AES computation

Input: state share $\mathbf{s}_i$ satisfying $s = \bigoplus \mathbf{s}_i$; key shares $\mathbf{k}_i$ satisfying $\mathbf{k} = \bigoplus \mathbf{k}_i$
Output: ciphertext shares $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n$ satisfying $\mathbf{c} = \bigoplus \mathbf{c}_i$

$(\mathbf{k}_i^0, \mathbf{k}_i^1, \ldots, \mathbf{k}^{\mathsf{N_r}}) \leftarrow \mathsf{SecKeyExp}(\mathbf{k}_i)$
$\mathbf{s}_i \leftarrow \mathsf{AddRoundKey}(\mathbf{s}_i, \mathbf{k}_i^0)$
first Nr-1 rounds
for $r = 1$ **to** $\mathsf{N_r} - 1$ **do**
  $\mathbf{s}_i \leftarrow \mathsf{ShiftRows}(\mathbf{s}_i)$
  for $a = 1$ **to** 4, **for** $b = 1$ **to** 4 **do** ▷ SubBytes
    $((\mathbf{s}_1)_{a,b}, (\mathbf{s}_2)_{a,b}, \ldots, (\mathbf{s}_n)_{a,b}) \leftarrow \mathsf{SecSbox}((\mathbf{s}_1)_{a,b}, (\mathbf{s}_2)_{a,b}, \ldots, (\mathbf{s}_n)_{a,b})$
  $\mathbf{s}_i \leftarrow \mathsf{MixColumns}(\mathbf{s}_i)$
  $\mathbf{s}_i \leftarrow \mathsf{AddRoundKey}(\mathbf{s}_i, \mathbf{k}_i^r)$
  last round
for $a = 1$ **to** 4, **for** $b = 1$ **to** 4 **do** ▷ SubBytes
  $((\mathbf{s}_1)_{a,b}, (\mathbf{s}_2)_{a,b}, \ldots, (\mathbf{s}_n)_{a,b}) \leftarrow \mathsf{SecSbox}((\mathbf{s}_1)_{a,b}, (\mathbf{s}_2)_{a,b}, \ldots, (\mathbf{s}_n)_{a,b})$
$\mathbf{s}_i \leftarrow \mathsf{ShiftRows}(\mathbf{s}_i)$
$\mathbf{c}_i \leftarrow \mathsf{AddRoundKey}(\mathbf{s}_i, \mathbf{k}_i^{\mathsf{N_r}})$
return $(\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n)$

Remark 3.16. Naturally, Rivain and Prouff also provided the security proof for the above construction, proving the t-privacy as explained in the ISW model. We decided to not sketch their proof but we refer the original paper. Such a decision has been taken for a simple reason: as we will see in our contributions, we have mostly dealt with a slightly stronger definition: SNI. Moreover, the Rivain-Prouff countermeasure has been proven to achieve not only t-privacy but also SNI. Thus, in the following section, we are going to introduce the work by Barthe

and others called *Strong Non-Interference* security and to show that the above construction is indeed SNI.

3.3 Strong Non-Interference

In [BBD+16], the authors proposed a new methodology to approach probing security in a more precise and scalable manner. They exploited the intrinsic connection between probabilistic information flow and probing privacy, in order to, not only verify masked algorithms but also to show how to systematically achieve privacy from an unprotected description of those algorithms. Among many contributions proposed in the paper, we will focus mainly on their stronger notion of probing security: Strong Non-Interference (SNI) privacy. Moreover, they showed how many well-known masked protocols were "unconsciously" achieving such stronger security guarantees. Many, but not all: as already mentioned in the previous section, Barthe et al. noticed a flaw in the Rivain-Prouff RefreshMasks (see Algorithm 3.3) and it, indeed, does not satisfy the strong non-interference property. Consequently, they proposed a substitute refreshing algorithm that enjoyed the SNI security and they showed how guaranteeing privacy for the whole gadget simply from the strong non-interference property of the refreshing algorithm.

In a bit we are going into the details of this new definition and the fixed RefreshMasks, but, for the sake of completeness, we before briefly mention the other contributions the paper provided even if they were not directly impacted our work. The authors also proposed: 1) an automatic tool to check whenever a gadget respects the strong non-interference properties for some of the protocols presented in the literature for $t \leq 6$; 2) a type-based enforcement of probing security; 3) a certified masking transformation able to take as input an arithmetic expression and returns a masked gadget satisfying their SNI security proposing along with a comparison among the various type systems from the state of the art. Anyway, we didn't have usage for our contributions, thus they just have mentioned and we can now present the formal security notion we worked with.

3.3.1 NI and SNI definitions

To better understand the differences between probing security and strong non-interference, it may be useful to use Barthe et al.'s terminology. Thus, we now recall the classic ISW privacy redefined as in [BBD+16]:

Definition 3.17 (t-NI security [BBD+16]). *Let G be a gadget taking as input n shares $(a_i)_{1\leq i\leq n}$ and n shares $(b_i)_{1\leq i\leq n}$, and outputting n shares $(c_i)_{1\leq i\leq n}$. The gadget G is said to be t-NI secure if for any set of t_1 probed intermediate variables and any subset $\mathcal{O}$ of output indices, such that $t_1 + |\mathcal{O}| \leq t$, there exist two subsets I and J of input indices which satisfy $|I| \leq t$ and $|J| \leq t$, such that the t_1 intermediate variables and the output variables $c|_O$ can be perfectly simulated from $a|_I$ and $b|_J$.*

We stress the fact that probing securities (no matter the way are defined) are *not* composable; that is, the guarantees implied by two different t-NI gadgets may not be maintained when the two gadgets are combined. As a matter of fact, for example, the NI definition does not allow *per se* to reduce the number of shares to $t+1$, i.e. $2t+1$ are still needed to satisfy the full construction of AES. This is due to the fact that, informally, NI does not guarantee enough independence between internal and output probes. In this sense, the SNI states this isolation-constraint must be respected. Moreover, differently from before, under the strong non-interference is possible to use only $t+1$ shares to guarantee the probing security of AES.

Definition 3.18 (t-SNI security [BBD+16]). *Let G be a gadget taking as input n shares $(a_i)_{1\leq i\leq n}$ and n shares $(b_i)_{1\leq i\leq n}$, and outputting n shares $(c_i)_{1\leq i\leq n}$. The gadget G is said to be t-SNI secure if for any set of t_1 probed intermediate variables and any subset O of output indices, such that $t_1 + |O| \leq t$, there exist two subsets I and J of input indices which satisfy $|I| \leq t_1$ and $|J| \leq t_1$, such that the t_1 intermediate variables and the output variables $c|_O$ can be perfectly simulated from $a|_I$ and $b|_J$.*

In fact, the difference between the two above definitions is in the size of the sets I and J. For the SNI, such cardinality cannot exceed the number of internal probes t_1. In other words, it has to be independent of the number of probes in $\mathcal{O}$ that, however, needs to be simulated. This constraint represents that level of isolation mentioned before and it can be applied in large systems to ensure composability: it is easy to derive the t-SNI of a system composed of only t-SNI secure components.

3.3.2 SNI Gadgets

As anticipated at the beginning of the section, along with the definitions, the authors analysed the literature and showed how many of those protocols achieved strong non-interference. In particular, the Rivain-Prouff masking, protecting AES against high-order attacks, has been proven SNI, this is the reason why RP contraction works with a number of shares equal to $t+1$. Specifically, their work focused on the SecMult (see Algorithm 3.2) and the RefreshMasks. Concerning the refreshing, the algorithm proposed in [RP10] has been proved insecure against attacks of an order $\lceil (n-1)/2 \rceil + 1$, see [CPRR14]; consequently, it needed to be fixed. Barthe et al. introduced a multiplication-based mask refreshing, which achieves strong non-interference.

Multiplication. The multiplication gadget did not need changes, the SNI has just to be proven. Since in Section 3.2 we did not provided any security proof referring to the SNI section, we provide here the proof[5] and the exact algorithm used.

Lemma 3.19 (t-SNI of SecMult [BBD+16]). *Let $(a_i)_{1\leq i\leq n}$ and $(b_i)_{1\leq i\leq n}$ be the input shares of the SecMult operation from [RP10] (see Algorithm 3.2), and let $(c_i)_{1\leq i\leq n}$ be its output*

[5] For the sake of simplicity, we actually refer to the proof provided in [CGPZ16]. Nevertheless, the proof is almost identical to the one proposed in the original paper.

Algorithm 3.9 SecMult from [BBD+16]

Input: shares a_i satisfying $\bigoplus_{i=1}^{n} a_i = a$, shares b_i satisfying $\bigoplus_{i=1}^{n} b_i = b$
Output: shares c_i satisfying $\bigoplus_{i=1}^{n} c_i = a \cdot b$

1: **for** $i = 1$ **to** n **do**
2: $\quad c_i \leftarrow a_i \cdot b_i$
3: **end for**
4: **for** $i = 1$ **to** n **do**
5: $\quad$ **for** $j = i + 1$ **to** n **do**
6: $\quad\quad r \leftarrow \mathbb{F}_2$ ▷ referred by $r_{i,j}$
7: $\quad\quad c_i \leftarrow c_i \oplus r$ ▷ referred by $c_{i,j}$
8: $\quad\quad r \leftarrow (a_i \cdot b_j \oplus r) \oplus a_j \cdot b_i$ ▷ referred by $r_{j,i}$
9: $\quad\quad c_j \leftarrow c_j \oplus r$ ▷ referred by $c_{j,i}$
10: $\quad$ **end for**
11: **end for**
12: **return** $(c_1, \ldots, c_n)$

shares. For any set of t_1 intermediate variables and any subset $\mathcal{O}$ of output shares such that $t_1 + |\mathcal{O}| < t$, there exist two subsets I and J of indices with $|I|, |J| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $a_{|I}$ and $b_{|J}$.

Proof. As usual, the goal is to build a pair of sets I,J such that every probe can be simulated by $a_{|I}$ and $b_{|J}$, with the additional constraint that $|I|$,$|J| \leq t_1$. By construction, each probed wire is in a limited number of forms: can be an input variable (i.e. a_i or b_i), can be a multiplication between inputs with the same index (i.e. $a_i b_i$) or different indexes (i.e $a_i b_j$), can be the auxiliary variables r (i.e $r_{i,j}$ or $r_{j,i}$) but also the intermediate operation computed for $r_{j,i}$ (i.e $a_i b_j \oplus r_{i,j}$), lastly, can be an output variable (i.e c_i) or an intermediate computation of it (i.e $c_{i,j}$). For the moment consider the intermediate probes, excluding the outputs c_i, those can be categorised into four groups:

1. The first group is composed of probes in the form: a_i, b_i or $a_i b_i$. In such cases the index i is added to both I and J.
2. The second group is composed of probes in the form: $r_{i,j}$ or $c_{i,j}$[6]. In such cases the index i is added to both I and J.
3. The third group is composed of probes in the form: $(r_{ij} \oplus a_i b_j)$. In such case, if i or j is already inserted in I and J (i.e at least one of the indexes belongs to the previous two groups) then both i and j are added to I and J.
4. The forth group is composed of the last remaining form: $a_i b_j$. In such a case the index i is added to I while the index j is added to J.

A couple of remarks are needed to better understand the upcoming simulation of those probes. The first is about the cardinalities of the two sets: after the computation of Group1 and Group2

[6] Note that by definition must hold that $i \neq j$.

holds that $I = J$ (for simplicity say U is defined as the common values of I and J after the second step) so $|I| = |J|$ and only one index is added per set; furthermore, by construction for Group3 is required that $i \in U$ or $j \in U$ as a consequence adding $\{i, j\}$ to both sets still consists in augmenting by one their cardinalities; finally, Group4 adds one index per set. In sum, $|I|,|J| \leq t_1$. The second is about the interconnection of $r_{i,j}$ in the computation of the partial sums of $c_{i,z}$ with $1 \leq i < j \leq z \leq n$. By construction, for $i < j$ the auxiliary value $r_{i,j}$ is required to compute all $c_{i,z}$ for $z \geq j$; in addition $r_{i,j}$ is required to compute $a_i b_j \oplus ri, j$, which is used also to compute $r_{j,i}$, which, in turn, is required to compute $c_{j,z}$ for $z \geq i$. Consequently, if $i \notin U$ (or respectively $j \notin U$) has to hold that $r_{i,j}$ (or $r_{j,i}$ resp.) does not belong to Group2, thereby it is not required in any computation of $c_{i,z}$ (or $c_{j,z}$ resp.). Hence, for the simulation of the internal probes, it is possible to identify four cases:

Case 1: $i \in U \wedge j \in U$. In those cases both indexes are in U, thus it is possible to perfectly simulate all the probes in the form: $a_i b_j$, $a_j b_i$, $r_{i,j}$, $r_{i,j} \oplus a_i b_j$ and $r_{j,i}$ exactly like they would have been computed in the real circuit.

Case 2: $i \in U \wedge j \notin U$. In those cases the probes can assume only two forms: either $r_{i,j}$ (if $j \notin J$) and a random assignment perfectly simulates them; or $r_{i,j} \oplus a_i b_j$ but, in that case, by construction must hold that $j \in J$ so it can be perfectly simulated as well.

Case 3: $i \notin U \wedge j \in U$. In those cases neither $r_{i,j}$ or any partial sum $c_{i,z}$ must not have been probed, otherwise i would belong to U. Consequently, the probe has form $r_{j,i}$ (if $i \notin J$) or $r_{i,j} \oplus a_i b_j$ (if $i \in J$). The first case is straightforward and a random assignment perfectly simulates it; the latter can be computed like in the real circuit from $a_i b_j \oplus r_{i,j} = a_j b_i \oplus r_{j,i}$.

Case 4: $i \notin U \wedge j \notin U$. In those cases the probe can only be in the form $r_{i,j} \oplus a_i b_j$ but both i and j are not in U. Thus, $r_{i,j}$, a_i and b_i have not been probed and a random assignment cannot be distinguished from a real execution of the circuit.

The only remaining forms excluded from above are the partial sum $c_{i,z}$, but those probes can be indirectly and indistinguishably simulated from every $r_{i,j}$ with $i \in U$ (which, in turn, can be perfectly simulated as explained in Case 1, 2 and 3). This concludes the simulation of every internal probe; we now consider the simulation of probes in the set $\mathcal{O}$, i.e. the output variables $c_{|\mathcal{O}}$. The first subset of output probes can be easily simulated and they are the c_i where $i \in U$; as said before for such variable is possible to simulate every partial sum $c_{i,z}$ and that includes the final sum c_i. Thus, we consider now only the c_i with $i \notin U$. For such probes, it is possible to construct a subset of indexes V in the following way: for every observed variable belonging to Group3 (i.e $r_{i,j} \oplus a_i b_j$) such that i,$j \notin U$, j is added to V if $i\mathcal{O}$, otherwise i is added to V. It is easy to see that since only the probes in Group3 are taken into account must hold that $|U|+|V| \leq t_1$. This implies that $|U|+|V|+|\mathcal{O}| \leq t$. Consequently, what is needed to be proven is that there must exist at least an index, say $j^\star \in [1, n = t + 1]$, such that $j^\star \notin U \cup V \cup \mathcal{O}$; in other words, there is at least an auxiliary $r_{i,j^\star}$ in the computation of c_i which has not been probed, neither is used in the computation of any other $c_{i'}$ for $i' \in \mathcal{O}$. Thus, if it can be proven then the adversary can not distinguish a random value from a real computation of the output share. Given any index $i \in \mathcal{O}$, it is possible to rewrite the computation of c_i in the following

way:

$$c_i = a_i b_i \oplus \bigoplus_{j \neq i} r_{i,j} = r_{i,j^\star} \oplus \left(a_i b_i \oplus \bigoplus_{j \neq i, j^\star} r_{i,j} \right)$$

Let analyse such $r_{i,j^\star}$: 1) we know that $i \notin U$, it implies that $r_{i,j^\star}$ or any other partial sum $c_{i,z}$ has been probed; 2) we know that $j^\star \notin U$, this implies that $r_{j^\star,i}$ or any other partial sum $c_{j^\star}, z$ has been probed; 3) we know that $c_{j^\star}$ has not been probed, otherwise would have held that $j^\star \in \mathcal{O}$, which is not by definition; 4) assuming that $i < j^\star$, we know that $r_{i,j^\star} \oplus a_i b_{j^\star}$ has not been probed, otherwise $j^\star \in V$ since $i \in \mathcal{O}$; 5) assuming that $j^\star < i$, we know that $r_{j^\star,i} \oplus a_{j^\star} b_i$ has not been probed, otherwise $j^\star \in V$ since $j^\star \notin \mathcal{O}$. Therefore, is it possible to claim that neither $r_{i,j^\star}$ nor $r_{j^\star,i}$ have been observed or used elsewhere. And this means that a random assignment for c_i is indistinguishable from its real computation.

□

Refreshing. As said, the refreshing needed more attention. It has to be redefined to achieve strong non-interference, the authors provided the following candidate, which has been proved and accepted as best practice:

Algorithm 3.10 RefreshMasks

Input: shares x_i satisfying $x = \bigoplus x_i$
Output: shares y_i satisfying $x = \bigoplus y_i$

1: **for** $i = 1$ **to** n **do**
2: $\quad y_i \leftarrow x_i$
3: **for** $i = 1$ **to** n **do**
4: $\quad$ **for** $j = i + 1$ **to** n **do**
5: $\quad\quad r \xleftarrow{\$} \mathbb{F}_{2^k}$ ▷ refereed by $r_{i,j}$
6: $\quad\quad y_i \leftarrow y_i \oplus r$ ▷ refereed by $y_{i,j}$
7: $\quad\quad y_j \leftarrow y_j \oplus r$ ▷ refereed by $y_{j,i}$
8: **return** $(y_1, \ldots, y_n)$

Lemma 3.20 (t-SNI of RefreshMasks [BBD+16]). *Let $(x_i)_{1 \leq i \leq n}$ be the input shares of the* RefreshMasks *operation from [BBD+16], and let $(y_i)_{1 \leq i \leq n}$ be its output shares. For any set of t_1 intermediate variables and any subset $\mathcal{O}$ of output shares such that $t_1 + |\mathcal{O}| < t$, there exists a subset I of indices with $|I| \leq t_1$, such that the t_1 intermediate variables, as well as the output shares $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

Proof. The proof starts, as usual, defining the set of indexes I as stated in the lemma. Pretty straightforwardly, by definition, every internal probe can leak values: y_i, $r_{i,j}$ or $y_{i,j}$ or $y_{j,i}$ with $i < j$, I is constructed as follow:

1. for y_i, $r_{i,j}$ or $y_{i,j}$ then i is added to I.
2. for $y_{j,i}$ then j is added to I.

Should be easy to see that, since for each case always one index is added to the set, then the cardinality of I is at most as the number of internal probes, i.e. $|I| \leq t_1$. Concerning the simulation itself, the author gave a helpful visualization which we propose for the sake of clearness: the algorithm can be represented by the following matrix, where each final refreshed share y_i is computed XORing each element of line i.

$$\begin{pmatrix} x_1 & 0 & r_{1,2} & r_{1,3} & \cdots & r_{1,n} \\ x_2 & r_{1,2} & 0 & r_{2,3} & \cdots & r_{2,n} \\ x_2 & r_{1,3} & r_{2,3} & 0 & \cdots & r_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n & r_{1,n} & r_{2,n} & r_{3,n} & \cdots & 0 \end{pmatrix}$$

Moreover, each partial sum $y_{i,j}$ is computed XORing the first $j+1$ element of line i. Thus, by construction, whenever an element in the i-th row is observed i is in I and x_i can be used to perfectly simulate the entire row, since the rest can be computed simply at random (like it would have been generated in the real circuit). This can be applied to simulate every internal probe. To conclude the proof is remained to simulate the output probes which index is in $\mathcal{O}$. It can only be that $i \in I \cap \mathcal{O}$ or $i \in \mathcal{O}$ but not in I. The latter case is straightforward, since $i \notin I$ then the adversary has not probed any partial sum $y_{i,j}$ and, in turn, it is impossible to distinguish y_i from a random value. Regarding $i \in I \cap \mathcal{O}$, since i is also in I then the entire row has already been perfectly simulated and y_i can be computed as it would have been in the real circuit, XORing every element in the row. Hence, both internal and output probes can be perfectly simulated from $x_{|I}$, making the refreshing t-SNI.

□

3.4 Region Probing Model

3.4.1 Leakage Rate and Regional Probing Model

So far, we mainly focus on practices and enhancements at the time level, which, in turn, allowed to enlarge the threshold t. Namely, faster is the computation and more shares can be used in the masking and more secure is the implementation. Thus, the value of t became an important parameter to evaluate the security of a masked scheme: more knowledge the adversary can "safely" (i.e without recovering sensitive information) obtain and stronger is the level of security. Unfortunately, it has been recognized that the aforementioned reasoning does not always lead to more secure schemes. A part of the community started to (also) look at what has been called the *leakage rate*: the ratio between the threshold t and the total amount of wires in the circuit. It is important to notice that also the leakage rate needs to be treated as a guideline and not as an absolute parameter for evaluations, in fact, the rate still does not consider practical aspect like the well-established fact that, for instance, writing in-memory operations leak more than arithmetic ones.

In 2016, Andrychowicz et al. introduced the first improvement in the leakage rate in [ADF16]; compared to the implicit $\mathcal{O}(1/n)$ rate represented by the ISW constructions. It should be easy to see the way in which the AND gate is masked (see Algorithm 3.1) any input share (independently from a_i or b_i) is required in the computation at least n times, making the $\mathcal{O}(1/n)$ bound tight. The authors proposed a way to securely mask any circuit achieving a leakage rate of $\mathcal{O}(1/\log n)$. We are not going to discuss their results about the leakage ratio but we will get into the slightly different model that they worked on: the *Regional Probing Model*. To prove security in the classical probing model it is required to (perfectly/statistically/computationally) simulate the view from any adversary observing the value of (up to) t, freely chosen, wires in the *entire* transformed circuit. While in the regional probing model, a significantly stronger adversary is taken into account, where it can probe t wires per sub-circuit. That is, the transformed circuit is conceptually divided into *regions* and the adversary is allowed to observe t variable in each region. Typically, a region is associated with a transformed gadget; concretely, we consider any masked AND/XOR gate (or an AND/XOR gadget) a region. Moreover, by definition, a wire connecting two gates can therefore belong (and must be considered in) to both regions. Such model intrinsically improves the leakage rate. The authors provided a rigorous definition of the t-regional privacy based on the Ideal-RealWorld simulation paradigm (we refer to [ADF16]), but we reputed sufficient to consider the definitions presented in Section 3.1 with the difference that the adversary can, now, probe t wires per region, instead of the whole circuit C'.

3.4.2 About [ISW03], Regional Privacy and Re-Randomization Property

To be truthful, the region probing model was already considered in [ISW03], with one region per gadget. Ishai at al. claimed that security in this model can be deduced from their constructions due to the re-randomization property of the outputs of the AND gadget. Namely, for each original output bit, the encoded outputs are $(n-1)$-wise independent even given the entire n-encoding of the inputs; they claimed this would imply security against a stronger type of adversary who may observe at most t' wires in each gadget, where $t' = \Omega(t)$. However, we argue that this re-randomization property is actually not enough to achieve security in the region probing model: we exhibit a simple counterexample, i.e. a gadget achieving the re-randomization property but insecure in the region probing model. Consider a mask refreshing algorithm taking as input $x_1, \ldots, x_n$ and outputting $y_1 = x_1 \oplus r_1; \ldots; y_{n-1} = x_{n-1} \oplus r_{n-1}$ and $y_n = x_n \oplus r_1 \oplus \cdots \oplus r_{n-1}$, for random $r_1, \ldots, r_{n-1}$; here y_n is computed from left to right. Such mask refreshing achieves the re-randomization property as the output shares $y_1, \ldots, y_n$ are $(n-1)$-wise independent, even given all the input shares $x_1, \ldots, x_n$. However such mask refreshing does not achieve region probing security. In fact, if we iterate such mask refreshing multiple times, it is possible to see that the adversary can accumulate knowledge and eventually recover the original secret, by probing only a constant fraction of the wires in each mask refreshing gadget. More precisely, assuming that we can probe $n/4 + 2$ internal variables

at each iteration; at the first iteration, the adversary probes variables $x_1, \ldots, x_{n/4}$, x_n and $x_n \oplus r_1 \oplus \cdots \oplus r_{n/4}$. Therefore the adversary can compute:

$$\begin{aligned} y_1 \oplus \cdots \oplus y_{n/4} &= (x_1 \oplus r_1) \oplus \cdots \oplus (y_{n/4} \oplus r_{n/4}) \\ &= x_1 \oplus \cdots \oplus x_{n/4} \oplus x_n \oplus (x_n \oplus r_1 \oplus \cdots \oplus r_{n/4}) \end{aligned}$$

Therefore from the knowledge of $x_1 \oplus \cdots \oplus x_{n/4}$, the adversary can keep the knowledge of $y_1 \oplus \cdots \oplus y_{n/4}$ by spending only 2 additional probes within the mask refreshing. Let $z_1 = y_1 \oplus r'_1, \ldots, z_{n-1} = y_{n-1} \oplus r'_{n-1}$ and $z_n = y_n \oplus r'_1 \oplus \cdots \oplus r'_{n-1}$ be the second iteration of the mask refreshing algorithm. The adversary can now probe $y_{n/4+1}, \ldots, y_{n/2}$, y_n and $y_n \oplus r'_1 \oplus \cdots \oplus r'_{n/2}$; see Figure 32 for an illustration. The attacker can then compute as previously:

$$z_1 \oplus \cdots \oplus z_{n/2} = (y_1 \oplus \cdots \oplus y_{n/4}) \oplus y_{n/4+1} \oplus \cdots \oplus y_{n/2} \oplus y_n \oplus (y_n \oplus r'_1 \oplus \cdots \oplus r'_{n/2})$$

Similarly, at the 3rd iteration, the adversary can compute the XOR of 3/4 of the output shares, and after the 4th iteration, it can eventually recover the XOR of the n shares. The attack can be adapted when probing any constant fraction of the n shares; see Figure 32. Therefore the above mask refreshing algorithm is not secure in the region probing model, despite achieving the re-randomization property.

3.4.3 Privacy in the Stateless Regional Probing Model

The simpler way to achieve security in the region probing model is actually via the t-SNI notion explained in Section 3.3. The authors showed that the notion allows for securely composing masked algorithms; i.e. the t-SNI of a full construction can be proven based on the t-SNI of its component gadgets. However, the notion *per se* is not enough, as well as $n = t + 1$ shares are not sufficient to ensure it. Thus, to achieve privacy in the region probing model, we considered the ISW masking countermeasure from Section 3.1.1, but in which we additionally performed an $(n-1)$-SNI mask refreshing algorithm as inputs of each XOR and AND gadgets. For this purpose, we consider RefreshMasks from Lemma 3.20 which guarantees $(n-1)$-SNI security. Note that, in Lemma 3.20, it is stated the t-SNI property, while, here, we are claiming its $(n-1)$ tolerance; this may be confusing, but both statements are equivalent since in the algorithm is considered a number of shares $n = t + 1$, consequently $t = n - 1$. Moreover, as before, each region is defined as comprising an AND or XOR gadget, and the mask refreshing of the corresponding output variable so that each output $z^{(j)}$ is used only once in the next region (see Figure 33 for an illustration, where it is considered a general f fan-out). From the composability of the SNI and both previous lemmas Lemma 3.19 and Lemma 3.19, it is straightforward to deduce the $(n-1)$-SNI of the "refreshed" AND gadget.

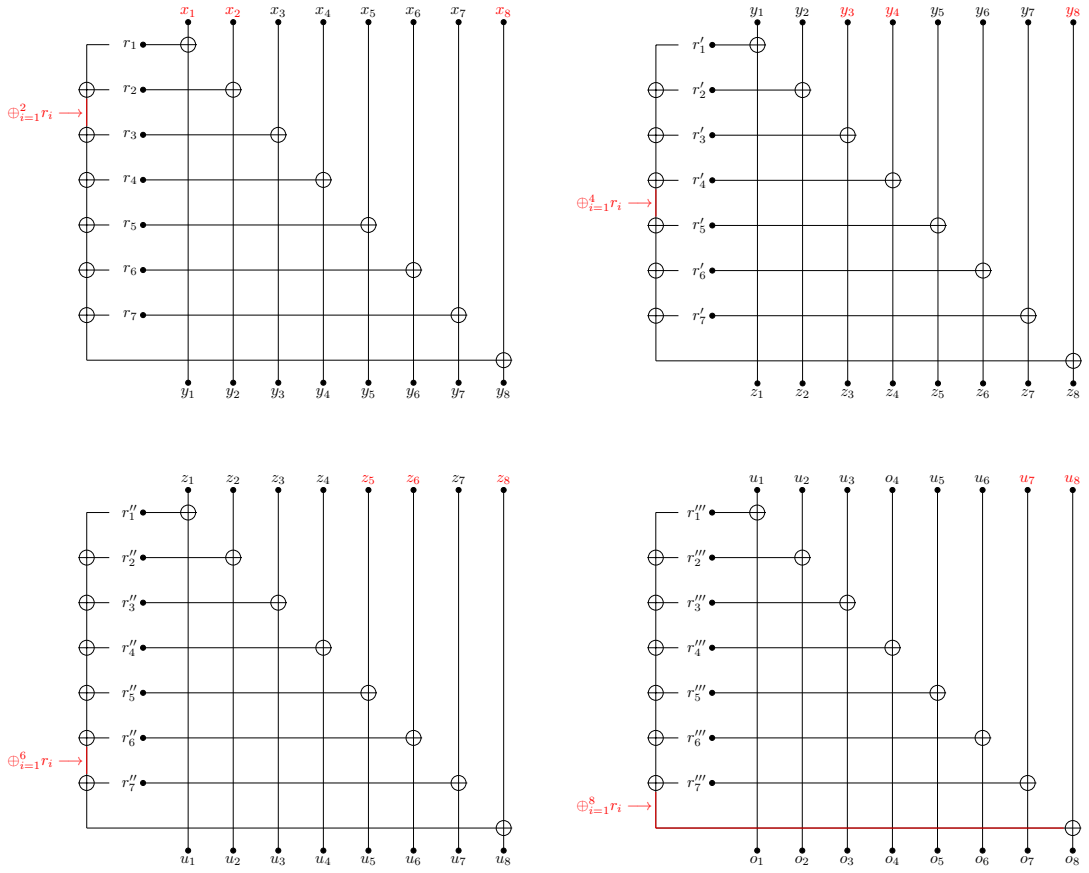

Fig. 32: Counterexample with $n = 8$, the red values represent the probes of the attacker.

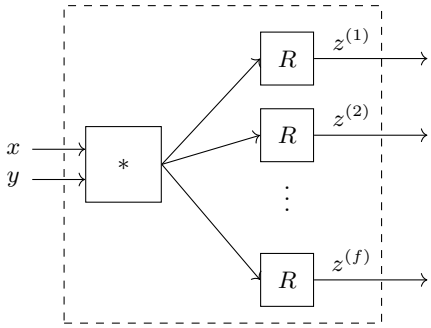

Fig. 33: A region comprises a AND or XOR gadget, and the mask refreshing of the output variable.

Thus, before proving the t-regional privacy for the whole circuit $\widetilde{C}$, we need to explicitly demonstrate that even the XOR gadget from the ISW construction augmented with a RefreshMasks per output achieves $(n-1)$-SNI property. Recalling Section 2.1, we assume that such a gadget has up to fan-out 2; actually, we assume fan-out exactly 2 since the other case is easier and straightforward to deduce from the above proof.

Lemma 3.21. *Let G be the* XOR *gadget from Section 3.1.1, where an $(n-1)$-*SNI *mask refreshing is applied after it. Such a gadget achieves $(n-1)$-*SNI *security.*

Proof. Let $(a_i)_{1\leq i\leq n}$ and $(b_i)_{1\leq i\leq n}$ be the input shares of the XOR gadget, and $(c_i)_{1\leq i\leq n}$ be the corresponding output shares; let $(d_i)_{1\leq i\leq n}$ the output shares of the first Refresh and $(e_i)_{1\leq i\leq n}$ the output shares of the first Refresh. Let t_3,t_1 and t_2 be the number of probes within the XOR, $\mathsf{Refresh}_1$ and $\mathsf{Refresh}_2$ gadgets respectively, with $t_3+t_1+t_2+|\mathcal{O}|<n$, where $\mathcal{O}=\mathcal{O}_1\cup\mathcal{O}_2$ is the set of output variables to be simulated with $\mathcal{O}_1\subset(d_i)_{1\leq i\leq n}$ and $\mathcal{O}_2\subset(e_i)_{1\leq i\leq n}$; see Figure 34 below.

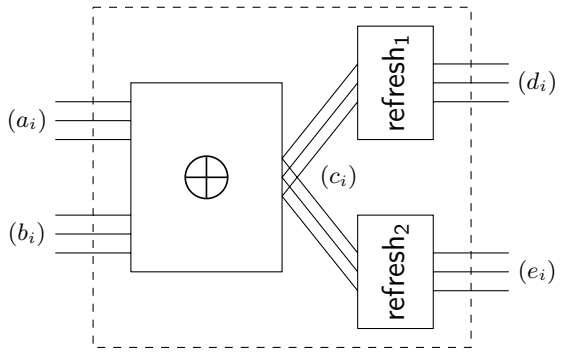

Fig. 34: XOR gadget composed with RefreshMasks.

Now, since by assumption the $\mathsf{Refresh}_1$ gadget is $(n-1)$-SNI and $t_1+|\mathcal{O}_1|<n$, the t_1 probes within $\mathsf{Refresh}_1$ and the output variables $d_{|\mathcal{O}_1}$ can be simulated from $c_{|R_1}$, with $|R_1|\leq t_1$; similarly, the $\mathsf{Refresh}_2$ gadget is $(n-1)$-SNI too and $t_2+|\mathcal{O}_2|<n$, thus, the t_2 probes within $\mathsf{Refresh}_2$ and the output variables $e_{|\mathcal{O}_2}$ can be simulated from $c_{|R_2}$, with $|R_2|\leq t_2$. Furthermore, by the internal structure of the XOR (see Section 3.1.1), the variables $c_{|R_1\cup R_2}$ can be simply simulated from $a_{|R_1\cup R_2}$ and $b_{|R_1\cup R_2}$. Lastly, the t_3 probes within the XOR gadget can be perfectly simulated from $a_{|S}$ and $b_{|S}$, with $|S|\leq t_3$. Note that no further assumption, but from the t-privacy, has been considered for the XOR component, for this reason, the set S has cardinality t_3. Additionally, we would like to remark that no specification has been made about the c_is that have been probed by the adversary, namely when they were counted as inputs of $\mathsf{Refresh}_i$ or outputs of XOR; nevertheless, as long as they are counted once, the affiliation is irrelevant. Therefore the output variables $d_{|\mathcal{O}_1}$ and $e_{|\mathcal{O}_2}$ along with every other internal probes

the within the full gadget can be perfectly simulated from $a_{|I}$ and $b_{|I}$ with $I = R_1 \cup R_2 \cup S$; since by definition, $|I| \leq |R_1| + |R_2| + |S| \leq t_3 + t_1 + t_2 < n - |\mathcal{O}|$ as required by Definition 3.18, the whole gadget is $(n-1)$-SNI. Clearly, the 1-output case would follow the same reason, with the difference of having only one set R, but the inequalities will be respected anyway.

□

We can now state the following theorem, proving that the ISW construction for t-privacy in the stateless model, modified adding a $(n-1)$-SNI as input of each AND and XOR gadget, is secure in the regional settings. This is done by exploiting the SNI property of every single gadget. More precisely:

Theorem 3.22 (t-privacy in the region probing model). *Let C be a stateless circuit of fan-out 2. Let (T, I, O) be the* ISW *transformer with $n = 2t + 1$ shares, where an $(n-1)$-*SNI *mask refreshing is applied as input of each* XOR *and* AND *gadgets. The transformed circuit is t-private secure where the adversary can put at most t probes per region, each of size $\mathcal{O}(t^2)$.*

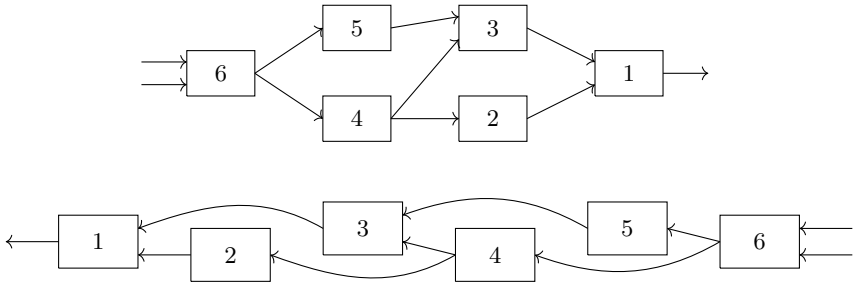

Fig. 35: Example of a circuit (top), and the same circuit displayed in reverse topological sort order (bottom).

Proof. We consider the circuit as a set of q gadgets $G_1, \ldots, G_q$ that we order as a direct acyclic graph from output to input in a reverse topological sort order; see Figure 35 for an illustration. We know that each gadget G_i achieves the $(n-1)$-SNI property and let denote by $(x_i^{(j)})_{1 \leq i \leq n}$ the input shares of the composed circuit, for $1 \leq j \leq m$, where m is the number of inputs in the original circuit corresponding to the q gadgets $G_1, \ldots, G_q$. We prove by recurrence on q that the composition of the q gadgets achieves the following property: any set of t probes in each gadget can be perfectly simulated from the knowledge of the input shares $x_{|I_j}^{(j)}$, for subsets I_j with $|I_j| \leq t$ for all $1 \leq j \leq m$. The base case $q = 1$ is straightforward from the $(n-1)$-SNI property of the gadget G_1. Thus, let now consider q gadgets $G_1, \ldots, G_q$ with input shares $(x_i^{(j)})_{1 \leq i \leq n}$ for $1 \leq j \leq m$, and an additional gadget G_{q+1}. By assumption any set of t probes in each gadget $G_1, \ldots, G_q$ can be perfectly simulated from the knowledge of the input shares $x_{|I_j}^{(j)}$, with $|I_j| \leq t$ for all $1 \leq j \leq m$. Let $(y_i)_{1 \leq i \leq n}$ be the output shares of the inner

AND or XOR components of G_{q+1}, and let $(x_i^{(m+1)})_{1\leq i\leq n}$, $(x_i^{(m+2)})_{1\leq i\leq n}$ be the corresponding input shares. Since the original circuit C has fan-out 2, the output shares of G_{q+1} are used by at most two gadgets G_{j_1} and G_{j_2}. But by construction, the two refreshing masks applied to $(y_i)_{1\leq i\leq n}$ will output two different sets of shares $(y_i^{(1)})_{1\leq i\leq n}$ and $(y_i^{(2)})_{1\leq i\leq n}$. For this reason, the probes in I_{j_1} and I_{j_2} cannot exceed the threshold t; thus, we can define $\mathcal{O} = I_{j_1} \cup I_{j_2}$ be the output shares of G_{q+1} that must be simulated; it holds that $|\mathcal{O}| \leq |I_{j_1}| + |I_{j_2}| \leq t$. Since G_{q+1} is $(n-1)$-SNI and $|\mathcal{O}| + t \leq 2t < n$, the t probes within G_{q+1} and the output shares can be perfectly simulated from $x_{|I_{k+1}}^{(k+1)}$ and $x_{|I_{k+2}}^{(k+2)}$, with $|I_{k+1}|, |I_{k+2}| \leq t$. This proves the inductive property for the $q+1$ gadgets $G_1, \ldots, G_{q+1}$. Eventually, this proves that the transformed circuit is t-private secure where the adversary can put at most t probes per region.

□

3.4.4 Privacy in the Stateful Regional Probing Model

Symmetrically, we adapted the ISW construction for the stateful circuits (see Section 3.1.1) to obtain the t-privacy in the regional probing model. Like in [ISW03], our construction proceeds as follows; see Figure 36 for an illustration.

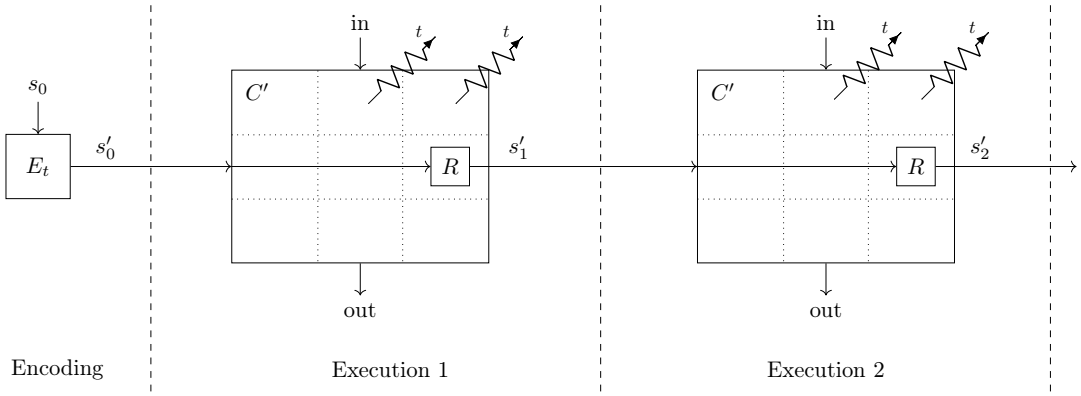

Fig. 36: Illustration of the stateful model. The initial encoding s'_0 used in the first execution gets refreshed into s'_1 before getting passed to the next execution. The adversary can put t probes per region within each execution, where the position of the probes can be changed between executions.

We used the stateless transformer $T(C,t)$ secure in the region probing model from above, with t probes per region. Let denote with $E_t(x)$ the encoding used by the stateless transformer, where x is the input being encoded. The initial state s_0 of C is encoded as $s'_0 = E_t(s_0)$.[7] At

[7] Here we can use E_t instead of E_{2t} in [ISW03] because we consider a circuit C' already secure in the region probing model.

the i-th invocation, the circuit $C' = T(C, t)$ takes as input an encoded state s'_i and outputs an encoded state s'_{i+1} that is passed to the next circuit execution. Note that the encoded state s'_i must be refreshed after each execution; otherwise, the adversary could recover the internal state by probing the encoded state t probes at a time; in the circuit C' this is done by using an $(n-1)$-SNI mask refreshing R as output. Recall that for a block-cipher, the internal state corresponds to the key whose encoding must be refreshed after each execution. The regular input in of C is unprotected, and need not be encoded before getting fed into C'; for each execution of C', this input is first encoded using E_t and the output out is decoded using the corresponding D_t. This implies that these inputs and outputs are known to the adversary so that they can be given for free to the simulator. Thus, the perfect privacy in the stateful model is derived from the perfect privacy in the stateless case, thanks to the region probing model. Namely, a sequence of invocations of the stateful circuit C' can be unwound into a larger stateless circuit $\widetilde{C}$; in the unwound circuit, the adversary can corrupt up to t wires in each region of each circuit produced by the stateless transformation. This means that every new circuit execution corresponds to adding more regions in the unwound circuit (see Figure 36). However, the adversary is allowed to move its probes between circuit executions; therefore in the unwound circuit $\widetilde{C}$, the probes corresponding to the i-th execution must be simulated without knowing the position of the probes from the $(i+1)$-th execution. To perform these successive simulations we had to consider a slightly stronger definition than t-SNI security just for mask refreshing between successive executions, which we called *free-t-SNI* security, where in a given gadget the set of input variables I that must be known for the simulation, does not depend on the set of output variables O to be simulated; formally: under this notion, the output variables $c_{|O}$ must be uniformly and independently distributed for any $O \subsetneq [1, n] \setminus I$, and this must be true even conditioned on the probed variables and $c_{|I}$. This means that even after the simulation of the probed variables has been provided to the adversary, it is still possible to simulate the variables in $c_{|O}$, simply by generating uniformly and independently random values (this was not necessarily the case with the original t-SNI notion).

Definition 3.23 (free-t-SNI security). *Let G be a gadget taking as input n shares $(a_i)_{1 \le i \le n}$ and outputting n shares $(c_i)_{1 \le i \le n}$. The gadget G is said to be free-t-SNI secure if for any set of $t_1 \le t$ probed intermediate variables, there exists a subset I of input indices with $|I| \le t_1$, such that the t_1 intermediate variables and the output variables $c_{|I}$ can be perfectly simulated from $a_{|I}$, while for any $O \subsetneq [1, n] \setminus I$ the output variables in $c_{|O}$ are uniformly and independently distributed, conditioned on the probed variables and $c_{|I}$.*

The RefreshMasks (see Algorithm 3.10 from Section 3.3.2) satisfies the above free t-SNI notion. The proof will be sketched since is essentially the same as the proof of the ISW multiplication algorithm from [CGPZ16, Appendix B.1]. That is, the proof constructs a subset $I = U$ of indices such that the t_1 probes and $c_{|I}$ can be perfectly simulated from $a_{|I}$, and in the proof the other output variables $c_{|O}$ from any $O \subsetneq [1, n] \setminus I$ are simulated by generating a random value. Therefore the output variables in $c_{|O}$ are uniformly and independently distributed, conditioned on the probed variables and $c_{|I}$. This led us to formulate the following lemma:

Lemma 3.24 (free-$(n-1)$-SNI of RefreshMasks). *The* RefreshMasks *algorithm from [BBD+16] is free-$(n-1)$-*SNI.

Finally, as we made for the stateless setting we provide proof for the construction which satisfies the t-privacy also in the regional model, where the adversary probes t wires per gadget.

Theorem 3.25 (Perfect privacy, region stateful model). *There exists a perfectly t-private stateful circuit transformer secure in the region probing model which maps any stateful circuit C of size $|C|$ and depth d to a randomized stateful circuit of size $\mathcal{O}(|C| \cdot t^2)$ and depth $\mathcal{O}(d \log t)$.*

Proof. The proof is essentially the same as the proof of Theorem 3.22 from above. The difference is that in the unwound circuit $\widetilde{C}$, the adversary can move its probe between regions corresponding to different executions of the circuit C'. This implies that the simulation for the i-th execution must be performed before knowing the position of the probes for the $(i+1)$-th execution. This is not a problem thanks to the characteristic of the new free-$(n-1)$-SNI property of RefreshMasks (see Lemma 3.24). Namely, the input shares of the next gadget (that must be known for the next simulation), either belong to the set $c_{|I}$ of outputs shares from the previous gadget and are already simulated, or to a set $c_{|O}$ of output shares that can be simulated by generating uniformly distributed values. Therefore the simulation for the stateful model can be performed by running a sequence of simulations from the stateless model.

□

Remark 3.26. As obvious as it is, one can obtain privacy in the region model also for the statistical setting, for both stateless and stateful cases. It is enough to consider the two adapted ISW constructions for the regional probing model in the original proofs, respectively from Section 3.1.4 and Section 3.1.5. The proofs would still hold and, from them, one can deduce the statistical privacy even for adversary probing t wires per gadget.

Remark 3.27. From the next Section on, unless it is not explicitly specified, we will always refer to the regional probing model. Namely, whenever in the next chapters it is stated t-private or t-privacy or any other references to the previous definitions, it will still mean with respect to the regional probing model.

3.5 Software Probing Model

In the next chapter, we will present our work on the first improvement made regarding statistical privacy (see Section 3.1.4). And we will refer to the *Software Probing Model*. Sometimes, for the sake of clearness, it is useful (and practical) to work in the RAM model used in the algorithmic analysis; see [MS08, Section 2.2]; especially when software implementations are analysed. In this model, each memory access takes a unit of time, and every memory cell can store an integer whose bit-size is logarithmic in the input size; for a polynomial-time algorithm,

this enables to store array indices in a single cell. Moreover, in the software probing model, it is assumed that during the execution the adversary can only probe the input address and output value of a RAM cell that is read during a table look-up, but not the content of the internal wires of the circuit implementation of the RAM. This software probing model was already used for example in the high-order table look-up countermeasure from [Cor14]. For simplicity, as you will see, we are going to still describe our construction in terms of an expanded circuit $\widetilde{C}$ as in [ISW03]. But, the software implementation of our circuit $\widetilde{C}$ is augmented with a RAM unit, where the adversary can only probe the input address and the input/output value, but not the internal wires of the RAM. Nevertheless, we would stress the fact that, for completeness, we also provide in Section 4.2.4 a pure circuit description of our countermeasure, with a proof of security in the standard wire probing model.

4

Secure Wire Shuffling in the Probing Model

In this chapter, we are going to present our first work, published at CRYPTO21, [CS21]. The masking countermeasure used in the perfect privacy is quite practical and has been widely studied with numerous improvements as we presented in the previous chapter, but the inherent quadratic factor can not be lowered. For this purpose Ishai et al. relaxed the privacy to its statistical form, achieving quasi-linear complexity. Up to our knowledge, such proposal construction has never been under further investigation. Our work describes an improved construction in the statistical model that is better asymptotically and practically, since we argue that the original construction from [ISW03] was, despite its theoretical potential, essentially unpractical. To briefly summarize the construction in Section 3.1.4, their gadgets have complexity $\mathcal{O}(t \log t)$ and they proceed in two steps: 1) it considers statistical security in the weaker random probing model (Section 3.1.3), achieved through the masking countermeasure against $t = k$ probes with $2k + 1$ shares (where k is the security parameter); 2) the resulting transformed circuit from the previous step is expanded in such a way that each wire is mapped into ℓ new wires mostly dummies (see Figure 31), so that only a relatively small subset of wires, randomly distributed, in the circuit would take care of the actual computation. We called this construction the *Wire Shuffling Countermeasure*, as it consists in randomly shuffling the position of the signal among those ℓ wires.

Since it will be the most significant change in our proposed construction, it can be helpful to recall the functioning of the final transformed gadget $\widetilde{G}$: it takes 2ℓ inputs, $(a_i)_{1 \leq i \leq \ell}$ and $(b_i)_{1 \leq i \leq \ell}$, where the information v and v', from the intermediate circuit, are located at index $j \in [1, \ell]$ and $j' \in [1, \ell]$; then, for each index $1 \leq i \leq \ell$ a pair of $(\mathsf{key}_i, \mathsf{val}_i)$ is formed (we refer to Section 3.1.4 for how such pairs are computed) in such a way that running a sorting network algorithm on $(\mathsf{key}_1, \mathsf{val}_1), (\mathsf{key}_2, \mathsf{val}_2), \cdots (\mathsf{key}_\ell, \mathsf{val}_\ell)$, according to key_i, it would put v and v' in random but adjacent positions; finally, the i-th output share would be computed looking at val_i, val_{i+1} and val_{i+2}, leaving the final output $v * v'$, with $* \in \{\mathsf{AND}, \mathsf{XOR}\}$, in a uniformly random position $j'' \in [1, \ell]$. Additionally, recall that the logarithmic factor in the gadget complexity is given by the sorting algorithm. It is in this part of their scheme that our improvement took place: we managed to replace the sorting algorithm with a simpler technique,

achieving linearity, i.e. $\mathcal{O}(t)$ complexity, for each gadget. As in [ISW03], we randomly shuffled the position of the signal v_i among the ℓ wires, independently for each original wire i of the intermediate circuit. However, we explicitly computed the index position $j_i \in [1, \ell]$ of each signal v_i among the ℓ wires; whereas in ISW this position was only implicitly determined by the value of the ℓ wires, as one of them would contain the signal v_i while the others would get the dummy value \$.

Informally, consider the two wires i and i' in C', for which the signal is located at positions $j \in [1, \ell]$ and $j' \in [1, \ell]$ in the expanded circuit $\widetilde{C}$. Since the positions j and j' of the signal are now explicitly computed, we don't need to use a sorting network anymore. Instead, we can simply generate a new random index $j'' \in [1, \ell]$, and cyclically shift the information corresponding to wire i by $\Delta = j'' - j$ positions modulo ℓ, and similarly by $\Delta' = j'' - j'$ positions for wire i'. For both inputs, the signal is now located at the common position $j + \Delta = j' + \Delta' = j''$; consequently, it is possible to process the signal directly at such position j''. As a first advantage, a such cyclic shift has the main advantage that it can be computed in linear time instead of $\mathcal{O}(\ell \log \ell)$, hence we can get statistical security with time complexity $\mathcal{O}(|C| \cdot t)$ the whole circuit instead of $\mathcal{O}(|C| \cdot t \log t)$ proposed in [ISW03]. In addition, of minor importance but still appreciable, this construction is also much easier to implement in practice, as we can use a simple table look-up for the cyclic shifts, instead of a complex sorting network.

We stress the fact that the main difference between our and the original ISW construction is that the index positions of the signal values are now explicitly computed in the final circuit $\widetilde{C}$. This had to be taken into account in our security proofs since those index positions can be probed by the adversary, so we may as well assume that the adversary knows all those index positions. We crucially relied on the fact that, as in [ISW03], the adversary learns those positions only at the evaluation phase (being generated only at the running time); namely, *after* it has committed its probes in the circuit. Therefore when the adversary learns the exact locations it is actually too late: we showed that even those new information, it can only learn the signal values with probability at most p. This means that, as in the original ISW, the adversary cannot learn more from the worst-case probing of the final circuit $\widetilde{C}$, than from the p-random probing of the intermediate circuit C'; we will see that from this the worst-case statistical security can be achieved for our transformed circuit $\widetilde{C}$.

Everything above introduced has been considered assuming the circuit is stateless. For the stateful construction, we must add additional countermeasures; clearly, if the adversary is able to know the position of the signal v_i at the end of one execution, it can directly probe v_i at the beginning of the next execution. This holds for memory cells that must be transmitted from one execution to the next; for instance, the AES key needs to be stored and passed through subsequent executions. In ISW this is has been achieved by using a t-private encoding of a random cyclic shift for each pack of ℓ wires. Such t-private encoding has complexity $\mathcal{O}(t^2)$, and since for every memory cell this cyclic shift requires a circuit of size $\mathcal{O}(\ell \log \ell)$, the additional complexity would be $\mathcal{O}(st^2 \ell \log \ell)$, which gives a complexity of $\tilde{\mathcal{O}}(st^3)$ for s memory cells. In order to get an improved complexity we proceed as follows: for each wire i from C' at the end of one execution, we perform a random permutation of the $\ell = \mathcal{O}(t)$ corresponding

wires in $\widetilde{C}$, but without processing the index location explicitly. For this, we used a sequence of $\log_2 \ell$ layers, wherein each layer the information in all wires of index j and $j + 2^m$ is randomly swapped, for $0 \leq m < \log_2 \ell$. The complexity is then $\mathcal{O}(s\ell \log \ell) = \mathcal{O}(st \log t)$, and in turn, the circuit overall complexity is $\mathcal{O}(|C| \cdot t \log t)$. In the following Table 4.01, we have summarized the time and circuit complexities. We will see in detail that, asymptotically, in the stateless model our construction improves the time complexity but not the circuit complexity; while in the stateful model, we improve both the time and circuit complexities.

		time complexity (RAM model)	circuit complexity
Stateless model	ISW, Theorem 3.11	$\mathcal{O}(\lvert C\rvert \cdot t \log t)$	$\mathcal{O}(\lvert C\rvert \cdot t \log t)$
	Theorem 4.12	$\mathcal{O}(\lvert C\rvert \cdot t)$	$\mathcal{O}(\lvert C\rvert \cdot t \log t)$
Stateful model	ISW, Theorem 3.14	$\mathcal{O}(\lvert C\rvert \cdot t \log t + s \cdot t^3 \log t)$	$\mathcal{O}(\lvert C\rvert \cdot t \log t + s \cdot t^3 \log t)$
	Theorem 4.19	$\mathcal{O}(\lvert C\rvert \cdot t + s \cdot t \log t)$	$\mathcal{O}(\lvert C\rvert \cdot t \log t)$

Table 4.01: Time and circuit complexity of our new construction vs ISW, where s is the number of memory cells that must be passed from one execution to the other.

Furthermore, we describe an AES implementation of our shuffling countermeasure, which we compare with an AES implementation of the perfectly private masking countermeasure. We are going to see that, in practice, our shuffling construction outperforms the masking countermeasure for $t \geq 6\,000$. We also provided the source code in [Cor21].

Before getting into the details of our construction, in the next section, we will explain an additional minor difference with respect to Section 3.1.4 and Section 3.1.5. In order to make the readability of the proofs easier, our construction works in a slightly different random probing model; in this customized model the leakage will not take place at the wire level, instead, it leaks at the gate level revealing, still with probability p, both input and output values. We called it the *Random Gate-Probing Model*. Nevertheless, we will show the equivalence between the two models.

4.1 Random Gate-Probing Model

As anticipated above, for proving the security of our new construction, it was more efficient to work in a slight variant of the random probing model for the intermediate circuit C', in which we assume that every gate of the circuit leaks all its information with probability p, instead of every wire. In our variant, whenever a gate leaks, all its input and output wires are leaked (see Figure 41 for an illustration); we call this variant the random gate-probing model. We also assume that the input wires in C' are also leaking with probability p; technically, this corresponds to a "copy gate" applied to each input which also is leaking with probability p.

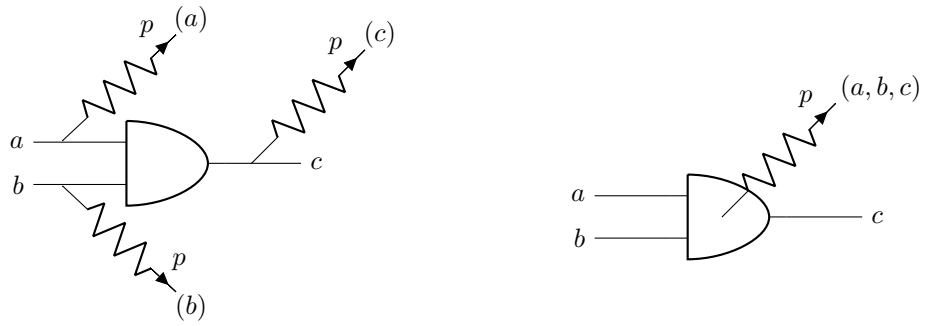

Fig. 41: Random probing model (left): each wire leaks with probability p. Random gate-probing model (right): each gate leaks with probability p, in which case its inputs and output (a, b, c) are leaked.

Given a circuit C and a set of its wires W, we define C_W as the value of the wires belonging to W; thus, the security in such new model is definable as follows:

Definition 4.1 (Random gate-probing security). *Consider a randomized circuit C' and a random sampling W of its internal wires, where each gate G_i of C' leaks with independent probability p_i. The circuit C' is said (p, ε)-random gate-probing secure if for any $(p_i)_i$ with $p_i \leq p$, there exists a simulator $\mathcal{S}_{C'}$ such that $\mathcal{S}_{C'}(W) \stackrel{id}{=} C'_W(\mathsf{Encode}(\boldsymbol{x}))$ for every plain input $\boldsymbol{x}$, except with probability at most ε over the sampling of W.*

After a deeper look, it is possible to notice that our above definition is slightly stronger than Theorem 3.6 from [ISW03]. Specifically, in Theorem 3.6 the simulator needs to produce both the random sampling W and the leaking values, whereas in Theorem 4.1 the simulator $\mathcal{S}_{C'}$ is given W as input and it must perfectly simulate the leaking values (except with probability at most ε) over the sampling of W. Namely, W is fixed by the adversary and, for any chosen set W, the simulator is outputting $|W|$ indistinguishable values. As it was for Lemma 3.7, the masking countermeasure is proven secure in the random gate-probing model via Chernoff's bound. Since the proof is essentially the same as the proof of Lemma 3.7 (see Lemma 3.1.3) is therefore omitted.

Lemma 4.2. *There exists a circuit transformer $T(C, k)$ producing a circuit C' of size $\mathcal{O}(k^2|C|)$, such that T achieves $(\Omega(1/k), \varepsilon)$-random gate-probing security, where ε is a negligible function of the security parameter k.*

4.2 Shuffling Countermeasure in Stateless Setting

In this section, we describe our new construction that achieves worst-case probing security with running time $\mathcal{O}(t)$ instead of $\mathcal{O}(t \cdot \log t)$ in [ISW03]. For the moment, let's consider stateless

circuits only; we will consider stateful circuits in the next Section 4.3. We are going to explain the construction (Section 4.2.1) and the security proof (Section 4.2.2); later we will propose an improvement for the time complexity with respect to the constant k factor (Section 4.2.3); and lastly, we will provide the results in the pure circuit model (Section 4.2.4).

4.2.1 Description

Our construction proceeds in two steps: first we transform the original circuit C into an intermediate circuit $C' = T(C, k)$ with $n = 2k+1$ shares, Theorem 3.22 from Section 3.4. Then we transformed the circuit C' into an expanded circuit $\widetilde{C}$ as follows.

Wires. For each wire i in C' we consider a set of ℓ wires in $\tilde{C}$, labelled $(i, 0), \ldots, (i, \ell-1)$ and an index j_i. Let $a_0, \ldots, a_{\ell-1}$ be the value of the ℓ wires. The circuit $\widetilde{C}$ maintains the invariant that if wire i in C' has value v, then this value appears at position j_i in $\widetilde{C}$, that is $a_{j_i} = v$, while the value of the over wires is arbitrary.

Encoding and decoding. The usage of the explicit shuffling index required a redefinition of the encoding and decoding algorithms. We slightly modified the Encode' and Decode' of the intermediate circuit C' as follow:

- Encode'. To encode a value v, first generate at random an index $j \xleftarrow{\$} [0, \ell-1]$ and output the encoding $(j, (0, \ldots, 0, v, 0, \ldots, 0))$, where v is at the j-th position.
- Decode'. Given $(j, (a_0, \ldots, a_j, \ldots, a_{\ell-1}))$, return a_j.

A positive consequential characteristic of the explicit computation of the positional index is that we don't need the dummy element \$ for the $\ell - 1$ other wires, this allowed us to save a flag bit at the implementation level, a bit that would have been necessary to recognize the dummy value; in fact, as shown above, at the encoding phase we could simply assign them to 0.

Algorithm 4.1 Gate $*$ processing

Input: Encodings $(j, (a_0, a_1, \ldots, a_{\ell-1}))$ and $(j', (b_0, b_1, \ldots, b_{\ell-1}))$
Output: Index j'' and array $(c_0, c_1, \ldots, c_{\ell-1})$ such that $c_{j''} = a_j * b_{j'}$

1: $j'' \leftarrow_{\$} [0, \ell)$
2: $\Delta = j'' - j$, $\Delta' = j'' - j'$
3: For all $0 \le i < \ell$, let $a'_i \leftarrow a_{i-\Delta}$ and $b'_i \leftarrow b_{i-\Delta'}$ ▷ $a'_{j''} = a_j$, $b'_{j''} = b_{j'}$.
4: For all $0 \le i < \ell$, let $c_i \leftarrow a'_i * b'_i$ ▷ $c_{j''} = a_j * b_{j'}$.
5: **return** $(j'', (c_0, c_1, \ldots, c_{\ell-1}))$

Gates. Consider a gate G in the intermediate circuit C', taking as input a and b, and outputting $c = a * b$ where $* \in \{\mathsf{XOR}, \mathsf{AND}\}$. Moreover, let $(a_i)_{0 \le i < \ell}$ and $(b_i)_{0 \le i < \ell}$ be the corresponding input wires for the expanded gadget $\widetilde{G}$ in $\widetilde{C}$, and let j and j' be the corresponding

indexes, with $a_j = a$ and $b_{j'} = b$. The processing of the gadget $\widetilde{G}$ proceeds generating a random index $j'' \leftarrow [0, \ell - 1]$, and making the sparse-shares $(a_i)_{0 \le i < \ell}$ and $(b_i)_{0 \le i < \ell}$ pass through a CyclicShift, where the cyclic shifts are computed based on $\Delta = j'' - j$ and $\Delta' = j'' - j'$. This would produce a change in the position of the ℓ-array (a_i) by $j'' - j$ positions modulo ℓ and, similarly, of the ℓ-array (b_i) by $j'' - j' \mod \ell$ positions. In this way the input signals a and b would then be located at common position j''; in turn, the gadget would apply the $*$ operation to each position $0 \le i \le \ell - 1$ independently. Clearly, the j''-th output would carry the original signal $c = a * b$ as it would have been computed in G belonging to C'. We provide a formal description in Algorithm 4.1 above, where all indices computations are performed modulo ℓ; see also Figure 42 for an illustration.

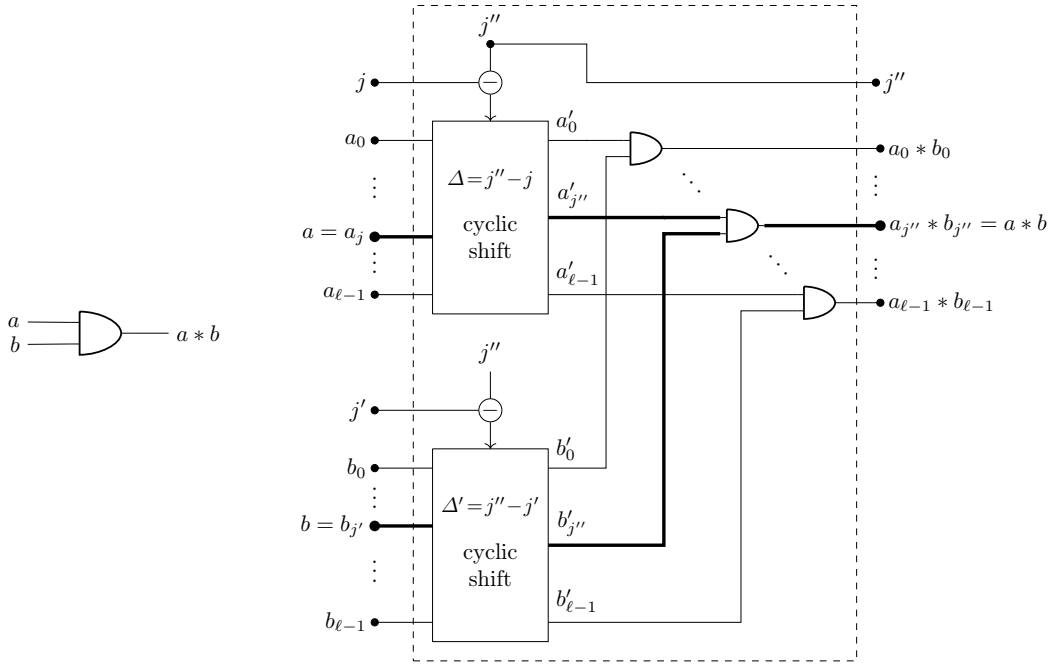

Fig. 42: Original gate in C' (left) and shuffling gadget in $\tilde{C}$ (right). The bold wires contain the original signal value from C'; the other wires contain only dummy values.

Exploiting the visual representation of Figure 42 above, we would immediately address the maintained soundness property of our new gadget since it is a part of the security definition. This implies that in the next section we will provide just privacy proof, claiming that is enough to achieve Definition 3.8. We claim the following:

Theorem 4.3. *The transformed gadget $\widetilde{G}$ defined above achieves the soundness property, according to Definition 3.8.*

Proof. The intermediate circuit $C' = T(C, k)$ computes the same function as C. Moreover, every expanded gate in $\tilde{C}$ computes the same gate as C'. Namely, consider the gate $c = a * b$ in C'. In the final circuit $\tilde{C}$ we have $c_{j''} = a'_{j''} * b'_{j''} = a_{j''-\Delta} * b_{j''-\Delta'} = a_j * b_{j'} = a * b = c$ as required. Therefore $\tilde{C} = \tilde{T}(C, k)$ computes the same function as C.

□

Remark 4.4. Note that Algorithm 4.1 can be implemented via a table look-up. Namely, the ℓ wires can be stored in an array $T[i]$ for $0 \leq i < \ell$, with $T[j] = v$ for the signal index j, and the cyclic shift is performed as in Step 3, that is with the loop $T'[i] \leftarrow T[i - \Delta \bmod \ell]$ for each $0 \leq i < \ell$. Moreover, the running time of Algorithm 4.1 is $\mathcal{O}(\ell)$ per gadget.

We just explained how to compute both AND and XOR gates; to complete the description of $\widetilde{T}$, it remains to describe the expansion of a random gate in C'. It is managed similarly to in the original circuit C: with a gate taking no input and outputting a random bit (see Section 2.1). Such gate would be expanded into a gadget outputting $(j, (r_0, \ldots, r_{\ell-1}))$ with $r_i \leftarrow \{0, 1\}$ for all $0 \leq i < \ell$ and $j \leftarrow [0, \ell)$. This concludes the description of every type of gate in the process of transforming C' into $\widetilde{C}$.

4.2.2 Shuffling Security and Composition

As in [ISW03], our goal is to show that the adversary does not learn more from the worst-case probing of the final circuit $\widetilde{C}$ than what it would have learnt from the p-random (gate)-probing of the intermediate circuit C'. For this we proceed with a similar compositional approach showed in [BBD+16]: 1) we introduced a new security definition for a single gadget in the expanded circuit $\widetilde{C}$, 2) we proved that our shuffling gadget from the previous section satisfies such definition, and 3) we showed how to get security for the full circuit $\widetilde{C}$ by composition.

Gadget. The main benefit of this approach is that 3) only depends on 1), therefore we could later modify the shuffling gadget and still get security for the full circuit, as long as the shuffling gadget satisfies the security definition. We called it *ℓ-shuffling security* and it is defined as follows:

Definition 4.5 (Shuffling security). *We say that a randomized gadget $\widetilde{G}$ achieves ℓ-shuffling security if any set of t probes, excluding the input wires of the gadget, can be perfectly simulated from scratch, except with probability at most t/ℓ, where the probability is taken over the randomness used by the gadget.*

As one may notice the input wires of the gadget are excluded from the simulation. This choice has been taken to simplify the proofs and reasoning on the fact that when it will be used in the composition of the full circuit such wires have a duality essence; namely, in the composition, the selected among the input wires of a gadget can be handled by the simulation of the output wires of the previous gadget, with the only exception of the input wires of circuit $\widetilde{C}$, which

we will see how to handle separately. In sum, this assumption will improve the readability of the proof and will not impact the security anyway. Given this needed digression, we can now proceed with phase 2) mentioned at the beginning of the Section: proving that the construction from Section 4.2.1 achieves our ℓ-shuffling security.

Lemma 4.6. *The gadget $\tilde{G}$ as described in Algorithm 4.1 is ℓ-shuffling secure.*

Proof. We must construct a simulator that can simulate any set of t probes, with failure probability at most t/ℓ. In the simulation the input indices j, j' are fixed, as well as the input arrays $(a_0, \ldots, a_{\ell-1})$ and $(b_0, \ldots, b_{\ell-1})$. From the definition the adversary cannot probe those input arrays; in particular, the adversary cannot probe the signal $a = a_j$ and $b = b_{j'}$. Moreover, the adversary must commit to the position of the probes before the execution of the gadget. Therefore in the simulation, the probes have fixed positions while the index j'' is randomly and uniformly distributed in $[0, \ell)$. This implies that for a fixed $i \in [0, \ell)$, the variable a'_i contains the secret value a with probability at most $1/\ell$; the same holds for the variables b'_i and c_i. Consequently, the t probes can be perfectly simulated, except with probability at most t/ℓ. □

Composition. We now prove the worst-case statistical t-privacy of the final circuit $\tilde{C}$, from the p-random gate-probing security of the intermediate circuit C'; see Figure 43 for an illustration.

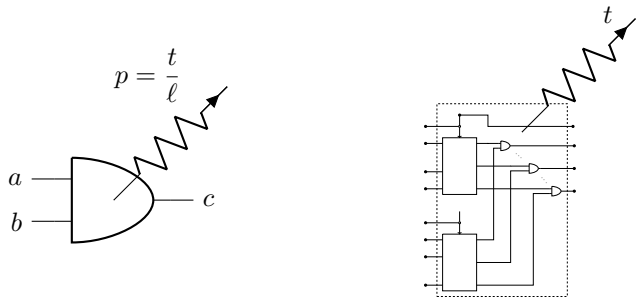

Fig. 43: A set of t probes in a shuffling gadget in $\tilde{C}$ (right) correspond to a gate-leaking probability at most $p = t/\ell$ in C' (left).

Namely, we claim that the existence of an adversary able to break the t-statistical privacy of our transformed $\widetilde{C}$ circuit would result in an attacker able to break the p-random gate-probing security of C'; a statement that, in turn, would jeopardise the veracity of Lemma4.2 (which is a contradiction). Thus, let's state the following Lemma:

Lemma 4.7. *Suppose that C' is (p, ε)-random gate probing secure. Then, the circuit $\tilde{C}'$ constructed as described above with $\ell := t/p$ achieves ε-statistical security in the worst case against t probes.*

Proof. We must construct a simulator $\tilde{\mathcal{S}}$ for the t probes of the final circuit $\widetilde{C}$, using the already proved simulator $\mathcal{S}'$ for the gate average-case security of the intermediate circuit C'. According to Lemma 4.2, it exists a simulator $\mathcal{S}'$ that receives as input a random sampling W of the gates, where each gate leaks with independent probability $p_i \leq p$, and $\mathcal{S}'$ is able to provide a perfect simulation of the gates in W, except with probability at most ε over the sampling W.

Therefore, our simulator $\tilde{\mathcal{S}}$ can proceed as follows. First, it receives from the adversary the set of probed wires belonging to the final circuit $\widetilde{C}$. Then, it generates randomly every index position of circuit $\widetilde{C}$, like they would have been computed in a real execution. Thus, as illustrated in Figure 43, a set of t_i probes in a shuffling gadget $\widetilde{G}_i$ of $\widetilde{C}$ corresponds to a leaking probability $p_i = t_i/\ell$ in the corresponding gate G_i of C'. More precisely, according to Lemma 4.6, each gadget $\widetilde{G}_i$ can be perfectly simulated from scratch, except with probability at most t_i/ℓ. Therefore, our simulation for the full circuit $\widetilde{C}$ will first run the simulators $\mathcal{S}_i$ for all gadgets independently; when any such simulator $\mathcal{S}_i$ fails (this happens with probability at most $p_i = t_i/\ell \leq t/\ell = p$), we include the corresponding gate G_i in the sampling W.

Moreover, from Definition 4.5 and Lemma 4.6, the failure probabilities are independent, because in Definition 4.5 the failure probability is taken over the gadget randomness, and such randomness in each gadget is generated independently. Consequently, the gates G_i is included in W with independent probability $p_i \leq p$ as required in Definition 4.1. Eventually, our simulator $\tilde{\mathcal{S}}$ runs the intermediate circuit simulator $\mathcal{S}'$ over the sampling W, and receives a simulation of the input wires of the corresponding gates $G_i \in W$. Those simulated values are, now, used to perfectly simulate the t_i probes within each expanded gadget $\widetilde{G}_i$, for the case when $\mathcal{S}_i$ has failed. Since the simulator $\mathcal{S}'$ from the intermediate circuit C' provides a perfect simulation except with probability ε over the sampling W, our final simulator $\tilde{\mathcal{S}}$ provides a perfect simulation of the t probes for $\widetilde{C}$ except with probability ε. This proves the security for the whole circuit, since, by definition, ε is negligible.

□

Furthermore, our construction has better running time $\mathcal{O}(|C| \cdot t)$ with respect to the ISW construction; while it maintains the same circuit complexity $\mathcal{O}(|C| \cdot t \cdot \log t)$. From Lemma 4.3 and Lemma 4.7, we could claim the existence of a faster statistical t-private transformer.

Theorem 4.8. *There exists a statistically t-private stateless transformer (T, I, O), such that $T(C, k)$ transforming a circuit C into a circuit $\widetilde{C}$ of running time $\mathcal{O}(|C| \cdot k^3 \cdot t)$ and size $\mathcal{O}(|C| \cdot k^3 \cdot t \cdot (\log k + \log t))$. The depth of $\widetilde{C}$ is the same as that of C, up to* polylog *factors.*

Proof. Lemma 4.3 proved the soundness of $\widetilde{C}$, while Lemma Lemma 4.7 ensured its privacy. This is enough to achieve statistical privacy in the stateless worst-case model. The intermediate circuit C' has size $\mathcal{O}(|C|k^2)$, while the final circuit has running time $\mathcal{O}(|C|k^2\ell)$ and size

$\mathcal{O}(|C|k^2\ell\log\ell)$. Fixing $p = t/\ell$ and $p = \Omega(1/k)$ to achieve gate average-case privacy for C', we get running time $\mathcal{O}(|C| \cdot k^3 \cdot t)$ and size $\mathcal{O}(|C| \cdot k^3 \cdot t \cdot (\log k + \log t))$.

□

4.2.3 Improved Time-Complexity

From the proof of Lemma 4.7 the circuit $\widetilde{C}$ is actually secure in the region probing model, where the adversary can put t probes per gadget in $\widetilde{C}$. We can, however, further optimize the circuit complexity, with respect to the security parameter k, if we only require security against a total of t probes in the full circuit $\widetilde{C}$. Namely, in this case, it is possible to consider each gadget of $\widetilde{C}$ having t_i probes with the condition that $\sum_i t_i \leq t$, instead of $t_i \leq t$ for all i in the proof of Lemma 4.7. This means that for each corresponding gate in the intermediate circuit C', we can consider a leakage probability $p_i = t_i/\ell$ such that $\sum_i p_i \leq \mu$ over the full circuit C', with $\mu = t/\ell$. Note that, in this way, we lose the better leakage ratio considered in Definition 4.1 (where we required $p_i \leq p = t/\ell$ for all gates). In particular, if we take $t/\ell > 1$, we can tolerate a leakage probability $p_i = 1$ for a fraction of the gates in C', as long as $\sum_i p_i \leq t/\ell$ over all gates of C'; see Figure 44 for an illustration.

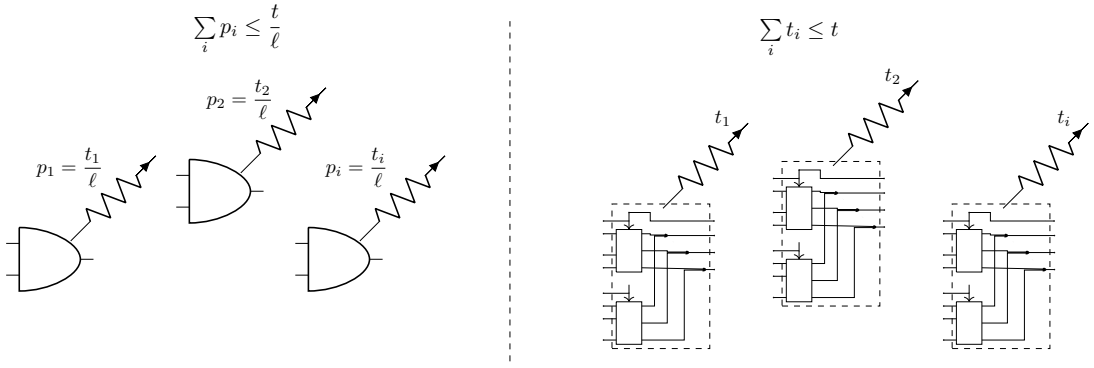

Fig. 44: A total of t probes in the final circuit $\tilde{C}$ (right) corresponds to a total of leaking probabilities at most t/ℓ in the intermediate circuit C' (left).

Nevertheless, we may handle this weaker condition by modifying the definition of random gate probing security as follows:

Definition 4.9 (Random Σ-gate-probing security). *Consider a randomized circuit C' and a random sampling W of its internal wires, where each gate G_i of C' leaks with independent probability p_i. The circuit C' is said (μ, ε)-random Σ-gate-probing secure if for any $(p_i)_i$ with $\sum_i p_i \leq \mu$, there exists a simulator $\mathcal{S}_{C'}$ such that $\mathcal{S}_{C'}(W) \overset{id}{=} C'_W(\mathsf{Encode}(\boldsymbol{x}))$ for every plain input $\boldsymbol{x}$, except with probability at most ε over the sampling of W.*

Note that here $\sum_i p_i$ is the average number of leaking gates in the circuit C'. Due to the condition $\sum_i p_i \leq \mu$, when applying Chernoff's bound on the intermediate circuit $C' = T(C, k)$ secure against k probes, we could prove random Σ-gate-probing security with a much-improved $\mu = \Omega(k)$, instead of $\Omega(1/k)$. We would like to stress that this newly considered trade-off in the balance leakage ratio vs threshold has to be done with the scope of improving the practical implementation results (see Section 4.4). The following Lemma provides concrete values for $\mu(k)$ and $\varepsilon(k)$ for the intermediate circuit C' based on the masking countermeasure:

Lemma 4.10. *There exists a circuit transformer $T(C, k)$ producing a circuit C' of size $\mathcal{O}(k^2|C|)$, such that T achieves (μ, ε)-random Σ-gate-probing security for $\mu = \Omega(k)$ and ε a negligible function of the security parameter k. In particular, one can take $\mu = k/4$ and $\varepsilon = 2^{-k/(12 \log 2)}$.*

Proof. Let $X_1, \ldots, X_m$ be independent Bernoulli random variables, where $X_i = 1$ if the i-th gate of C' is probed, and $X_i = 0$ otherwise. We assume that $\sum_i \Pr[X_i] \leq \mu$ with $\mu = k/4$. Let $X = X_1 + \ldots + X_m$ be the total number of leaking gates in circuit C'. The circuit $C' = T(C, k)$ is perfectly secure against an adversary probing at most k variables; hence it is perfectly secure against the leakage of $k/2$ gates; namely, a gate can be perfectly simulated from the knowledge of its two inputs. Therefore we can provide a perfect simulation of the probes if $X \leq k/2$. To compute the simulation failure probability we apply the Chernoff bound for $\delta \geq 1$ (see **??**):

$$\Pr[X \geq (1 + \delta) \cdot \mathbb{E}[X]] \leq \exp\left(-\frac{\delta}{3}\mathbb{E}[X]\right)$$

We fix δ such that $(1 + \delta)\mathbb{E}[X] = 2\mu$. Since $\mathbb{E}[X] = \sum_i \Pr[X_i] \leq \mu$, we have $\delta \geq 1$ as required. We obtain:

$$\frac{\delta}{3}\mathbb{E}[X] = \frac{\delta}{3} \cdot \frac{2\mu}{1 + \delta} \geq \mu/3$$

Finally, using $\mu = k/4$, we obtain:

$$\Pr[X \geq k/2] \leq \exp(-k/12)$$

The simulation failure probability is therefore upper-bounded by $\varepsilon = \exp(-k/12) = 2^{-k/(12 \log 2)}$. □

Note that the above circuit C' does not need to be secure in the region probing model with k probes per region; namely in the proof of Lemma 4.10 only the total number of probes matters. Therefore, as a further improvement, the scheme only requires $n = k+1$ shares with appropriate mask refreshing as in [BBD+16] (instead of $n = 2k + 1$ used in our previous construction). Eventually, we obtain a statistically t-private stateless transformer with complexity $\mathcal{O}(|C| \cdot k \cdot t)$, instead of $\mathcal{O}(|C| \cdot k^3 \cdot t)$ in Theorem 4.8. The proof would follow as previously: firstly proving the worst-case security of $\widetilde{C}$ from the average-case security of C'.

Lemma 4.11. *Suppose that C' is (μ, ε)-random Σ-gate-probing secure. Then, the circuit $\widetilde{C}$ constructed as described above with $\ell := t/\mu$ achieves ε-statistical security in the worst case against t probes.*

Proof. The proof is essentially the same as the proof of Lemma 4.7. Instead of having each gadget $\widetilde{G}_i$ simulator $\mathcal{S}_i$ fail with probability $p_i \leq p$, the failure probabilities are still independent but with the looser condition $\sum_i p_i \leq \mu = t/\ell$. This gives a sampling W of the gates G_i in C' with the same condition $\sum_i p_i \leq \mu$. Since C' is (μ, ε)-random Σ-gate-probing secure, this implies that $\widetilde{C}$ achieves ε-statistical security in the worst case against t probes.

□

Theorem 4.12. *There exists a statistically t-private stateless transformer (T, I, O), such that $T(C, k)$ transforms a circuit C into a circuit $\widetilde{C}$ of running time $\mathcal{O}(|C| \cdot k \cdot t)$.*

Proof. From Lemma 4.11, the circuit $\widetilde{C}$ achieves worst-case statistical t privacy in the stateless model. Its running time is $\mathcal{O}(|C| \cdot k^2 \cdot \ell)$. With $\ell = t/\mu$ and $\mu = k/4$, the running time is $\mathcal{O}(|C| \cdot k \cdot t)$.

□

4.2.4 Shuffling for Pure Circuit Model

Before passing to the stateful model, for the sake of completeness, we consider the pure circuit model; specifically, here is not allowed the use of RAM, as explained in Section 3.5. The construction described in Section 4.2.1 has been implemented using table look-ups and it has been proved secure in the software probing model, where the adversary can only probe the input address and input/output value of a RAM cell, but not the content of the internal wires of the circuit implementation of the RAM. However, it is easy to obtain a pure circuit implementation of the construction, using a circuit implementation of a cyclic shift, with complexity $\mathcal{O}(\ell \log \ell)$; that is maintaining the original ISW complexity. As a major consequence, the adversary is now allowed to probe also the wires within the cyclic shift. Therefore we use the same approach as in [ISW03, Lemma 2], see Section 3.1.4, with $\ell = \mathcal{O}(t/p^7)$ instead of $\ell = \mathcal{O}(t/p^4)$. Thus, since our expanded gate has complexity $\mathcal{O}(\ell \log \ell)$ instead of $\mathcal{O}(k \cdot \ell \log \ell)$ in [ISW03], the final complexity will be $\mathcal{O}(|C| \cdot k^9 \cdot t \cdot (\log k + \log t))$ instead of $\mathcal{O}(|C| \cdot k^{10} \cdot t \cdot (\log k + \log t))$. This is sufficient to prove the following:

Theorem 4.13. *There exists a statistically t-private stateless transformer (T, I, O), such that $T(C, k)$ transforms a circuit C to a circuit $\widetilde{C}$ of size $\mathcal{O}(|C| \cdot t \cdot \log t)$. The depth of $\widetilde{C}$ is the same as that of C, up to* polylog *factors.*

After this section we will come back to the RAM model, namely, the constructions we are going to see for the stateful settings use RAM implementation of cyclic shifts as well; consequently, we anticipate now, that similarly, we obtained the same result in the pure circuit model for the stateful settings using the randomizing network construction from Section 4.3.2.

Theorem 4.14. *There exists a statistically t-private stateful transformer T, such that $T(C, k)$ maps a circuit C with s memory cells to a circuit $\widetilde{C}$ with complexity $\mathcal{O}(|C| \cdot t \cdot \log t)$. The depth of $\widetilde{C}$ is the same as that of C, up to* polylog *factors.*

4.3 Shuffling Countermeasure in Stateful Setting

We finally discuss our countermeasure(s) in the more useful and practical stateful model. We propose two constructions in the worst-case statistical stateful model that achieve a better complexity bound than the ISW construction recalled in Section 3.1.5. Before that let us briefly recall that in our stateless construction from Section 4.2, the position $j \in [0, \ell - 1]$ of the signal v_i among the ℓ wires is explicitly computed; it is therefore assumed that it is known to the adversary at the end of a given execution. For the stateful model, this means that without any additional countermeasure, the adversary could directly probe the signal v_i at the beginning of the next execution; this holds for the hidden state that must be transmitted from one execution to the other (see Figure 45 for an illustration).

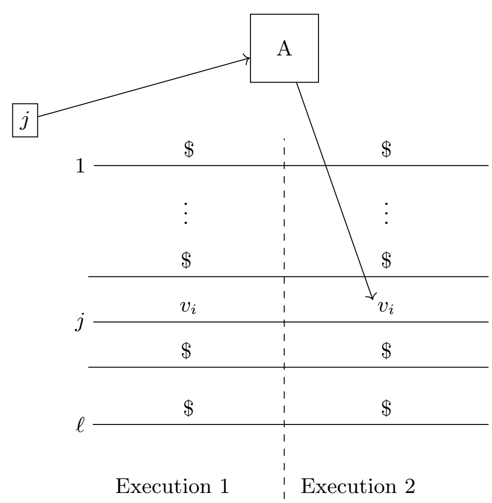

Fig. 45: Without additional countermeasure, the adversary learning the index position j at the end of a given execution can directly probe the signal v_i at the beginning of the next.

Because of that, the adaptive case of the stateful model requires a stronger guarantee compared to Theorem 4.5. We had to extend the ℓ-shuffling security by ensuring that even after having observed t probes in the gadget, any set O of output probes can be simulated from scratch, except with probability at most $2t/\ell$. Note that the construction in Figure 42 from the stateless case cannot satisfy this definition, since the adversary could directly probe the output position j'' of the signal. Thus, we introduce what we called *strong* ℓ-shuffling security:

Definition 4.15 (Strong ℓ-shuffling security). *We say that a randomized gate G achieves strong ℓ-shuffling security if any set S of t probes (excluding the input wires) and any set O of output probes, can be perfectly simulated from scratch, except with probability at most $2t/\ell$, where the set O is chosen adaptively from the value of the probes in S.*

It is interesting to notice that in the original ISW construction for the statistical privacy in the context of stateful circuits, the authors used a perfectly t-private random cyclic shift; and such a countermeasure indeed satisfies our Definition 4.15. Since from the t-privacy the adversary gets no information about the position of the output signal, and therefore for a total of $2t$ probes the probability to recover the signal is at most $2t/\ell$; the complexity of the ISW construction for a single gadget is $\mathcal{O}(\ell t^2)$.

As mentioned above, we actually propose two solutions which improve the original time complexity: the first will exploit several consecutive classical (i.e not masked) cyclic shifts, the intent was to use a number of cyclic shifts greater than the threshold, so that the adversary can not probe all. The second (and more efficient) will use a particular randomizing network on the ℓ-tuple of output wires. The *Iterated Cyclic Shift* countermeasure saves a linear factor, while the *Randomizing Network* a square one, both compared to the $\widetilde{\mathcal{O}}(|C| \cdot t \log t + s \cdot t^3 \log t)$ complexity of [ISW03]. Additionally, both constructions, indeed, achieve strong ℓ-shuffling security. To complete the section, we are going to show that any construction satisfying Definition 4.15 enables obtaining worst-case statistical security in the stateful model.

4.3.1 Iterated Cyclic Shifts

Our first construction consists of a sequence of $t+1$ random cyclic shifts with uniformly and independently distributed shifts $\Delta_1, \ldots, \Delta_{t+1} \leftarrow [0, \ell - 1]$, see Figure 46 for an illustration.

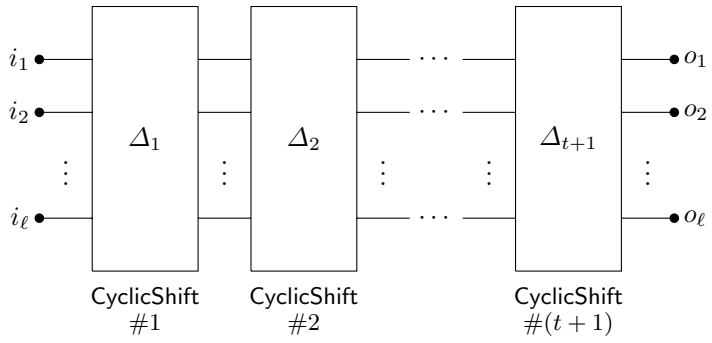

Fig. 46: First construction: sequence of $t+1$ random cyclic shifts.

As in [ISW03], the construction is used as output for every hidden state bit that must be transmitted from one execution to the next. While, as opposed to our stateless construction,

the index position of the signal is not explicitly computed in the cyclic shifts; the position of the signal is only implicitly determined by the value of the wires in $\{0, 1, \$\}$, where \$ is the dummy value (technically it is a 0). At the end of a given execution, the gadget must, therefore, convert from a representation with explicit signal index j to the representation with wire values in $\{0, 1, \$\}$; this can be done with complexity $\mathcal{O}(\ell)$. Then, the index position of the signal is computed again explicitly as the beginning of the next execution. The construction satisfies the strong ℓ-shuffling security, since with at most t internal probes, one of the $t + 1$ random cyclic shifts is necessarily not probed, and, consequently, the adversary does not get any information about the position of the output signal. As anticipated, the overall cost of this first construction is $\widetilde{\mathcal{O}}(\ell t)$, instead of $\widetilde{\mathcal{O}}(\ell t^2)$ for the ISW construction with the perfectly t-private random cyclic shift.

Lemma 4.16. *The gadget described above is strong ℓ-shuffling secure.*

Proof. We consider the following sequence of two security games:

Game_1: we follow Definition 4.15, in which the adversary first chooses t probes (excluding the input wires), obtains their value, and then chooses another t probes among the output wires of the gadget. The adversary wins the game if he observes the signal value x. We denote by S_1 the event that the value x is revealed by the first t probes, and by T_1 the event that the value x is revealed by the t output wires chosen by the adversary. It holds that $\Pr[S_1] \leq t/\ell$.

Game_2: we proceed as in Game_1 above, but now we modify the way the output of the first t probes is generated, in that we never reveal the value x; instead we always leak the dummy value \$. Let T_2 be the event in Game_2 that the value x is revealed by the t output wires. We see that events T_1 and T_2 are identical unless one of the first t probes would reveal x. This means $T_1 \wedge \neg S_1 = T_2 \wedge \neg S_1$.

We claim that in Game_2 the adversary gets no information about the index position of the signal in the output variables. Namely, from the first set of t probes the adversary can only receive the dummy value \$; moreover, the adversary can only probe at most t of the $t + 1$ shifting index Δ_i of the cyclic shifts. Since at least one of those $t + 1$ random cyclic shifts has not been probed, the distribution of the signal position after this cyclic shift is independent of the adversary's view and uniform in $[0, \ell)$; this also holds for the signal position in the output variables. This implies $\Pr[T_2] \leq t/\ell$. Eventually, we obtain:

$$\Pr[S_1 \vee T_1] = \Pr[S_1] + \Pr[T_1 \wedge \neg S_1] = \Pr[S_1] + \Pr[T_2 \wedge \neg S_1] \leq \Pr[S_1] + \Pr[T_2] \leq 2t/\ell$$

Therefore, the $2t$ probes can be simulated except with probability at most $2t/\ell$.

□

4.3.2 Randomizing Network

Instead, our second construction consists of a network of $\log_2 \ell$ layers, wherein the m-th layer for $0 \leq m < \log_2 \ell$ the information in all wires of index i and $i+2^m$ is swapped with independent

probability 1/2; see Figure 47 for an illustration, where for simplicity we assume that ℓ is a power of 2. Letting $j' \in \{0, \ldots, \ell-1\}$ be the index position of the signal before the randomizing network, at layer m the m-th bit of the signal position is therefore randomly flipped. Since this is done for all layers $0 \leq m < \log_2 \ell$, at the end the output index of the signal is randomly distributed in $\{0, \ldots, \ell - 1\}$.

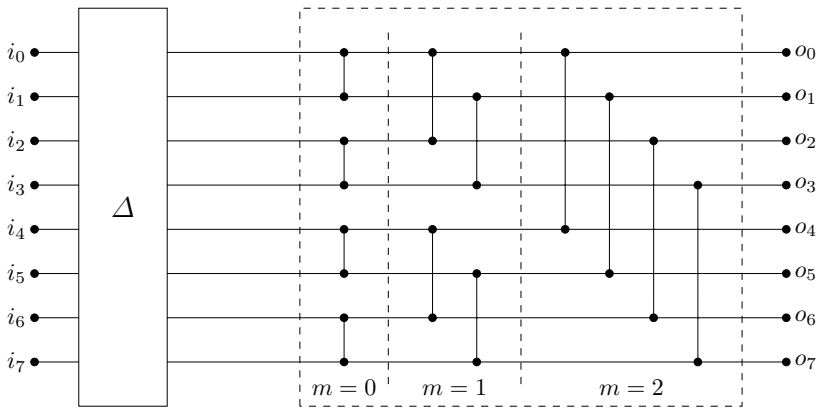

Fig. 47: Second construction: random cyclic shift and randomizing network for $\ell = 8$.

As in the previous construction, the index position j' is only known as input and not computed explicitly during the swaps: the position of the signal is only implicitly determined by the value of the wires in $\{0, 1, \$\}$; the index position of the signal is computed again explicitly as the beginning of the next execution; see Figure 48 for an illustration. For a single gadget, our second construction has complexity $\widetilde{\mathcal{O}}(\ell)$, instead of $\widetilde{\mathcal{O}}(\ell t)$ in our first construction and $\widetilde{\mathcal{O}}(\ell \cdot t^2)$ in [ISW03]. Moreover, our second construction has depth polylogarithmic in t, instead of linear in t in our first construction.

Finally, to satisfy the strong ℓ-shuffling security (Definition 4.15), we must prepend a random cyclic shift; otherwise, since, by definition, the input index j is fixed, the adversary could directly probe the j-th wire after the first layer, and learn the signal with probability 1/2. Thus, we could state and prove the following:

Lemma 4.17. *The gadget described above is strong ℓ-shuffling secure, with circuit complexity $\mathcal{O}(\ell \cdot \log \ell)$.*

Proof. The proof is essentially the same as the proof of Lemma 4.16 from the previous section. We assume that when the adversary probes a swap, it learns the two input wires of the swap, and additionally a bit b corresponding to whether a swap of the wire values occurred or not. We consider two security games Game_1 and Game_2. In Game_1 we follow Definition 4.15; while, as previously, in Game_2 we always leak the dummy value \$; moreover, when a swap is probed, we replace the bit b with an independently generated random bit. As previously, we claim

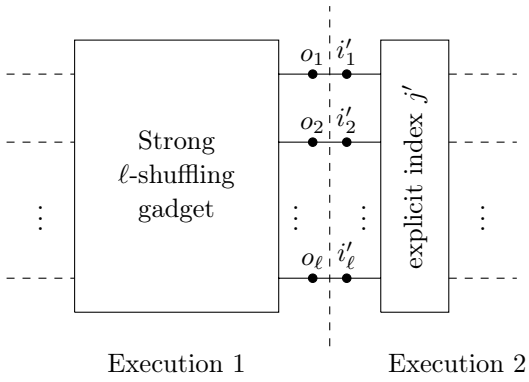

Fig. 48: Thanks to a strong ℓ-shuffling gadget, the adversary does not get information about the index position of the signal at the end of an execution.

that in Game_2 the adversary learns nothing about the position of the signal among the output wires. Denoting by S_1 the event in Game_1 that the value x is revealed by the first t probes, we have $\Pr[S_1] \leq 2t/\ell$, since the adversary can now learn 2 wires at a time. Denoting by T_1 and T_2 the same events as in the proof of Lemma 4.16, we obtain as previously:

$$\Pr[S_1 \vee T_1] \leq \Pr[S_1] + \Pr[T_2] \leq 2t/\ell + t/\ell = 3t/\ell$$

Therefore the $2t$ probes can be simulated except with probability at most $3t/\ell$. To satisfy Definition 4.15 we can therefore use $3\ell/2$ wires instead of 2ℓ.

□

4.3.3 Composition in the statistical stateful model

As in the stateless case, we showed that the worst-case statistical privacy of $\widetilde{C}$ in the stateful model follows from the p-random gate-probing security of C', based on the ℓ-shuffling security (Definition 4.5) and strong ℓ-shuffling security of the gadgets.

Lemma 4.18. *Suppose that C' is (p, ε)-random gate probing secure. Then, the circuit $\tilde{C}$ constructed as described above with $\ell := 3t/(2p)$ achieves stateful ε-statistical security in the worst case against t probes per execution.*

Proof. The proof is essentially the same as the proof for the stateless case (see Lemma 4.7 in Section 4.2.2). But, for the stateful case, we first obtain a p-random secure intermediate circuit C' from Theorem 3.5. The proof then proceeds as in the stateless case. In a given execution the adversary probes in $\widetilde{C}$ generate a random sampling W of the gates in C', where from Definition 4.5 each gate leaks with probability at most p. Moreover, Definition 4.15 implies that the input wires for the next execution of C' also leak with probability at most p. Then,

from the p-random probing security of C', this implies worst-case t-privacy in the stateful model of the final circuit $\widetilde{C}$.

□

Eventually, due to the randomizing network construction, which has complexity $\mathcal{O}(\ell \log \ell)$, the complexity of the final circuit $\widetilde{C}$ is $\mathcal{O}(|C| \cdot t \log t)$ instead of $\mathcal{O}(|C| t \log t + s \cdot t^3)$ in [ISW03]. Therefore as opposed to ISW our construction has quasi-linear complexity even for a large number s of memory cells. Recall we already provided the same result for the pure circuit model in Section 4.2.4.

Theorem 4.19. *There exists a statistically t-private stateful transformer T, such that $T(C, k)$ maps a circuit C with s memory cells to a circuit $\widetilde{C}$ with complexity $\mathcal{O}(|C| \cdot t \cdot \log t)$. The depth of $\widetilde{C}$ is the same as that of C, up to* polylog *factors.*

Proof. From Lemma 4.18, circuit $\widetilde{C}$ achieves statistical privacy in the stateful worst-case model, with circuit complexity $\mathcal{O}(|C| \cdot \ell \log \ell)$. With $\ell = \mathcal{O}(t)$, the circuit complexity is finally $\mathcal{O}(|C| \cdot t \cdot \log t)$.

□

4.4 Implementation

As a final part of this Chapter and our work at [CS21], we provide some details about the prototype of our stateless construction from Section 4.2.1. For performance reasons, as explained in Section 4.2.3, our implementation took into consideration the non-regional model.

Security parameters. Recall that the construction proceeds in two steps. Starting from the original circuit C, we first constructed an intermediate circuit C' based on the classical masking countermeasure with perfect security against k probes, where k is the security parameter. From Chernoff bound, the intermediate circuit C' is also secure in the random gate-probing model; more precisely, according to Lemma 4.10, the circuit C' achieves the (μ, ε)-random Σ-gate-probing security with parameters $\mu(k) = k/4$ and $\varepsilon(k) = 2^{-k/(12 \log 2)}$. Here $\mu(k) = k/4$ denotes the number of gates that are probed on average, and $\varepsilon(k) = 2^{-k/(12 \log 2)}$ the probability of simulation failure (for the unlucky case when the number of leaking gates in C' is too large). Therefore, to get $\varepsilon = 2^{-80}$ security, we must fix $k = 668$. For the intermediate circuit C' we use $n = k + 1$ shares with appropriate mask refreshing, as in [BBD+16]. In the second step, every wire from the intermediate circuit C' must be expanded into ℓ wires in the final circuit $\tilde{C}$, where according to Lemma 4.11 it is required that $\ell = \lceil t/\mu \rceil = \lceil 4t/k \rceil$ to get security against t probes. This implies that security against $t = k \cdot \ell/4$ probes as a function of ℓ is achieved. Furthermore, for $\varepsilon = 2^{-80}$ and $k = 668$, this gives security against $t = 167 \cdot \ell$ probes as a function of the parameter ℓ[1]. Since the running time of our construction is $\mathcal{O}(\ell)$, with those

[1] We see that it would not make sense to use $\ell \leq 4$, since the intermediate circuit C' already provides perfect security against $k = 668$ probes.

parameters we can claim it is equal to $\mathcal{O}(t)$ for security against t probes, instead of $\mathcal{O}(t^2)$ for the masking countermeasure.

Number of operations. We also provide a comparison among the concrete number of operations between the masking countermeasure and our construction. For simplicity, we consider a single AND gadget. From Algorithm 3.9, the AND gadget in the intermediate circuit C' performs a total of $n \cdot (7n-5)/2$ operations, with $n = t+1$ shares for perfect security against t probes. This includes $n \cdot (n-1)/2$ random generations, and $n \cdot (3n-2)$ boolean operations. In turn, this gives $N_m = (t+1) \cdot (7t+2)/2 \simeq 7t^2/2$ operations as a function of the maximum number of probes t. We now consider circuit $\widetilde{C}$ corresponding to the expansion of the AND gadget in C'. Every random generation in the intermediate circuit C' requires $\ell+1$ operations in $\widetilde{C}$. From Algorithm 4.1, every boolean operation in C' requires $5\ell+3$ operations in $\widetilde{C}$. Hence, the total number of operations is:

$$N_s = n \cdot (n-1)/2 \cdot (\ell+1) + n \cdot (3n-2) \cdot (5\ell+3) \simeq \frac{31}{2} \cdot n^2 \cdot \ell$$

With $n = k+1$ and $\ell = 4t/k$, we get $N_s \simeq 62 \cdot k \cdot t$. Finally, with $k = 668$, the number of operations is therefore $N_s \simeq 41\,416 \cdot t$ for worst-case security against t probes. We refer to Table 4.41 for a summary of the operation count.

	Masking countermeasure	Shuffling countermeasure
#rand	$n^2/2$	$\frac{1}{2} \cdot n^2 \cdot \ell$
#bool	$3n^2$	$15n^2 \cdot \ell$
#op	$7n^2/2$	$\frac{31}{2} \cdot n^2 \cdot \ell$
#op	$\mathbf{7t^2/2}$	$\mathbf{41\,416 \cdot t}$

Table 4.41: Number of operations for worst-case security against t probes, where $n = t+1$ for the masking countermeasure, and $n = k+1$ and $\ell = 4t/k$ for the shuffling countermeasure, with $k = 668$; we only keep the high-order terms.

Since the masking countermeasure has complexity $7t^2/2$ and our shuffling countermeasure has complexity $41\,416 \cdot t$, the two countermeasures have equal complexity for $7t^2/2 = 41\,416 \cdot t$, which gives $t \simeq 12 \cdot 10^3$. Therefore we expect our shuffling countermeasure to beat the masking countermeasure for a number of probes $t \geq 12 \cdot 10^3$.

AES implementation. As one additional contribution, we implemented the AES algorithm augmented with our shuffling countermeasure, which we compare with an AES implementation of the masking countermeasure, using the same parameters as above. For our shuffling construction used the following optimization: in the implementation of SecMult (see Algorithm 3.9, when accumulating $c_i \leftarrow c_i \oplus r$ (lines 7 and 9), for a given index i we always used the same signal position j_i among the ℓ wires. This is because the SecMult algorithm would

t	668	2004	3340	4676	6012	7348	8684	10020
Masking (s)	1.4	12	34	70	111	187	235	310
Shuffling (s)	52	63	78	91	102	119	134	141

Table 4.42: Running time of AES implementation, as a function of the number of probes t. We use $n = t + 1$ for the masking countermeasure and $\ell = 4t/k$ for the shuffling countermeasure. Implementation on a 3,2 GHz Intel processor, running on a single core.

remain secure in an extended model of security where the adversary could obtain all successive values of the c_i variables with a single probe. We summarize the timings in Table 4.42; see also Figure 49. We see that our shuffling construction outperforms the masking countermeasure for a number of probes $t \geq 6\,000$, with a running time of approximately 2 minutes for $t \simeq 6\,000$. We provide the source code in [Cor21].

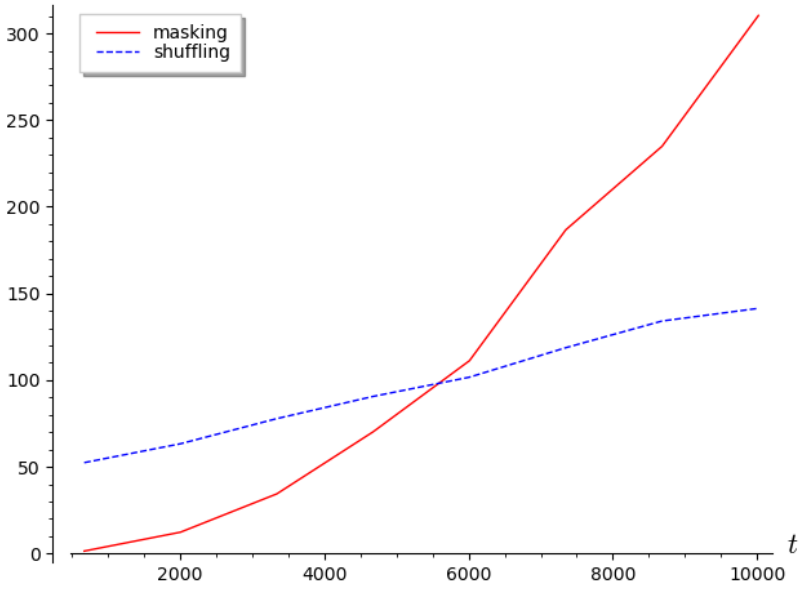

Fig. 49: Running time (in seconds) of the masking and shuffling countermeasure for AES, to get security against t probes. Implementation on a 3,2 GHz Intel processor, running on a single core.

5

Rivain-Prouff on Steroids: Faster and Stronger Masking of the AES

In this chapter, we are going to introduce our second work, submitted at CARDIS22. This has been a joint work with Luan Cardoso dos Santos, François Gérard and Johann Großschädl. Our paper is focused on obtaining a faster implementation of the Rivain-Prouff masking scheme; we propose a more efficient table-based multiplication in $\mathbb{F}_{2^8}$. This new technique facilitates a further improvement in the exponentiation function (see Algorithm 3.5), which, as we are going to see, has been adapted to work faster and with less randomness. Hence, it is possible to, conceptually speaking, divide our work into three contributions: 1) improved table-based multiplication technique, via a new approach for the 0-operands; 2) new exponentiation function, the power-to-254 is performed slightly differently and it uses less randomness (i.e. RefreshMasks); 3) twice-faster implementation of masked AES with order 1. For the sake of fairness, our personal contribution concerns mainly the new SecExp254, while the implementation has been entrusted to the more expert hands of Luan, François and Johann. We directly worked on the algorithm and its SNI property; for this reason, this chapter will recall the table-based multiplication, informally outlining our differences with the classical technique (see Section 5.1); then, we will give a full explanation of the new exponentiation algorithm (see Section 5.2); lastly, we will briefly mention the practical results of our masked AES, comparing it with some of the well-established implementation in the literature (see Section 5.3).

5.1 Table-Based Multiplication

We start recalling the classical table-based multiplication of elements of $\mathbb{F}_{2^8}$ technique. At its core, it essentially involves two look-up tables: the *Log table*, where it is stored the order of every 255 non-0 elements of the field (based on the generator $g(x) = x + 1$); and the *AntiLog table*, where it is stored the elements themselves, i,e, every element corresponding to the orders in the range $[0, 254]$. It should be clear that, if we consider only the non-0 elements, `AntiLog[Log[a]] = a` is always respected. Therefore, supposing $a(x)$ and $b(x)$ are two non-0 elements of the field, their product $c(x) = a(x)b(x)$ is computed as follow:

1. The Log table is accessed to obtain $u = \text{ord}(a(x))$ and $v = \text{ord}(b(x))$.
2. The modular sum of the orders is computed $s = u + v \mod 255$.
3. The AntiLog table is used to recover the multiplication element corresponding to order s.

Consequently, three table lookups and a modular addition are required to perform a table-based multiplication, [DR02, Gla07]. Since it has been used in our implementation, it is worth mentioning that is possible to extend the AntiLog table in order to avoid the modular sum of the orders and treat it as a simple addition. In particular, the table is duplicated in such a way that `AntiLog[255+i] = AntiLog[i]` for $i \in [0, 254]$; in this way, $u + v$ can be added without the modular subtraction. The disadvantage of this technique is that the AntiLog table doubles in size with an overall 768 bytes length, but the efficiency benefits overcome the memory costs if considering that the two tables are static and so they can be stored in non-volatile memory and do not occupy the RAM.

Treatment of 0-Operands. As specified above, the table-based multiplication took into account only non-0 elements; since 0 is not an element of the multiplicative group and, therefore, it has no multiplicative order. But for implementation reasons, Log and AntiLog tables are treated as a 256-entries array (when the non-0 operands are 255). This has been taken as best practice since in this way it is possible to exploit the integer representation of the field elements as the index access for the lookups. Consequently, the 0 element is required but it needs a special treatment: assigning $\mathsf{Log}[0] = 255$ and $\mathsf{AntiLog}[255] = 0$ the aforementioned condition `AntiLog[Log[a]] = a` holds for any $a(x) \in \mathbb{F}_{2^8}$. In turn, in the multiplication algorithm has to check now whenever $a(x)$ or $b(x)$ are equal to 0 and in case compute artificially the multiplication. Specifically, in addition to the above three steps, a fourth one is required where a 0-check is performed over the product factors and in case is set the result to 0. Clear, this slows down the time complexity of the multiplication; for a pseudocode description of the whole algorithm, we refer to Subsection 7.2 of [Gla07].

This check is indeed time-consuming; to reduce such time, we propose a different approach for the 0-element case. In particular, it can be completely removed at the cost of increasing the size of both the Log and AntiLog tables. The idea comes from the observation that no sum in the order of two non-0 elements in $\mathbb{F}_{2^8}$ is strictly greater than 509. Thus, we can rearrange the tables in this way: first, it is set $\mathsf{Log}[0] = 509$, and second, it is possible to extend the AntiLog table from 512 to 1024 entries, in particular for every $509 \leq i \leq 1023$ is $\mathsf{AntiLog}[i] = 0$. In other words, this modification will have the two following consequences: the 0-element has order 509[1] and every sum of orders that involves at least one 0-element will have an order greater than 509; in turn, accessing the AntiLog table with orders greater than 509 will result in the value 0. Note that the above modifications in the tables have no impact on any products involving both non-0 elements, the yield correct result can be deduced from the classical table-based multiplication. Therefore, with our new technique the fourth checking step can be completely

[1] Please note that from a strictly mathematical point of view the 509 order makes no sense; but, as will see, it still guarantees the correct final computation of the product.

avoided; and when applied reduces the time complexity of the `GF2P8MULB`[2] macro by up to $36,4\%$. However, this speed-up comes at the expense of larger look-up tables. The Log table still consists of 256 entries but needs to be able to accommodate the 9-bit integer 509, which means it has to be of 16-bit type `hword` instead of 8-bit type `byte`. Furthermore, the AntiLog table requires 1024 entries instead of 512, whereby all entries in the upper half (i.e. index 509 and above) are 0. Hence, the two tables become twice as large, i.e. the size of both tables amounts to 1536 bytes altogether, which is still relatively small compared to the non-volatile memory consumption of other masked AES implementations.

Reducing the Number of Table Look-Ups. We didn't mention it before, but with the same reasoning and a slight change, it is possible to reduce the number of lookups for the original exponentiation algorithm in [RP10] (see Algorithm 3.5). Such exponentiation-based secure inversion consists of four masked multiplications, three other masked arithmetic operations in $\mathbb{F}_{2^8}$ (i.e. squaring, fourth-power, 16th-power), and two mask refreshings. However, the three other arithmetic operations are all linear in $\mathbb{F}_{2^8}$ and they can, therefore, be performed separately on each of the n shares. In addition, the three operations have in common that each of them can be implemented with a 256-byte look-up table. Moreover, if we just do not consider the mask refreshings for a moment, the first two steps of the inversion in Algorithm 3.5 are a squaring followed by a multiplication[3], and this operation sequence (i.e. a multiplication preceded by a linear operation that requires just an ordinary table look-up) appears two further times, namely in lines 4 and 6, and in lines 7 and 8. Based on this observation, it is possible to reduce the total number of table look-ups during an exponentiation by modifying the squaring table (resp. fourth-power/16th-power table) to contain the order of the square $\mathrm{ord}(b)$ instead of the actual square b. Note that this modification does not increase the size of the table, but allows one to accelerate the exponentiation by using the order read from the table as an operand in a subsequent multiplication, which saves a look-up into the Log table. In the ideal case, i.e. when both operands are given as orders, the multiplication only consists of an addition and a single table lookup.

Remark 5.1. This optimization to reduce the number of table lookups is not particularly effective when applied to the original masked exponentiation given in Algorithm 3.10 because of the mask refreshings between the squaring and multiplication at line 2 and line 5 (which we ignored in the above description of our idea). However, in the next Section, we will introduce a modified variant of the exponentiation that is much better suited for our optimization and, therefore, faster than the original exponentiation of Rivain and Prouff.

[2] `GF2P8MULB` is the instruction for multiplication in $\mathbb{F}_{2^8}$ for modern x64 processors.

[3] To be precise, these two steps are a masked squaring and masked multiplication. However, for the explanation of our optimization, it does not matter whether these operations are masked or not.

5.2 New Exponentiation-Based Inversion

In this Section, we are going to introduce the new exponentiation algorithm; the exponentiation $x \mapsto x^{254}$ is the main operation of the masking scheme. For this reason, it is crucial to find the most efficient way to compute it using the $\mathbb{F}_{2^8}$ multiplication technique described previously; the goal is to minimize the number of table lookups by minimizing the number of conversions between the normal domain and the order domain. There are four main operations ("gadgets") that are used in a secure exponentiation: SecMult, SecSquare, SecPow16, and RefreshMasks. The optimal representation for those operations is as follows:

1. RefreshMasks: As mentioned before, this operation injects fresh randomness into the shares. Since there is no trivial way to do it from the order domain, the inputs and the outputs should be in the normal domain.
2. SecMult: Since the first operations in the SecMult algorithm are ordinary multiplications in $\mathbb{F}_{2^8}$, the order domain is preferred for the inputs. Conversely, the last step is somewhat similar to a mask refreshing and, thus, the outputs are in the normal domain.
3. SecSquare and SecPow16: These are the most flexible operations. Since the tables are precomputed and the lookups are performed share by share, there is no efficiency difference between the two representations and the implementer is free to choose them for inputs and outputs, but of course the lookup tables have to be generated accordingly.

Algorithm 5.1 new_SecExp254 computation

Input: shares $x_0, x_1, \ldots, x_d$ satisfying $x = \bigoplus_{i=0}^{d} x_i$
Output: shares $e_0, e_1, \ldots, e_d$ satisfying $e = \bigoplus_{i=0}^{d} e_i = x^{254}$

1: $(z_i)_{0\le i\le d} \leftarrow \mathsf{SecSquare}((x_i)_{0\le i\le d})$ ▷ $\bigoplus_i z_i = x^2$
2: $(z_i)_{0\le i\le d} \leftarrow \mathsf{RefreshMasks}((z_i)_{0\le i\le d})$
3: $(y_i)_{0\le i\le d} \leftarrow \mathsf{SecMult}((z_i)_{0\le i\le d}, (x_i)_{0\le i\le d})$ ▷ $\bigoplus_i y_i = x^2 \cdot x = x^3$
4: $(z_i)_{0\le i\le d} \leftarrow \mathsf{SecSquare}((y_i)_{0\le i\le d})$ ▷ $\bigoplus_i z_i = (x^3)^2 = x^6$
5: $(y_i)_{0\le i\le d} \leftarrow \mathsf{SecMult}((z_i)_{0\le i\le d}, (x_i)_{0\le i\le d})$ ▷ $\bigoplus_i y_i = x^6 \cdot x = x^7$
6: $(z_i)_{0\le i\le d} \leftarrow \mathsf{SecSquare}((y_i)_{0\le i\le d})$ ▷ $\bigoplus_i z_i = (x^7)^2 = x^{14}$
7: $(y_i)_{0\le i\le d} \leftarrow \mathsf{SecMult}((z_i)_{0\le i\le d}, (x_i)_{0\le i\le d})$ ▷ $\bigoplus_i y_i = x^{14} \cdot x = x^{15}$
8: $(y_i)_{0\le i\le d} \leftarrow \mathsf{SecPow16}((y_i)_{0\le i\le d})$ ▷ $\bigoplus_i y_i = (x^{15})^{16} = x^{240}$
9: $(e_i)_{0\le i\le d} \leftarrow \mathsf{SecMult}((y_i)_{0\le i\le d}, (z_i)_{0\le i\le d})$ ▷ $\bigoplus_i e_i = x^{240} \cdot x^{14} = x^{254}$

In the original Rivain-Prouff exponentiation, the order of operations is such that, when using our gadgets, some extra conversions between representations would be needed. For example, at the end of Algorithm 3.5, the two SecMult operations are sequential, which means that a conversion from normal to order domain is required in between. To improve efficiency, we propose a new exponentiation algorithm in Algorithm 5.1. In this algorithm, we can see that the four SecMult operations are separated by SecSquare (resp. SecPow16) operations, which

means that, when the squaring and 16th-power table are pre-computed to output the result in the order domain, no extra conversions are needed except after the RefreshMasks. Furthermore, one less RefreshMasks is carried out compared to the original exponentiation. This modification invalidates the security proofs we saw in [RP10]; for this purpose given the changes the t-SNI property of our new exponentiation algorithm is faced in the following lemma:

Lemma 5.2 (t-SNI of SecExp254). *Let $(x_i)_{1\leq i\leq t+1}$ be the input shares of the x^{254} operation and let $(y_i)_{1\leq i\leq t+1}$ be the output shares. For any set of t_1 intermediate variables and any subset $\mathcal{O}$ of output shares such that $t_1 + |\mathcal{O}| \leq t$, there exists a subset $\mathcal{I}$ of indexes with $|\mathcal{I}| \leq t_1$, such that those t_1 intermediate variables, as well as the output shares $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|\mathcal{I}}$.*

Proof. The proof will exploit the SNI property of SecMult (Lemma 3.19) and RefreshMask (Lemma 3.20), as well as the linearity of the squaring operation (note the exponentiation to 16 can be seen as four subsequent squarings).

Let $\mathcal{I}$ be the set of indices corresponding to the internal variables observed by the adversary, such that $|\mathcal{I}| \leq t_1$. And let $\mathcal{O}$ be the set of indices corresponding to the output probes observed by the adversary, such that $t_1 + |\mathcal{O}| \leq t$. Moreover, consider a partition of $\mathcal{I} = \bigcup_{i=1}^{9} \mathcal{I}^i \leq t_1$ depending on the subgadget in which observation occurs (see Figure 51).

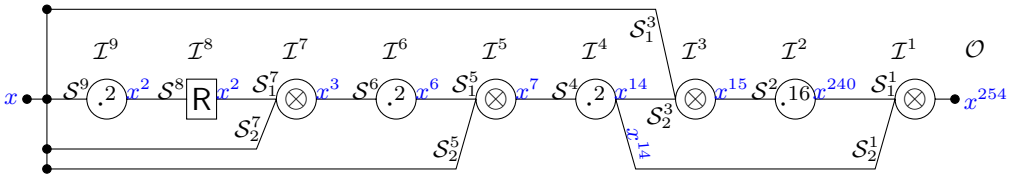

Fig. 51: x^{254} as a composition of squaring ($\cdot^2$), multiplication ($\otimes$) and refreshing mask (R) gadgets.

Thus, to prove the SNI property of Algorithm 5.1, by Definition 3.18 there must exist a subset of input indices $\mathcal{S}$ with $|\mathcal{S}| \leq t_1$, such that the t_1 intermediate variables along with the output variables corresponding to $\mathcal{O}$ can be perfectly simulated from $x_{|\mathcal{S}}$. We are going to build such set $\mathcal{S}$ starting from the output probes and continuing backwards, subgadgets by subgadgets, until the input shares. Formally:

Gadget 1. By assumption, $|\mathcal{I}^1 \cup \mathcal{O}| \leq t_1 + |\mathcal{O}| \leq t$. This allows us to exploit the SNI property of the multiplication $\otimes$ gadget. Namely, there exists, by Theorem 3.19, two sets of indices $\mathcal{S}_1^1$, $\mathcal{S}_2^1$ such that $|\mathcal{S}_1^1|, |\mathcal{S}_2^1| \leq |\mathcal{I}^1|$. In particular, it is possible to simulate the gadget probes along with the observed variables $\mathcal{O}$ from its input shares corresponding to indices in $\mathcal{S}_1^1$ and $\mathcal{S}_2^1$.

Gadget 2. As a composition of squarings over a field of characteristic 2, the gadget $\cdot^{16}$ is linear. Hence, by its NI property, there exists a set of indices $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |\mathcal{I}^2| + |\mathcal{S}_1^1|$ and every probe in the last two gadgets (including the output probes) can be simulated from its input shares corresponding to $\mathcal{S}^2$ and $\mathcal{S}_2^1$. Note that, $|\mathcal{S}^2| \leq |\mathcal{I}^2| + |\mathcal{S}_1^1| \leq |\mathcal{I}^2| + |\mathcal{I}^1|$.

Gadget 3. From the previous steps, $|\mathcal{I}^3 \cup \mathcal{S}^2| \leq |\mathcal{I}^3| + |\mathcal{I}^2| + |\mathcal{I}^1| \leq t$. So, we can apply the SNI property of the $\otimes$ and claim that there exist two sets of indices $\mathcal{S}_1^3$, $\mathcal{S}_2^3$ such that $|\mathcal{S}_1^3|$,$|\mathcal{S}_2^3| \leq |\mathcal{I}^3|$ and every probe in the previous gadgets can be simulated from its input shares corresponding to $\mathcal{S}_1^3$, $\mathcal{S}_2^3$ and $\mathcal{S}_2^1$.

Gadget 4. From the previous steps, $|\mathcal{I}^4 \cup \mathcal{S}_2^3 \cup \mathcal{S}_2^1| \leq |\mathcal{I}^4| + |\mathcal{I}^3| + |\mathcal{I}^1| \leq t$. That joined to the NI property of $\cdot^2$, allows the existence of a set of indices $\mathcal{S}^4$ such that $|\mathcal{S}^4| \leq |\mathcal{I}^4| + |\mathcal{I}^3| + |\mathcal{I}^1|$ and every probe in the previous gadgets can be simulated from its input shares corresponding to $\mathcal{S}^4$ and $\mathcal{S}_1^3$.

Gadget 5. From the previous steps, $|\mathcal{I}^5 \cup \mathcal{S}^4| \leq |\mathcal{I}^5| + |\mathcal{I}^4| + |\mathcal{I}^3| + |\mathcal{I}^1| \leq t$. This means we can use the SNI property of $\otimes$ and define two sets of indices $\mathcal{S}_1^5$, $\mathcal{S}_2^5$ such that $|\mathcal{S}_1^5|$,$|\mathcal{S}_2^5| \leq |\mathcal{I}^5|$ and every probe in the previous gadgets can be simulated from its input shares corresponding to $\mathcal{S}_1^5$, $\mathcal{S}_2^5$ and $\mathcal{S}_1^3$.

Gadget 6. From the previous steps, $|\mathcal{I}^6 \cup \mathcal{S}_1^5| \leq |\mathcal{I}^6| + |\mathcal{I}^5| \leq t$. Hence, by its NI property, there exists a set of indices $\mathcal{S}^6$ such that $|\mathcal{S}^6| \leq |\mathcal{I}^6| + |\mathcal{I}^5|$ and every probe in the previous gadgets can be simulated from its input shares corresponding to $\mathcal{S}^6$, $\mathcal{S}_2^5$ and $\mathcal{S}_1^3$.

Gadget 7. From the previous steps, $|\mathcal{I}^7 \cup \mathcal{S}^6| \leq |\mathcal{I}^7| + |\mathcal{I}^6| + |\mathcal{I}^5| \leq t$. So, we can apply the SNI property of the $\otimes$ and claim that there exist two sets of indices $\mathcal{S}_1^7$, $\mathcal{S}_2^7$ such that $|\mathcal{S}_1^7|$,$|\mathcal{S}_2^7| \leq |\mathcal{I}^7|$ and every probe in the previous gadgets can be simulated from its input shares corresponding to $\mathcal{S}_1^7$, $\mathcal{S}_2^7$, $\mathcal{S}_2^5$ and $\mathcal{S}_1^3$.

Gadget 8. From the previous steps, $|\mathcal{I}^8 \cup \mathcal{S}_1^7| \leq |\mathcal{I}^8| + |\mathcal{I}^7| \leq t$. By Theorem 3.20, also the refreshing mask R is proven to be SNI. Thus, there exists a set of indices $\mathcal{S}^8$ such that $|\mathcal{S}^8| \leq |\mathcal{I}^8|$ and every probe in the previous gadgets can be simulated from its input shares corresponding to $\mathcal{S}^8$, $\mathcal{S}_2^7$, $\mathcal{S}_2^5$ and $\mathcal{S}_1^3$.

Gadget 9. Lastly, $|\mathcal{I}^9 \cup \mathcal{S}^8| \leq |\mathcal{I}^9| + |\mathcal{I}^8| \leq t$. Hence we can, again, apply the NI property of $\cdot^2$ and define a set of indices $\mathcal{S}^9$ such that $|\mathcal{S}^9| \leq |\mathcal{I}^9| + |\mathcal{I}^8|$ and every probe in the previous gadgets can be simulated from its input shares corresponding to $\mathcal{S}^9$, $\mathcal{S}_2^7$, $\mathcal{S}_2^5$ and $\mathcal{S}_1^3$.

In conclusion, from the above statements, we ensured the existence of a simulator for each gadget. Furthermore, let $\mathcal{S} = \mathcal{S}^9 \cup \mathcal{S}_2^7 \cup \mathcal{S}_2^5 \cup \mathcal{S}_1^3$, we can compose these simulators to perfectly simulate the computation of x^{254} from $x_{|\mathcal{S}}$. As the last thing to prove, it must hold that the cardinality of such set of indices has to be less or equal than the number of intermediates probed variables t_1. But, from Step 3, Step 5, Step 7, Step 9 and $\mathcal{I} = \bigcup_{i=1}^{9} \mathcal{I}^i \leq t_1$:

$$|\mathcal{S}| \leq |\mathcal{S}^9| + |\mathcal{S}_2^7| + |\mathcal{S}_2^5| + |\mathcal{S}_1^3| \leq |\mathcal{I}^9| + |\mathcal{I}^8| + |\mathcal{I}^7| + |\mathcal{I}^5| + |\mathcal{I}^3| \leq t_1$$

□

Before continuing with the next section concerning our implementation, we think it is worthy to address one particular comment given to us during the reviews phase of our submission.

In fact, more than one reviewer have suggested to use the *Probe Isolation Non-Interference* (or PINI) security notion from [CS18] for the above exponentiation gadget, given the fact that it is considered stronger than the SNI counterpart. Let's start recalling its definition; but, we will state it explicitly for 1-input gadgets since it is the case for Algorithm 5.1. For the sake of clarity, we use the definition as it was proposed in [CGZ20][4]:

Definition 5.3. *Let G be a gadget with input shares $(x_i)_{1\leq i\leq n}$, and output shares $(y_i)_{1\leq i\leq n}$. The gadget G is* PINI *if for any $t_1 \in \mathbb{N}$, any set of t_1 intermediate variables and any subset O of output indices, there exists a subset $I \subset [1, n]$ of input indices with $|I| \leq t_1$ such that the t_1 intermediate variables and the output shares $y|_O$ can be perfectly simulated from the input shares $x|_{I\cup O}$.*

It should be clear that in the case of 1-input gadgets the two definitions are (almost) identical since they both require a unique set for a unique input. That is, SNI requires that every probe needs to be simulated (only) from $x|_I$, and the PINI definition requires the same set I but it can simulate every probe from $x|_{I\cup O}$. Consequently, the SNI privacy requires even fewer input shares, since $|I| \leq |I \cup O|$. This allows us to claim that, in this specific 1-input gadget case, SNI implies PINI: the simulator $\mathcal{S}$ described in the proof of Lemma 5.2 satisfies the PINI definition as well. In conclusion, Algorithm 5.1 is indeed PINI, but we decided to stay with the SNI security since, in this specific case, it gives better guarantees.

Lemma 5.4. *The exponentiation-to-254 algorithm described in Algorithm 5.1 is* PINI.

5.3 Implementation Aspects and Results

Regarding the final part of our work, we also propose a 1-order implementation of our modified masked AES. Every transformation, i.e. SubBytes, ShiftRows, MixColumns and AddRoundKey, has been implemented in Assembly language to exclude the theoretical possibility that the compiler itself could leak. As mentioned, we put the focus on our main contribution and, while we are going to give an overview of the results and leakages, we will only discuss implementation aspects of the SubBytes transformation, referring to the original paper for a full detailed panoramic of all the whole AES.

Implementation Aspects of SubBytes. SubBytes is, from an implementer perspective, by far the most challenging round transformation because it has a significant impact on the overall execution time and is particularly leakage-sensitive due to its non-linear nature. It consists of 16 S-box operations; our masked inversion is based on the exponentiation technique we presented in Section 5.2 and uses (variants of) the optimized low-level gadgets introduced in Section 5.1. In particular, Algorithm 5.1 has three advantages over the original exponentiation method of Rivain and Prouff. First, it allows us to better exploit our optimization to reduce the overall

[4] Actually, in [CGZ20], the definition considers gadgets with any arbitrary number of inputs; while we explicitly consider only 1-input and 1-output gadgets only.

number of table look-ups and is, therefore, faster. Second, it needs only one RefreshMasks gadget instead of two, thereby reducing the number of random bytes from six to five. Lastly, as explained in the last section, it achieves SNI, which is a stronger notion of probing security that is not met by the original (i.e. flawed) exponentiation algorithm from [RP10]. On the other side, there are two aspects related to the new exponentiation that deserve further explanations. First, Algorithm 5.1 provides only a high-level description of our exponentiation using the four main gadgets (i.e. SecMult, SecSquare, SecPow16, and RefreshMasks), but omits gadgets for auxiliary operations like the conversion of operands between the normal domain and the order domain. For example, at the very beginning of the exponentiation, the shares x_i of the input operand x have to be converted from the normal domain to the order domain so that they can be fed into the SecMult gadgets at lines 3, 5, and 7. In addition, as explained in the last section, the RefreshMasks gadget operates in the normal domain, and therefore its output masks have to be converted from the normal domain to the order domain because they are used as input for the first SecMult gadget in line 3. The input of RefreshMasks is the output of the first SecSquare gadget in line 1, which would normally be given in the order domain. However, instead of converting the SecSquare output from the order domain to the normal domain, we implemented a second SecSquare gadget that operates fully in the normal domain, i.e. both its inputs and outputs are elements of $\mathbb{F}_{2^8}$ and not orders. Moreover, besides the inversion in $\mathbb{F}_{2^8}$, an S-box operation also involves and affine transformation, which is a linear operation and can be performed share-by-share using a 256-byte look-up table. At the highest level, our implementation of the first-order masked SubBytes transformation consists of a simple loop that is iterated eight times and executes in each iteration a pair of two masked S-box operations (i.e. two inversions followed by two affine transformations). Each first-order masked S-box requires five bytes of randomness (four for the SecMult gadgets and one for RefreshMasks), which amounts to 80 random bytes for the complete SubBytes.

Results. We give in Table 5.31 a comparison between our work and the previous implementations on a similar architecture. It is somewhat hard to make a direct comparison since even if the platforms are similar, the techniques used for the masking schemes are quite different and might not offer a similar level of security, neither in theory nor in practice.

The most straightforward scheme to compare to is naturally the Rivain-Prouff masking scheme studied in [GR17] since it is the starting point of our work. We can see that the techniques described in Section 5.1 and Section 5.2 largely improve the masking scheme as the cycle count is almost divided by two. The cycle-count for [SS17] is from Table 2 of [BWG+22]. The reason why this value (17.5k cycles) is twice the cycle-count reported on the authors' GitHub repository[5] ($3439.5 + 5288.1 \approx 8.75k$) is that the implementation in [SS17] is bit-sliced and processes blocks pairwise, but on GitHub the cycles/block are given. The minimum execution time is $8.75 \cdot 2 = 17.5k$ (but two blocks are encrypted). For Goudarzi-Rivain, the cycle-count of 49,329 corresponds to "Standard AES (KHL)" in [GR17, Table 3] (resp. [GR16, Table 16]) for $d = 2$: $7640 \cdot 4 + 6229 \cdot 2 + 6311 = 49,329$. Note that d in [GR17] and [GR16] is

[5] See `https://github.com/Ko-/aes-armcortexm`.

Table 5.31: Comparison of masked AES implementations for ARM Cortex-M3/M4 microcontrollers. [1]Schwabe et al's size is given for a full AES-CTR implementation, no size for AES block encryption was given in their paper. [2] [AP20] indicates two versions of similar performances but [BWG+22] does not report which one was used for the benchmarks. [3] See [FMPR10] for a discussion of the security of affine masking.

Reference	Implementation details	Exec. time (cycles)	Code size (kB)	Theoretical security	Practical security
[GR17]	RP masking	49,329 (ARM7)	4.8	Probing	unknown
[SS17]	Bitsliced	17,500 (M4)	39.9[1]	Probing?	t-test (failed [BWG+22])
[GSDM+19]	Bitsliced	6,800 (M4)	25.2	Probing	t-test (failed [BWG+22])
[AP20]	Fixsliced	6,200 (M4)	22 or 4[2]	Probing	t-test (failed [BWG+22])
ANSSI (compact)	Affine[3]	53,072 (M4)	5.5	Affine	Attacked ([BS20, MS21])
ANSSI (unrolled)	Affine[3]	29,920 (M4)	25.0	Affine	Attacked ([BS20, MS21])
Our work	RP masking	25,413 (M3)	3.2	SNI	t-test (passed)

the number of shares and not the masking order. Besides, our implementation is significantly more compact with a code size improvement of over 30%[6]. On the affine masking side, the ANSSI implementation is beaten on both metrics when considering first-order security. The contenders are thus the bitsliced/fixsliced implementations offering better speed results than ours and that are capable to process two blocks in parallel. We can first observe that their code is larger. However, we should fairly assume that large implementations were unrolled and that they did not try to optimize this metric. Regarding security, [GSDM+19] and [AP20] are reducing to the bare minimum the amount of randomness required to mask the whole scheme (only 2 bits). While they used a formal verification tool to assess the security in the probing model, using such an aggressive technique is quite naturally raising concerns for the practical security of the scheme and would not fit in our theoretical framework. Also, their technique does not generalize to arbitrary orders. More importantly, all the bitsliced/fixsliced schemes are actually leaking according to the recent analysis in [BWG+22]. This means that all those schemes need to be fixed before proper benchmarking can be performed. Since it is impossible to tell beforehand how large the performance penalty incurred by this fix will be, the cycle-count reported does not offer the possibility to compare to our result.

[6] While our code size is relatively small, we need six tables with an overall size of 3328 bytes: Log-table (512B), AntiLog-table (1024B), two squaring-tables (one with outputs in normal-domain for subsequent RefreshMasks, one with outputs in order-domain for subsequent SecMult, each 512B), 16th-power-table (512B), affine-transformation-table (256B). A fast Rivain-Prouff implementation like [GR17] needs 1792 bytes for tables (Log-table 256B, AntiLog-table 512B, squaring-table 256B, 4th-power-table 256B, 16th-power-table 256B, affine-transformation-table 256B)

6

Conclusions

We have seen how, since their first appearance, side-channel attacks represent a serious threat to any cryptosystem, and for so, they have been meticulously analysed both in academia and in the industry sector. Moreover, the increasing usage of software applications in our daily life is pushing security to become more suitable for embedded systems, which are limited in computational power. Side-channel countermeasures are, surely, not exempted from such challenges. Motivated by that, we presented, in this thesis, our contributions on the provable security of cryptographic implementations against side-channel attacks.

We showed how those countermeasures passed from being built ad hoc, i.e customizing specific hardware to resist against specific attacks, to being incorporated at the implementation level. Over the years, the seminal work proposed by Ishai, Sahai and Wagner in [ISW03] established itself at the very top among the most widely used protection strategies. In particular, they provided the very first provable secure model able to achieve privacy against a wide range of side-channel attacks. They exploited a secret sharing scheme to mask sensitive data, an approach already investigated by Chari et al. in [CJRR99], and that was identified as a good fit to prevent attacks based on power and electromagnetic leakage or differential analysis. Furthermore, the ISW probing model responded to the necessity of protection against high-order attacks; while, before it, masking was mainly constrained to the first-order (as in [CJRR99]) or second-order (like in [SP06]).

A decade after Ishai et al.'s work, in 2010, Rivain and Prouff showed an efficient way to adapt the ISW schemes to be applied to the block-cipher AES. Specifically, in [RP10], they were able to extend the ISW multiplication algorithm, over $\mathbb{F}_2$, to work and still be secure over any general $\mathbb{F}_{2^n}$. Their work influenced the side-channel field towards several improvements that are leading the masking countermeasures to achieve more practical performances. Barthe et al. strengthened the probing security with more independency between input and output shares, defining the so-called strong non-interference security. It is considered among the strongest, with the probe isolation non-interference [CS18], security definitions against side-channel attacks. Through SNI, it has made it possible to reduce the number of shares from $2t + 1$ to $t + 1$.

Thus, after retracing the milestones in the context of the ISW model, we saw our main contribution: secure wire shuffling in the probing model. We have described the first improvement made to the wire shuffling countermeasure from [ISW03]; namely, the construction achieving statistical privacy in the worst-case setting, having running time $\mathcal{O}(t \log t)$. Through a simpler way to process the "adjacent-position" property, carried out via a more expensive sorting network in [ISW03], we were able to lower the complexity to linear, i.e. $\mathcal{O}(t)$. Our construction is somehow practical in that for an AES implementation we can beat the classical masking countermeasure for a reasonable running time. However, such a crossover point occurs for $t \simeq 6\,000$; consequently, it is still probably unpractical for embedded implementations.

However, those theoretical achievements produced in the last decade came at the expense of relatively poor performance in terms of real-world implementations; especially if we consider masked implementation of AES based on the Rivain-Prouff technique. We saw how the efficiency of this technique is mainly driven by the efficiency of the multiplication in the field. Those multiplications are usually performed using so-called Antilog and Log tables, which map elements to their order in the multiplicative group of the field and vice-versa. We explained the classical way to multiply two elements: firstly, mapping them to their orders, then adding the orders (modulo 255), and finally mapping the result back to an element corresponding to their product. But, since the inversion is only defined in the multiplicative group, multiplications by 0 need to be treated as special cases; which, in turn, would increase additionally the computational time.

Thus, starting from the investigation proposed by Goudarzi and Rivain in [GR17, GR17], we demonstrated that the Rivain-Prouff masking scheme can be made much faster, namely by almost a factor of two in comparison with the, to this date, best implementation in the literature. We achieved this speed-up by (i) an optimization of the multiplication in $\mathbb{F}_{2^8}$ that allows us to overcome the special treatment of 0-operands and avoid branch-and-compare instructions, and (ii) a new exponentiation technique that reduces the overall number of table look-ups. Furthermore, we saw that our masking scheme is not only (almost) twice as fast as the original Rivain-Prouff scheme, it also satisfies SNI, which gave to it also strong theoretical guarantees.

List of Figures

List of Tables

Acknowledgements

First and foremost, I would express my most profound appreciation to Prof. Jean-Sébastien Coron for guiding me during those four years. I am highly grateful to him for offering me the opportunity to join his research group. I have truly appreciated his invaluable supervision, advice, and patience during my PhD study. So much has been learnt from our meetings.

I also extend my sincere gratitude to my group colleagues; their friendly support has played an essential role in this professional journey. Thanks to Razvan, Luca, Vitor, and Rajeev, who, over the years, left the group to continue their bright careers. Thanks to Pierrick and François to have shared with me this concluding phase. And, last but not least, thanks to Agnese. We started and (almost) finished our PhDs together, and much would not have been possible without your helpful reminders and mathematical explanations.

I also wish to thank Ingrid Verbauwhede, François-Xavier Standaert, Alex Biryukov, David Naccache and Jean-Sébastien Coron; it is an honour to have them on my defence committee. I particularly thank David and Alex for being members of my CET and for having kindly and meticulously supervised my work every year.

A special thank you to my co-authors Jean-Sébastien Coron, Johann Großschädl, François Gerard, and Luan Cardoso dos Santos for our collaboration.

Infine, un ringraziamento alla mia famiglia, a tutte le amiche e tutti gli amici di Roma per aver pazientemente ascoltato quattro anni di lamentele. La vostra lontana vicinanza non è mai passata inosservata né sarà mai dimenticata. Many thanks also to the whole Luxembourgish fellowship, the cold weather would have been unbearable without the warmth of friends.

Grazie di cuore!

References

[ADF16] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with o(1/log(n)) leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 586–615, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An $O(n \log n)$ sorting network. pages 1–9, 1983.

[AP20] Alexandre Adomnicai and Thomas Peyrin. Fixslicing aes-like ciphers: New bitsliced aes speed records on arm-cortex m and risc-v. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):402–425, Dec. 2020.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129, 2016.

[BDD+01] Therese Biedl, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer, and Stephen G. Kobourov. Tight bounds on maximal and maximum matchings. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation*, pages 308–319, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 1–18, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[BS20] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):1–25, 2020.

[BWG+22] Arthur Beckers, Lennert Wouters, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. Provable secure software masking in the real-world, 2022. `https://www.esat.kuleuven.be/cosic/publications/article-3461.pdf`.

[CGPZ16] Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. Faster evaluation of sboxes via common shares. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, pages 498–514, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[CGZ20] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 342–375, Cham, 2020. Springer International Publishing.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 398–412, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

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

[Cor21] Jean-Sébastien Coron. Implementation of higher-order countermeasures, 2021. Publicly available at `https://github.com/coron/htable/`.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 28–44, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CPRR14] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption*, pages 410–424, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 742–763, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[CRV14] Jean-Sebastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. Cryptology ePrint Archive, Paper 2014/890, 2014. `https://eprint.iacr.org/2014/890`.

[CS18] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. Cryptology ePrint Archive, Paper 2018/438, 2018. `https://eprint.iacr.org/2018/438`.

[CS21] Jean-Sébastien Coron and Lorenzo Spignoli. Secure wire shuffling in the probing model. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 215–244, Cham, 2021. Springer International Publishing.

[DR02] Joan Daemen and Vincent Rijmen. *Specification of Rijndael*, pages 31–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[FMPR10] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. *IACR Cryptol. ePrint Arch.*, 2010:523, 2010.

[Gla07] Brian R. Gladman. A specification for Rijndael, the AES algorithm. Technical report, available for download at `http://ccgi.gladman.plus.com/oldsite/cryptography_technology/rijndael/aes.spec.v316.pdf`, 2007.

[GR16] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? Cryptology ePrint Archive, Report 2016/264, 2016. `http://ia.cr/2016/264`.

[GR17] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Jean-Sébastien Coron and Jesper B. Nielsen, editors, *Advances in Cryptology — EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597. Springer Verlag, 2017.

[GSDM+19] Hannes Gross, Ko Stoffelen, Lauren De Meyer, Martin Krenn, and Stefan Mangard. First-order masking with only two random bits. In *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*, TIS'19, page 10–23. ACM, 2019.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[MS08] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008.

[MS21] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected AES implementation on ARM. *IACR Cryptol. ePrint Arch.*, page 592, 2021.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 413–427, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[SP06] Kai Schramm and Christof Paar. Higher order masking of the aes. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, pages 208–225, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[SS17] Peter Schwabe and Ko Stoffelen. All the aes you need on cortex-m3 and m4. In Roberto Avanzi and Howard Heys, editors, *Selected Areas in Cryptography – SAC 2016*, pages 180–194, Cham, 2017. Springer International Publishing.