Preventing differential side-channel attacks in hardware: from generic to specific solutions

Charles Momin

February 2024

École polytechnique de Louvain
UCLouvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. François-Xavier Standaert	*UCLouvain*
Pr. Sylvain Guilley	*TELECOM-ParisTech*
Pr. Amir Moradi	*TU Darmstadt*
Pr. Thomas Peters	*UCLouvain*
Pr. David Bol	*UCLouvain*

Acknowledgements

This work is the outcome of five years spent in the cryptography research group at UCLouvain. I would like to thank everyone I have encountered, who, sometimes indirectly or unexpectedly, contributed to the completion of this thesis.

First, I would like to express my deepest gratitude to my supervisor, François-Xavier Standaert, who not only introduced me to the world of research but also provided continuous guidance as well as challenging questions. He let me the freedom to explore my own investigation paths and to make my own choice while providing invaluable support when needed.

I am thankful to my jury and committe members, who took the time to read and insightfully discuss this thesis. In particular, I would like to thank Sylvain Guilley and Amir Moradi for being in my jury and for their thorough comments on my manuscript and Thomas Peters for the different collaborations and for being the secretary of the jury. Finally, I would like to thank David Bol for being the president of the jury and to Christof Paar for being a member of my committe.

Special thanks to all my colleagues at the Crypto Group, whose team spirit and camaraderie contributed to make the experience unforgettable. Thank you for the joint collaboration and daily support, for the enriching technical and non-technical discussions, and for the hearty laughs over a coffee, during a squash match, a bike ride, culinary discoveries of all kinds or our traditional beer meetings.

Finally, I would like to thank my friends, my family, and my wife, who have supported me and sometimes indirectly endured the consequences of certain publication processes. The interest you occasionally showed in my work has been a boost and attempting to answer your questions has always been a enriching exercise for me.

Abstract

Information security relies on cryptographic algorithms preventing third parties from diverting systems or knowledge from their intended use. These algorithms, deployed on electronic devices, are exposed to side-channel attacks that exploit the circuit's data-dependent physical behavior.

In hardware, the countermeasures' ecosystem ranges from generic solutions applicable to any algorithm and providing adaptable security levels but typically implying significant overheads on the resulting circuit's performances, to others having lower overheads but requiring at least algorithmic tweaks and possibly coming with some inherent physical security but little flexibility. This thesis explores the challenges and trade-offs faced when implementing three intertwined yet different solutions.

First, masking consists in randomizing the values manipulated by an implementation, making them difficult to exploit. This common generic strategy is applicable to any algorithm but involves significant overheads. Yet, we illustrate here that these can be amortized and that implementations providing a good trade-off between area and latency are achievable.

Second, fresh re-keying is another strategy whereby an easy to protect re-keying function is used together with the long term key to generate a fresh session key used for few algorithm's executions only, thereby limiting the amount of exploitable key manipulations performed. Two distinct re-keying functions are evaluated here.

On the one hand, a novel easy-to-mask function, whose implementation mitigates common limitations encountered with masking while keeping the possibility to adapt the security level.

On the other hand, a function that limits by design the number of exploitable observations when attempting to mount an attack without relying on built-in protections but doesn't leave much room for adaptations. We show that physical security can be achieved with limited overheads but at the cost of using conservative instances parameters.

Contents

1 Introduction

1.1 Information security and modern cryptography

For centuries, information security has been a recurring concern in society. While it is easy to imagine its usefulness in strategic fields, such as intelligence or military applications, it can find application in a multitude of other areas, often to ensure that someone has good control over access to information or services. Whether in accordance to the law in force or not, information security aims to develop technical solutions which, if properly developed, prevent a third party from diverting a system or knowledge from its intended use case. Examples of applications include the protection of sensitive information for industrial, commercial or private purposes (e.g., protection of intellectual property, application of professional secrecy, secure messaging or storage) or identity verification (e.g., bank transfers, ratification of official documents, etc.).

With this goal in mind, cryptography appeared as a discipline studying and aiming to produce methods for protecting messages. Numerous attempts have been proposed over the centuries with some mechanisms dating back to antiquity. However, the advent of task automation, the improvement of the computational power (electronics being a perfect example linking the two) and the undeniable interest in and use of reverse engineering methods (potentially with considerable resources) now necessitates the development of more robust methods. Therefore, modern cryptography seeks to rigorously study the security of a solution against an adversary, using formal proofs based on assumptions about the latter's capacity (defining the so-called threat model). More particularly, nowadays security mechanisms are in fact mathematical algorithms that are applied to the data to be protected together with a secret parameter (called the key). These algorithms are public and are designed such that the security lies only in the knowledge of the key (in opposition to the security through obscurency). Put in another way, that means that it is hard for someone to compromise the security of the data without knowing the secret key value, even if he has access to all the public knowledge covered by the threat model.

Considering the various practical application scenarios, the concept of information security can have different meanings and may consider adversaries with different capabilities. On the one hand, confidentiality aims to prevent third parties from accessing information that is not intended for them. On the other hand, authenticity is used to verify the source of information, which may or may not be public. Although independent, the two are often combined to create more robust systems proven secure in the black-box model (i.e., considering only the input/output behavior of an algorithm without taking into account internal computations or implementation matter into consideration).

Schemes are usually built to achieve computational security, i.e., it's difficult for an adversary to compromise security based on "reasonable" resources (either time or memory). In such a context, designer try to ensure that the best strategy an adversary can put in place to compromise security is to try out all possible keys one by one (denoted as key enumeration) until he finds the one that is being used. Such a strategy becomes exponentially more complicated the larger the key in question. As an example, the recent NIST standardization competition in symmetric cryptography required keys of at least 128-bit, which would take billions of years to enumerate today (and will still take several hundred million years in a decade's time).

1.2 The Side-channel threat

Nowadays, cryptographic algorithms are implemented on electronic devices to which an adversary can have physical access in some situation. This only fact forces us to consider a more powerful adversary than the one assumed in the black-box framework insofar as the latter has the ability to observe and even interfere with the circuit physical behaviour. In this regard, Side-Channel Attacks (SCAs) depict the class of attack taking advantage of a dependency between the physical behavior of a circuitry and its internal computation. First demonstrated by Kocher et al. ' Differential Power Analysis (DPA) [KJJ99] that was using electrical power as a source of information, SCAs were generalized to other phenomena such as electromagnetic wave [GMO01; QS01] emissions or the latency of operations that are manipulating sensitive data.

Digital electronics basics To explain the origin of leakage, it's required to understand the basic working mechanism of the digital electronic systems in use today. Generally speaking, a digital circuit can be seen as a set of blocks, each of which performs a basic boolean logic operation

(i.e., relying on binary logical values that are either true or false), and which together perform a complex functionality. In practice, the logic states are represented by the voltage value of a circuit internal (capacitive) node, lying above a specific threshold (resp. below) to depict the true state (resp. false). More specifically, there are two types of logic circuits with distinct operating methods. On the one hand, combinational logic is the set of circuits that perform a function for the values given to it as inputs (i.e., its output is adapted appropriately as soon as its input changes). On the other hand, sequential logic implements functions on the basis of input values, but can also be based on past inputs (and therefore requires the existence of a memory block). To this end, a register is a basic circuit performing basic memory functions. It stores (and holds) the value of its input when a transition occurs on its control signal. It is said to be triggered by a rising edge if storage occurs when the control signal changes from a false to a true logic value (or by a falling edge for the opposite transition). Today's systems are often based on both types of logic (typically by putting combinational logic between registers stages), and use a periodic signal (denoted the clock) to synchronize operations between registers.

The origin of leakage Moving down a level of abstraction, both types of logic are actually implemented using transistors. These act like voltage-controlled switches, i.e. they let current through if a sufficient electrical voltage (which can be seen as true logic) is applied between two of their terminals. In CMOS technology, a combinational logic gate is implemented by a transistor array that connects the gate's output node ether to the voltage source or to the ground, depending on the value applied to its input. In this context, the Boolean logic value true (resp. false) is reached by charging (resp. discharging) the capacitive node at the gate output. The time required for this charge/discharge phenomenon can vary for different nodes (due to specific network architecture or imperfections), leading to situations where logic values do not propagate at the same speed through the combinational circuit. These timing differences can lead to the appearance of temporary logic value transitions that stabilize once all the input values on which a node depends are settled (denoted a *glitch*).

Taking these considerations into account, current consumption can be a (side-) information channel about the values manipulated by a circuit. In particular, a first current consumption (called dynamic dissipation) is caused by value transitions occurring within a circuit at each clock cycle. It originates in the transition of the output node of logic gates, or

through the propagation of glitches, and can be different for each node (e.g., due to differences in capacitance between them). In addition to transitions, a second consumption (i.e., static dissipation) comes from the fact that a certain amount of current is needed to maintain a node at its logic value even in the absence of a transition (due to the existence of parasitic leakage current). These variations in current intensity within the circuit wires are also the source of electromagnetic wave emanations, the direction of propagation of the latter depending on the geometry configuration of the circuit.

A generic side-channel security framework When trying to evaluate a device' security against side-channel attacks, the most basic framework consists in considering that an adversary is capable of measuring a circuit when the latter performs computation involving a sensitive data (e.g., the secret key). More specifically, depending on the application, it is generally considered that the adversary is capable of observing a bounded amount of different executions manipulating the same key value. Sometimes, it is also considered that the adversary can repeat the same execution multiple times (e.g., in order to reduce the noise level through averaging). These observations (often denoted as *traces*) consist of several samples measured for a single execution, generally acquired using an appropriate sensor for the exploited physical phenomenon. The traces quality (in terms of informativeness) may vary between experimental setups and depends on different factors (e.g., sampling frequency, number of samples, resolution or the bandwidth of the measuring device or probe used) such as studied in [BUS21].

Under these considerations, an attack is any strategy that allows an adversary to find with a significant probability the value of a sensitive data involved in the targeted algorithm (e.g., the key) only relying on the measurements and the public knowledge (e.g., the algorithm itself and the inputs/outputs data if available). Attacks are generally divided into several sub-attack targeting smaller part of the data to recover in order to reduce complexity (denoted a divide and conquer attack). Two categories of attack are usually considered: the first one consists in directly mounting an attack against the targeted device during a so-called online phase. The DPA [KJJ99] and the Correlation Power Analysis (CPA) [BCO04] are two of the most-known examples. The second one additionally assumes that the adversary has access to a device similar to the one he is targeting prior to the online phase. This so-called training phase allows an adversary to estimate the conditional

distribution $\hat{\mathrm{p}}(l|x)$ (also referred as a *template* [CRR02]) of the measured value l provided that the internal value x is manipulated.

When trying the build a model, a classical assumption is that the leakage consists of a deterministic part (modeled as a function of the targeted state) and an additive noise. The noise can also be split in two parts: the first one is called the algorithmic noise and reflects the impact on the observed leakages of internal states not covered. The second one is called the physical noise and aims to model the non-deterministic part of the leakage (e.g., due to thermal noise). Under the assumption that the models correctly reflect the physical behavior of the device, they can be used to mount a maximum likelihood attack where the most probable key candidate is kept out of all the candidates using generally significantly fewer traces than without profiling [SMY09]. Under such circumstances, assessing the security of a device against side-channel attacks means evaluating the number of traces (i.e., the data complexity) required during the online phase in order to break the security of the implemented algorithm.

The quality of an attack is often quantified by the reduction in key entropy it enables for a given number of traces (i.e., the number of possible values the key can take on given physical observations). More practically, an attack is considered successful if the key rank it achieves falls below the threshold considered to enable key enumerations (seeing an attack as a way of sorting key values from most likely to least likely, the rank is the position of the correct key value when the attack is carried out).

Worst-case side-channel security When trying to assess the scalability of an attack a strategy consists in mounting an attack with the number of traces for which we are seeking to evaluate security. Apart from the fact that this may require a significant investment of time and resources, this approach has a major disadvantage: there is no way of knowing in advance whether an attack will succeed or not (and is sometimes more akin to trial and error between the best attacks proposed in the literature). Another approach consist at trying to evaluate the worst-case security level, that is, the theoretical lowest data complexity required to mount an attack. An alternative strategy consists thus in relying on information theory in order to evaluate how much information about the target value can be extracted from the leakage. In particular, the Mutual Information (MI) between a random variable X and the observed leakage L can be used to quantify the amount of information (expressed in bits) recovered about one by observing the second and thus to bound

the data complexity. It turns out that it is difficult to know the exact distribution of the leakage, making it impossible to compute the MI in practice. Instead, one strategy is to rely on bounds of the latter such as the Perceived information (PI) or the Gaussian Hypothetical Information (gHI) [Bro+19]. In view of the threat posed by SCA, the development of secure implementation against the latter has been an active research topic for the last decades as detailed next.

1.3 Countermeasures

The first countermeasures were typically proposed at low abstraction levels due to the physical nature of the leakage. For example, hardware countermeasures can target a reduction of the side-channel information by blurring the signal into noise in the time or amplitude domains [CCD00; Man04], or by reducing this signal thanks to special (dual-rail) circuit technologies [TV03; TV04]. In addition, these countermeasures can be improved using two strategies. On one hand, the first operate at the implementation-level and aims to amplify the signal reduction that can be obtained, which can be achieved by masking [Cha+99; GP99]. On the other hand, the second is more related to the algorithmic level and aims to limit the possible exploitation of a given signal level, which can be achieved by re-keying [Med+10].

Masking The underlying principle of masking is to replace every variable of an execution by secret-shared intermediate variables (denoted a *sharing*) that are individually independent of the secret data manipulated in such a way that only the combination of all shares reveals information about the secrets. A common example is the Boolean masking, where the secret bit x is secret-shared in the d shares sharing $(x_0, x_1, \ldots, x_{d-1})$ such that $x = \bigoplus_{i=0}^{d-1} x_i$ with where the value of $d-1$ shares are randomly assigned. The operations are typically split in elementary operations (e.g., simple logic gates) that are achieved by dedicated circuitry computing securely over the shared data (denoted the *gadgets*). Using this technique, the adversary is forced to collect information on all individual shares before combining them to reconstruct the sensitive intermediates. It turns out that learning information about the secrets from partial information based on their shares is a hard problem. To be precise, if the leakage of the individual shares is sufficiently noisy and independent, masking is capable of providing exponential security in the number of shares (i.e., proportional to the inverse the product of

the MI of each share) as supported by theoretical formalization [ISW03; PR13; DDF14; DFS15].

Yet, securely masking a circuit remains a difficult task when taking into account the physical defaults (such as glitches [MPG05], transitions [Cor+12] and couplings [Cnu+17]) that can break the independence assumptions. Another challenge is that the security of small gadgets may not directly extend to their combination, leading to so-called composition issues [Cor+13; Bar+16]. The absence of a formal analysis that takes these two issues into account has led to the discovery of higher-order security flaws (i.e., when more than a single share is considered) in most of the schemes proposed previously [Fau+18; Moo+19]. Recently, the composable notion of glitch-robust probe-isolating non-interference (PINI) [CS20] together with the HPC masking scheme [Cas+21] were introduced as a solution to these challenges (followed by variants with different trade-off such as HPC3 [KM22]). The trivial composition properties of the latter enables the verification of implementation at (potentially large) scale (e.g., using the fullVerif tool [Cas+21]) and makes it an interesting target for automated generation of masked implementations (e.g., using the AGEMA tool [Kni+22b]).

Finally, while recent implementations relying on scheme (provably) secure against the aforementioned physical defaults allow efficient masked hardware implementations by leveraging parallelism to simultaneously operate on the individual shares (e.g., [CS21; KSM22; MCS22; Kni+22a; Cas+23b]), they also come with the need of fresh random bits. Naturally, full ciphers composed of many such gadgets also need many bits of fresh randomness per cycle, especially if those implementations leverage parallelism, are optimized for low latency and if higher-order protection is required. Hence, it is not uncommon for parallel masked hardware implementations of full block ciphers to require hundreds or even thousands of independent, uniformly distributed, and unpredictable random bits per cycle, which may turn out to be challenging to implement in an efficient manner.

Re-keying The concept of fresh re-keying was introduced as a result of the limitations encountered when trying to mask common cryptographic primitives. Its underlying idea is to leverage a separation of duties between an easy to protect "re-keying function", that is only used to produce a fresh key k^* used by a cryptographically strong function to process the message. The intuition behind this is that this fresh key is used a finite number of times (or even only once), thus limiting in practical terms the applicability of attacks requiring several (potentially

many) traces to succeed. To this end, a first observation is that specifying the properties necessary for the re-keying function turns out to be challenging and requires adjusting the trade-off between the re-keying function's efficiency and the physical assumptions needed for its secure implementation (as reflected by the different adversarial model already considered [Med+10; Med+11; Dzi+16]).

In order to obtain a protected re-keying function, a first approach is to design one that is better suited to masking (i.e., that aims to reduce the impact of the limitations set out above). For this purpose, hard learning problems have been shown to be an important source of cryptographic hardness. Popular instances of such problems include Learning Parity with Noise (LPN), Learning With Errors (LWE), which is a generalization to larger moduli [Reg05], and Learning With Rounding (LWR), which is a deterministic version [BPR12]. In the context of masking, a point of particular interest is their algebraic structures, which feature key homomorphism properties. These allow masking to be implemented at a cost that tends linearly with the number of shares, while limiting independence issues linked to physical defaults.

Another approach is to use a Leakage-Resilient Pseudo Random Function (LR-PRF). These are not based on the implementation of a strong low-level countermeasure, but rather seek to limit the quantity of observations that can be exploited to carry out an attack (under the assumption that a small quantity of information can be extracted from each observation). Existing solutions are organized in the form of alternating structures which can be implemented at a competitive cost compared with higher-order masking. As a practical example, Medwed et al. propose a LR-PRF based on AES executions organized as a tree [MSJ12]. They also point out that a parallel architecture is needed to ensure physical security when an unbounded number of measurements are measure for the same inputs.

1.4 Thesis outline

The following chapters focus on the efficient implementation of circuits protected against side-channel attacks and their integration in side-channel resistant modes of operations. They are based on several papers to which I had the opportunity to contribute. Next is provided a brief description of the topics they cover, ommiting the second chapter which consist in a general background.

The third chapter focuses on the implementation of symmetric modes of operations that are secure against SCAs. It highlights the fact that,

depending on the security objective being pursued, the level of protection required when implementing different primitives may differ. More particularly, it may range from relying on protection uniformly (e.g., using high-order masking for every primitive) to using protection for specific bloc(s) while leaving those performing the bulk of the computation with light (or no) protection. It presents a framework for analyzing the security levels achieved by modes of operations, which translates into practical implementation goals (DPA vs SPA security). Some concrete cases study drawn from candidates proposed in the NIST LWC competition are analyzed to illustrate the trade-off that may be encountered. I finally demonstrate (based on experimental) the potential performance gains enabled by leveled implementations, mainly in terms of energy consumption (throughput will also be covered in Chapter 4 and Chapter 6).

Based on the observation made in the third chapter that SPA security is reasonably easy to achieve in hardware, the remaining chapters focus on the implementation of secure solutions against DPA attacks.

The fourth chapter focuses on the efficient implementation of primitives protected with the standard masking countermeasure. It specifically addresses the HPC2 scheme (a variant of the HPC scheme introduced). In the first section, I evaluate the performances that can be achieved in the practical use case of the leveled Spook mode of operation (a second-round candidate in the LWC competition). I start by detailing the architecture of an unprotected version used as the basis for the implementation of Shadow-512 in the protected version. This highlights the resource-sharing possibilities offered by Spook. I then complete the analysis with a detailed architecture of a masked version of Clyde-128 and combine the two to present the first implementation results of a protected hardware implementation of Spookv2. I conclude the section by illustrating the significant throughput gains enabled by Spook (complementing the energy efficiency results from the first chapter). Next, another section focuses on the use case of AES. I begin by presenting a generic optimization strategy applicable to the implementation of the nonlinear layers with HPC2. I then demonstrate that significant latency gains can be achieved by combining this strategy and leveraging the inherent pipeline architecture of HPC2' AND2 gadgets. To this end, I present three architectures with different levels of parallelism and compare their implementation results with (non-optimized) implementations auto-generated by the AGEMA tool. Finally, I discuss some aspects of physical security achieved by all the protected architec-

tures presented in the chapter, including experimental results, formal composition analysis, and public security evaluation.

Slightly deviating from the traditional masking usage, the fifth chapter tackles the implementation of re-keying schemes. More specifically, I focus on the performance gains that can be achieved by leveraging hard(-physical) learning problems. I begin by specifying 'crypto-physical dark matter' a re-keying scheme based on the Learning With Physical Rounding (LWPR) assumption, a new instance of hard-physical learning problem. I demonstrate that this proposal is promising by achieving significant performance gain compared to existing solution (e.g., latency improvement at scale of order of magnitude) while being more secure than others under the Hamming weigh leakage model assumption. As the latter sometimes lacks realism, I consolidate the validity of an analysis aimed at generalizing the security of the scheme when considering more general classes of leakage. In particular, I experimentally validate that the assumptions made are reasonable. I also conduct a physical analysis (following a worst-case strategy) validating that recovering the long-term key by mounting a side-channel attack against crypto-physical dark matter proves to be challenging.

In contrast to traditional masking, the sixth chapter focuses on the hardware implementation of primitives protected against DPA, based on circuits that do not incorporate built-in protection. Specifically, I investigate the security achieved by protected mode of operation incorporating a LR-PRF similar to those proposed by Medwed et al. in [MSJ12], considering cryptographic coprocessors as a study case. I begin by emphasizing that a lack of analysis of physical security requirements (as presented in the first chapter) can lead to practical flaws by experimentally mounting an attack against the integrity of an existing mode. Then, I perform an in-depth experimental evaluation aiming to assess whether the security requirements of the LR-PRFs are reasonable in practice. I also discuss the limitations faced by such solutions. Additionally, I present implementation results for an integration of the LR-PRF into a leveled mode aiming to achieve the same security properties as the one targeted by the attack conducted at the beginning of the chapter.

2 Background

This chapter introduces the basic concepts used in side-channel analysis. It begins by detailing what characterizes the leakages and presents tools and hypotheses commonly encountered in the literature. It then briefly explains what a side-channel attack is, presents some of common strategies and detail some tools that will be used in this manuscript. Finally, it concludes with a brief explanation of the countermeasures which will be addressed in this work.

2.1 Notations

Despite explicitely defined otherwise, the following notations apply in the manuscript. The realization of a random variable X is represented by x. Additionally, vectors a depicted using bold symbols. It follows that the notation $\boldsymbol{X}$ represents a vector of random variables. The leakage trace conditioned to the internal state value y is denoted $\boldsymbol{l}^y$. The different elements contained in a vector are depicted using an index, such that the vector $\boldsymbol{v}$ containing l element is denoted by $\boldsymbol{v} = [v_0, \ldots, v_{l-1}]$ where v_i denotes the i-th element of $\boldsymbol{v}$. A set is represented in calligraphic letters, such that the notation $y \in \mathcal{Y}$ depicts that y belong to the set $\mathcal{Y}$. When a set of distinct leakage traces of a same length is considered, the vector containing the t-th element of each is represented by $\boldsymbol{l_t}$.

2.2 Side-Channel Setting and Leakage Charaterization

Black-box model Common security applications are based on cryptographic primitives devices that generally manipulate a secret value that is fixed during several executions. In the context of encryption algorithms, the secret usually takes the form of an n-bit value referred as a *key*, where n is called the security parameter. Classical cryptanalysis methods consider that an adversary $\mathcal{A}$ tries to recover information about the key by observing the black-box behavior of a cryptographic primitive (i.e., by observing the input/output behavior of the executions using the fixed key). The precise nature of the observed behaviour depends on the

adversarial model considered: the latter defines the adversary's capabilities by restricting, for example, which values he has access to (input, output or both) and determines whether he can arbitrarily choose them. Under such considerations, cryptographic algorithms are designed such that no efficient attack strategy better than enumerating the 2^n different keys exists.

Outside the black-box model (or the side-channel threat) In practice, the primitives are executed on a physical device and $\mathcal{A}$ is empowered with additional information. The latter finds its source in the dependency between the internal values manipulated by the device and its physical behaviour during the execution of the primitive. For electronic devices, typical information channels are power consumption or the emanation of electro-magnetic waves that can be measured during the executions of the targeted implementation (referred next as the *target*). The so-acquired measurements associated to a single execution are referred as the *time-samples* and form a vector commonly known as a *trace*. A time-sample is considered to be a representation of the combination of the instantaneous current consumption of all the target' components. Under this consideration, each time-sample depends on multiple internal states values (while all can be considered when relying on power measurements, this assumption can be refined for EM wave measurements, which are usually more localized on a part of the target) and may contain information about several intermediate variables manipulated during the target' executions.

Leakage variation: the noise Even when considering that some internal states are fixed, distinct measurements resulting from multiple executions with the same value are sensitive to variations known as *noise*. Two types of noise are often considered. The *physical noise* is directly linked to the physical nature of the phenomena being measured, and finds its source in the noise intrinsic to the components of the targeted circuit (e.g., the thermal noise). The second is linked to the fact that an adversary is in practice not able to properly model all the internal states at once (due to computational complexity), but rather focuses on fewer exploitable ones. In this context, the consumption of non-targeted values is considered as another source of noise denoted the *algorithmic noise*.

Intermediate variable leakage From the acquired traces, $\mathcal{A}$ may obtain the leakage related to a specific intermediate variable. More concretely,

a leakage trace associated to the intermediate state y usually takes the form of a vector of random variables

$$\boldsymbol{L}^y = \mathsf{L}\left(y, \boldsymbol{N}^y\right) = \left[L_0^y, L_1^y, \ldots, L_{N_p-1}^y\right]$$

where L is called the leakage function, $\boldsymbol{N^y}$ is the noise variable vector and the element L_t^y is the t-th time sample and N_p depicts the trace length.

Points of Interest and the Signal to Noise Ratio Not all time samples are necessarily equally informative with respect to an intermediate variable and some may even be mere noise. Intuitively, one might expect that only the time samples measured when the device manipulates the variable would leak meaningful information. This is why side-channel analyses usually involve a pre-processing phase aimed at selecting the relevant time samples from the traces, denoted the Points-Of-Interest (POIs). Such selection directly implies reducing the dimensionality of the leakages, which is of particular interest when considering multivariate attacks (as explained with more details in Section 2.3). To this end, Signal-to-Noise Ratio (SNR) is one of the commonly used tools [Man04]. The latter allows assessing the quality of of the signal exploitable from the mean (i.e., first statistical moment) of the time samples (also known as first-order leakage). Without assumption on the leakage function, the SNR with respect to a specific intermediate variable is calculated for each time index of a given set of traces as follows:

$$\mathsf{SNR}_t(Y) = \frac{\hat{\mathsf{Var}}_{y\in\mathcal{Y}}\left(\hat{\mathsf{E}}\left[\boldsymbol{l}_t^y\right]\right)}{\hat{\mathsf{E}}_{y\in\mathcal{Y}}\left(\hat{\mathsf{Var}}\left[\boldsymbol{l}_t^y\right]\right)}$$

where $\hat{\mathsf{Var}}$ and $\hat{\mathsf{E}}$ denote respectively the sample variance and the mean operators, and $\boldsymbol{l}_t^y$ is the vector of leakages belonging to the class associated to the value y. Concretely, it is composed of the t-th time sample from all the traces measured when $Y = y$. The intuition behind the element $\hat{\mathsf{E}}\left[\boldsymbol{l}_t^y\right]$ is that we seek to characterize the signal related to the targeted intermediate variable only. To achieve this, it is important to ensure that the intermediate values other than the characterized one vary within a single class, which is why the SNR is usually computed using random operands (e.g., key and plaintext). The SNR faces some limitations: first, it is rather difficult to interpret when the exploitable leakage does not lie into the mean. Second, it cannot distinguish time samples that are related to bijectively connected variables.

Additive noise A standard assumption is to consider leakage as a combination of a deterministic part depending on the internal state and a random and independent additive noise. Under this assumption, leakage can be expressed as

$$\boldsymbol{L}^y = \delta(y) + \boldsymbol{N}$$

where δ is a function defining the deterministic contribution of the intermediate variable value on the leakage for each time sample. Similarly, each time sample can be expressed as

$$L_t^y = \delta_t(y) + N_t$$

where δ_t and N_t respectively are the deterministic and the random noise contribution of each time sample.

Under the additive noise assumption, the SNR can be rewritten as

$$\mathsf{SNR}_t(Y) = \frac{\mathsf{Var}(\delta_t(Y))}{\mathsf{Var}(N_t)}$$

where $\mathsf{Var}(\delta_t(Y))$ intuitively represents the signal of the intermediate variable (i.e., the variation of the leakage that only depends on the different values taken by it) and $\mathsf{Var}(N_t)$ denotes the random noise. Besides, the latter is often assumed to follow a Gaussian distribution (while it has been observed experimentally for the inherent physical noise [MOP07], this assumptions may be refined for the algorithmic noise depending on the nature of the deterministic leakage).

2.3 Side-Channel Attacks

General setting The general side-channel attack setting against an encryption algorithm considers that $\mathcal{A}$ has access (during a so-called *online-phase*) to the leakages resulting from the executions of a cryptographic primitive using a fixed key. The attacks exploit the fact that the observed leakage depends on key sensitive intermediate variable (i.e., whose distribution is related to the key value) manipulated during the execution of the algorithm. Generally, the intermediate variables depend only on few key bits that form a set denoted the *subkey*. A *divide-and-conquer* is a common strategy consisting in trying to recover each subkey independently rather than trying to recover the full key at a time. The quality of an attack is often evaluated based on the number of traces required to perform it, known as the *attack data complexity* (next denoted q_a). In practice, $\mathcal{A}$ does not need to recover all the subkeys to be successful when mounting an attack. Indeed, it is enough for him

to recover sufficiently subkeys such that he is able to recover remaining ones by enumeration.

There are two types of attacks: non-profiled attacks and profiled attacks. The first assumes that the adversary has no prior knowledge of the exact behaviour of the targeted device's leakage function. Instead, $\mathcal{A}$ relies on assumptions and uses a *model*, which is a function that approximates the true leakage function based on the value of an intermediate variable. A common example of model is the Hamming Weight (HW) that basically depicts the amount of bits set to '1' in a binary word. The possible subkeys' values (called the *candidates*) are then distinguished using a statistical test between the measured leakage values and their modeling under the assumption that a candidate is used. The quality of these attacks depends directly on the accuracy of the model used. However, it turns out that characterizing the leakage function accurately is a challenging task and may vary from one device to another. In general, simplified (yet sound) models are used and attacks relying on the latter may have suboptimal data complexity. Well known examples are the Kocher et al. ' DPA [KJJ99] or the Correlation Power Analysis (CPA) [BCO04].

Profiled attacks improve the situation by allowing the adversary to profile the leakage function during an additional *training phase*, which precedes the attack' online phase. In particular, it is assumed that during the training phase, $\mathcal{A}$ has access to a device identical to the targeted one, with which he can interact and typically obtain traces for executions of its choice. Similar to the data complexity of the attack, the *training data complexity* is the number of traces used to profile the leakage function (denoted next q_t). In practice, profiled attacks enable a reduction in the data complexity of the attack at the expense of a (possibly computationally complex and data-intensive) learning phase [SMY09].

Gaussian Template Attack Introduced by Chari et al. [CRR02], the Gaussian template attack is a profiled attack that involves modeling the true (unknown) Probability Density Function (PDF) of the leakages with a Gaussian distribution. In practice, the goal of the profiling phase is to evaluate $\hat{\mathsf{f}}(\boldsymbol{l}|y)$, the joint distribution of n_p time samples conditioned to the value of an intermediate state, which takes the following form

$$\hat{\mathsf{f}}(\boldsymbol{l}|y) = \frac{1}{\sqrt{(2\pi)^{n_p}\,|\boldsymbol{\Sigma_y}|}} \exp\left(-\frac{1}{2}\,(\boldsymbol{l}-\boldsymbol{\mu_y})^{\intercal}\,\boldsymbol{\Sigma_y^{-1}}\,(\boldsymbol{l}-\boldsymbol{\mu_y})\right)$$

where $\boldsymbol{l}$ is a leakage observation, $\boldsymbol{\mu_y}$ and $\boldsymbol{\Sigma_y}$ are respectively the leakage mean vector and the covariance matrix conditioned to the interme-

diate variable value y. In practice, these are approximated from the q_t profiling traces and we rather refer next to their estimation denoted $\hat{\boldsymbol{\mu}}_{\boldsymbol{y}}$ and $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{y}}$. Under the assumptions that all the $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{y}}$ are equal, a single covariance matrix can be used (denoted $\hat{\boldsymbol{\Sigma}}$ in such a case) and its estimation may leverage the benefits from a pooled estimation [CK13]. Similarly to the SNR computation, random operands are usually used during the learning phase in order to characterize the leakage associated to the targeted variable only and avoid bias. The probability densities thus created are called *templates*.

These templates can be used to approximate $\hat{\mathsf{p}}\left(y|\boldsymbol{l}\right)$, the probability that the intermediate variable value is y conditioned to the leakage realization $\boldsymbol{l}$. Using Bayes' theorem, the latter can be expressed as follows:

$$\begin{aligned}\hat{\mathsf{p}}\left(y|\boldsymbol{l}\right) &= \hat{\mathsf{Pr}}\left[\boldsymbol{Y}=y|\boldsymbol{L}=\boldsymbol{l}\right] \\ &= \frac{\hat{\mathsf{f}}\left(\boldsymbol{l}|y\right)}{\sum\limits_{y^*\in\mathcal{Y}}\hat{\mathsf{f}}\left(\boldsymbol{l}|y^*\right)}\end{aligned}$$

where $\mathcal{Y}$ is the set of all the values that y can take (e.g., the values spanning from 0 to 255 when considering a byte as a target) and y^* is the candidate of for the intermediate value.

When the targeted intermediate variable can be expressed as a deterministic function ϕ of a subkey and a public variable, the candidates can be discriminated during the attack phase using the following maximum likelihood estimator

$$\hat{k} = \arg\max_{k^*} \prod_{j=0}^{q_a-1} \hat{\mathsf{p}}\left(y_j = \phi(x_j, k^*)|\boldsymbol{l}_j^a\right)$$

where $\hat{k}$ is the subkey candidate value selected, $\boldsymbol{l}_j^a$ depicts the j-th trace of the attack set and y_j is the estimated value of the intermediate variable based on the public value x_j and k^*.

While template modelling could be applied using all the time samples (i.e., $n_p = N_p$), these are usually constructed over a reduced amount of dimensions to decrease the profiling complexity (i.e., $n_p << N_p$). A first strategy consists in considering only relevant POIs (e.g., by using the SNR). Yet, if the amount of time sample with signal is still too large, a solution is to rely on dimensionality reduction techniques (such as relying on linear subspace detailed hereunder).

Linear subspace template Profiled side-channel attacks can be performed by building templates in a linear subspace [Arc+06]. Compared

to the PDF estimation defined above, the adversary estimates the following leakage's conditional PDF:

$$\hat{\mathsf{f}}\left(\boldsymbol{l}|y\right) = \frac{1}{\sqrt{(2\pi)^{n_d}\left|\hat{\boldsymbol{\Sigma}}\right|}} \exp\left(-\frac{1}{2}(\boldsymbol{W}\boldsymbol{l} - \hat{\boldsymbol{\mu}}_y)^{\intercal}\hat{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{W}\boldsymbol{l} - \hat{\boldsymbol{\mu}}_y)\right)$$

where $\boldsymbol{l}$ is a leakage vector of length n_p and y a sensible variable value. The matrix $\boldsymbol{W}$ with dimensions $n_d \times n_p$ is the linear projection from the leakage domain (i.e., directly sampled from a scope) to its linear subspace with n_d dimensions. In the context of this work, the latter is computed using Linear Discriminant Analysis (LDA). The vector $\hat{\boldsymbol{\mu}}_y$ of dimension n_d is the approximated mean of the leakage in the linear subspace conditioned to y and is estimated in practice. Finally, the matrix $\hat{\boldsymbol{\Sigma}}$ is the estimated pooled covariance matrix within the linear subspace. The likelihood strategy followed with standard templates can be applied in the same manner using this distribution.

When dealing with subspace-based template attacks, the parameters n_p and n_d have to be carefully chosen. This can typically be done by selecting the value maximizing the Perceived Information (PI) after projection [Bro+19].

Consideration about the data complexity Depending on the considered threat model, the adversary has sometimes the option to reduce the amount of noise by relying on *averaging*. To do so, $\mathcal{A}$ repeats several executions using the same fixed value and collects the corresponding traces. Then, he builts a single averaged trace by simply averaging the aforementioned acquired traces (this technique is in practice used in Chapter 5 and Chapter 6). Under the assumption of additive Gaussian noise, averaging using n_a traces directly multiplies the SNR by n_a. Although it increases the number of traces used for the attack, it does not necessarily increase the number of different executions (e.g., with different plaintexts in the context of encryption). In practice, the data complexity must be interpreted as a multiple $q_a = n_a \cdot q_a^m$, with q_a^m being the number of different executions conducted. Attacks are often categorized based on the value of q_a^m: Simple Power Analysis (SPA) typically uses a low value (i.e., $q_a^m = 1$ for the strongest SPA) of q_a^m, whereas Differential Power Analysis (DPA) considers an arbitrary number of different executions.

2.4 Countermeasures

Here, we briefly describe the countermeasures considered in this work. The purpose of this section is to provide the basics, and further details will be given in the chapters dealing with them.

2.4.1 Masking

Introduced in [Cha+99], additive masking is a common countermeasure that has been the topic of numerous research works in the past two decades. It consists in encoding each intermediate variable y of a computation by a tuple of d values $\boldsymbol{y} = (y_0, \ldots, y_{d-1})$ such that each value y_i taken individually is independent of the value y but respecting the property that $y = \sum_i y_i$. The tuple $\boldsymbol{y}$ is denoted as the *sharing* of y and the y_i are depicted as the *shares*. Correspondingly, the operations are replaced by gadgets that operate on sharings. This strategy is generic and can be applied to any algorithm. The amount of shares d is often denoted as the masking order.

By employing this technique, $\mathcal{A}$ is forced to exploit jointly the information on all shares rather than being limited to a single internal state. Under the assumption that the shares leak independently and in a sufficiently noisy manner, the security of masking against side-channel attacks grows exponentially and one can assess the required data complexity by [DFS15]

$$q_a^m \geq \frac{cst}{\mathsf{MI}\left(Y; \mathsf{L}\left(Y\right)\right)^d}$$

where MI depicts the mutual information between the random intermediate variable and the measured leakage and cst is a small constant depending on the size of the key hypothesis and the target success rate [Ché+19].

Probing-model The probing model is a commonly used model in order to formally evaluate masking security[ISW03]. Using the latter, computations are represented as an abstract arithmetic circuit and $\mathcal{A}$ may obtain the values of t wires within the circuit. Under these circumstances, a circuit is said to be t-probing secure if the values manipulated by any set of t intermediate variables within the circuit are jointly independent of sensitive values. More intuitively, the t-probing security of a circuit ensure that $\mathcal{A}$ has to target at least $t + 1$ intermediate variable to recover the secret. The first gadgets secure under the probing model where the addition and the multiplication over $\mathsf{GF}(2)$ (next generalized

Algorithm 1 Masked Boolean addition with d shares.

Input: Sharings $\boldsymbol{a}$, $\boldsymbol{b}$
Output: Sharing $\boldsymbol{c}$ such that $c = a \oplus b$

for $i = 0$ to $d-1$ **do**
  $\boldsymbol{c}_i \leftarrow \boldsymbol{a}_i \oplus \boldsymbol{b}_i$

Algorithm 2 ISW multiplication with d shares.

Input: Sharings $\boldsymbol{a}$, $\boldsymbol{b}$
Output: Sharing $\boldsymbol{c}$ such that $c = a \wedge b$

for $i = 0$ to $d-1$ **do**
  for $j = i+1$ to $d-1$ **do**
    $r_{ij} \xleftarrow{\$} \mathbb{F}_2$
    $r_{ji} \leftarrow r_{ij}$
for $i = 0$ to $d-1$ **do**
  $\boldsymbol{c}_i \leftarrow \boldsymbol{a}_i \wedge \boldsymbol{b}_i$
  for $j = 0$ to $d-1$; $j \neq i$ **do**
    $\boldsymbol{c}_i \leftarrow \boldsymbol{c}_i \oplus (\boldsymbol{a}_i \wedge \boldsymbol{b}_j) \oplus r_{ij}$

to any finite field in [RP10]) that are respectively presented in Algorithm 1 and Algorithm 2. While addition has a cost that follows a linear trend with the number of shares, the multiplication' overheads follow a quadratic trend with the number of shares: more specifically, it requires d^2 multiplications and $d(d-1)/2$ random bits. The latter are necessary to preserve security: without them, we would obtain the shares values $\boldsymbol{c}_i = \boldsymbol{a}_i \wedge b$ depending on the operand b which goes against the principle of masking (and breaks probing security).

2.4.2 Re-keying

Re-keying Instead of modifying the way in which the operations of an existing primitive are implemented in order to integrate a countermeasure, leakage-resilient cryptography aims to design primitives with inherently stronger side-channel security properties. Their security is based on physical hypotheses that enable a more formal assessment of the practical feasibility of an attack. To this end, an intuitive observation is that repeated manipulations of a long-term key within a primitive' execution come with limitations in terms of security because the adversary has the ability to combine multiple leakages related to the key value. To address this issue, the re-keying strategy consists in replacing the use of a fixed long-term key by temporary keys, which are used for a few of operations. The amount of data used with a single key is

then reduced, limiting the opponent's ability to combine several distinct leakages. In practice, the temporary keys (or session keys) are derived from the long-term key while minimizing its usage (e.g., by generating session keys recursively based on the long-term key as the initial value).

Fresh re-keying Closely related to re-keying, the concept of *fresh* re-keying was introduced by Medwed et al. [Med+10] as an alternative to circumvent the inherent limitations of masking when trying to protect an encryption function (e.g., a block cipher). Its underlying idea is to leverage a separation of duties between a *re-keying function* RK, that is easy to protect against side-channel attacks (i.e., embedding security against DPA) and the encryption function processing the message. The re-keying function is only used to produce a fresh session key k^* each time a message block need to be processed and takes as inputs the (fixed) long term key k and a public random nonce r, a depicted in Figure 2.1 for a block cipher case. In such a case, the encryption function only requires security against SPA (similarly to what was achieved with re-keying). From a more top level point of view, this strategy is generic and does not imply modification of the algorithm to protect, but is more akin to a wrapper applied on top of the latter and rather implies some tweaks when integrating it.

In the original proposal, the specifications of the function are heuristic and can be summarized by the following two informal guidelines: it should be easy to implement securely (i.e., cheaper than directly protecting the encryption function), and it should be computationally challenging to guess bits of k from k^*. Regarding the protection against side-channel attacks, the authors suggest exploiting key-homomorphic re-keying function to reduce the computational overheads of masked implementation. In particular, they highlight that modular multiplication can be implemented in a masked manner with overhead following a linear trend.

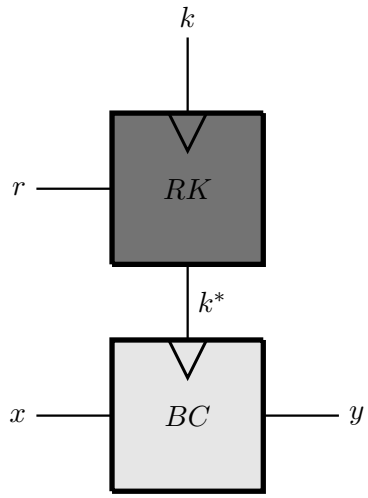


Figure 2.1: Fresh re-keying for a block cipher

3 Physical security in Symmetric Cryptography

Symmetric cryptography refers to the algorithm using a secret key that is shared among different parts. The diversity of applications in which they are deployed makes them a prime target for side-channel attacks and the development of secure implementation against the latter has been an active research topic for the last decades. Interestingly in this matter, the development of low-level side-channel countermeasure and the one of cryptographic primitives tailored to prevent leakage have followed quite independent paths. This can, in part, be explained by the very different abstraction used to analyze them. While low level countermeasures are typically evaluated experimentally based on statistical metrics [Sta16], proving the security of the aforementioned symmetric algorithms against leakages is rather based on cryptographic reductions leveraging some physical assumptions.

The research for cryptographic primitives with inherently improved security against physical leakage started with heuristic proposals such as [Koc03; Pet+08; Med+10; GFM14]. It was then formalized by Dziembowski and Pietrzak under the framework of leakage resilient cryptography [DP08], which has been the inspiration of many follow-up works and designs. Proposals for cryptographic primitives such as simple Pseudo Number Generators (PRGs), stream ciphers, Pseudo Random Function (PRFs) and Pseudo Random Permutation (PRPs) followed [Pie09; Yu+10; YS13; SPY13; Sta+10; FPS12; ABF13; DP10]. Concretely, such leakage-resilient primitives typically require some type of bounded leakage assumption, which is easier to fulfill for primitive that are naturally amenable to key evolution scheme than other where each execution requires the manipulation of a long-term secret key such as performed with PRF or PRP [BGS15]. The concept of leveled implementations introduced by Pereira et al. closely followed this finding: it aims at combining the minimum use of PRF or PRP that is heavily protected against side-channel attack relying on (possibly expensive and leading to poor performances) hardware-level (and/or implementation-level) countermeasure with other primitives requiring much fewer protections (or

even no protection at all) that are used to process the bulk of the computation [PSV15].

These seed results on basic cryptographic primitives (such as PRGs, PRFs and PRPs) triggered analyzes of complete functionalities like encryption and authentication schemes that were naturally followed by Authenticated Encryption (AE) mode of operations mixing both guarantees. This point is further supported by the latest algorithm AE lightweight standardization initiative from NIST, which stipulated resistance to side-channel attacks as the second evaluation criterion in its call for algorithms [SN17]. Strong integrity properties have been investigated, considering only the presence of leakage during encryption [Ber+18] or both encryption and decryption [Ber+17]. It turns out that such properties can be satisfied with weak physical assumptions for the bulk computation, which motivated a systematic analysis of composite security definition, enabling different physical requirement for integrity and confidentiality guarantees with leakage [Guo+19].

All taken into account, the quest for sound physical assumptions that are at the same time realistic (e.g., falsifiable and can be quantified by evaluation laboratories) and sufficient for proofs is of particular interest [KR19]. Combined with the massive amount of definitions capturing all the possible combinations of security targets for confidentiality and integrity with leakage [Guo+19], the complexity of these theoretical investigations is therefore calling for a systematization effort towards the concrete interpretation of leakage security proofs, in order to determine how these results can help developers in the design of secure and efficient implementations.

In this respect, this chapter presents a simplified framework allowing to focus on a reduced amount of relevant security targets. Besides, it also allows translating the physical assumptions used in the leakage security proofs into practical guidelines, stated in terms of security against Simple Power Analysis (SPA) and Differential Power Analysis (DPA), as can be analyzed with state-of-the art tools such as [CRR02; SLP05] and their extensions. Despite SPA and DPA requirements are only necessary conditions for the formal physical security assumptions to hold, this analysis allows pinpointing the minimum level of efforts that implementers must devote to protect different parts of an AE implementation in function of the security target. This framework is used to illustrate that reasoning about security against leakage can be viewed as a trade-off between mode-level and implementation-level (or hardware-level) protection. To give a simple example, a physical security property (e.g., the aforementioned ciphertext integrity with leakage) can theoretically

be obtained from standard modes of operation like OCB [RBB03], given that all the block cipher executions in the mode are strongly protected against DPA (which has high cost). Modes with better resistance against leakage can relax this DPA security requirement for certain parts of the implementations. Interestingly, the literature already includes different modes of operation corresponding to different leakage security targets. This allows us to illustrate the physical security trade-off based on actual algorithms, including candidates to the NIST LWC standardization process[1]. Eventually, the analysis is completed with a discussion about the interest of leveled implementations compared to uniformly protected implementations based on hardware implementation case study from an energy point of view.

In this chapter, my contribution has primarily focused on the experimental part and the concrete trade-off evaluations in Section 3.3. Namely, I was involved in the design and the implementation of a toy hardware version of Spookv1 and participated to the experimental setup used as well as the performances evaluations.

3.1 Simplifying framework

In this section, we present the simplifying framework that we will use in order to reason about leakage-resistant AE modes based on concrete (SPA and DPA) attacks. For this purpose, we first propose an informal decomposition of the modes that we will consider, and then list the design tweaks that such modes can leverage in order to reach different leakage security guarantees.

3.1.1 Leakage-resistant AE modes decomposition

We decompose the AE modes of operation under investigation into four informal parts, as illustrated in Figure 3.1 for a simple Inner Keyed Sponge (IKS) design [Ber+11]. An optional Key Generation Function (KGF) is used to generate a fresh key K^* based on a long-term master key K and a nonce N. Next, the message processing part uses the (optionally fresh) key in order to encrypt the message blocks. A Tag Generation Function (TGF) finally uses the result of the message processing part and outputs a tag for message authentication. The tag is only verified (i.e., compared to the genuine tag) in case of decryption.

No claim is made regarding the generality of this decomposition. As will be clear next, it nicely matches a number of recent AE schemes with

[1] `https://csrc.nist.gov/projects/lightweight-cryptography`

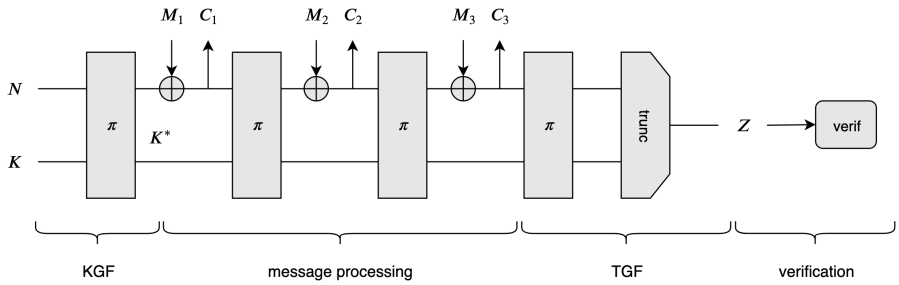


Figure 3.1: Exemplary decomposition of an AE scheme.

different levels of security against side-channel attacks, but other modes may not be directly adaptable to this (simple) framework. This limitation seems however natural when trying to specialize leakage-resistance analyzes for practical AE schemes. It also has to be noted the associated data are ignored in the discussions, since it has limited impact on leakage analyzes.

3.1.2 Design tweaks and security levels

The main design tweaks that enhance mode-based side-channel security are:

1. *Key evolution.* As formalized by Dziembowski and Pietrzak, updating the ephemeral keys of an implementation so that each of them is only used – and therefore leaks – minimally can improve confidentiality with leakage [DP08].

2. *Strengthened KGF and TGF.* As formalized by Berti et al., employing key and tag generation functions in such a way that the computation of their output (resp., input) given their input (resp., output) is not straightforward can enhance security with leakage [Ber+18; Ber+17]. This can for instance be achieved by preventing that recovering an ephemeral secret during message processing leads to the long-term master key.

3. *Two-pass designs.* As formalized by Guo et al. (resp., Barwell et al.) in the leakage-resistance setting [Guo+19] (resp., leakage-resilience setting [Bar+17b]), two-pass modes can improve message confidentiality with decryption leakages, if the tag verification does not require the manipulation of sensitive plaintexts.

Based on these three main design tweaks, we select a number of practically-relevant security targets that reflect existing AE schemes from the leakage-

resistance definition zoo of [Guo+19]. For this purpose, we first recall Guo et al. 's definitions of integrity and confidentiality with leakage in an abstracted way, which will be sufficient to guide our attack-based analysis in the next section.

Definition 1 (Ciphertext Integrity with Leakage [Ber+17], Informal.)**.** *In the ciphertext integrity with leakage security game, the adversary can perform a number of queries to encryption and decryption oracles enhanced with leakage functions, that capture the implementation of an AE scheme. His goal is to produce a valid fresh ciphertext and the implementation is considered secure if the adversary cannot succeed with good probability. Variants that we will use next include: ciphertext integrity with (nonce-respecting adversary and) leakage in encryption only (CIL1) and ciphertext integrity with misuse-resistance (i.e., no constraint on nonce) and leakage in encryption and decryption (CIML2).*

Definition 2 (Confidentiality with Leakage [Guo+19], Informal.)**.** *In the Chosen Ciphertext Attack (CCA) with leakage security game, the adversary can perform a number of queries to encryption and decryption oracles enhanced with leakage functions, that capture the implementation of an AE scheme. During a so-called "challenge query", he picks up two fresh messages X_0 and X_1 and receives a ciphertext Y_b encrypting X_b for $b \in \{0, 1\}$, with the corresponding leakage. His goal is to guess b and the implementation is considered secure if the adversary cannot succeed with good advantage. Variants that we will use next include: chosen ciphertext security with nonce-respecting adversary and leakage in encryption only (CCAL1), chosen ciphertext security with nonce misuse-resilience (i.e., fresh challenge nonce) and leakage in encryption only (CCAmL1) and chosen ciphertext security with nonce misuse-resilience and leakage in encryption and decryption, including decryption of the challenge Y_b (CCAmL2).*

In our notations, small caps are for resilience to misuse or leakage and capital letters for resistance.

Based on these definitions, we list our security targets and their link with the aforementioned design tweaks. We insist that these links hold for the AE schemes investigated in the next section. We do not claim they are necessary to reach the security targets and admit other design ideas could be leveraged. We further reckon a finer-grain analysis may be useful in order to analyze other modes.

- **Grade-1a.** CIL1 and CCAL1 security thanks to key evolution.

- **Grade-1b.** CIML2 security thanks to strengthened KGF and TGF (and only black box security guarantees for message confidentiality).

- **Grade-2.** CIML2 and CCAmL1 security thanks to a combination of key evolution (i.e., Grade-1a) and strengthened KGF and TGF (i.e., Grade-1b).

- **Grade-3.** CIML2 and CCAmL2 security thanks to a combination of key evolution and strengthened KGF and TGF with a two-pass design.

We also denote as **Grade-0** the AE schemes without leakage-resistant features.

3.2 From leakage-resistance to side-channel security

In this section, we first discuss how the physical assumptions used in leakage security proofs can be translated into minimum requirements for implementers. Next, we illustrate what are these minimum requirements for concrete instances of existing AE schemes. From the current literature, we identify Grade-0, Grade-1a, Grade 2 and Grade-3 AE schemes, which suggests the design of an efficient Grade-1b instance as an interesting scope for further investigations. We show that the minimum requirements suggested by security proofs are (qualitatively) tight and that failing to meet them leads to realistic SPA and DPA attacks.

3.2.1 Translating physical assumptions into implementation goals

Leakage security proofs for AE schemes rely on physical assumptions. As mentioned in introduction, the quest for sound and realistic assumptions is still a topic of ongoing research. In this section, we observe that the (sometimes strong) assumptions needed in proofs can be translated into minimum security requirements, expressed in terms of security against practical side-channel attacks.

Integrity with leakage requirements can be limited to the KGF, TGF (and optionally verification functions) for AE schemes with good leakage properties, and are extended to all the components of a mode of operation without such good properties (the detailed formal analysis

can be found in the original paper [Bel+20a]). The simplest assumption is to consider the underlying blocks to be leak-free [PSV15]. A recent relaxation introduces a weaker requirement of unpredictability with leakage [Ber+19]. In both cases, these assumptions need that implementations manipulating a long-term key limit the probability of success of key-recovery DPA attacks, which we express as:

$$\Pr\left[\mathcal{A}_{\mathsf{kr}}^{\mathsf{L}(.,.)}(X_1, \mathsf{L}(X_1, K), \ldots, X_q, \mathsf{L}(X_q, K)) \rightarrow K | K \xleftarrow{\text{u}} \{0,1\}^n\right] \approx 2^{-n+q\cdot\lambda(r)}, \tag{3.1}$$

where $\mathcal{A}_{\mathsf{kr}}^{\mathsf{L}(.,.)}$ is the key recovery adversary able to make offline calls to the (unkeyed) leakage function $\mathsf{L}(.,.)$, $X_1, \ldots, X_q$ the different inputs for which the primitive is measured, K the long-term key of size n bits and $\lambda(r)$ the (informal) amount of information leakage obtained for a single input X_i measured r times (i.e., repeating the same measurement multiple times can be used to reduce the leakage noise). For security against DPA, it is required that this probability remains small even for large q values, since there is nothing that prevents the adversary to measure the target implementation for many different inputs. Such DPA attacks reduce the secrecy of the long-term key exponentially in q. Hence, preventing them requires a mechanism that counteracts this reduction. For example, masking can be used for this purpose and makes $\lambda(r)$ exponentially small in a security parameter (i.e., the number of masks or shares) [Cha+99; DFS15].

Confidentiality with leakage requirements are significantly more challenging to nail down. For the KGF and TGF parts of the implementation, they at least require the same DPA security as required for integrity guarantees. For the message processing part, various hypothesis exist in the literature (e.g., *Only computation leaks assumption and bounded leakage* [DP08], *Oracle-free and hard-to-invert leakage function* [Yu+10] or *Simulatability* [SPY13]) aiming to capture the same

intuition that an ephemeral key manipulated minimally also leaks in a limited manner, preserving some of its secrecy. As a result, they share the minimum requirement that the probability of success of a key-recovery SPA attack remains small. Such a probability of success has a similar expression to the one of Equation 3.1, with as only (but paramount) difference that the number of inputs that can be measured is limited by design. Typically, $q = 2$ for leakage-resilient stream ciphers where one input is used to generate a new ephemeral key and the other to generate a key stream to be XORed with the plaintexts [Pie09; Yu+10; YS13; SPY13].

Note that because of these limited q values, the possibility to repeat measurements (by increasing the r of the leakage expression $\lambda(r)$) is an important asset for SPA adversaries. As will be detailed next, this creates some additional challenges when leakages can be obtained with nonce misuse or in decryption.

Besides the aforementioned requirements of security against key-recovery DPA and SPA, definitions of leakage-resistance provide the adversary with the leakage of the challenge query. In this setting, another path against the confidentiality of an AE scheme is to directly target the manipulation of the message blocks (e.g., in Figure 3.1, this corresponds to the loading of the M_i blocks and their XOR with the rate of the sponge). Following [Sta19], we express the probability of success of such Message Comparison (MC) attacks with:

$$\Pr\left[\mathcal{A}_{\mathsf{mc}}^{\mathsf{L}(.,.)}(X_0, X_1, \mathsf{L}(X_b, K)) \rightarrow b | K \xleftarrow{\text{u}} \{0,1\}^n, b \xleftarrow{\text{u}} \{0,1\}\right]. \quad (3.2)$$

In this case, the adversary has to find out whether the leakage $\mathsf{L}(X_b, K)$ is produced with input X_0 or X_1. This type of attack depends mainly on the similarities between the leakage values of the two messages and may be easier to carry out than a key recovery (e.g., worst-case leakage considering the Hamming weight leakage function). So implementers have to deal with the goal to minimize the message leakage with lower-level countermeasures, but trying to minimize the manipulation of sensitive messages is a desirable feature.

Summary. The heuristic security requirements for the different parts of an AE scheme with leakage (following the decomposition of Section 3.1.1) include:

- **For integrity guarantees:**
  - *For the KGF and TGF:* security against (long-term key) DPA.
  - *For the message processing part:* security against (ephemeral key) DPA or no requirements (i.e., full leakage of ephemeral device states).
  - *For tag verification:* security against DPA or no requirements.
- **For confidentiality guarantees:**
  - *For the KGF and TGF:* security against (long-term key) DPA.
  - *For the message processing part:* security against (ephemeral key) DPA or (ephemeral key) SPA and security against MC attacks.

As detailed next, for some parts of some (leakage-resistant) AE schemes, different levels of physical security are possible, hence enabling leveled implementations. For readability, the discussions will be illustrated with the following color code: blue for the parts of an AE scheme that require DPA security, light (resp., dark) green for the parts of an AE scheme that require SPA security without (resp., with) averaging, light (resp., dark) orange for the parts of an AE scheme that require security against MC attacks without (resp., with) averaging and white for the parts of an AE scheme that tolerate unbounded leakages. We draw the tag verification in light gray when it is not computed (i.e., in encryption).

We note that precisely quantifying the implementation overheads associated with SPA and DPA security is highly implementation-dependent, and therefore beyond the scope of this chapter. For example, the number of shares necessary to reach a given security level in software (with limited noise) may be significantly higher than in (noisier) hardware implementations [DFS15].

Yet, in order to provide the reader with some heuristic rule-of-thumb, we typically assume that preventing SPA implies "small" overheads [Vey+12] while preventing DPA implies "significant" overheads [GR17; GMK17]. In fact, the only requirement for the following reasoning to be practically-relevant is that enforcing SPA security is significantly cheaper than enforcing DPA security, which appear to be supported by the literature for the hardware case. There are several possible explanations: the first is the level of physical noise that seems to be inherently higher in hardware targets [BUS21]. This can be amplified by algorithmic noise from the usually more parallel architectures of specialized coprocessors (aiming for efficiency). These also have the impact of reducing the amount of leakage points available, thus limiting the application of multivariate model or horizontal attacks [Bat+16; BS21].

In the rest of this section, we illustrate the taxonomy of Section 3.1.2 with existing modes of operation. For each grade, we first exhibit how leakage-resistance translates into leveled implementations and discuss the results as well as their applicability to other ciphers. [2]

3.2.2 Grade-0 case study: OCB-Pyjamask

Mode-level guarantees. The OCB mode of operation does not provide mode-level security against leakage. This is due to the use of the same long-term key in all the (T)BC invocations of the mode. As a

[2] Interested readers can find the formal security analysis in the original publication, which not covered in this thesis. [Bel+20a].

result, all the (T)BC calls must be strongly protected against DPA. For completeness, a uniformly protected implementation of OCB-Pyjamask is illustrated in Figure 3.2.

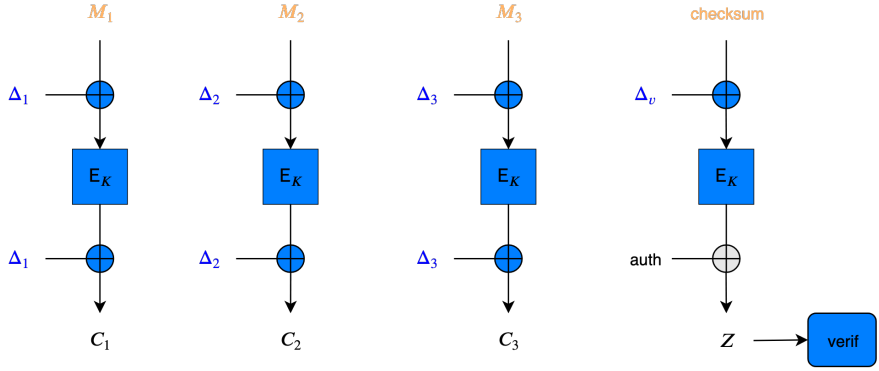


Figure 3.2: OCB, leveled implementation.

Discussion. As mentioned in the chapter introduction and will be further illustrated in Section 3.2.6, the absence of mode-level leakage-resistance does not prevent a mode of operation to be well protected against side-channel attacks: it only implies that the protections have to be at the primitive, implementation or hardware level. In this respect, the Pyjamask cipher is better suited to masking than (for example) the AES and should allow efficient masked implementations, thanks to its limited multiplicative complexity. A similar comment applies to the NIST lightweight candidates SKINNY-AEAD and SUNDAE-GIFT.

3.2.3 Grade-1a case study: PHOTON-Beetle

Mode-level guarantees. As detailed in [Bel+20a], PHOTON-Beetle is CCAL1 and CIL1 under physical assumptions that its KGF is leak-free, the leakage of the unprotected permutation calls are non-invertible and that the leakages of the XOR operations are bounded. As discussed in Section 3.2.1, such hypotheses translate into SPA security requirements for the message processing part, and DPA security for the KGF permutation call. Therefore, it can be implemented in a leveled fashion as illustrated in Figure 3.3. Note that nonce repetition is prevented in the CCAL1 and CIL1 security games, which explains the light gray and orange color codes (for SPA security without averaging).

Proofs' qualitative tightness. As soon as nonce misuse or decryption leakages are granted to the adversary, the following DPA becomes possible against the message processing part of PHOTON-Beetle: fix the

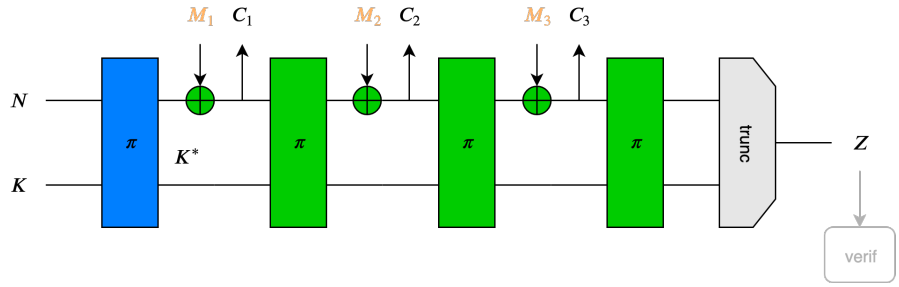


Figure 3.3: PHOTON-Beetle, leveled implementation, CCAL1, CIL1.

nonce and the ephemeral key K^*; generate several plaintext (or ciphertext) blocks M_1 (or C_1); recover the capacity part of the state including M_1 (or C_1) and finally inverse the permutation to recover the long-term key K. The number of plaintext/ciphertext blocks that can be measured in this way equals 2^r, where r (i.e., the rate of the sponge) equals 128 in the case of PHOTON-Beetle. This is (considerably) more than needed to perform a successful side-channel attack against a permutation without DPA protections [MOP07]. Hence, we conclude that for any stronger security target than CCAL1 and CIL1, uniform protections are again needed.

Discussion. A similar analysis applies to many IKS designs in the literature, for example the NIST lightweight candidates Gimli and Oribatida and the CAESAR competition candidate Ketje. It formalizes the intuition that when encrypting without misuse, it is not necessary to protect the message processing part of IKS modes as strongly as their KGF. But this guarantee vanishes with nonce misuse or decryption leakage because it is then possible to control the ephemeral keys and the KGF is invertible. Hence, for stronger security targets, lower-level countermeasures have to be uniformly activated, the cost of which again depends on the structure (e.g., multiplicative complexity) of the underlying primitives.

3.2.4 Grade-2 case studies: Ascon and Spook

Mode-level guarantees. Ascon and Spook are CCAmL1 and CIML2 under different sets of physical assumptions for confidentiality and integrity guarantees [Bel+20a]. They represent interesting case studies where the composite nature of Guo et al. 's security definition enables different practical requirements for different parts of a mode and different security targets. We note that the previous requirement of DPA security for the KGF and TGF cannot be relaxed, so it will be maintained in this and the next subsection, without further discussion. By contrast, the security requirements for the message processing and tag verification parts can significantly vary, which we now discuss.

We start with Ascon's CCAmL1 requirements, illustrated in Figure 3.4. They translate into SPA security (without averaging) for the message processing part even with nonce misuse. This is because (*i*) in the

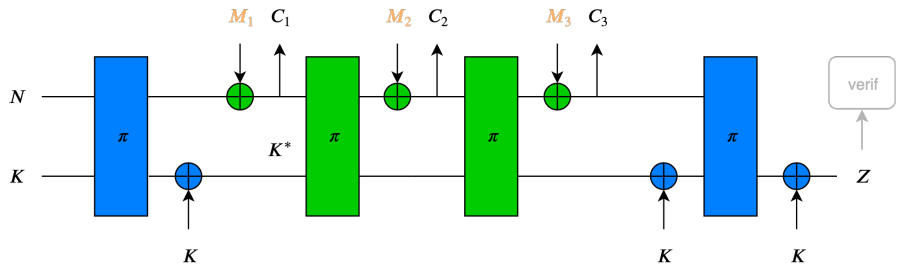


Figure 3.4: Ascon, leveled implementation, CCAmL1.

misuse-resilience setting, the challenge query of Definition 2 comes with a fresh nonce (i.e., nonce misuse is only granted during non-challenge queries), and (*ii*) even a full permutation state leakage obtained for non-challenge queries (e.g., thanks to the same DPA as described against PHOTON-Beetle) does not lead to the long-term key K on which confidentiality relies (thanks to the strengthened KGF). A similar situation holds for Spook and is illustrated in Figure 3.5.

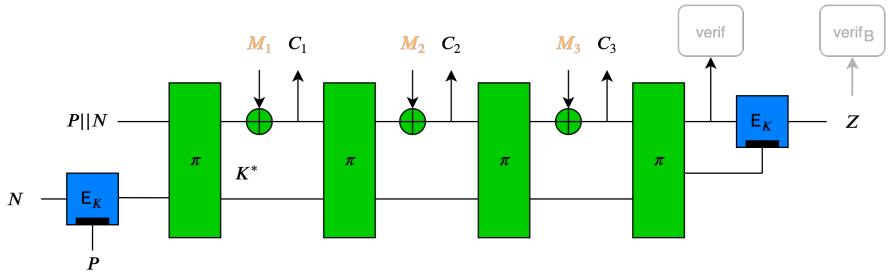


Figure 3.5: Spook, leveled implementation, CCAmL1.

We follow with Spook's CIML2 requirements, illustrated in Figure 3.6. The main observation here is that integrity is proven in a weak (unbounded leakage) model where all the intermediate permutation states are given in full to the adversary. This is possible thanks to the strengthening of the KGF and TGF which prevents any of these ephemeral states to leak about long-term secrets and valid tags. In the case of Spook, even the tag verification can be implemented in such a leaky manner

(thanks to the inverse-based verification trick analyzed in [Ber+17]). Optionally, a direct tag verification $\mathsf{verif}_\mathsf{B}$ can be used but requires DPA protections. A similar situation holds for the integrity of Ascon and is illustrated in Figure 3.7, with as only difference that it can only be implemented with a direct DPA-secure tag verification. Note that without an inverse-based or DPA-secure verification, it is possible to forge valid messages without knowledge of the master key [Ber+17], which is for example critical for secure bootloading [OC15]. The practical feasibility of the attack scenario has been demonstrated in [Bel+20a]

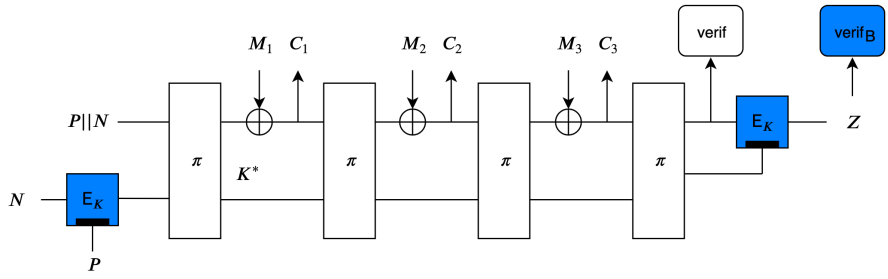


Figure 3.6: Spook, leveled implementation, CIML2.

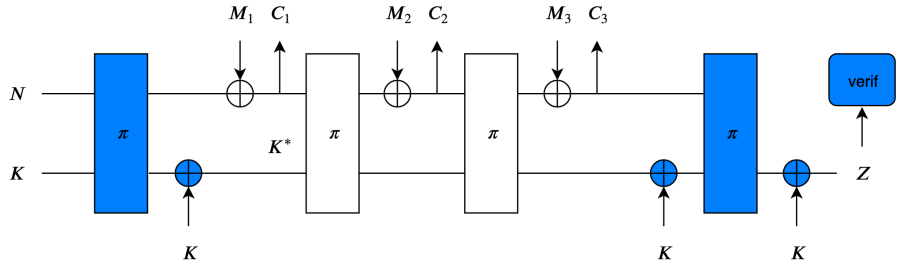


Figure 3.7: Ascon, leveled implementation, CIML2.

Proofs' qualitative tightness. From Figure 3.4 and 3.5, it can be observed that as soon as decryption leakages are granted to the adversary, a DPA attack against the confidentiality of the messages becomes possible. The beginning of the attack is exactly the same as the one against PHOTON-Beetle: fix the nonce and the ephemeral key K^*; generate several plaintext (or ciphertext) blocks M_1 (or C_1) and recover the

capacity part of the state including M_1 (or C_1). This time, the full state leakage cannot lead to the long-term key K but it still allows recovering all the decrypted messages in full. Note that this attack actually targets a weaker notion than CCAmL2 since it only exploits the decryption leakage, without access to the decryption oracle. Yet, as discussed in [Ber+17], it is a quite practical one in case of applications where IP protection matters (e.g., when a code or content can be decrypted and used but not copied).

Discussion. A similar analysis applies to other IKS designs with strengthened KGF and TGF, for example the NIST lightweight candidates ACE, WAGE and SPIX and the CAESAR competition candidate GIBBON. The TBC-based TET scheme also offers the same guarantees [Ber+20]. These designs reach the best integrity with leakage but due to their one-pass nature, cannot reach CCAmL2. The main concrete differences between Ascon and Spook are: (*i*) The KGF and TGF of Ascon are based on a permutation while Spook uses a TBC for this purpose, and (*ii*) the tag verification of Spook can be implemented in a leaky way with an inverse TBC call or in a DPA-protected way with a direct TBC call, while the tag verification of Ascon can only be implemented in the DPA-protected manner.

3.2.5 Grade-3 case studies: ISAP and TEDT

Mode-level guarantees. Leveled implementations of Ascon and Spook reach the highest security target for integrity with leakage (i.e., CIML2) but they are only CCAmL1 without uniform protections. ISAP and TEDT cross the last mile and their leveled implementations are proven CCAmL2 in [Bel+20a], while also maintaining CIML2 security. The integrity guarantees of ISAP and TEDT follow the same principles as Ascon and Spook and we therefore next focus on their practical requirements for confidentiality with decryption leakage.

We start with the ISAP design for which a leveled implementation is illustrated in Figure 3.8. For now skipping the details of the rekeying function RK which aims at providing "out-of-the-box" DPA security without implementation-level countermeasures such as masking, the main observation is that ISAP is a two-pass design where the tag verification does not require manipulating plaintext blocks. Hence, as long as the KGF, TGF (instantiated with RK) and the default tag verification are secure against DPA, the only attack paths against the confidentiality of the message are a SPA against the message processing part and a MC

attack against the manipulation of the plaintext blocks. In both cases, averaging is possible due to the deterministic nature of the decryption.

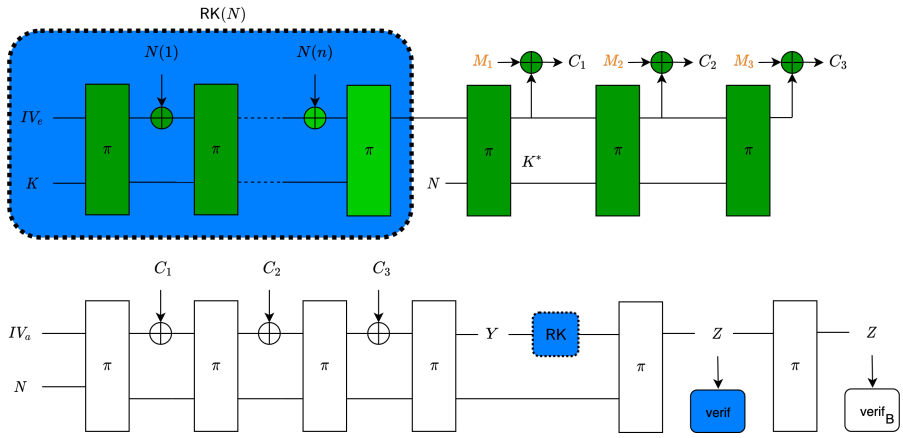


Figure 3.8: ISAP, leveled implementation, CCAmL2.

The default tag verification of Figure 3.8 must be secure against DPA. An exemplary attack path that becomes possible if this condition is not fulfilled is the following: given a challenge ciphertext (C, Z), flip some bits of C leading to related ciphertexts $C', C'', \ldots$ (which, due to the malleability of the encryption scheme, correspond to messages $M', M'', \ldots$ with single bits flipped compared to the target M); forge valid tags for these ciphertexts thanks to the leaking message comparison (as experimentally validated in [Bel+20a]) and finally perform a DPA against M thanks to the related messages' decryption leakage, which breaks the SPA requirements guaranteed by the proofs and leads to the same (practical) IP protection issue as mentioned for Ascon and Spook.

Alternatively, ISAP also comes with a tag verification that provides similar guarantees as Spook's inverse one at the cost of another permutation call.

TEDT's CCAmL2 requirements, illustrated in Figure 3.9, are mostly identical: the only difference is that the RK function is replaced by a TBC which must be secure against DPA thanks to masking or other low-level countermeasures, and optionally enables an inverse-based tag verification.

Discussion. TEDTSponge, a sponge-based variant of TEDT with similar guarantees, is proposed in [Guo+20a]. Besides their DPA resistant tag verifications, the main difference between ISAP and TEDT is their KGF and TGF.

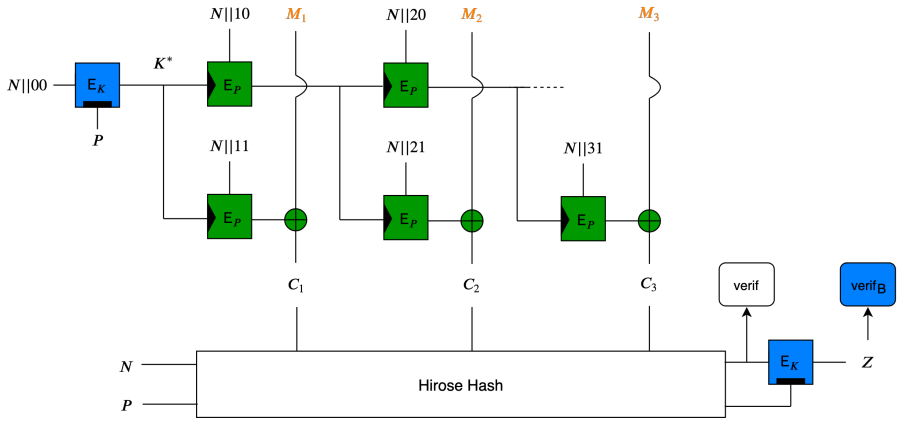


Figure 3.9: TEDT, leveled implementation, CCAmL2.

3.2.6 Summary

The investigations of this section support the conclusion that security against side-channel attacks can be viewed as a trade-off between mode-level and implementation-level (or hardware-level) protections. In general, even the highest security targets (i.e., CCAmL2 and CIML2) can be reached by modes without any leakage-resistance features like OCB, but then require strong low-level countermeasures for all the implementation parts. As the security targets and the quantitative security levels needed by an application increase, it is expected that leveled implementations will lead to better performance figures, which will be further analyzed in the next section (and in the next chapters).

3.3 Design choices and concrete evaluations

The framework of Section 3.1 allowed us to put forward a range of AE schemes with various levels of leakage-resistance in Section 6.3. These modes of operation leverage a combination of design ideas in order to reach their security target. In this section, we analyze a concrete question related to benefits enabled by leveled-implementation.

3.3.1 Uniform vs. leveled implementations

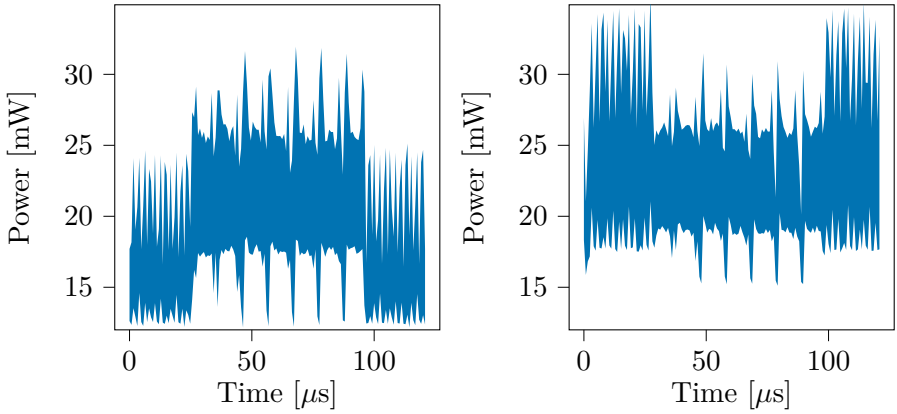
Research question. One important pattern shared by several designs analyzed in the previous section is that they can enable leveled implementations where different parts of the designs have different levels of security against side-channel attacks. This raises the question whether and

to what extent such leveled implementations can be beneficial. In software, it has already been shown in [Ber+20] that gains in cycles can be significant and increase with the level of security and message size. We next question whether the same observation holds in hardware, which is more tricky to analyze since enabling more speed vs. area trade-offs. Precisely, we investigate whether leveled implementations can imply energy gains (which captures the general design's efficiency [Ker+12]).

Experimental setting. In order to investigate the energy efficiency aspects of leveled implementations in presence of side-channel protections, we have designed FPGA and ASIC leveled implementations of Spook (a detailed similar architecture will be provided in 4). We applied the masking countermeasure with $d = 2$ and $d = 4$ shares (with the HPC2 masking scheme [Cas+20a]) to the Clyde 128-bit TBC (used as KGF and TGF in Spook), and no countermeasure to the Shadow 512-bit permutation. The FPGA implementations have been synthesized, tested and measured on a Sakura-G board, running on a Xilinx Spartan-6 FPGA at a clock frequency of 6 MHz. As a case study, we encrypted a message composed of one block of authentication data and six blocks of plaintext. ASIC implementations have been designed using *Cadence Genus 18* with a commercial 65 nm technology, and the power consumption has been estimated *post-synthesis*, running at a clock frequency of 333 MHz.

Experimental results. The power consumption versus time of the Spook FPGA implementation is shown in Figure 3.10. Its main phases can be recognized: first, the Clyde KGF, then the 7 Shadow executions (the two first consuming less power correspond to the one after the KGF and the AD processing) and finally the Clyde TGF. The power consumption of Shadow is independent of the masking order d, while the one of Clyde increases with d. The figure intuitively confirms the energy gains that leveled implementations enable when higher-order masking is required. We note that larger architectures for Clyde would not change the picture: latency could be reduced down to 48 cycles but this would cause significant area and dynamic power consumption increases.

We confirm this intuitive figure with performance results for the ASIC implementations of Spook. For this purpose, we have extracted energy estimations for one execution of a 32-bit (i.e., 8 S-boxes and 1 L-box) architecture of Clyde (about 3.4 nJ for $d = 2$ and 8.1 nJ for $d = 4$) and one execution of (unprotected) Shadow (about 1.2 nJ) independently, in order to easily study the contributions of the two primitives. Assuming that only the execution of the primitives consumes a significant amount of energy, we can then estimate the energy consumption per byte for both Spook (i.e., 2 Clyde executions and $n + 1$ Shadow executions where

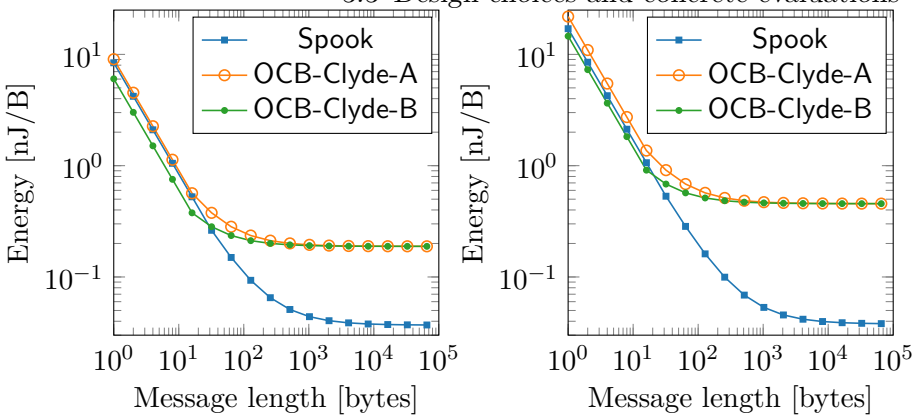


(a) $d = 2$ (b) $d = 4$

Figure 3.10: Power consumption of a leveled implementation of Spook with a masked Clyde on a FPGA (Xilinx Spartan 6) at clock frequency $f_{CLK} = 6\,\text{MHz}$, for the encryption of one block of associated data and 5 blocks of message. For this experiment, Clyde performs in 156 cycles (4 S-boxes, 1 L-box) and Shadow performs in 48 cycles (128-bit architecture)

n is the number of 32-byte message blocks) and OCB-Clyde-A (resp., OCB-Clyde-B), assuming $n + 2$ (resp., $n + 1$) Clyde executions (where n is the number of 16-byte message blocks). The OCB mode was used as an example of Grade-0 mode, and we used the Clyde TBC in order to have a fairer comparison between the modes. The A (resp., B) variant models the case where the OCB initialization is not amortized (resp., amortized) over a larger number of encryptions. The estimated energy per byte encrypted on ASIC is shown in Figure 3.11. For short messages (at most 16 bytes) and for both masking orders, OCB-Clyde-B consumes the least (with 2 Clyde executions), followed by Spook (2 Clyde and 2 Shadow executions), and OCB-Clyde-A is the most energy-intensive (with 3 Clyde executions). For long messages, both OCB-Clyde-A and -B converge to 1 Clyde execution per 16-byte block, while Spook converges to 1 Shadow execution per 32-byte block. In that scenario, a leveled implementation of Spook is therefore much more energy-efficient than OCB (e.g., 5 times more efficient for $d = 2$, and the difference increases with d).

Discussion. Both the FPGA measurements and the ASIC synthesis results confirm that leveled implementations can lead to significant energy gains for long messages. This derives from the fact that the energy



(a) $d = 2$ (b) $d = 4$

Figure 3.11: Energy consumption of leveled Spook and uniform OCB-Clyde implementations on ASIC (65 nm technology), in function of the message length.

per byte of a protected primitive is larger than for a non-protected one. More interestingly, even for short messages our results show that leveled implementations can be beneficial. For example, Spook requires only two TBC executions. Hence, it is always more energy-efficient than OCB, excepted when the OCB initialization is perfectly amortized, and the message length is less than 16 bytes. Furthermore, even in this case, the Spook overhead is only 35% for $d = 2$ and 15% for $d = 4$.

These energy gains come at the cost of some area overheads since the leveled nature of the implementations limits the possibility to share resources between their strongly and weakly protected parts. In the case of Spook studied in this section, the total area requirements of a 2-share (resp., 4-share & 8-share) leveled implementation is worth 53,897 (resp., 90,311 & 191,214) μm^2. The Shadow-512 part of the implementation is only 22% of this total for the 2-share implementation and decreases to 13% (resp., 6%) with 4 shares (resp., 8 shares).[3]

3.3.2 Forward vs. inverse tag verification

As demonstrated in [Bel+20a], a practical DPA can be mounted against a software unprotected tag verification, which may lead to forgery. A similar attack will also be demonstrated in Chapter 6, supporting the need to evaluate the trade-off between implementing a protected tag ver-

[3] All the results in this subsection include the cost of the PRNG used to generate the shares (we used a 128-bit LFSR) and the area costs include the interface.

ification or an inverse based tag verification. To this end, we evaluate the cost of protecting the tag verification against DPA with masking. Masking the XNOR operations is cheap since it is an affine operation. Masking the AND operations is more costly and implies performing 127 two-input secure AND gates (for a 128-bit tag), with a multiplicative depth of at least 7. The overall cost can be estimated as about 20% in hardware area (compared to Clyde 32-bit architecture of Section 3.3.1). The second method is only applicable to TBC-based TGF. It computes the inverse of the TBC on the candidate tag, allowing a secure comparison with unbounded leakage. Being unprotected, the comparison is cheap but the inverse leads to overheads. For example, for the Clyde TBC, the inverse does not change the execution time, but increases the hardware area by 24%. We conclude that protecting the tag verification leads to limited overheads in front of other implementation costs, with a (mild) simplicity and latency advantage for the inverse-based solution.

3.4 Concluding remarks

This chapter presents a top-level view of the security objectives targeted when implementing common symmetric cryptographic algorithms, namely the mode of operations. It shows that security against side-channel attacks can be viewed as a trade-off between mode-level and implementation-level (or hardware-level) protections. These take the form of two implementation guidelines, called the SPA and the DPA security, which aim to quantify the number of different observations allowed in order to carry out an attack against a particular primitive (the first typically considers this number to be low). The latter are in fact necessary conditions which have the advantage of being useful for setting up proof, but which can be quantified by evaluators as well.

Based on these guidelines, this chapter shows how security guarantees (i.e., confidentiality and integrity) can be obtained in the presence of leakage and misuse. In particular, it shows that the two can in fact be analyzed independently (allowing more genericity depending on the applications needs), and that the strongest guarantee of integrity (integrity with misuse resistance and leakage in encryption and decryption) can already be obtained for single-pass modes, which have the advantage of being more efficient. On the other hand, the strongest guarantee of confidentiality requires the use of a (less efficient) two-pass mode.

This chapter also demonstrates that the protected TGF (required to achieve integrity guarantees) can lead to small gain in latency compared

to the implementation of a protected tag verification, at the cost of a slight increase in surface area.

Moreover, it is shown that taking into account the leakage at the design phase can lead to performances improvement by relying on the leveled implementation. In particular, Section 3.3.1 demonstrates that such architectures can effectively reduce energy consumption compared with uniform protection for sufficiently long messages. This could be adapted to other performance metrics such as latency. Generally speaking, these architectures leave the door open to different implementation compromises when long messages have to be processed, focusing on the (weakly protected) primitive carrying out the bulk of the computation. However, the execution(s) of the secure primitive against DPA is no longer amortized for small messages, approaching almost the inefficient performances scenario with uniform protection.

As discussed in Section 3.2.1, it is often accepted that SPA security is easy to achieve for hardware implementations. In practice, this translates to the fact that primitives requiring SPA protection are not implementing dedicated countermeasures as long as they are implemented with a certain level of parallelism. While this may seem positive from an implementation cost perspective, it turns out that the practical cost of implementing the leveled mode is primarily composed of that of the DPA primitive when large levels of protection are used (e.g., Shadow-512 represents 13% of the total area with 4 shares in Section 3.3.1).

Taking these considerations into account, the following chapters focus on the efficient and cost-effective implementation of primitives protected against DPA attacks.

4 DPA security through masking

As introduced in Section 1.3, masking is a standard countermeasure against SCAs. Its underlying principle is to compute over secret-sharings with d shares that are individually independent of the intermediate variable manipulated by a device, which increases the security exponentially with d. To this end, all the internal states manipulated by an implementation are transformed into sharings and the corresponding operations are implemented using gadgets, which are subcircuits that perform the operations on the sharings in a secure manner.

Masking benefits from a strong theoritical background and several works have emerged over the past 20 years in order to formally assess masking security. These works build upon the original contribution that introduced the t-probing model introduced in Section 2.4.1 [ISW03; DDF14; DFS15; Bar+16]. and demonstrate its relevance when more natural leakage models are considered. Due to its simplicity, t-probing remains a commonly analyzed tool when evaluating the security of a masking scheme.

Probing model limitation and common implementation flaws Although the design of efficient masking schemes has been extensively studied, implementing an (efficient) cryptographic primitive in a masked fashion still incurs a cost following a quadratic trend and technical challenges. On the one hand, physical defaults considered are the glitches or the transitions that inherently occur into the implementation' circuitry during the manipulation of sequential intermediate variables. On the other hand, composition issues occur when the composition of t-probing secure gadgets leads to a circuit that is not t-probing secure anymore.

The absence of a formal security analysis in the presence of the aformentioned challenges has already led to security reductions at high-order (i.e., breaking the t-probing security) for existing schemes [Moo+19]. In order to provide an intuitive example about the issue induced by glitches, we consider a hardware instanciation of the ISW multiplication (Algorithm 2 in Section 2.4.1). In that specific case, considering that the circuitry layout is such that all randomness bits propagate slower than the shares of $\boldsymbol{a}$ and $\boldsymbol{b}$ in the combinational logic computing $\boldsymbol{c}_i$, the share

$\boldsymbol{c}_i$ is not independent of the unshared value b, which breaks the probing security.

To address this, the robust probing model emulates physical defaults as an extension of the set of intermediate variables obtained by a single probe [Fau+18]. More intuitively, it reflects the fact that the value manipulated by the probed wire is in practice subject to physical phenomena and may pass through various intermediate values before stabilizing. As a glitch propagates through the entire combinational logic and is stopped by a register, robust probing model emulates it with an extended probe such that the resulting observed wires include all the wires belonging to the combinational circuit computing the probed wire. For transitions, we extend a probe by considering that the value of the probed wire is observed for two consecutive cycles. Similarly, it is possible to extend probes while accounting for both defaults by applying the propagation rules successively.

Hardware Private Circuit (HPC) and PINI framework The HPC masking scheme was originally proposed as an arbitrary-order masking scheme with glitch robust probing security in [Cas+20a]. In addition to being glitch-robust, its gadgets are proven trivially composable at any order using the Probe Isolation Non Interference (PINI) framework [CS20]: that is, if all the gadget composing a circuit are PINI, the resulting circuit is PINI too. Its composability property is based on a simulatability framework. A set of probes is simulable if there is a way to simulate all the probed values by knowing certain inputs. In this context, simulation means being able to generate the probed values following the same joint distribution as if it was the real case. If such simulation is possible, it intuitively means that the probed values are independent of input values not known to the simulator. Its security has latter been extended to glitches+transitions robust security under some assumptions on the structure taken by the circuits: intuitively, a gadget cannot manipulate shares from the same sharing during two consecutive clock cycles, otherwise the transition would have the effect of recombining them. A solution to circumvent the issue is to implement the adjacent executions in parallel.

For HPC masking schemes, the implementation of linear gadgets (e.g., XOR) are done sharewise (similarly to the boolean masked addition presented in Section 2.4.1, Algorithm 1) and can be implemented only relying on combinational logic (i.e., no sequential gate is required). As often encountered with masking scheme, the implementation of the non-linear gadgets (e.g., AND) is more complex due to the cross product

mixing differents inputs shares. The design of such gadgets is an active research area and different multiplication gadgets have been proposed. They typically rely on the use of sequential gates (i.e., registers) carefully placed in order to stop the propagation of the glitches while ensuring probing security. Various solutions have been proposed, with different trade-offs in terms of latency, surface area, operating field and need for randomness [Cas+20a; KM22; KSM22].

In this section, we focus on the implementation of secure (possibly high-order, i.e. $d > 2$) masked primitives with the HPC2[1] masking scheme (the variant operating on $\mathsf{GF}(2)$ having the lowest randomness requirements). With the properties provided by its PINI property, it can be used to implement Substitution–Permutation Network (SPN) architecture while providing glitch+transition probing security at arbitrary security order. However, the register layers inherent to the non-linear operations gadgets can have a significant overhead on the cost and on the performances of the circuit. In this context, the question arises whether these limitations can be amortized. To this end, this chapter addresses two exploration tracks, as explained hereunder.

First, the Spook case study (masked with HPC2) is used to demonstrate experimentally that masking is relevant in the case of leveled implementations. In particular, complementary to the energy gains highlighted in the first chapter, we experimentally validate the gains in latency and throughput that can be obtained using levelling, thus amortizing the performance overheads of formally proven masking scheme secure at arbitrary order.

A second area of study begins with the observation that, due to the insertion of registers blocking the propagation of glitches, complex non-linear gadget implemented with HPC2 scheme often follow a pipeline architecture. Although this increases latency, it also has the advantage of increasing the throughput of the corresponding gadgets, allowing certain degrees of freedom for the implementation of efficient serial architectures. Moreover, the particular asymmetrical architecture of AND2 HPC2 gadgets leaves room for optimized architectures. Following this lead, our analysis begins by showing a generic method for implementing optimized non-linear layers with HPC2 and presents results from masked hardware implementations of AES. These are compared with (non-optimized) auto-generated implementations based on simple syn-

[1] the optimizations presented in this chapter have been successfully extended to new techniques leveraging on others HPC variants (such as the low-latency gadget HPC3) that are presented in the work [Cas+23b]. Additionnal details about the latter are provided in the chapter conclusion.

Algorithm 3 HPC2 ANDgadget with d shares.

Input: Sharings $\boldsymbol{a}$, $\boldsymbol{b}$
Output: Sharing $\boldsymbol{c}$ such that $c = a \wedge b$.

for $i = 0$ to $d-1$ **do**
  for $j = i+1$ to $d-1$ **do**
    $r_{ij} \xleftarrow{\$} \mathbb{F}_2;\ r_{ji} \leftarrow r_{ij}$
for $i = 0$ to $d-1$ **do**
  $\boldsymbol{p}_{ii} \leftarrow \mathsf{R}(\boldsymbol{a}_i \mathsf{R}(\boldsymbol{b}_i))$
  for $j = 0$ to $d-1, j \neq i$ **do**
    $\boldsymbol{p}_{ij} \leftarrow \mathsf{R}(\overline{\boldsymbol{a}_i} \wedge \mathsf{R}(r_{ij})) \oplus \mathsf{R}(\boldsymbol{a}_i \wedge \mathsf{R}(\boldsymbol{b}_j \oplus r_{ij}))$
  $\boldsymbol{c}_i \leftarrow \bigoplus_{j=0}^{d-1} \boldsymbol{p}_{ij}$

chronization mechanisms, highlighting that some interesting implementation trade-offs are achievable while still maintaining the security.

In this chapter, I personally contributed to the design (including the generic optimization strategy), hardware implementation, and preliminary side-channel evaluation of all the presented architectures (namely, Spookv1/v2 unprotected and protected Spookv2 with a masked implementation of Clyde128, as well as the 8/32 and 128-bit architectures of the AES encryption). I was also part of the organising team of the CHES2020 and CHES2023 challenges, leading the hardware part of both and resulting in the open-source publication of two protected cores as well as the setup of a semi-automated analysis framework used to evaluate the challenges submissions.

4.1 Background

4.1.1 HPC2 Masking Scheme

The hardware private circuits masking scheme introduced in [Cas+21] come in two flavors: we next focus on the HPC2 variant. The HPC2 scheme operates over GF(2) (i.e., bitwise) and instantiates two gadgets. First, the gadget performing the field addition is implemented following Algorithm 2 in Section 2.4.1, does not require randomness and have no latency (i.e., can be implemented in purely combinational logic). The multiplication gadget (i.e., AND) is depicted in Algorithm 3, where $\mathsf{R}(\cdot)$ is an operator representing the fact that the value between parenthesis is hold by a register.

The latter requires the state-of-the art amount of randomness. Precisely, each HPC2 ANDgadget requires $d(d-1)/2$ bits. A specificity of

this gadget is its latency asymmetry: if the first input sharing enters the gadgets at cycle t, the second input sharing is expected to enter the gadget at cycle $t + 1$ and the output is produced at cycle $t + 2$. Other gadgets performing basic operations have been designed (e.g., MUXes).

4.1.2 fullVerif

The open source tool fullVerif is an automated composition-based program that can verify that the assumptions of the HPC2 security proofs are fulfilled by a Hardware Description Language (HDL) design. In order to be checked by fullVerif, the modules' definitions as well as their ports should be annotated with specific verilog attributes. These attributes indicate signal's properties such as their types (i.e., *sharing*, *control* or *randomness*) or their sizes and time validity to the tool. Different composition strategies can be specified for each module in the design. The tool works based on a netlist of the architecture together with a simulation file in order to build a dataflow graph. The latter is then used to proceed to different security checks as listed in [Cas+21].

4.1.3 Spook

is an Authenticated Encryption with Additionnal Data (AEAD) scheme. It is a Round 2 candidate to the NIST LightWeight Cryptography (LWC) competition.[2] Its primary design goals are resistance against side-channel attacks (SCAs) and energy efficient implementation. Spook is built on the TETSponge AEAD mode, that is based on two primitives: a Tweakable Block Cipher (TBC) used in the key generation and tag generation parts of the mode, and a permutation used in the the sponge part of the mode [Guo+20b]. The TBC of Spook is Clyde-128 and (in the primary variant of the mode) its permutation is Shadow-512. Both algorithms are named after their block size. Next, we focus on the improved version of Spook (Spookv2) introduced in [Bel+20b] in order to increase the security margins of Shadow-512 while also improving the performances of some of its components.

In more detail, both primitives share the same components. In particular, they both rely on the same instances of 4-bit S-boxes. They also share the same linear layer (referred to as the L-box) operating on 64 bits. Additionally, Shadow-512 also incorporates a diffusion linear layer (referred to as the D-box) operating on 128 bits. Clyde-128 consists of the application of 12 rounds, each composed of a layer of 32 S-boxes followed by a layer of 2 L-boxes and a constant addition. An

[2] `https://csrc.nist.gov/projects/lightweight-cryptography`.

additional tweakey addition is performed at each round with an even index. Shadow-512 consists of 6 stages, each composed of two rounds (denoted as A and B). Round A is similar to that of Clyde-128 and involves the application of 128 S-boxes followed by 8 L-boxes and a constant addition (different from the one used in Clyde-128). Round B is similar to Round A but replaces the linear layer with a diffusion layer composed of 4 D-boxes. Visual representations of the algorithms and various components can be found in the Appendixes of [Bel+20b]

4.1.4 Advanced Encryption Standard

The AES is a symmetric key algorithm standardized by the NIST in 2001. Its variant AES-128 operates on 128-bit state with 128-bit secret key. Both the state and the key can be represented as 4x4 matrices of 16 bytes each. The algorithm is composed of 10 rounds, each one being composed of 4 different operations. First, the `SubByte` operation is a non-linear substitution operating on each byte independently. Second, the `ShiftRows` operation is a cyclic shift of 1, 2 or 3 positions of each byte in the last three rows. Third, the `MixColumns` operates over each column of the state independently. The later are considered as polynomials with coefficients over GF(2^8) and are multiplied modulo $x^4 + 1$ by the fixed polynomial $03x^3 + 01x^2 + 01x + 02$. Finally, the `AddRoundKey` operation adds the round key by performing a bitwise XOR operation. It has to be noted that the `MixColumns` operation is not performed in the 10-th (i.e., last) round and an `AddRoundKey` is performed before the first round.

Each round key is derived with a key scheduling algorithm applied at each round on the previous round key. First, the last column is rotated by one byte up. Next, the SubByte operation is applied on each byte of the resulting column and the round constant is added using a bitwise XOR operation. The first column of the new round key is obtained by applying a bitwise XOR between the column obtained after the round constant addition and the first column of the key. Finally, the new value of the i-th column is obtained by bitwise XORing the i-th column of the key and the $i - 1$-th column of the new round key.

4.2 Leveled Implementation of Spookv2

4.2.1 Contextualization

General framework Since the NIST LWC competition is aimed at standardizing "lightweight cryptographic algorithms that are suitable for use in constrained environments", evaluating their implementation results

is of primary importance for the comparison of different candidates. In particular, the Spook mode of operation being designed with the SCA concern in mind, it is expected that it can be implemented with strong protection against SCAs (Grade-2 security, as explained in Chapter 3) with limited overheads. More into the details, the TETSponge mode used in Spook aims at enabling so-called leveled implementation, which minimize these overheads by ensuring strong security guarantees against leakage with only two calls to a strongly protected (e.g., masked) TBC while only requiring weak protections (or in parallel hardware implementations, no protection at all) for the other permutation calls used to process the message. Relevant constrained environments include hardware platforms such as FPGAs and ASICs. Therefore, in order to enable a fairer comparison between implementation, a common hardware API (next denoted as the LWC HW API) was proposed by the Athena Project benchmark initiative, which we use next [Kap+].

Organization Next are presented two hardware implementations of Spook: an unprotected one and a leveled, protected one (where Clyde-128 is masked). In addition to performance improvements for the unprotected core compared to the preliminary results reported in [RD] and [Bel+20b], this section also presents the first results of side-channel protected implementations for Spook (the latter relying in part on the unprotected one). Both implementations are compliant with the LWC HW API and have been designed in a general architecture where API handling features are common for the two implementations, and only the block implementing core primitives differs. More into the details, the high level view of both architectures is depicted in Figure 4.1. The crypto core is the main module which implements the cryptographic functionality and is optimized to be as small and efficient as possible, at the expense of an increased of complexity in its interfaces: it has many control signals, and the input data must be provided in fixed-sized, already padded blocks. The interface of the crypto core is however identical for both unprotected and masked implementations. Next, we will focus on the architecture of the crypto core itself and refer to the original API definition [Kap+] the for a detailed explanation of the expected behaviour of control signals, as well to [MCS21] for a detailed explanation of how the primitive are integrating the latter at lower-level.

4.2.2 Unprotected Architecture

Shared primitive Core The architecture of the unprotected Spook version is oriented towards the area requirements specified for the NIST

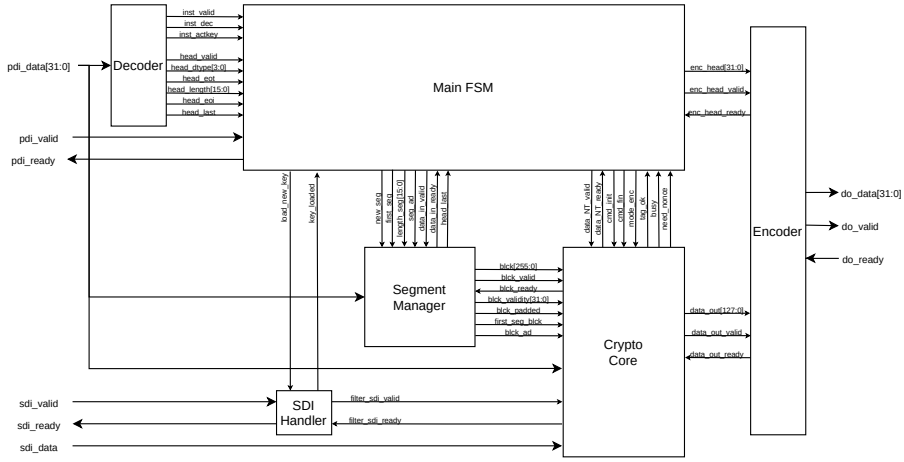


Figure 4.1: General framework.

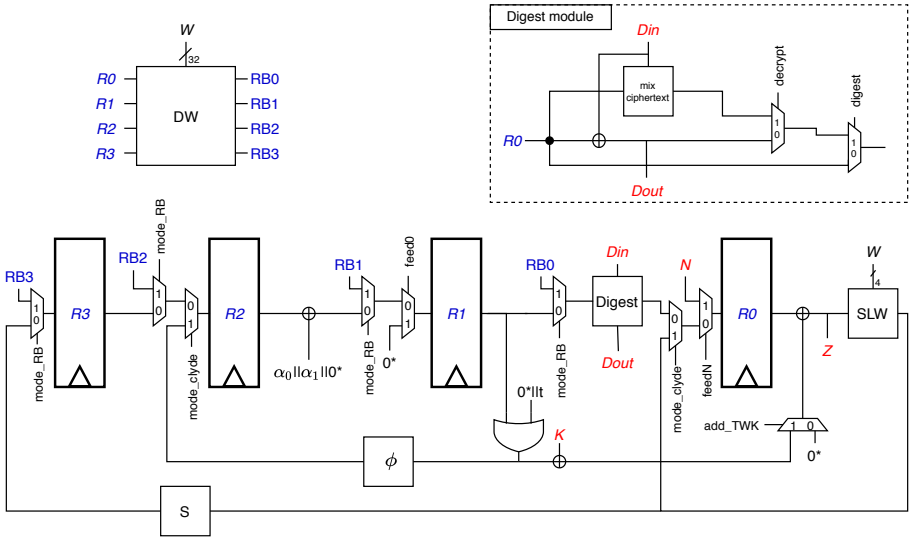


Figure 4.2: Unprotected crypto core: resource sharing architecture

benchmarking effort[3] while optimizing throughput and energy consumption. The main functionality of the primitive core is the implementation of the Clyde-128 (tweakable block cipher) and Shadow-512 (permutation) cryptographic primitives. The primitive core architecture is introduced in [Bel+20b].

The architecture operates on 128 bits, hence it requires 4 cycles to execute a round of Shadow-512 and 1 cycle for a round of Clyde-128. Furthermore, thanks to the similarities between Clyde and Shadow (same S-box, L-box, etc.), the ability to compute Clyde-128 can be added on top of the implementation of Shadow-512 with little additional logic (roughly 10% based on FPGA implementation in [Bel+20b]).

On Figure 4.2, each bus is 128-bit wide unless indicated otherwise. The IOs (in red) are the nonce, key, tag, bytes to digest (denoted as `Din`, that are AD padded plaintext or ciphertext) and the digested bytes (denoted as `Dout`, that are plaintext or ciphertext). The symbols S, L, D and W in logic blocks correspond respectively to the S-box, L-box, D-box and constant addition.

The `mix ciphertext` module is only used during a decryption process to mix the partial input ciphertext block with the state of the sponge before it is used during the following execution of Shadow-512. The signals α_0 and α_1 control the addition of the domain separation bits.

To compute Clyde-128, the initial plaintext and tweak are respectively stored in the registers `R0` and `R1`. The control signal `mode_RB` is unset (i.e., equals 0) while `mode_clyde` is set. In this way, the state of Clyde-128 cycles through the `R0` register and the SLW logic (and the tweakey addition) at the rate of one round per clock cycle. Therefore, the full Clyde-128 computation takes 12 cycles. The tweak flows through registers `R1`, `R2` and through the ϕ logic, producing a valid updated tweak every two clock cycles.

For Shadow-512, the four 128-bit bundles $b_0, \ldots, b_3$ are stored in the registers `R0` to `R3`. Those registers act as a large circular shift register with two shifting modes. Shadow-512 uses two rounds that differ in their linear layer. For Round A (and the S-box of round B), the data cycles from `R3` to `R0`, then through the SLW unit and the S unit back to `R3`, computing a full round in four cycles (all MUXes' controls are unset). For Round B (except its S-box), data is cycling inside the same `Ri` register: the signal `mode_RB` is set, forwarding the data to the DW unit (that updates a part of it input and shifts the other part) for 4 cycles.

[3] `https://cryptography.gmu.edu/athena/LWC/LWC_Suggested_FPGA_Design_Goals.pdf`

When starting the Spook operation, the N and 0* values are loaded in `R0` and `R1` respectively during the first clock cycle. Then, in 12 cycles, the first call of Clyde-128 is computed, producing the fresh seed B.

Next, during the first round of the first execution of Shadow-512, the initial state (i.e., $0^*||N||0^*||B$) is loaded sequentially per bundle (except for B), using again the control signals `feed0` and `feedN`. Shadow is then executed (in 48 cycles per execution), and the digest unit is used at the beginning of each execution when AD/P/C needs to be fed. Finally, the Clyde tag computation takes 12 cycles, with the signal `t` set in order to ensure that the MSB of the tweak is high.

Two LFSRs are used in order to generate the constants of the primitives: a 4-bit LFSR for Clyde and a 32-bit LFSR for Shadow.

4.2.3 Protected Architecture

Overview The architecture of the protected Spook version aims at providing high security against leakage by leveraging on the leveled architecture enabled by the TETSponge mode. In particular, the side-channel security (and therefore the area cost, speed and energy consumption) of the Clyde-128 and Shadow-512 implementations significantly differ. The first one protects a long-term key and is masked to ensure DPA security. The second one has to ensure weaker and cheaper SPA security and is thus designed with an unprotected parallel architecture. As explained in Chapter 3, such an implementation offers strong security properties (i.e., ciphertext integrity with misuse-resistance and leakage in encryption and decryption and CCA security with misuse-resilience and leakage in encryption only). As for performance, such a leveled implementation aims maintaining the throughput and energy efficiency of the unprotected core at the cost of a larger circuit.

The protected core contains two independent sub-cores interacting together: the Shadow core and MSKClyde core. The first one computes the Shadow512 primitive and does not use any special countermeasures. The other computes both the direct and the inverse operations of the Clyde128 primitive and is masked. The Shadow core follows the same architecture as the primitive core described in Section 4.2.2. The main difference is that the logic related to the computation of Clyde-128 (i.e., The Φ logic, the tweakey addition, the 4-bits constant addition and the bypass multiplexer) are removed. The MSKClyde is on the other deeply changed and its architecture is detailed next.

Architecture of the masked Clyde module The MSKClyde core is a direct application of the HPC2 masking on the Clyde128 algorithm.

As depicted in Figure 4.3, it takes as input an unshared tweak (i.e., clyde_tweak), an unshared plaintext/ciphertext (i.e., clyde_din), a shared key (i.e., key_sharing) and outputs a shared ciphertext/plaintext (i.e., dout) that is then recombined outside the core. The randomness (i.e., $2N_{SB} \cdot d(d-1)$ bits per S-box) is provided by the rnd0 and rnd1 busses to the S-boxes, and is generated by dedicated PRNGs based on parallel and unrolled 128-bit LFSRs located outside the core.

The core operates over a d128-bit long shared representation of the state (i.e., state_sharing). To make this possible considering the unshared inputs, the processed data first goes through a cst_msk unit that trivially shares its input by encoding the unmasked data as the first share and putting $d-1$ share to the constant 0. When a new execution starts, the signal new_run is asserted during one clock cycle. This has the effect of fetching the inputs in the core and resetting the submodules. This step is followed by the first constant and tweakey additions performed by the Add_W_TWK core, whose input is generated by the W unit and the δ unit, providing respectively the values of the 4-bit constant (i.e., W) and the 128-bit δ used to compute the tweakey. Both the W unit and the δ unit are similar to the 4-bits LFSR and the Φ unit of the unprotected core, except that an additional register holds the evolving values of δ. The FSM Clyde core controls the update of the latter by asserting the signals upd_W and upd_delta during one clock cycle. The signals add_W and add_TWK enable the addition of the corresponding value inside the Add_W_TWK core. The shared value of the key is fed by the Key Holder and does not change during the Clyde-128 computation.

The Clyde-128 rounds are computed by alternating between the S-box layer logic and the L-box layer logic. The Clyde FSM core drives the datapath according to the layer under computation and enables the addition of the constant and of the tweakey when required. Two architectural parameters can be changed: the number of parallel masked 4-bit S-boxes implemented in the S-box layer logic (denoted by N_{SB}) and the number of parallel masked 64-bit L-boxes in the L-box layer (denoted by N_{LS}). By using multiple S-boxes/L-boxes in parallel and combining them with a shift register strategy, the area versus latency trade-off of the Clyde-128 core can be adjusted. In particular, the latency of the L-box layer is $2/N_{LS}$ with N_{LS} being equal either to 1 or 2. For the S-box layer, the latency is $3 + 32/N_{SB}$ where N_{SB} can take any of the following values: 1, 2, 4, 8, 16 and 32. The additional three cycles are due to the latency of a single masked S-box that comes from the HPC2 AND2 gadgets architecture [Cas+20b].

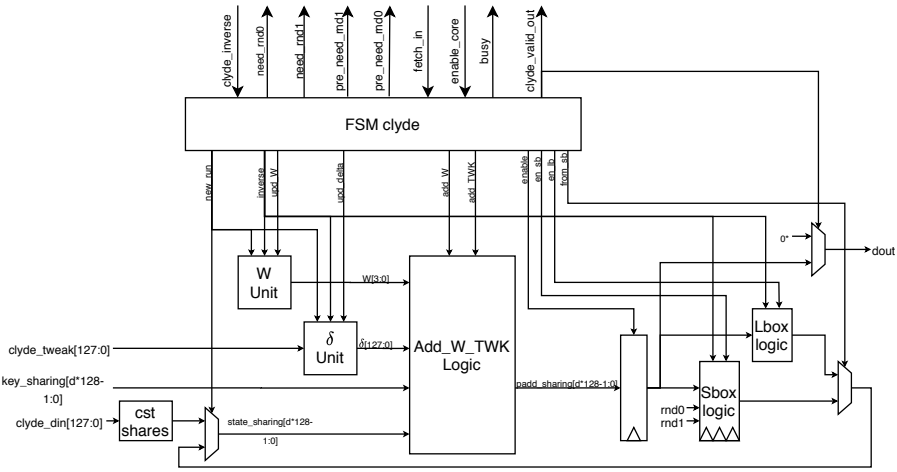


Figure 4.3: Protected Clyde-128 core.

Each block developed above can compute its inverse operation (used for the inverse based tag verification tweak, as detailed in Chapter 3), making straightforward the implementation of the inverse Clyde-128 primitive. Except for the masked S-boxes, the inverse functionality of each unit is implemented using an independent and parallel dedicated logic block. Both the direct and the inverse functionality logic blocks are fed with the input data and their corresponding output are muxed. The appropriate value is chosen depending on the value of the signal inverse.

In order to reduce the logical cost, the implementation of the inverse S-box reuses the already existing logic for the implementation of the direct one. The S-box being nearly involutive, its inverse is obtained by applying a simple linear layer at the inputs and outputs of the already existing core, as depicted in Figure 4.4. This costs 5 XOR gates and 2 multiplexers while allowing to reuse the costly logic of the masked AND gates already implemented.

4.2.4 Implementation Results

Next are presented the results obtained for FPGA and ASIC implementations of the two cores version presented in Section 4.2.2 and Section 4.2.3. In particular, the protected implementation was synthesized adopting $N_{SB} = 8$ and $N_{LB} = 1$ as degree of parallelization.

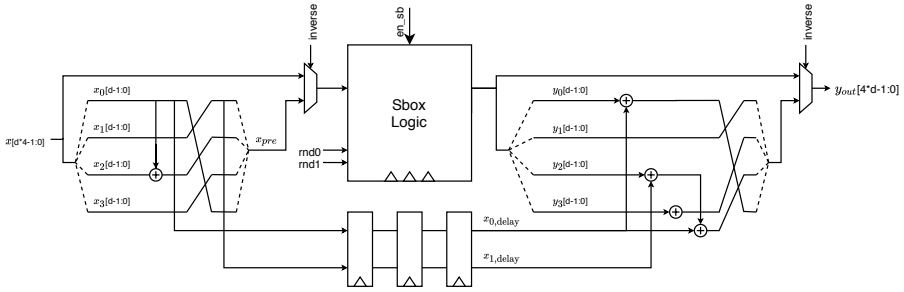


Figure 4.4: Protected S-box supporting direct and inverse operation.

Instance	Regs	LUTs	Freq [MHz]	Latency [cycles]	Throughput [Mbps]
Unprotected	1527	2067	206.8	48	1102.9
Protected ($d = 2$)	5499	6340	200	48	1066.6
Protected ($d = 3$)	6209	9111	200	48	1066.6
Protected ($d = 4$)	8367	11555	200	48	1066.6

Table 4.1: Artix-7 implementation results (xc7a50tcsg325-3)

FPGA Target The results provided are based on an Artix7 FPGA (xc7a50tcsg325-3) implementation performed with the Xilinx Vivado v2019.2.1 toolset. These are shown in Table 4.1 and are given post place-and-route (i.e., post-implementation). The table includes standard FPGA metrics, namely the amount of slice registers and look-up tables required, the maximal clock frequency, the latency and the corresponding throughput (for long messages). More specifically, the maximum throughput is improved by 112 Mbps (11.7%) compared to the results reported in [Bel+20b]. This is achieved with a slight resources increase (45 additional registers (3%) needed for the pipelining of control signals). The protected core is larger than the unprotected one (expected by the additional masked Clyde-128 core), but it achieves the same throughput.

ASIC Target The numbers provided for the ASIC target are based on a 65nm technology implementation performed with the a 65nm technology design kit. The tools are Cadence Genus v18.10-p003_1 for the synthesis. The implementation relies on the use of a clock-gating strategy in order to reduce the dynamic power when sub-blocks are in idle state. The results are shown in Table 4.2 and are given post synthe-

Instance	Area [kGE]	Frequency [MHz]	Throughput [Mbps]
Unprotected	18.14	588	3124
Protected ($d = 2$)	48.716	588	3124
Protected ($d = 3$)	67.766	588	3124
Protected ($d = 4$)	88.81	588	3124

Table 4.2: ASIC 65nm implementation results (post synthesis)

sis. The table includes standard ASIC metrics, namely the area, the maximal clock frequency and the throughput (for long messages)[4]. The latency is the same as for the FPGA implementation. The results for the unprotected core exhibit a throughput improvement of 975 Mpbs (45%) compared to the results reported in [Bel+20b] at the cost of an increase in area of 5%[5]. Again, the protected core is larger than the unprotected one, but it achieves the same throughput.

For short messages, the throughput is lower than the one shown in Table 4.1 and Table 4.2 due to the latency inherent to the initial/final executions of Clyde-128. This takes 24 cycles for the unprotected core and $12*(2/N_{LS}+32/N_{SB}+3)$ cycles for the protected core. In Figure 5.8, we illustrate the throughput of the implementations for various message lengths. It shows that the overhead cost of initialization/finalization is small when the message is larger than 1 kB. Furthermore, the number of implemented S-boxes in the protected Clyde-128 implementation can be reduced with limited impact on the latency.

4.2.5 A Note on The Physical Security

The protected Clyde-128 implementation has been verified with the fullVerif tool, ensuring probing security in the presence of glitches of its architecture. Besides, it has also undergone a worst-case public evaluation as part of the CHES2020 conference challenge[6]. More specifically, candidates had access to an open-source implementation[7], a 10M traces profiling dataset (with variable keys and plaintexts and with knowledge of the randomness used for masking) and 5 validation datasets (with fixed keys, variable plaintexts and without knowledge of the random-

[4]Power estimation for short message can also be found in [MCS21]

[5]The area results reported in Table 4.2 have been adapted compared to [MCS21] to take only the cell area into account.

[6]`https://ches.iacr.org/2020/challenge.php`

[7]`https://www.spook.dev/implementations`

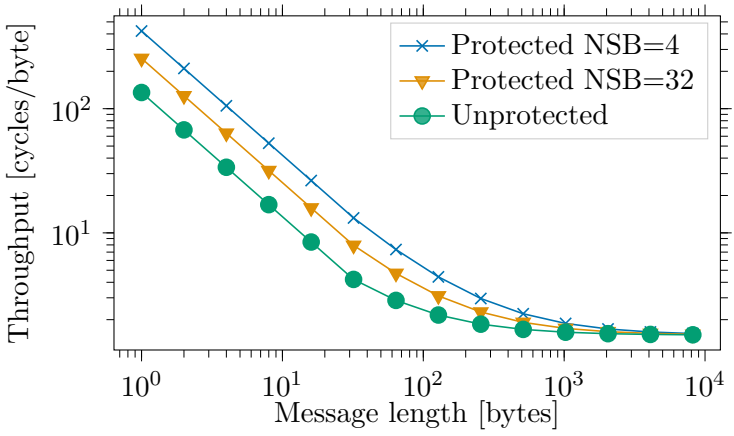


Figure 4.5: Throughput of Spook encryption for protected and unprotected implementations with no AD. For the protected implementation, there is one L-box implemented, and the number of S-boxes is either 4 or 32.

ness used for masking) each containing 5M traces. The aim of this evaluation was to perform an attack reducing the rank of the 128-bit key below 2^{32}. After 3 months of evaluation, no attack meeting this criterion was proposed.

4.3 Advanced Encryption Standard

4.3.1 Contextualization

AGEMA is an open source tool presented in [Kni+22b]. It automatically generates masked hardware netlists based on unprotected HDL implementations and relies on specific annotations of the IO ports at the top level of the hierarchy in order to identify which part of the circuit should be masked. Working on top of a synthetized netlist of the full architecture, it propagates the signals annotations across the hierarchy before implementing the masked parts of the circuit. It offers the choice between different masking schemes relying on the PINI property (among which HPC2), as well as different optimization strategies depending on the chosen masking scheme. To ensure the proper synchronization of the signals across the hierarchy, the tool relies on two approaches: pipelining (i.e., introducing registers to synchronize all the inputs of each masked gadgets) or clock gating (i.e., modulating the reg-

isters' clock signals). Combined with fullVerif, it represents a significant step towards improving the usability of masking schemes in hardware.

Motivations and Organizations The advances achieved by AGEMA naturally raise the question whether the implementations obtained compete with manually optimized ones. In other words, can all the architecture-level optimizations that an experienced designer would exploit be automated and what are the limitations faced by existing automation tool, and in our particular case AGEMA.

While masking a circuit with only sharewise gadgets is a simple transformation, using the HPC2 is more complex because these gadgets introduce additional latency in the circuit. This means that masked non-linear sub-circuits such as S-boxes in SPNs often have a high latency. As shown in [Kni+22b], clever usage of clock gating to properly synchronize all the signals in the circuit will on the one hand reduce the global cost, but will result in an implementation with (potentially significantly) larger latency. On the other hand, the synchronization achieved through pipelining will not improve the latency either and will increase area cost. However, this configuration effectively increases the throughput of the subcircuit (at the reasonably reduced cost of synchronization register, given that the amount of the latter is minimized). Although this translates directly into an increase in overall throughput, it can also be exploited to improve the latency of serial architectures at a reduced cost, which is of particular interest when searching for best latency vs area trade-off.

With this consideration in mind, a first question is whether there is a way of limiting the number of synchronization registers added to a masked pipeline implementation. To this end, the section begins by presenting a generic optimization strategy applicable in non-linear layers implemented with HPC2. This strategy is then applied to the AES S-box, which is itself integrated into AES architectures with different levels of parallelism: one using one S-box instance (denoted 8-bit) and a round based (i.e., 128-bit) using 20 S-boxes (4 for the key scheduling and 16 for the round computation). These are then compared with those obtained with AGEMA, illustrating that the latency gains achieved for the 8-bit is significantly improved but tend to vanish for the larger architectures. Additionaly, we propose a new 32-bit optimized architecture that achieves good trade-off and is believed to be inherently more difficult to be automatically generated because of the operation re-ordering it implies and which requires a deep understanding of the operations to

perform. The physical security of the architectures are discussed and supported by experimental results.

4.3.2 Non-linear Layer Implementation with HPC2

Because of its significant cost both in logic and randomness, the masked implementation of the non-linear layer in cryptographic primitives (e.g., S-boxes) requires a particular attention. First, taking into account the constraint that the HPC2 masking scheme instantiates gadgets operating over GF(2) only, implementers need to rely on bitwise representation of the operation they intend to implement, composed of basics operation over GF(2) (e.g., XOR2, AND2 and NOT gates). In general, representation with minimal amount of AND2 gates and AND depth allows achieving better performances: They inherently decrease both the latency and the number of required register stages for computing the targeted operation, while also minimizing the logic overhead needed to generate the randomness.

Due to the asymmetry at their inputs, naively replacing basic unprotected operations in complex circuit with HPC2 gadgets may result in a suboptimal area and latency. For example, two different implementations (i.e., unprotected and masked) of an AND3 gate are shown Figure 4.6. While the area and latency for the two unprotected implementations are identical, they differ for the protected ones. On the left, 4 additional registers are required to ensure the circuit functionality, and the computation is performed in 4 cycles. On the right, only 2 registers are required and the computation is performed in 3 cycles. Such area and latency overheads are incurred by implementations that add a register on one of the inputs of the AND2 gadget to achieve symmetric latency (which simplifies the design process).

Based on this observation, it may be tempting to look for optimal symmetric-latency gadget-based circuits for more complex operations (e.g., AND3 gates). However, using a composition of such larger gadgets is still not optimal. For example, two implementations of an AND5 operation are shown in Figure 4.7. On the left, the implementation directly instantiates twice the optimized AND3 construction of Figure 4.6 and requires 10 additional registers to compute the results in 6 cycles. On the right, a more optimized implementation only requires 3 additional registers and performs the computation in 4 cycles. Instead of the "larger gadgets" approach, optimizing directly the latency of a full logic block (e.g., a full S-box) helps reduce the number of registers instantiated in the architecture.

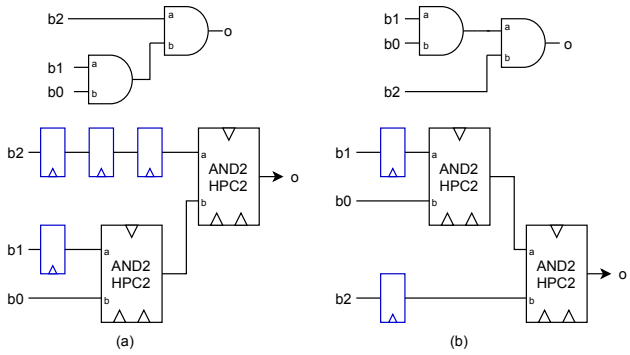


Figure 4.6: Example of AND3 gate implementation using HPC2 gadgets.

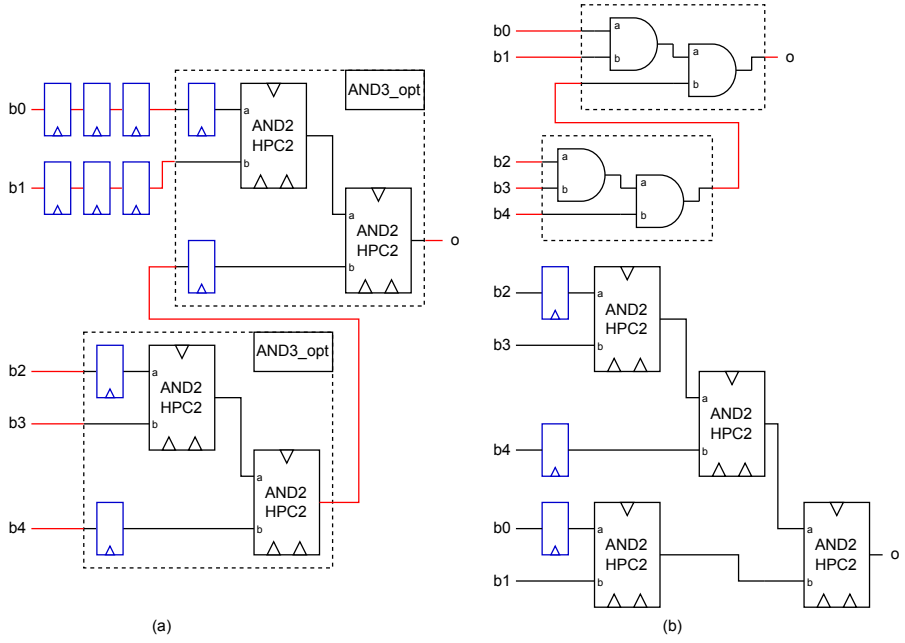


Figure 4.7: Example of AND5 gate implementation using HPC2 gadgets.

Generic block latency optimization algorithm. The optimization of a block is efficiently performed as follows. Starting from the input signals of the logic block to optimize, the information about the exact time of validity of each signal is propagated across the circuitry, level by level. For some gadgets that have more than one input, the latency requirements of all the inputs may not be met. In such cases, a solution is to insert registers on the input path in order to meet the timing constraints. In the specific case where the latest signal (i.e., the signal being valid with the largest latency) is connected to the input port with the lowest timing requirement (i.e., the input port at which a signal is first entering the gadget), then the input connections are switched. This optimization naturally leads to a solution with a reduced amount of additional registers while keeping the same functionality as the original circuit.

4.3.3 Masked AES S-box Implementation

To allow a fair comparison, we follow the design choice of [Kni+22b] and use the S-box design proposed by Boyar and Peralta [BP12]. The latter is composed of basics operation over GF(2) (i.e., XOR2, AND2 and NOT gates) and instanciates 34 AND2 gates with an AND-depth of 4. The naive implementation using the HPC2 gadgets results in an S-box with 8 cycles latency and requires 156 synchronization registers. By applying the latency optimization methodology presented in Section 4.3.2, the latency is reduced by 25 %, resulting in a S-box core operating in 6 cycles and requiring 94 additional registers (i.e., $\approx 65\%$ of sync. registers reduction).

4.3.4 8-bit serial implementation

As depicted in Figure 4.8, the 8-bit serial implementation takes as input the shared value of the key `sh_key` and the (unshared) plaintext `pt`. The latter is first represented as a valid sharing by concatenating $d-1$ zero shares to each bit. The architecture of this implementation is organized around two mains blocks: the `KeyHolder` and the `StateHolder` which store and order the processing of the round key and of the state. The blocks feed each byte serially during the `SubByte` and `AddRoundKey` layers. Unless otherwise noted, each bus in the architecture is 8-bit wide. In addition, one S-box is instantiated with the block `MSKSbox`. The block `MSKmc` is a combinational logic that implements the `MixColumns` operation for a full column (i.e., 4 bytes) and consists in d instances of the

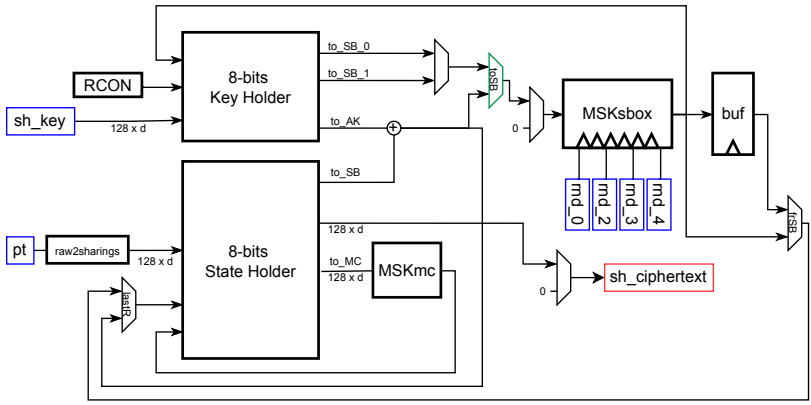


Figure 4.8: Global Architecture of the 8-bit serial implementation.

unprotected `MixColumns` operation logic, where each instance processes one state share.

The shared state and round key are stored in dedicated shift registers of shared bytes, as shown in Figure 4.9 and Figure 4.10(the numbers on the figure indicate the byte index in the unmasked state). The rounds are computed serially by performing the `SubByte` and `AddRoundKey` operations byte per byte. The state is processed row per row (i.e., starting with the bytes 0,4,8,12 and ending with the bytes 3,7,11,15). The four S-box executions occurring during the key scheduling are interleaved between the processing of the state. To do so, the mux `toSB` in Figure 4.8 is used to feed the S-box instance with a shared byte coming either from the key, either from the `AddRoundKey` result. This interleaving of key scheduling and round processing necessitates the insertion of a buffer at the output of the S-box to avoid loosing data in cases where the output of the S-box is valid and a key byte is provided at its input. Indeed, in such cases, the shift register holding the state is stalled, making any feedback from the S-box to the state holder impossible. We rely on the pipeline structure of the S-box instance to reduce the overall latency of the execution. That is, we do not wait for full S-box execution to be finished before feeding the S-box with valid data. Instead, we feed it at each clock cycle when its input data is available. Considering the latency of our S-box design, the full `AddRoundKey`, `SubByte` and the key scheduling operations of a round are performed in $\mathsf{SB}_{lat} + 16 + 4 = 26$ cycles.

The `ShiftRows` operation is performed in a single cycle by enabling the data flowing through the state holder with an appropriate routing defined by the MUXes `sh-i` (depicted in green in Figure 4.9). Finally, the `MixColumns` operation is performed in 4 cycles, using the MUXes

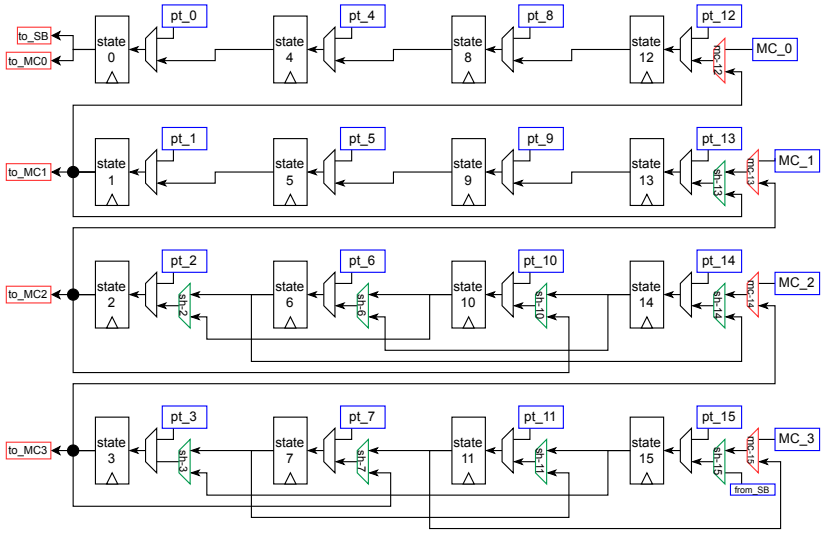


Figure 4.9: 8-bit implementation: architecture of the state holder.

`mc-i` to route the signals back from the `MC` instance (depicted in red in Figure 4.9). The last `AddRoundKey` operation is performed byte per byte in 16 cycles, using the mux `lastR` in Figure 4.8 to bypass the S-box instance. Overall, the latency of a full execution of our 8-bit implementation is equal to $9 \times (26+1+4)+(26+1)+16 = 322$ cycles.[8]

4.3.5 128-bit serial implementation

As depicted in Figure 4.11, the 128-bit architecture contains all the logic necessary to operate on a full 128-bit state, as well as the logic required to perform the key scheduling in parallel to the round computation. In more details, the state and the key sharing `sh_key` are stored in a register. After the `AddRoundKey` layer, the state in directly routed to the `SubByte` logic composed of 16 S-box instances. Directly following the latter, the `ShiftRows` operation is performed at the routing level before entering the logic for the `MixColumns` operation, itself composed of four independent blocks, each operating on one individual column of the shared state. Depending on the round counter, MUXes are used

[8] A MUX located at the output of the global core is used to control the proper release of the valid ciphertext. This is not strictly required in the context of our work, however practical integrations will likely add a logic block computing the recombined ciphertext. Without our output gating, this would lead to leaking unmasked internal states of the AES, defeating the masking countermeasure.

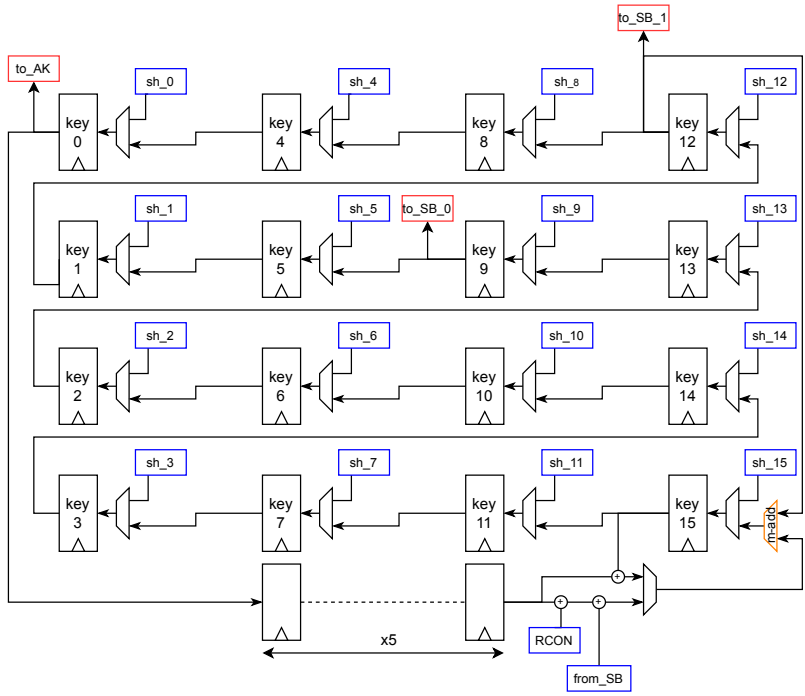


Figure 4.10: 8-bit implementation: architecture of the key holder.

to feed back to the state register the results either after the `ShiftRows` operation, or after the `MixColumns` operation.

For the key scheduling, four S-box instances are used order to process the 4 bytes of the fourth column of the key in parallel. As for the `ShiftRows` layer, the rotation occurring is performed as the routing level, requiring no additional logic. Finally, combinational XOR gadgets are placed at the output of the S-box in order to finalize the round key. The resulting round key is fed back to the key holder register. For this 128-bit architecture, it therefore follows that the latency of a full encryption is $10 \cdot (6 + 1) = 70$ cycles.

4.3.6 32-bit serial implementation

As shown in Figure 4.12, the module is organized around two datapath blocks performing the operations dedicated to the round computation (denoted `MSKaes_32bits_state_datapath`) and the key scheduling (denoted `MSKaes_32bits_key_datapath`). The module `MSKSbox` is shared between the two datapath blocks and implements the `SubBytes` layer for 4 masked bytes. In particular, it is composed of 4 parallel in-

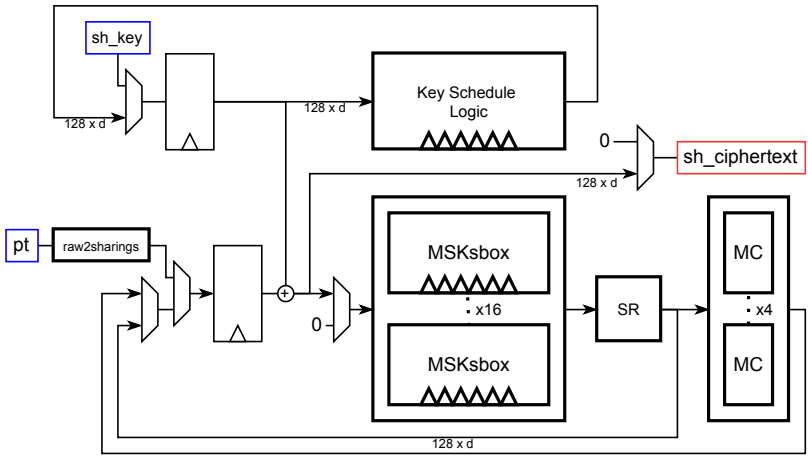


Figure 4.11: Global Architecture of the 128-bit serial implementation

stances of masked S-boxes. The bus `rnd` is used to provide the fresh randomness to the 4 S-boxes instances

Architecture of the `MSKaes_32bits_state_datapath` module

Figure 4.13 shows the detailed architecture of the module performing the round computations. It is organized as a shift register where each register unit holds a masked state byte. The module operates on 32-bit parts of the state and is also implementing the logic that computes the `AddRoundKey`, `ShiftRows` and `MixColumns` layers. In particular, these are implemented in purely combinational logic. Addition gadgets (i.e., XORs) are used to perform the key addition with key bytes coming from the round key (denoted `sh_4bytes_from_key`). The module `MC_unit` computes the result of the `MixColumns` operation for a masked column (i.e., 4 masked bytes). The `ShiftRows` layer is free, being implemented as a specific routing at the input of the `SubBytes` layer. In particular, the ordering of the bytes routed to the S-boxes (denoted `sh_4bytes_to_SB`) is selected such that the rotations over the rows are applied. Dedicated MUXes (controlled by `route_MC`) are used in order to bypass the `MixColumns` logic block when executing the last round. Other MUXes (controlled by `loop`) are used during the last key addition in order to bypass the `ShiftRows`, `SubBytes` and `MixColumns` layers. When a new execution starts, the masked plaintext bytes are loaded in the register through the MUXes controlled by `init`. Then, the `AddRoundKey` and `ShiftRows` layers are executed by propagating the data across the pipeline to the S-boxes. The `MixColumns` operation is performed when

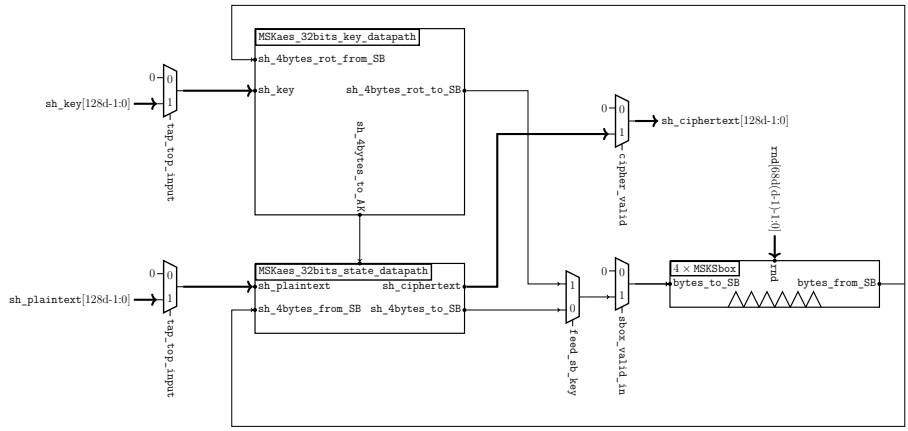


Figure 4.12: Global Architecture of the 32-bit serial implementation.

the result of the SubBytes layer is coming back to the core by asserting the signal route_MC.

Architecture of the MSKaes_32bits_key_datapath module

The module MSKaes_32bits_key_datapath is shown in Figure 4.14. It is organized as a shift register where each register unit holds a masked byte of the key (the numbers on the Figure indicate the byte index in the unmasked key). The module is split in 4 independent parts, each taking care of the key scheduling operation on a single row. The sharing of the 128-bit key is routed from the input with the control signal init.

Concretely, the key scheduling starts by sending the last column of the key (i.e., byte indexes 12, 13, 14 and 15) to the S-boxes. The RotWord operation is performed by the routing that sends the key bytes to the S-boxes. Once computed, the result of the SubBytes layer is routed back to the core through the MUX controlled by the signal from_SB. At the same time, the round constant is applied and the first column (i.e., byte indexes 0,1,2 and 3) of the new key is computed by adding its value to the column coming back from the S-boxes. The remaining three columns (i.e., byte indexes [4,5,6,7], [8,9,10,11] and [12,13,14,15] are then updated sequentially by XORing each byte with the value of the last byte updated in the same row. The signal loop is used to make the key shares loop across the key pipeline. This is required to keep the key material after the AddRoundKey operations while the SubBytes results of the key scheduling is still under computation.

As a result, the round function and the key scheduling algorithm are executed in parallel by interleaving the S-boxes usage appropriately.

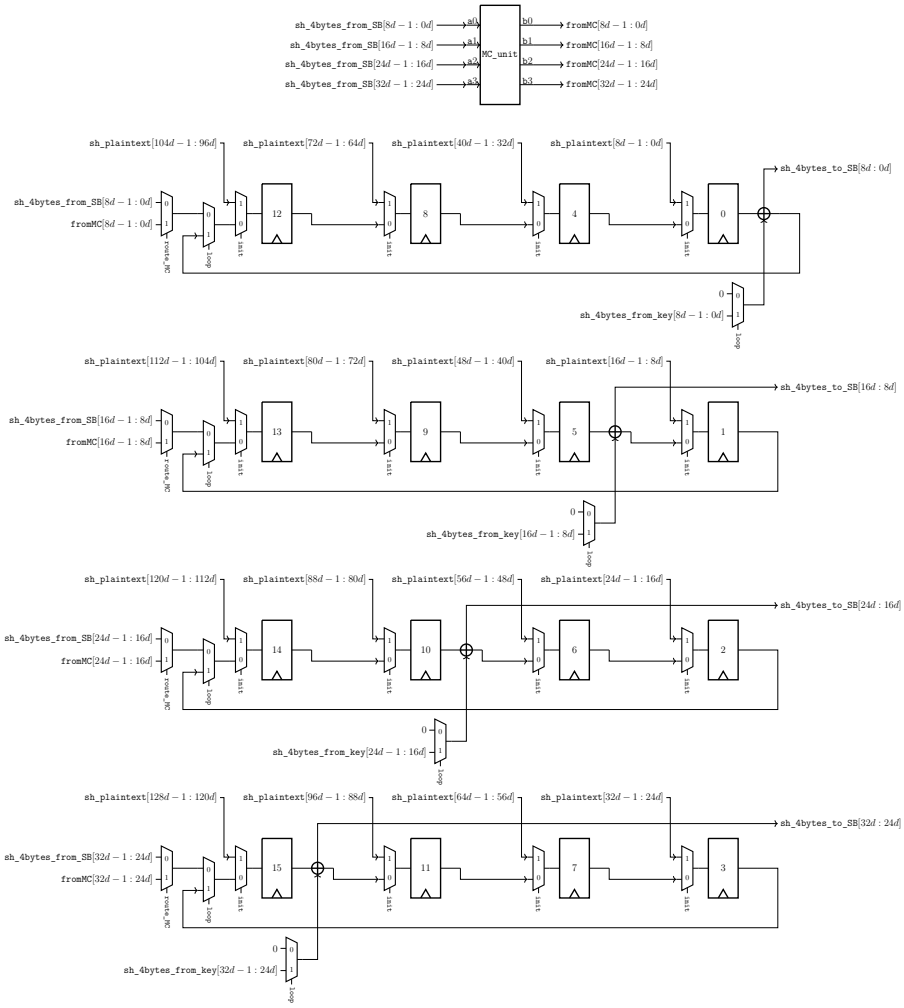


Figure 4.13: Global architecture of the `MSKaes_32bits_state_datapath` module. The value held by the DFF at index i is depicted by the signal `sh_reg_out[i]` in the HDL.

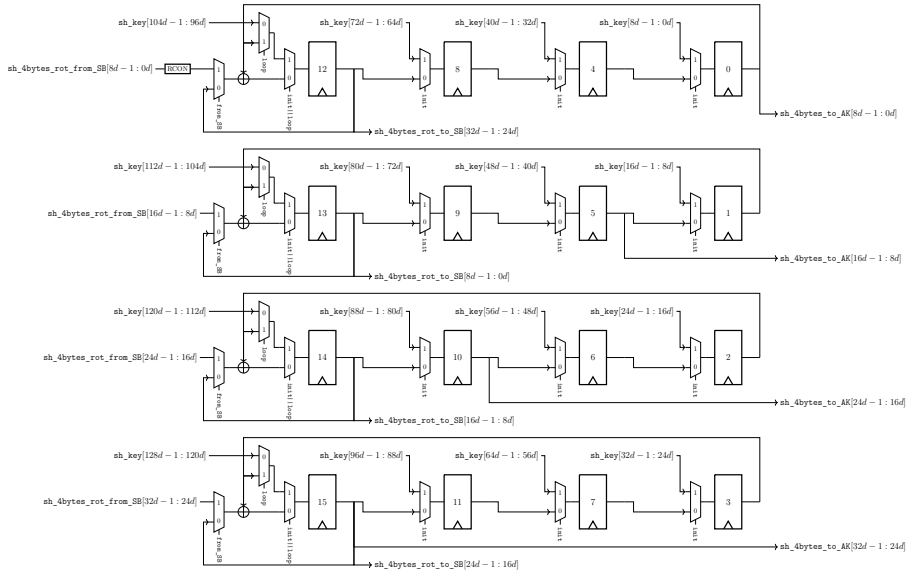


Figure 4.14: Global architecture of the module `MSKaes_32bits_key_datapath`. The value held by the DFF at index i is depicted by the signal `sh_m_key[i]` in the HDL.

In particular, the first cycle of the execution is used to start the key scheduling algorithm by asserting `feed_sb_key` and `sbox_valid_in`. During this cycle, both the module `MSKaes_32bits_state_datapath` and `MSKaes_32bits_key_datapath` are disabled. Then, the core enters into a nominal regime that computes a round in 10 cycles. A typical round starts with 4 clock cycles during which data is read from the state registers, XORed with the subkey and fed to the S-boxes, which performs the `AddRoundKey`, `ShiftRows` and `SubBytes` layers for the full state (one column per cycle). During these cycles, `sbox_valid_in` is asserted and data (state and subkey) loops over the shift registers. At the fifth cycle of a round, the module `MSKaes_32bits_key_datapath` is disabled in order to wait one cycle for the S-box results. At the same cycle, the S-boxes output the column of the new subkey value, which is processed by enabling the module `MSKaes_32bits_key_datapath` and asserting `from_SB` for one cycle. Next, during the last 4 cycles of a round, the S-boxes output the 4 columns of the state, on which the `MixColumns` layer is directly applied, and the result is stored in the state registers. At the same time, the subkey update is finalized, such that a new subkey is ready at the last cycle of a round. During this last cycle, the next key schedule round is started, and a new state round starts at the

following cycle. Finally, the last round is very similar to the regime mode except that the module `MC_unit`is bypassed. In particular, the signal `route_MC` is de-asserted and the shift registers are configured to make the data loop. No new key scheduling round is started during this last cycle. Overall, the resulting latency of a full encryption is therefore $1 + 10 \cdot (4 + 6) + 4 = 105$ cycles. An in-depth functional description including timing diagrams as well as additional architectural diagrams can be found in [SC23a].

4.3.7 Implementation results

Next are presented the post-synthesis ASIC implementation results obtained for the different architectures as well as the results obtained with AGEMA generated implementation. All the syntheses have been performed in a 65nm technology using Genus Synthesis Solution (Cadence).

Masked S-box Implementations Starting with the S-box implementation results depicted in Table 4.3, the new S-box architecture performs a computation in 6 cycles, which is 25% faster than the 8 cycles obtained by AGEMA for both pipeline and clock-gating synchronization strategies. As explained in 4.3.3, this is a direct effect of the input ports switching for AND2-HPC2 gadgets described in Section 4.3.2. Compared to the S-box with pipeline synchronization, our implementation is approximately 20% smaller (see Table 4.4). The first reason for this is the reduced number of synchronization registers enabled by the lower latency. A second reason for this difference lies in the implementation of the AND2-HPC2 gadget used by AGEMA: they require more registers than what is required, i.e., $4d(d-1) + 3d$ per multiplication gadget, while our implementation (which comes from fullVerif's library) only instantiates $2d + 7/2 \times d(d-1)$ registers per multiplication. Since both implementations are fully pipeline, they reach a throughput of one S-box evaluation per cycle.

The comparison with the S-box generated with the clock-gating synchronization strategy is more subtle. On the one hand, it avoids all synchronization registers, but on the other hand, clock gating logic is added, and the multiplication gadgets are more expensive. The increased cost of the multiplication gadgets dominates for $d \geq 3$, while for $d = 2$ the AGEMA implementation is a bit smaller than ours. In addition to having a higher latency, the AGEMA clock-gated S-box has a much lower throughput.

Instance	Share [count]	Seq. area [GE]	Area [GE]	Latency [cycle]	Throughput [exec/cycle]
AGEMA c.g.*	2	2009	2972	8	0.125
AGEMA c.g.*	3	4625	6822	8	0.125
AGEMA c.g.*	4	8329	12 290	8	0.125
AGEMA pipe.†	2	3024	3981	8	1
AGEMA pipe.†	3	6168	8360	8	1
AGEMA pipe.†	4	10 400	14 356	8	1
New	2	2273	3213	6	1
New	3	4831	6705	6	1
New	4	8354	11 515	6	1

* Clock-gating synchronization mechanism.
† Pipeline synchronization mechanism.

Table 4.3: ASIC 65nm S-box implementation results (post-synthesis).

Instance	Shares	Area [%]*	Latency [%]*	Area [%]†	Throughput [%]†
New	2	8	−25	−19	0
New	3	−2	−25	−20	0
New	4	−6	−25	−20	0

* Compared to AGEMA with clock-gating synchronization.
† Compared to AGEMA with pipeline synchronization.

Table 4.4: ASIC 65nm S-box implementation results comparison (post-synthesis).

Masked AES Implementations The differences highlighted for the S-box are amplified when considering a full encryption core, as shown in Tables 4.5 and 4.6. Starting with the 8-bit serial architecture, our new implementation has a 6.4 times lower latency than the ones generated with AGEMA for both synchronization mechanisms. This is due to the fact that the hand-crafted control mechanism takes full advantage of the pipeline architecture of the S-box without adding superficial pipeline levels, speeding up the computation of the `SubByte` operation by processing up to 6 bytes of the same AES encryption in parallel instead of 1. Regarding the throughput, the 8-bit pipeline AGEMA implementation is better then ours: it is able to optimally use the S-box pipeline while our implementation dedicates cycles to other operations.

As for the area, our new implementation is roughly 2.6 times smaller than the one generated with pipeline synchronization. This difference is mainly due to the very large number of registers needed to achieve a complete round pipeline for the full AES state and round keys in the AGEMA pipeline implementation. Compared to the implementations synchronized with clock-gating, the area of our new architecture is slightly higher, but are of the same order of magnitude. In more details, the overheads observed for the 8-bit implementations vary from 14 % to 4 % (for 2, 3 and 4 shares) and are caused by the synchronization registers used in our implementations. In particular, besides the registers introduced in the S-box, $48d$ registers are used in the global architecture ($40d$ in the key schedule, as shown in Figure 4.10, and $8d$ in the global datapath, as depicted in Figure 4.8).

For the 128-bit implementations, we also achieve a latency reduction and a throughput increase compared to the AGEMA implementations, but the gain is much smaller than in the 8-bit case. Our implementation is slightly larger than the AGEMA clock-gated one, while the pipeline one is larger than ours.

Finally, we also report the result of the new 32-bit implementation (an architecture that was not given in [Kni+22b]). It performs a full encryption in 105 cycles, which is similar to what is achieved by the round-based implementation generated by AGEMA. On top of that, its area turns out to be significantly lower than what is achieved for round-based architectures, making it an interesting alternative when the area vs. latency trade-off is considered.

4.3.8 Physical Security

Formal Composition Verification with fullVerif The implementations have been validated using the last version[9] of the fullVerif tool, guaranteeing glitch and transition robust probing security.

Preliminary Leakage Assessment The second step leverages the TVLA testing methodology in order to validate practical security and rule-out issues that cannot be caught by fullVerif such as those due to (post-) synthesis optimizations. Namely, a fixed vs. random t-tests was used as a preliminary leakage assessment [GGJR+11]. The measurements were performed on a Sakura-G board running at 6 MHz. We conducted the acquisitions with a PicoScope 5244D sampling at 500 MS/s with 12-bit

[9]The fullVerif tool has been extended to incorporate the "Optimized composition approach" presented in [CS21], which ensures security against both glitches and transitions

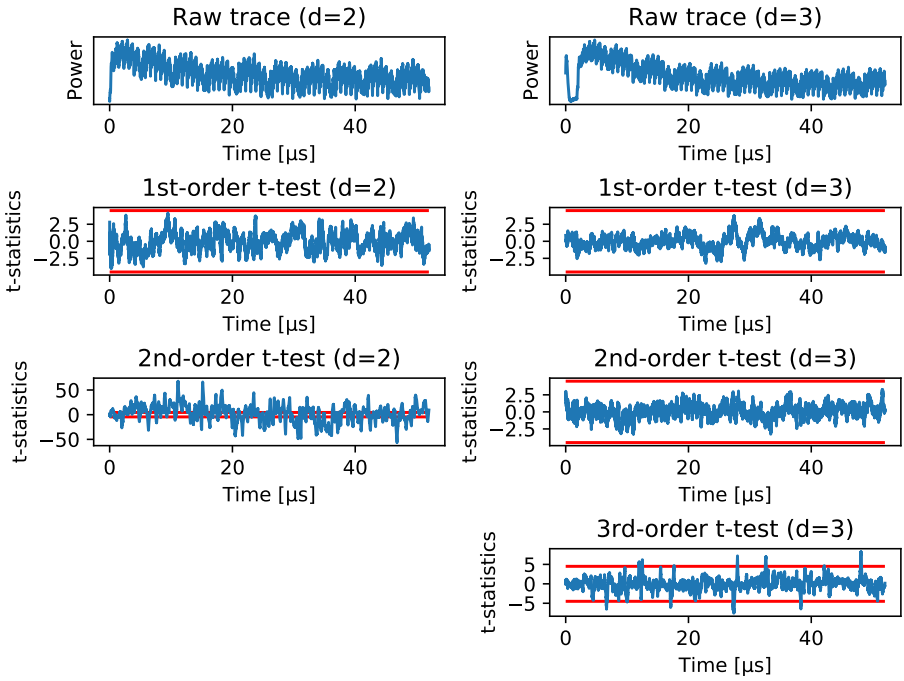


Figure 4.15: Fixed vs random T-test results (AES-128, 8-bit serial architecture).

resolution. As in [Kni+22b], each randomness bit was generated on-the-fly by a randomly seeded independent instance of the 31-bit maximum length LFSR presented in [Alf98]. As shown in Figure 4.15 for the 8-bit implementation, leakage can be observed starting at second order with 2 shares (1 M traces for both test orders) and at the third order with 3 shares (10 M traces for all test orders).

A non-specific fixed-vs-random Welch's T-tests has also been performed for the 32-bit implementation. This evaluation was carried out as a preliminary analysis of the SMAesH open-source core that relies on the 32-bit architecture[SC23b]. In particular, the power measurements were performed using a CW305-A100 evaluation board from NewAE. continuous supply voltage of 1 V is provided to VCC-INT by a low noise Keysight E36102B power supply through the dedicated banana jacks of the board, and the FPGA clock frequency is 1.5625 MHz (it is generated by the on-board PLL from the onboard crystal). We probe the shunt measurement at the dedicated low noise amplified signal point X4 with a digital oscilloscope Picoscope 6424E from PICO TECHNOLOGY. The measurements are performed at 5 GS/s with an enhanced vertical resolution of 10 bits. The clock of the oscilloscope is synchronized to the clock

of the FPGA using a 10 MHz clock derived from the PLL on the CW305 board. Such a synchronization, combined with the high sampling rate, ensures good trace alignment. To further reduce the trace misalignment caused by clock phase drift, a simple trace re-alignment is applied: the acquired traces are shifted by the (small) integer number of samples that maximizes the correlation to a reference trace. The size of traces is then reduced by aggregating (i.e., summing) 16 measurements samples into a single one, resulting in a practical sampling rate of 312.5 MHz and an equivalent vertical resolution of 14 bits. The randomness was generated on-the-fly by an embedded PRNG based on multiple instances of the Trivium stream cipher (see [SC23a] for a detailed description).

In particular, first- and second-order univariate t-tests have been conducted over the full execution. Following [Din+17], the significance threshold was set to 5.034, corresponding to a significance threshold $\alpha = 0.01$ for $n_L = 21000$ leakage points in the trace (this only includes the 10 AES rounds). We first performed a t-test with the countermeasure disabled, that is, when the PRNG seed is the same for all the traces. The result is shown in Figure 4.16. Significant first-order and second-order leakage can be observed in this setting (for all the AES rounds). Besides, the t-statistic at first order is larger than the one observed at second order (around 5 times larger). Second, Figure 4.17 shows the t-test results when the PRNG is enabled. In that configuration, no more leakage is detected with the t-test at first order, while peaks of t-value above the threshold are still observable along the time frame of an execution at second-order.

We do not detect first-order leakage after 1 million traces, while a significant amount of second-order leakage is detected. As usual with leakage detection, the preliminary results shown in Figure 4.154.164.17 do not guarantee that first-order leakage cannot be detected with more traces. Yet, it allows us to conclude that that masking is effective for the noise level of our implementation and that the best attacks will exploit second-order leakage.

Public Evaluation A public security evaluation of the 32-bit implementation was organized in the form of the CHES2023 conference challenge (i.e., the SMAesH challenge[10]). The latter is a side-channel analysis contest where participants were trying to mount side-channel attacks against the (open-source) SMAesH core[11] protected at the first order

[10] `https://smaesh-challenge.simple-crypto.org/`
[11] `https://github.com/simple-crypto/SMAesH`

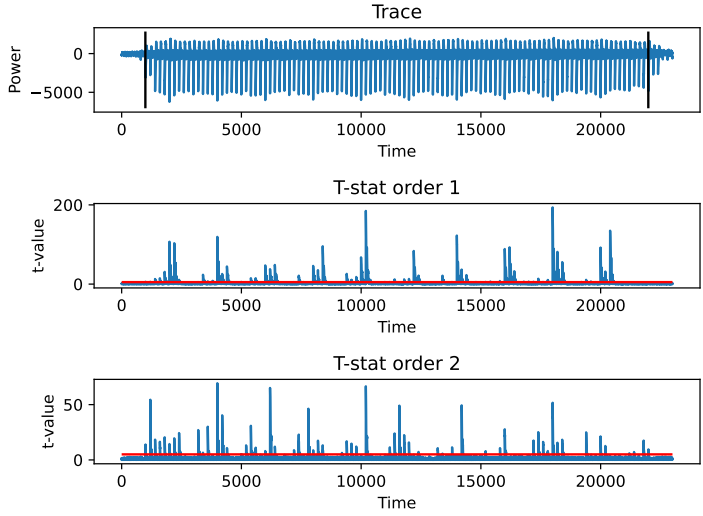


Figure 4.16: TVLA with PRNG disabled. The vertical black delimiters mark the range of 21000 samples corresponding to one AES encryption.

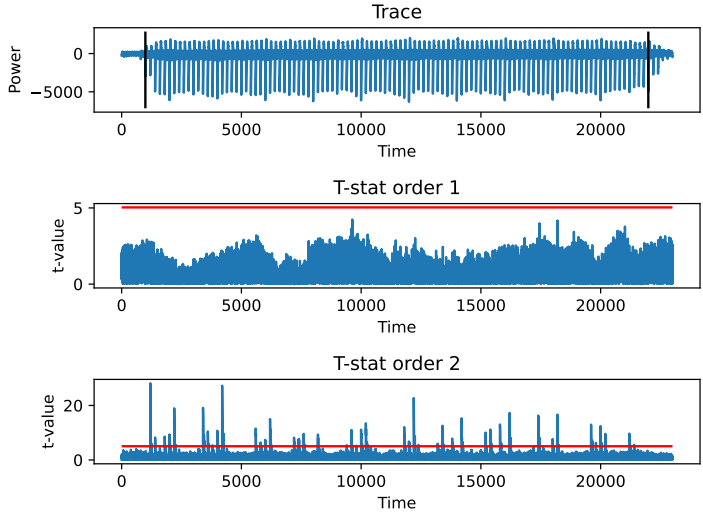


Figure 4.17: TVLA with PRNG enabled.

(i.e., using 2 shares). An attack was considered as successful if the resulting key guess had a rank below 2^{68}. Power measurements traces collected for two different FPGAs were made publicly available: a Spartan6 collected from a SakuraG board and an Artix7 collected from a CW305 board. In particular, three datasets (of $\approx$ 16M traces each) have been collected for each target: a training set with random inputs for each trace, a validation set with a (random) fixed key value used for every trace with a different plaintext and a test set (same as the validation set, but kept secret in order to evaluate submissions). Together with measurements, the input data of each execution were provided, namely the unmasked 128-bit plaintext, the unmasked 128-bit key. Additionally, the seed used to generate randomness as well as the shared key and plaintext values were provided for the Artix7 target[12]. Together with the documentation, an evaluation framework with a working demo attack was provided to help/motivate people developing an attack.

The evaluation lasted for 4 months and a total of 112 attacks (77 including 5 successful against the Artix7, 35 including 6 successful against the Spartan6) were submitted from 4 different teams. The evolution of the attack performances against the two targets are depicted in Figure 4.18. In particular, it shows that the current best attacks achieving a key enumeration complexity below 2^{68} require 290k traces against the Artix7 target and $\approx$ 900k traces against the Spartan6 target, illustrating that the architecture remains challenging to attack in practice considering these targets, even in a worst-case scenario. More information related to the attacks strategies can be found by directly downloading the (open-source) submissions from the challenge website.

4.4 Concluding remarks and recent extensions

This section investigates the possibility to reduce the overhead inherent to the pipeline architecture of masked design relying on the provably secure HPC2 masking scheme. In particular, the register stages required to ensure the security in the presence of glitches involve significant overheads on the area and latency of the masked circuit. A first cause is directly inherent to the sequential nature of multiplication gadgets, implying de facto more cycles (and logic) required to execute them. The other stems from the need for synchronization mechanisms, due to the asymmetrical nature of multiplication widgets. The generic strategy de-

[12] see `https://smaesh-challenge.simple-crypto.org/datasets.html` and `https://smaesh-challenge.simple-crypto.org/targets.html` for a complete description of the datasets

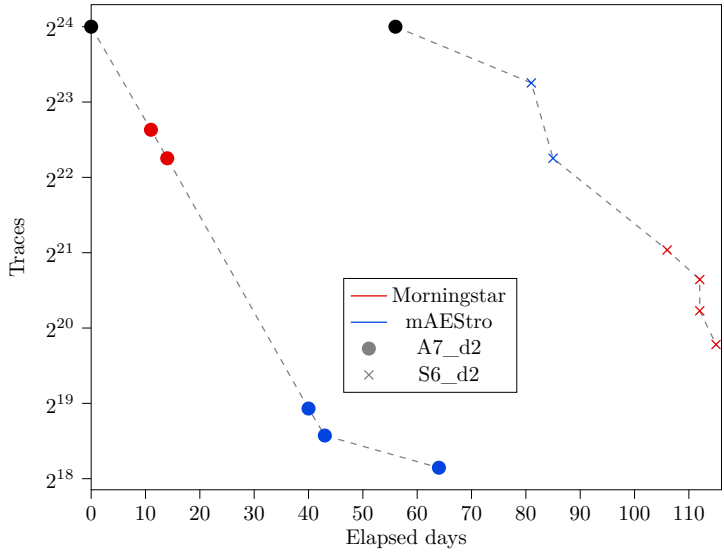


Figure 4.18: Attack complexicity to reach a key rank of 2^{68} during the SMAesH competition. Colors represents different teams while symbols represents different target.

scribed in Section 4.3.2 is a first step when trying to limit both (combined with the search for low multiplicative depth representations).

This idea forms the basis of an extension work aimed at further reducing latency and the cost of pipeline architecture using the HPC masking scheme [Cas+23b]. This leverages the fact that the HPC variants (with different trade-offs in term of latency, randomness requirement and area) can be combined together to achieve better latency versus area trade-off while leveraging the PINI property to maintain the security. Additionally, we explore the possibility of further optimizing synchronization registers, such as allowing operation rescheduling or gadget replications to avoid needlessly instantiating more expensive registers. As an example, area and latency gains of respectively 20% and 10% can be achieved when implementing the presented 32-bit AES with two shares, taking into account the randomness generation cost.

At a higher level, the integration of masked Clyde-128 in spook confirms that the throughput of the unprotected primitive can prevail for sufficiently long messages in the leveled implementation framework. This is made possible at the expense of additional implementation costs, which is quickly dominated by the DPA-protected primitive, as already confirmed in Chapter 3.

Eventually, a general observation is that efficient masked hardware implementations leverage on some level of parallelism to operate simultaneously on individual shares which also comes with the need of fresh random bits. More specifically, this chapter confirms that significant throughput of random bits are expected for implementations based on the HPC2 scheme. As an indicative example, the 32-bit AES implementation presented requires $68d(d-1)$ bits per cycle. Despite this huge demand, the efficient generation and distribution of these bits is often overlooked by assuming the existence of robust and high-entropy randomness sources, which leads to bias in the practical evaluation of the countermeasure cost. A recent study tries to tackle the issue [Cas+23c]. To this end, TRNGs are quickly discarded in favor of PRNGs because of their poor performance. Under such consideration, LFSRs are of particular interest because of their ability to generate many bits in one cycle at low cost, thanks to unrolling (i.e., repetition of the update function combinational logic, as for example used in the Spook implementation presented in [MCS21]). However, it is demonstrated that this strategy may lead to security issues if not handled with care (i.e., if the randomness generation is not analyzed together with the masking scheme). Instead, the study proposes the stream cipher Trivium as a promising alternative, being nearly as cheap to implement but enjoying the property of a stream cipher.

Instance	Share [count]	Seq. area [GE]	Area [GE]	Latency [cycle]	Throughput [exec/cycle]
AGEMA 8-bit c.g.*	2	4068	9356		
AGEMA 8-bit c.g.*	3	7678	16 319	2043	0.000 49
AGEMA 8-bit c.g.*	4	12 375	24 919		
AGEMA 8-bit pipe.†	2	25 055	30 338		
AGEMA 8-bit pipe.†	3	38 667	47 302	2043	0.0044
AGEMA 8-bit pipe.†	4	53 382	65 921		
New 8-bit	2	4790	10 634		
New 8-bit	3	8571	17 591	322	0.0031
New 8-bit	4	13 315	25 915		
New 32-bit	2	11 139	19 598		
New 32-bit	3	22 399	36 776	105	0.0095
New 32-bit	4	37 511	59 217		
AGEMA 128-bit c.g.*	2	40 198	63 613		
AGEMA 128-bit c.g.*	3	92 909	143 100	99	0.010
AGEMA 128-bit c.g.*	4	167 376	254 948		
AGEMA 128-bit pipe.†	2	86 317	109 725		
AGEMA 128-bit pipe.†	3	161 789	211 969	99	0.091
AGEMA 128-bit pipe.†	4	259 307	346 880		
New 128-bit	2	47 597	73 699		
New 128-bit	3	99 859	148 129	70	0.10
New 128-bit	4	171 274	249 011		

* Clock-gating synchronization mechanism.
† Pipeline synchronization mechanism.

Table 4.5: ASIC 65nm AES encryption implementation results (post-synthesis).

Instance	Shares	Area [%]*	Latency [%]*	Area [%]†	Throughput [%]†
New 8-bit	2	14	−84	−65	−30
New 8-bit	3	8	−84	−63	−30
New 8-bit	4	4	−84	−60	−30
New 128-bit	2	16	−30	−33	10
New 128-bit	3	4	−30	−30	10
New 128-bit	4	−2	−30	−28	10

* Compared to AGEMA with clock-gating synchronization.
† Compared to AGEMA with pipeline synchronization.

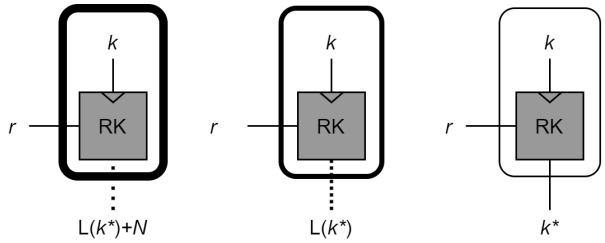
Table 4.6: ASIC 65nm AES enc. implem. results comparison (post-synthesis).

5 DPA security through fresh re-keying with (easier) masking

Protecting common symmetric primitives such as block cipher implementations against side-channel attacks is a difficult problem. Countermeasures like masking [Cha+99; ISW03] are expensive and are also error-prone due to physical defaults such as glitches [MPG05] or transitions [Cor+12], and due to composability issues [Cor+13; Bar+16]. Informally, this situation is caused by the complex (nonlinear) nature of the (tweakable) block ciphers. While the linear parts of an implementation can be trivially secret-shared with limited complexity overheads, the secure execution of its non-linear parts typically implies overheads that are quadratic in the number of shares. As a result of these limitations, the concept of fresh re-keying (as detailed in 2.4.2) was introduced [Med+10].

Theoretically, block ciphers and tweakable block ciphers are well understood primitives, with known instances that are secure and efficient (as long as they do not require strong protections against side-channel attacks). By contrast, specifying the properties of the re-keying function turns out to be challenging, and requires adjusting the trade-off between the re-keying function's efficiency and the physical assumptions needed for its secure implementation. Intuitively, this can be explained from the fact that, in addition to being easily protected against DPAs, re-keying function must also ensure some security properties regarding the generated session key k^*. Indeed, the session key is used sooner or later by the encryption function, which is only protected against SPAs. In that a situation, the amount of information about k that an adversary can extract from the leakage of k^* depends on both the re-keying and the circuit's leakage function.

Fresh re-keying comes thus with a broad design space, which is reflected by existing solutions that consider different adversarial models. On the first side of the adversarial spectrum and in addition to the fact that the re-keying function has to be protected against side-channel attacks, the heuristic solution presented in [Med+10; Med+11] assumes that the adversary can only observe the noisy leakages of the fresh key, as illustrated in Figure 5.1(a). On the other side of the spectrum, the



(a) Noisy leakages. (b) Noise-free leakages. (c) Unbounded leakages.

Figure 5.1: Fresh re-keying adversarial models (from weaker to stronger).

solution from [Dzi+16] rather consider that the adversary can observe the fresh key in full (i.e., unbounded leakages), as illustrated in Figure 5.1(c).

In this context, Hard-*physical* learning problems have been introduced as a possible solution to limit the implementation overhead of re-keying functions. Conceptually, their goal is to leverage the physical features of an implementation in order to generate errors or to round, as required to implement primitives based on hard learning problems. Put in another way, they aim to leverage the properties of the leakage function to limit or avoid the cost of expensive operations. Hard-learning problems have been shown to be an important source of cryptographic hardness. Popular instances of such problems include Learning Parity with Noise (LPN), Learning With Errors (LWE), which is a generalization to larger moduli [Reg05], and Learning With Rounding (LWR), which is a deterministic version [BPR12]. Intuitively, they express the difficulty of solving a linear algebraic system based on erroneous observations (even when it is straightforward based on error-free observations). They have found many applications for the design of basic cryptographic primitives and have also gained particular interest in the context of post-quantum cryptography, as witnessed by the signature scheme CRYSTALS-Dilithium [Duc+18] and the encryption scheme CRYSTALS-Kyber [Bos+18], which have been selected by the NIST to become future standards. From the protected implementation point of view, their algebraic structure makes a (key-homomorphic) part of their implementation quite amenable to masking against side-channel attacks.

Yet, the existing solutions still face limitations that make their practical use challenging. First, the proposal of Medwed et al. was a heuristic one, based on a list of necessary properties for the re-keying function. These properties include good diffusion and simplicity to protect against

side-channel attacks thanks to masking. Concretely, the instance chosen in [Med+10; Med+11] consists in a finite field multiplication ($k^* = k \cdot r$ over $\mathbb{F}_{2^\kappa}$), which is easy to mask since key-homomorphic. It was then shown by Belaid et al. that in the standard case where Hamming weight(HW) leakages are observed by an adversary, a lack of physical noise makes this solution easy to break. Indeed, the least significant bit of the Hamming weight of a value is the XOR of its bits. Hence, for a known r, the least significant bit of $\mathsf{HW}(k \cdot r)$ is a linear function of the bits of k. After κ leakages on average, solving the resulting system of equations therefore leads to the full master key [BFG14]. This attack was then improved in order to remain effective with lower SNR in [Bel+15; PM16; GJ19] and it was shown in the first part of [Dzi+16] that the "Learning Parity with (Gaussian) Leakage" (LPL) problem on which this fresh re-keying scheme relies indeed requires significant noise levels (i.e., more than what is available intrinsically in unprotected implementations).

Second, Dziembowski et al. studied the possibility to use a weak Pseudo-Random Function (wPRF) as re-keying function [Dzi+16]. Their proposal is to perform (several) inner product computations $\langle \boldsymbol{k}, \boldsymbol{x} \rangle$ where $\boldsymbol{k}$ and $\boldsymbol{x} \in \mathbb{Z}_{2^q}^n$ and to "round" the results by computing $\lfloor \langle \boldsymbol{k}, \boldsymbol{x} \rangle \rfloor_\rho$ which simply drops $q - \rho$ bits. The security of this re-keying can be reduced to a Learning With Rounding (LWR) assumption [BPR12; Alw+13], and it is almost key-homomorphic: only the carry bits involved in the shared computations break the key-homomorphism. Hence, by complementing this re-keying with a light error correction scheme (the cost of which increasing logarithmically with the number of shares), this wPRF can be efficiently masked. Yet, despite complexity overheads that are close to linear in the number of shares, such a solution suffers from the large key size that it requires (e.g., 128-bit security is conjectured for a key of $n = 128 \times q = 32$ bits), the manipulation of which leading to relatively poor performances.

Given these observations, this chapter focuses on a new re-keying function that considers the intermediate model depicted in Figure 5.1(b), assuming that an adversary has access to the noise-free leakage of the fresh key. It has to be noted that, as usual when introducing a new cryptographic primitive, the later aims at exhibiting relevant security and implementation properties which may open new research directions. In this respect, this chapter illustrates that, taking advantage of the implementation simplifications made possible by the key homomorphism properties and parallelism, the (newly) proposed re-keying scheme is more efficient than the one of Dziembowski et al. [Dzi+16] and at the

same time more secure than the one of Medwed et al. [Med+10] under the commonly used Hamming Weight model.

More into the details, the chapter focuses on the implementation aspects of the *crypto-physical dark matter* fresh re-keying scheme, based on the novel hard-physical learning problem Learning-With-Physical-Rounding (LWPR) [Duv+21]. In particular, it highlights the competitive implementation properties enabled. Informally, these properties are due to the fact that, contrary to re-keying schemes in the model of figure 5.1(c) where the mapping/rounding has to be computed securely (e.g., thanks to masking), the (physical) mapping/rounding introduced with the LWPR hard-physical learning problem never has to be computed securely in the model of figure 5.1(b), since it is performed by a leakage function. Besides, experimental results are provided in order to strengthen the security analysis of the scheme. In particular, a physical security analysis is provided for a proof-of-concept hardware implementation and the chapter concludes by presenting the results of an analysis conducted in order to assess the validity of adversarial model generalization to a broader class of leakage function than the Hamming weight leakage model intially considered.

In this chapter, I was in charge of all the experimental part. Namely, I worked on the design, implementation and performances evaluation of the masked architectures of the re-keying function presented. I was also in charge of the experimental setup and of the side-channel security analysis conducted in Section 5.2.3 as well as the leakage experimental characterization of Section 5.3.2.

5.1 Learning With Physical Rounding Problem: The crypto-physical dark matter fresh re-keying scheme

5.1.1 Background and Motivations

***Cryptographic dark matter* wPRF** At Crypto 2018, Boneh et al. proposed a wPRF candidate the is instantiated as follows [Bon+18]. First, it takes a secret matrix $\boldsymbol{K} \in \mathbb{F}_2^{m \times n}$ (possibly Toeplitz for efficiency) and a public vector $\boldsymbol{r} \in \mathbb{F}_2^n$. Next, it computes the product $\boldsymbol{K} \cdot \boldsymbol{r}$ and it interprets the output of this product as a vector of 0/1 values over $\mathbb{F}_3$. Finally, the output of the wPRF is the sum of these values modulo 3. In other words, their wPRF can be defined as:

$$\mathsf{F}_{\boldsymbol{K}}(\boldsymbol{r}) := \mathsf{map}(\boldsymbol{K} \cdot \boldsymbol{r}), \tag{5.1}$$

with $\mathsf{map} : \{0,1\}^m \to \mathbb{F}_3$ that maps $\boldsymbol{y} \in \{0,1\}^m$ to $\sum \boldsymbol{y}_i$ mod 3. The similarity between this function and the one of Dziembowski et al. is striking: the matrix multiplication can be seen as multiple inner products and the mapping function plays the role of the rounding. Its limitations for masked implementations are therefore similar: the mapping function is nonlinear over $\mathbb{F}_2^n$ which requires special care and implies overheads.

Re-keying with noise-free leakage: exploring *Crypto-physical dark matter* The main research question tackled in this first section is whether a secure re-keying scheme in the adversarial model of Figure 5.1(b) can be built, by leveraging some "crypto-physical dark matter". By this, we mean building a wPRF combining a matrix multiplication with a physical mapping that would not have to be computed explicitly (i.e., with a dedicated circuitry) and would rather be performed in an analog manner by an implementation's leakages. This section focuses on the implementation and physical security aspects of the proposed re-keying scheme, but a mathematical analysis of the relevant security properties (not a part of this thesis) can be found in the full publication [Duv+21]. Briefly, the latter put forward that multiplications in a (bigger) prime field can lead to secure (and efficient) candidates considering the frequently observed Hamming weight leakage function [MOP07]. Besides, relying on the (free) operation performed by the leakage function, masking with a small key and complexity overheads that are linear in the number of shares can theoretically be obtained.

5.1.2 Notations

In this chapter, we denote vectors with bold letters $\boldsymbol{v}$ and matrices with bold capital letters $\boldsymbol{M}$. We use the log notation for the logarithm in basis 2. For $n \in \mathbb{N}$, we denote by $[n]$ the set of integers from 1 to n and by $[0, n]$ the set of integers from 0 to n. Additionally, the term crypto-physical dark matter is used to refer to the re-keying operations and the term LWPR for the problem of recovering the long-term key of the resulting re-keying scheme.

5.1.3 Physical model & LWPR

In order to define the LWPR problem, we need to define the physical model we are working with. In this respect, the main issue is that we must specify crypto-physical dark matter computations that mix mathematical operations and physical ones. For this purpose, we first

observe that the elements of the vector $\boldsymbol{y} = \boldsymbol{K} \cdot \boldsymbol{r}$ are in $\mathbb{F}_p$, with p a prime number. We then formalize as "physical rounding" the function modeling the side-channel information that an adversary gets from noise-free leakages on this vector. The physical model we will consider for the rounding is a composition of two (more or less specialized) assumptions.

On the one hand, we assume that the leaking device computes on binary-represented data: each value in $\mathbb{F}_p$ is therefore represented with (at least) $\lceil \log p \rceil$ bits. We denote as $\mathsf{g} : \mathbb{F}_p \rightarrow \{0,1\}^{\lceil \log p \rceil}$ the function associating to each element of $\mathbb{F}_p$ the binary representation of its representative in $[0, p-1]$. We also define $\mathsf{g}_m : \mathbb{F}_p^m \rightarrow \{0,1\}^{m\lceil \log p \rceil}$ as: $\mathsf{g}_m(\boldsymbol{v}) := \mathsf{g}(v_1)||\mathsf{g}(v_2)|| \ldots ||\mathsf{g}(v_m)$. We argue that this assumption is quite generic and captures the reality of most embedded computing devices deployed in current applications.

On the other hand, we need a more specialized assumption defining how the physical (noise-free) leakages depend on the $m\lceil \log p \rceil$ bits provided by g_m. This role will be played by the leakage function. We denote the leakage function computed on the binary representation of the manipulated data as $\mathsf{L}_\mathsf{g}(.)$ and use it as a parameter of our investigations.

The generic LWPR problem is defined as follows.

Definition 3 (Learning with physical rounding)**.** *Let $p, n, m \in \mathbb{N}^*$, p prime, for (unknown) $\boldsymbol{K} \in \mathbb{F}_p^{m \times (n+1)}$. The $\mathrm{LWPR}_{\mathsf{L}_\mathsf{g},p}^{n,m}$ sample distribution is given by:*

$$\mathcal{D}_{\mathrm{LWPR}_{\mathsf{L}_\mathsf{g},p}^{n,m}} := (\boldsymbol{r}, \mathsf{L}_\mathsf{g}(\boldsymbol{K} \boxdot \boldsymbol{r})) \text{ for } \boldsymbol{r} \in \mathbb{F}_p^n \text{ uniformly random,}$$

where $\boldsymbol{K} \boxdot \boldsymbol{r} = \boldsymbol{K} \cdot (\boldsymbol{r}, 1)$ and $\mathsf{L}_\mathsf{g} : \mathbb{F}_p^m \rightarrow \mathbb{R}^{n_d}$ is the physical rounding function. Given query access to $\mathcal{D}_{\mathrm{LWPR}_{\mathsf{L}_\mathsf{g},p}^{n,m}}$ for a uniformly random $\boldsymbol{K}$, the $\mathrm{LWPR}_{\mathsf{L}_\mathsf{g},p}^{n,m}$ problem is (q, τ, μ, ϵ)-hard to solve if after the observation of q LWPR samples, no adversary can recover the key $\boldsymbol{K}$ with time complexity τ, memory complexity μ and probability higher than ϵ.

In this definition, $\boldsymbol{K} \boxdot \boldsymbol{r}$ denotes the crypto-physical dark matter operations. Note that $\boldsymbol{K}$ is multiplied with $(\boldsymbol{r}, 1)$ rather than $\boldsymbol{r}$. The justification for such a design can be found in the security analysis of [Duv+21]. As already mentioned, the later concerns the Hamming Weight leakage function which frequently observed for standard CMOS devices [MOP07]. More formally, we first denote the Hamming weight function $\mathsf{HW}(\boldsymbol{v})$, defined on any vector $\boldsymbol{v}$ of length $t \in \mathbb{N}^*$ with coefficients in $\{0,1\}$ as $\mathsf{HW}(\boldsymbol{v}) = \sum_{i=1}^{t} \boldsymbol{v}_i$, where the sum is performed in $\mathbb{Z}$. We then consider two possible implementations of it:

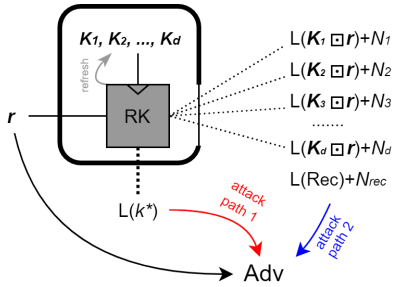


Figure 5.2: General adversarial model of our re-keying scheme.

- Parallel: $\mathsf{L}_{\mathsf{g}}^{p}(\boldsymbol{y}) : \boldsymbol{y} \mapsto \mathsf{HW}(\mathsf{g}_m(\boldsymbol{y})) = \mathsf{HW}(\mathsf{g}(\boldsymbol{y}_1)) + \mathsf{HW}(\mathsf{g}(\boldsymbol{y}_2)) + \ldots + \mathsf{HW}(\mathsf{g}(\boldsymbol{y}_m))$.
- Serial: $\mathsf{L}_{\mathsf{g}}^{s}(\boldsymbol{y}) : \boldsymbol{y} \mapsto \Big(\mathsf{HW}(\mathsf{g}(\boldsymbol{y}_1)), \mathsf{HW}(\mathsf{g}(\boldsymbol{y}_2)), \ldots, \mathsf{HW}(\mathsf{g}(\boldsymbol{y}_m))\Big)$.

The parallel case is reflecting a hardware implementation where the m vector elements are computed in parallel. The serial case is reflecting an intermediate level of parallelism implementation where the m vector elements are computed one by one. The serial case is a significantly more challenging context for securing implementations against side-channel attacks than the parallel case, since a b-bit Hamming weight function leaks approximately $\log(b)$ bits on average to the adversary.

5.1.4 A general adversarial model

While trying to break the LWPR assumption is one natural path to attack our re-keying scheme, it is not the only option. Side-channel security also relies on the fact that the re-keying function itself is well protected. Since crypto-physical dark matter computations are key-homomorphic, masking is a natural candidate for this purpose. It leads to the general adversarial model of Figure 5.2, where the first (red) attack path targets the leakage of the re-combined ephemeral key $\mathsf{L}(k^*)$ (i.e., the LWPR assumption) while the second (blue) attack path targets the noisy leakages of the shared computations $\boldsymbol{K}_1 \boxdot \boldsymbol{r}$, $\boldsymbol{K}_2 \boxdot \boldsymbol{r}$, $\ldots$, $\boldsymbol{K}_d \boxdot \boldsymbol{r}$ together with the leakage generated by the shares' recombination.

We next formalize this adversarial model, starting with a definition of our re-keying scheme and following with the key recovery experiment it aims to keep hard.

Definition 4 (Re-keying scheme)**.** *Let $n \in \mathbb{N}$ be a security parameter, and $d \in \mathbb{N}$ a number of shares. A re-keying scheme* RK *is made of the polytime algorithms:*

- $\mathsf{Gen}(1^n, d)$. *Generates the long-term key* $\boldsymbol{K}$ *and the initial sharing* $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_d$.
- $\mathsf{SharedMult}(\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_d, \boldsymbol{r})$. *Generates the shares of the ephemeral key* k^* *(i.e.,* $k_1^*, k_2^*, \ldots, k_d^*$*) from the* d *shares* $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_d$ *and a randomness* $\boldsymbol{r}$.
- $\mathsf{Rec}(k_1^*, k_2^*, \ldots, k_d^*)$. *Generates the ephemeral key* k^* *by recombining its shares.*
- $\mathsf{Refresh}(\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_d)$. *Replace the shares of the long-term key by fresh ones.*

Definition 5 (Side-channel key recovery experiment $\mathsf{Exp}^{\mathsf{skr}}_{\mathcal{A},\mathsf{RK}}(n, d)$)**.** *The experiment processes in three (setup, challenge and final) phases specified as:*

- *Setup phase. A long term key* $\boldsymbol{K}$ *is generated in function of the security parameter* n *and it is split into shares* $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_d$ *using the* $\mathsf{Gen}(1^n, d)$ *algorithm.*
- *Challenge phase. The adversary* $\mathcal{A}$ *performs* q *re-keying queries. For each query, the vector* $\boldsymbol{r}$ *is chosen at random, the* $\mathsf{SharedMult}$ *algorithm is computed on the shares of the long-term key and* $\boldsymbol{r}$*, and the shares of the long-term key are refreshed. The vector* $\boldsymbol{r}$ *is then given to* $\mathcal{A}$ *with the following leakages:*
  - $\mathsf{L}(\boldsymbol{K}_i \boxdot \boldsymbol{r}) + N_i$*, for* $i \in [d]$ *and some noise* N_i *– the shares' leakages.*
  - $\mathsf{L}(\mathit{Rec}) + N_{rec}$*, for some noise* N_{rec} *– the leakage from the shares' recombination.*
  - $\mathsf{L}(k^*)$*, the noise-free leakage from the ephemeral key* k^*.
- *Final phase. The adversary* $\mathcal{A}$ *outputs a candidate for the long-term key* k'*. The output of the experiment is defined to be* 1 *if* $k' = k$ *and* 0 *otherwise.*

We say that $\mathcal{A}$ *succeeds, or breaks the re-keying scheme* RK*, with probability* ϵ*, time complexity* τ*, memory complexity* μ *and* q *queries if after* q *queries in a challenge phase bounded by these time and memory complexities,* $\mathsf{Exp}^{\mathsf{skr}}_{\mathcal{A},\mathsf{RK}}(n, d) = 1$ *with probability* ϵ.

The leakage $\mathsf{L}(k^*)$ of Figure 5.2 has to be understood as all the leakage that can be obtained on k^*. The starting assumption is the one of an adversary who does not obtain significantly more information than the

Hamming weight of k^*. Analyzing more general classes of leakages is an important open problem and will be covered in Section 5.3.1. The first attack path from Figure 5.2 is the topic of the analysis from [Duv+21]. The second attack path and how to chose the number of shares to reach a given security level will be covered in Section 5.2.3. While the LWPR problem is stated for noise-free leakages (which is important since the ephemeral key is unshared and can be used in an environment with small amount of noise), the secure implementation of masking generally requires a certain level of noise. Yet, contrary to the masking of nonlinear operations for which this necessary level of noise may increase with the number of shares [Bat+16], the masking of a key-homomorphic primitive only requires a constant noise rate. Other advantages of the re-keying approach for masking are recalled in Section 5.2.3.

5.2 Implementation results

The analysis performed in [Duv+21] shows that a re-keying based on crypto-physical dark matter is more secure than the initial proposal of Medwed et al. [Med+10], under the realistic HW leakage assumption. Besides, a practical instance based on the Mersenne prime $p = 2^{31} - 1$ is proposed with the parameters $m = 4$ and $n = 4$ aiming for the generation of a 128-bit key. Such an instance is expected to provide a security such that the first attack path in Figure 5.2 is more challenging than targeting directly the long-term key shares via a side channel attack.

We next show that the proposed instance can lead to significantly more efficient performances results than the wPRF proposal of Dziembowski et al. [Dzi+16] by comparing a crypto-physical dark matter toy implementation with the FPGA implementation results for the LWP based wPRF presented in [BSS20]. We also confirm that high security can be obtained against the second attack path of Figure 5.2 and provide a side-channel security analysis of a masked crypto-physical dark matter implementation.

5.2.1 Experimental hardware architecture

The results provided next rely on the crypto-physical dark matter architecture illustrated in Figure 5.3. As the architecture in [BSS20], it is designed to leverage the key-homomorphism of the re-keying function by processing the shares serially. It is divided in two main blocks. The first one is composed of the different memories that hold the shares of K, the value of r and the randomness required to refresh the shares. The latter

is embedded into the memory blocks and is added to each word of K after it has been read from memory. The second block is the computation core. It includes the logic to perform the different dot product operations (organized as a pipeline for efficiency) and an accumulator that recombines the intermediate results. For security and performance (in particular latency) reasons, 5 multiplications are performed in parallel, leading to an internal bus of 160 bits.

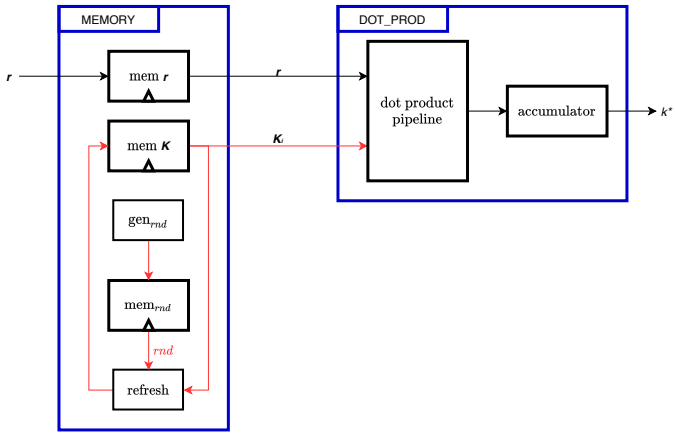


Figure 5.3: Overview of the crypto-physical dark matter hardware architecture.

The detailed architecture of the memory block is shown in Figure 5.4. The r memory consists of a 128-bit long register that is fed with the appropriate value at the beginning of an execution. Both the key and the randomness memories are composed of d (i.e., the amount of shares used) independent memories of $dm(n+1) \cdot 32 = d \cdot 640$ bits. In practice, these are mapped to BRAM blocks which are dedicated memory resources embedded in the FPGA. While processing a specific share, all the memories are read and the appropriate output is selected. The selection mechanism is straightforwardly done using a multiplexer for the randomness memory. An additional register barrier is added in front of the selection multiplexer in order to avoid physical defaults such as glitches that may lead to reductions of the security order [MPG05]. Additionally, the register barriers corresponding to memories that are not expected to be read are reset. The values that are read in memory are then directly forwarded to the computation pipeline.

A dedicated mechanism is used in order to refresh the key values that are forwarded to the computation core. Each share is directly re-

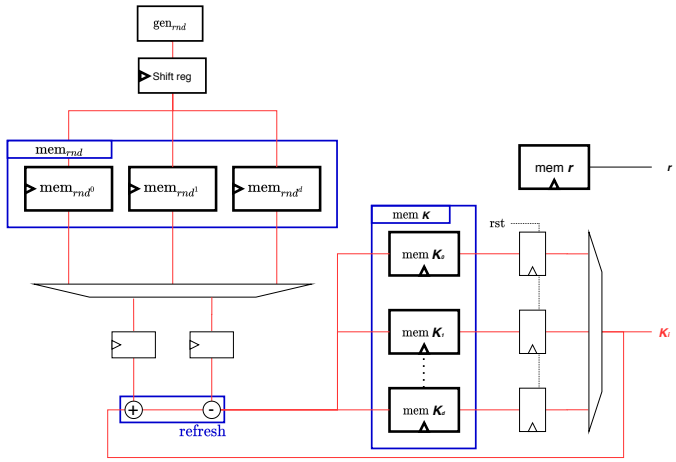


Figure 5.4: Architecture of the memory block.

randomized after being used, with a (linear) refresh mechanism that is similar to the one in [BSS20]. Namely, we first generate uniform 31-bit values thanks to four 128-bit LFSRs and output a uniform value over $\mathbb{F}_p$ thanks to a simple rejection sampling (i.e., we output a single 31-bit value that is different from 2^{31}, and use four LFSRs so that they jointly fail with low probability). We then refresh the key words and write them back in the appropriate memory.

The data output by the memory block directly feeds the computation core by entering the modular multiplication layer. As depicted in Figure 5.5, the latter in composed of 5 independent modular multiplications. We next refers to the amount of concurrent multiplication performed as the parallelism level P (with $P = 5$ for the architecture just described). We rely on the DSP resources embedded in the FPGA to do so, and more precisely on the multiplier units that they contain.

We use instances of the solution presented in [Kop+17] in order to perform the modular multiplications. This solution mixes an efficient utilization of the DSP resources combined with an adder tree specifically designed to limit the logic depth. Additionally, it allows easily incorporating the modular reduction with limited overheads.

Following the modular multiplication, the modular adder tree is split in a two-level pipeline to improve the maximal clock frequency that can be reached. It turns out the register between the second and the third addition layers has a significant impact on the clock frequency, improving

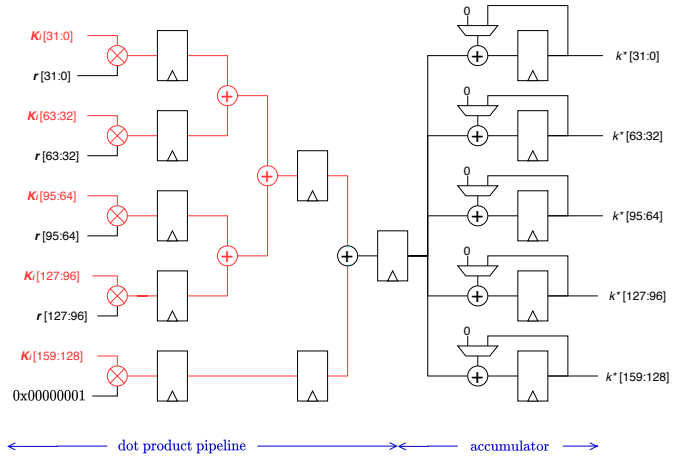


Figure 5.5: Architecture of the dot product pipeline.

its maximal value from by 60% at a low logic cost (see [Duv+21] for more implementation results).

Eventually, an accumulator ends the pipeline and is composed of 5 independent adders with feedback. As when a refreshed key share is written back in the memory, the data coming from the dot product pipeline is fed to all the adders and only a specific one is activated depending on the line processed. Following this configuration, the final session key value is obtained once all the lines of all the shares have been processed.

5.2.2 Performance evaluation

We next analyze different performance metrics for the new LWPR-based proposal and compare them with the LWR-based solution of Dziembowski et al. [Dzi+16]. In particular, the analysis performed demonstrate that a toy (FPGA) implementation of the latter performs competitively compared to the LWR-based solution presented [BSS20]. It should be remembered that even if significant differences are observed, these must be put into perspective when drawing conclusions, as the adversary models differ between the two proposals. For some relevant metrics, we report the $P = 4$ and $P = 8$ cases from [BSS20] which are the most comparable to our architecture using $P = 5$.

Starting with the size of the key storage, the Figure 5.6 simply illustrates the difference between the (128×32)-bit key that the LWR-based solution requires, to be compared with our (20×32)-bit key.

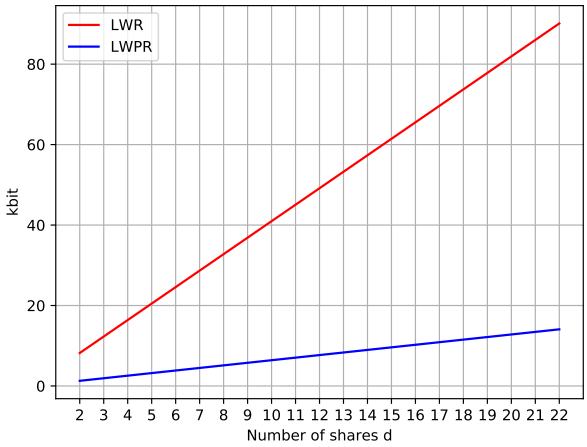


Figure 5.6: Key storage in function of the number of shares.

Next, the randomness requirements of the two proposals are illustrated in Figure 5.7. They are proportional to the key size, but the gap between the LWR-based solution and ours is amplified due to the fact that the output of each inner product in [Dzi+16] has to be rounded to 10 bits for security reasons, and $\lceil \log_2(d) \rceil$ bits are additionally lost due to the error correcting code that is needed in order to deal with the carry propagations that make the LWR-based wPRF almost key-homomorphic only.

Eventually, the metric that highlights the most performance gains of the crypto-physical dark matter is the latency given in Figure 5.8 It underlines that even when the LWR-based solution is implemented with a large level of parallelism (e.g., $P = 8$), the LWPR-based solution is orders of magnitude faster. Furthermore, the amount of cycles depicted for the LWPR-based solution takes into account the cycles requires to perform a protected TBC call, that is a full crypto-physical dark matter execution and an additional unprotected TBC call. More particularly, the lower part of the figure shows that for 20 shares, a full re-keyed TBC execution can be performed in approximately 160 cycles (to which one must add the cost to generate 12,500 random bits, as per Figure 5.7, which means a quite reasonable 80 bits per cycle if to be generated on-

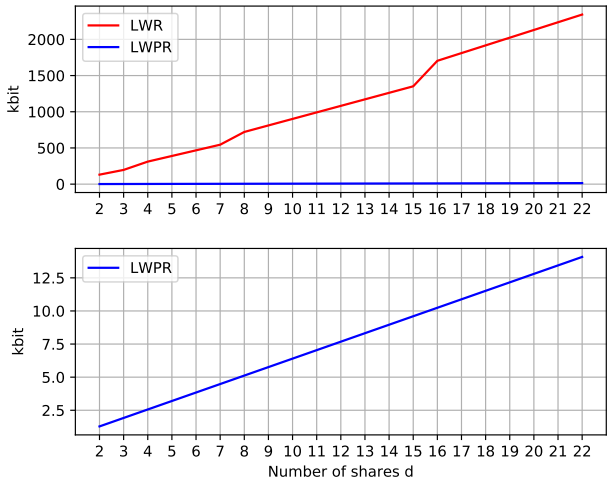


Figure 5.7: Randomness requirements in function of the number of shares.

the-fly) where the LWR-based re-keying requires roughly 10,000 cycles to generate the fresh key.

5.2.3 Physical security

We first recall some generic advantages of key-homomorphic primitives for masking, then describe how to evaluate/bound their concrete (e.g., power consumption) leakage following a worst-case strategy, and finally show how to choose the number of shares needed to reach a given security target thanks to this leakage bound.

Advantages of key-homomorphic primitives for masking. In short, and as carefully discussed in [BSS20], key homomorphic primitives enable a significant simplification of both the design of secure masked implementations and their evaluations.

A first interesting feature for this purpose is that they are trivial to analyze from a probing security viewpoint, since they enable the independent manipulation of the shares. Key-homomorhic primitives do not suffer from composability issues and the only refreshing they need is for the shares of their long-term key, which can be performed using cheap schemes (with linear overheads), as discussed in [Bar+17a], Section 8.2.

A second interesting feature is that they mitigate the risks of physical defaults (like glitches or transitions) leading to shares' re-combinations. This benefit again comes from the possibility to manipulate shares in-

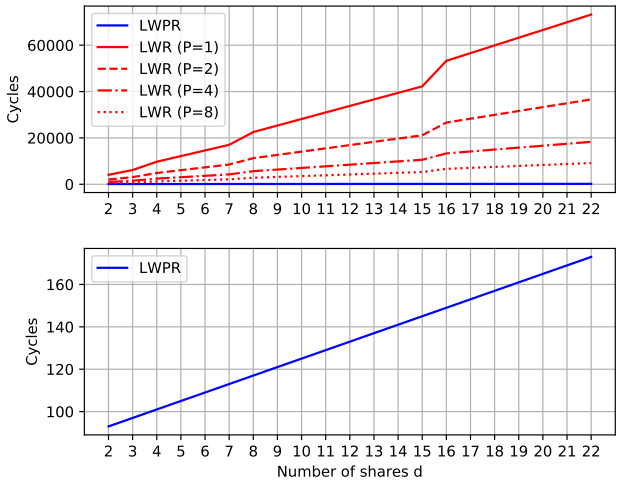


Figure 5.8: Latency in function of the number of shares.

dependently. In case of shares-serial implementations like the one we chose, they can additionally reduce the risks of couplings [Cnu+17].

A third advantage is that each key share is manipulated minimally (i.e., a constant number of times, independent of the security order). This ensures an inherently good resistance against horizontal attacks, formalized by a constant noise rate [Bat+16].

Eventually, a consequence of the previous advantages is that the evaluation of such masked implementations is scalable. For example, increasing the number of shares in a block cipher implementation usually implies some modification of the internal components that consequently requires repeating the (time-consuming) side-channel security evaluations for each security order. By contrast, increasing the number of shares of a key-homomorphic architecture boils down to re-using exactly the same component multiple times, avoiding the need to repeat evaluations.

Evaluation of the shares' leakages The side-channel security of a masked implementation depends on two main assumptions: the shares' leakages must be sufficiently noisy and independent [DFS19]. As discussed and evaluated in [BSS20], the independence is essentially guaranteed by design in a shares-serial key-homomorphic architecture. Since we use the same architecture, we do not detail the detection tests needed to confirm this assumption and rather focus on the level of noise in a prototype implementation.

For this purpose, we first synthetized our design for the Xilinx Spartan 6 FPGA available on the SAKURA-G board. Its clock frequency was set to 6 MHz. The synthesis was performed with the "keep hierarchy" flag avoiding the tool to trim out useful registers. The leakage signal was captured with a Tektronix CT-1 probe. This signal was sampled with a Picoscope 5244d at a rate of 500 MSamples/s, with 12-bit resolution.

An exemplary power traces is given at the top of Figure 5.9. It corresponds to a 3-share implementation that generates a fresh key in ≈ 12 cycles (i.e., 4×3 cycles, where the 4 factor is the number of 31-bit key words generated to obtain a 124-bit key and the 3 factor is the number of shares). As a standard first step in our evaluations, we estimated the 8-bit side-channel Signal-to-Noise Ratio (SNR) [Man04], which is represented at the bottom of the same figure. We observe that the level of SNR varies between bytes and selected the most leaking byte for our following (worst-case) investigations.[1]

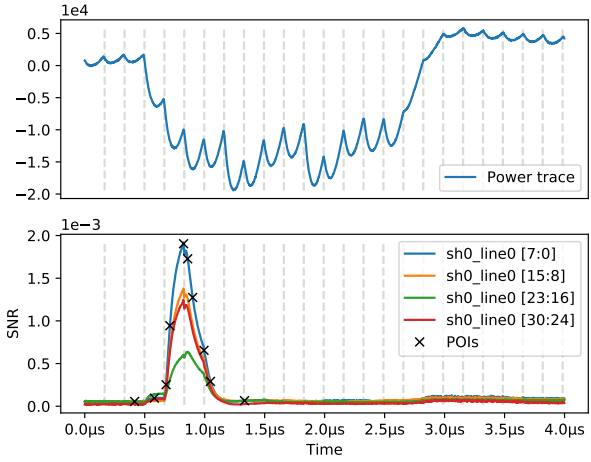


Figure 5.9: Top: exemplary leakage trace. Bottom: SNR for exemplary target bytes.

As a standard second step in our evaluations, we used the SNR plots to select Points-of-Interest (POIs) in the traces, represented by the crosses in Figure 5.9. We then estimated the information leakage that can be extracted from the corresponding multivariate distribution under a Gaussian assumption, using the information theoretic metrics and bounds

[1] All the SNRs in Figure 5.9 are for the first share. The following shares give slightly less leakage due to the progressive filling of the pipeline described in Figure 5.5, which generates algorithmic noise.

introduced in [Bro+19]. Namely, we estimated the Gaussian Perceived Information (gPI) and the Gaussian hypothetical information (gHI) using the sampling based estimation described in this reference. The convergence of these two metrics for the 10 POIs we selected is illustrated at the top of Figure 5.10. The gHI gives a bound on the amount of information that can be extracted with such a Gaussian model. We assume that it provides a reasonable approximation of the worst-case security level of our implementation (with the usual cautionary remark that better measurement setups and models can always improve the gPI and its corresponding gHI bound). Since we consider an 8-bit adversary while attacks based on 32-bit guesses can be performed by determined adversaries, we multiplied our leakage estimation by a factor 4 to make our approximations more conservative.

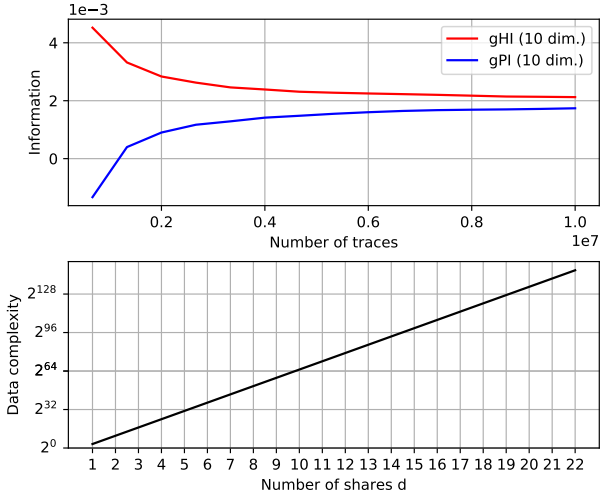


Figure 5.10: Top: convergence of IT metrics. Bottom: security level.

Selecting a number of shares Eventually, based on the previous estimations of the information leakages and assuming that the independence condition is fulfilled for our implementations, we use the results in [DFS19] in order to select the number of shares needed to reach a target security level. In short, they bound the data complexity N of any side-channel attack against a masked implementation by the inverse of the mutual information obtained on its shares (which we approximate with the aforementioned gHI) raised to the number of shares d. It leads

to the following inequality:

$$N \geq \frac{c}{\mathrm{MI}(\boldsymbol{K}; \boldsymbol{L})^d} \approx \frac{c}{\mathrm{gHI}(\boldsymbol{K}; \boldsymbol{L})^d},$$

with c a small constant depending on the size of the key hypothesis and the target success rate [Ché+19] (e.g., $c = 10$ appears to be a standard value for 8-bit guesses). The resulting data complexities are given at the bottom of Figure 5.10 in function of the number of shares. They confirm that high security can be obtained against the second attack path of Figure 5.2.

5.2.4 On the need of parallelism

As a reminder, the threat model described in 5.1.4 considers that the re-keying scheme uses a long term key to generate a fresh key that will then be used without masking for a single (e.g., block cipher) encryption. As a result, two natural paths are considered. The first one aims to recover the long term key from the (masked) product $\boldsymbol{K} \boxdot \boldsymbol{r} = \sum_{i=1}^{d} (\boldsymbol{K}_i \boxdot \boldsymbol{r})$ using side-channel. The analysis of the latter has been done in Section 5.2.3. The second option is to target the noise-free leakage of the value $\boldsymbol{K} \boxdot \boldsymbol{r}$ and try to recover the value of $\boldsymbol{K}$ out of it.

Given this threat model, the attack against serial LWPR implementations turns out to be pretty simple. Say we have an instance of LWPR with $p = 2^{31} - 1$ where the adversary can observe the Hamming weight of every 31-bit word of $\boldsymbol{K} \boxdot \boldsymbol{r}$. Then, she can filter the leakages and retain the ones with (worst-case) value 0, which has a single preimage. Every such event gives a linear equation in the key elements of the corresponding line of $\boldsymbol{K}$, and each such line can be recovered with $(n + 1)$ linear equations. Since p is only mid-size, these events happen with a concretely reachable probability $\frac{1}{p}$. As a result, after the generation $p\,(n + 1)$ LWPR samples, the full key is compromised with good probability.

Since it is hard to rule out that (close to) Hamming weight leakages can be observed in practice (as Section 5.3.2 will confirm experimentally), this attack suggests that serial instances of LWPR cannot be secure without additional countermeasure. A first candidate is parallelism (i.e., increasing the amount of recombined fresh-key words in a single cycle), as exploiting extreme leakage value (i.e., leakage value with a small pre-image set size) becomes exponentially less likely with non-injective leakage function (such as Hamming weight). Therefore, the investigation performed in the next section focus on parallel LWPR

instances. We note that shuffling appears also as a natural candidate: if well implemented, shuffling makes the m intermediate computations of a serial implementation hard to distinguish so that the adversary does not gain more information than if she was provided with the sum of these leakages, which emulates a parallel implementation [HOM06; Vey+12; UBS21]. Therefore, the investigation performed in the next section focus on parallel LWPR instances.

5.3 Generalization of the leakage model

The previous sections motivate and demonstrate the interests of LWPR-based re-keying scheme as an alternative to direct masking of a cryptographic primitive. In particular, Section 5.2 demonstrates that letting the leakage function perform the rounding avoid the need of designing a dedicated protected circuit to perform the latter and may lead to significant performances improvement. It follows that the aforementioned reasons advanced give a strong incentive for understanding the security of the LWPR assumption under realistic leakage functions. More particularly, the investigations performed in [Duv+21] suggest as fundamental problem the analysis of leakage functions that (even slightly) deviate from the Hamming weight.

This section presents the results of a security analysis considering a practical context of linear leakage function (i.e., functions that can be expressed as a weighted sum of the bits manipulated by an implementation [SLP05]). More into the details, the analysis presented in [Hof+23b] shows that security against algebraic cryptanalysis is preserved in this context (and can even be extended towards higher-order functions) under reasonable conditions on the accuracy of the weights. While a mathematical security analysis is provided in [Hof+23b], we rather put the focus on the experimental part confirming the practical relevance of the assumptions about realistic hardware implementations made in the latter.

5.3.1 LWPR for linear leakage functions

A more realistic leakage model While reasonable as a first step, considering security against Hamming weight leakages is oversimplifying since the different bits of an implementation can consume more or less power, due to different load capacitances. This is precisely what is modeled by regression-based attacks originally introduced by Schindler et al. [SLP05]. Its main idea is to approximate the deterministic part of

the leakage function as a weighted sum of n_b well-chosen basis function that we denote as β_i:

$$\mathsf{M}_{\mathsf{r}}(\boldsymbol{v}) = \sum_{i=1}^{n_b} a_i \cdot \beta_i(\boldsymbol{v}),$$

with the a_i's $\in \mathbb{R}$. For a t-bit value $\boldsymbol{v}$, a typical choice is to consider $n_b = t$ and to use the bits of $\boldsymbol{v}$ as basis functions.[2] We will denote the resulting class of leakage functions as linear models $\mathsf{M}_{\mathsf{r1}}(\boldsymbol{v})$, which generalizes Hamming weight leakages where $a_i = 1 \ \forall \ 1 \leq i \leq n_b$. The model can be refined by adding more basis functions. Typically, we can add the $\binom{t}{\delta}$ basis functions of degree δ, which we denote as quadratic leakage models $\mathsf{M}_{\mathsf{r2}}(\boldsymbol{v})$ when $\delta = 2$, cubic leakage models $\mathsf{M}_{\mathsf{r3}}(\boldsymbol{v})$ when $\delta = 3, \ldots$, until the bijective model $\mathsf{M}_{\mathsf{r}t}$ where $\delta = t$ and $n_b = 2^t$.

Profiling a leakage model then essentially boils down to estimate the vector of coefficients $\boldsymbol{a}$, by applying the least square method. For this purpose, the evaluator first collects a vector of n_p profiling traces $\boldsymbol{l} = [l^1, l^2, \ldots, l^{n_p}]$, corresponding to values $\boldsymbol{v}^1, \boldsymbol{v}^2, \ldots, \boldsymbol{v}^{n_p}$. She then builds a $n_p \times n_b$ matrix:

$$B = \begin{pmatrix} \beta_1(\boldsymbol{v}^1) & \beta_2(\boldsymbol{v}^1) & \cdots & \beta_{n_b}(\boldsymbol{v}^1) \\ \beta_1(\boldsymbol{v}^2) & \beta_2(\boldsymbol{v}^2) & \cdots & \beta_{n_b}(\boldsymbol{v}^2) \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1(\boldsymbol{v}^{n_p}) & \beta_2(\boldsymbol{v}^{n_p}) & \cdots & \beta_{n_b}(\boldsymbol{v}^{n_p}) \end{pmatrix}.$$

Eventually, the vector of coefficients minimizing the means square error is:

$$\hat{\boldsymbol{a}} = [\hat{a_1}, \hat{a_2}, \ldots, \hat{a_{n_b}}] = (B^T \cdot B)^{-1} \cdot B^T \cdot \boldsymbol{l}.$$

We note that the analysis in [SLP05] considers noisy leakages and therefore adds a probabilistic component to the previous deterministic function. We do not detail this part since LWPR (conservatively) assumes noise-free leakages.

Formal security analysis setting Trying to formally evaluate the security with a more general leakage function, a first observation is that the product $\boldsymbol{K} \boxdot \boldsymbol{r}$ is linear over $\mathbb{F}_p$. As a result, the analysis performed in [Hof+23b] focus on the cryptographic criteria of the function L_{g} considered as a function over $\mathbb{F}_p$. If the adversary can approximate well the real function L_{g} by a p-ary function (a n variables function from $\mathbb{F}_p^n$ to $\mathbb{F}_p$ with p a prime number), she obtains a system of equations over $\mathbb{F}_p$ where

[2] Or $n_b = t + 1$, with a constant term to capture DC effects in the measurements.

the unknowns are (affine combinations of) the key elements. This gives a modeling of $\mathsf{L}_{\mathsf{g}}(\boldsymbol{K} \boxdot \boldsymbol{r})$ as equations over $\mathbb{F}_p$ where the only unknowns are the $\mathbb{F}_p$ elements of $\boldsymbol{K}$. The analysis from [Hof+23b] evaluates the complexity required to solve the aforementioned system for a linear leakage model in the serial case (and generalizes it for parallel case) by considering these models as functions over $\mathbb{F}_p$ denoted s-bounded pseudo-linear functions and defined in Definition 5.2.

Definition 6 (s-bounded pseudo-linear functions)**.** *We denote as $\mathsf{F}_{\boldsymbol{a}}$ the function characterized by $\boldsymbol{a} \in \mathbb{F}_p^t$, where $t = \lceil \log p \rceil$ and defined as:*

$$\mathsf{F}_{\boldsymbol{a}} : \ \mathbb{F}_p \to \mathbb{F}_p, \ y \mapsto \sum_{i=0}^{t-1} a_i \cdot \mathsf{g}(y)_i. \tag{5.2}$$

We call $\mathsf{F}_{\boldsymbol{a}}$ s-bounded pseudo-linear if each a_i belongs to $[0, s]$, with $s \in \mathbb{F}_p$ a security parameter, and we call C_1^s the class of s-bounded pseudo-linear functions.

We recall that the g function associates elements of $\mathbb{F}_p$ to their binary representation. The name pseudo-linear comes from the fact that this function is linear in the bits of the binary representation of $\boldsymbol{a}$. However, it corresponds to a much higher degree p-ary function. This increase of the degree when projecting a binary representation into a prime field is the root of the LWPR hardness. In [Hof+23b], it is considered that an s-bounded $\mathsf{F}_{\boldsymbol{a}}$ is the p-ary function used to approximate L_{g}. This definition implicitly relies on two assumptions: one on the degree of the physical leakage function and another one on the number of values taken by the coefficients a_i. The practical relevance of these two assumptions is discussed in the following sections.

5.3.2 Experimental Framework

Prototype hardware implementation We based our experiments on a parallel hardware implementation operating on 31-bit words and running on a FPGA. The architecture of the computational core of our masked LWPR-based re-keying scheme is depicted in Figure 5.11. Concretely, the complete computation is performed by processing each share sequentially and the matrix multiplication for the share index d is split in $(N+1) \times M = (4+1) \times 4 = 20$ modular multiplications over $\mathbb{F}_p$ (i.e., the same instance parameters as in Section 5.2). More precisely, the core implements 4 modular multiplications in parallel (i.e., the results of the multiplications between the j-th column of a key share denoted $K^d_{*,0 \leq j < 5}$ and the j-th element of the nonce denoted r_j, where $r_4 = 1$).

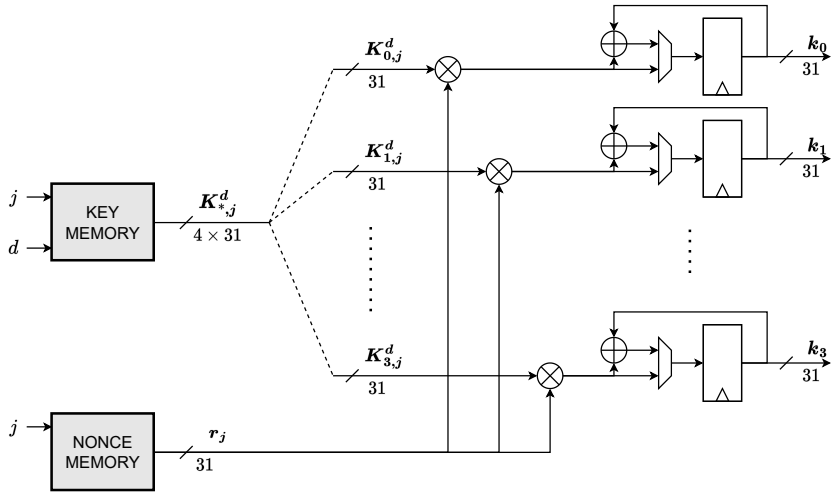


Figure 5.11: Masked LWPR-based re-keying: prototype hardware implementation.

We used the efficient modular multiplication architecture based on DSPs proposed in [Kop+17] for this purpose. The results obtained for the multiplications are then added in 4 accumulators (i.e., one for each row) dedicated to the reconstruction of the resulting fresh key.

As shown in Figure 5.12, the architecture of the key memory is designed to limit the impact of physical defaults on the implementation's security. In particular, the key shares are stored in independent physical memory units (i.e., BRAMs in the context of our FPGA implementation) and a register barrier is used before selecting the appropriate memory output. The latter is needed to avoid shares' recombinations that may be caused by glitches. Finally, a dedicated multiplexer at the input of the register barrier is used to drive the shares' values that are not currently processed to zero. Together with the fact that the shares are processed sequentially, this mechanism ensures the independence of the key shares.

Overall, the prototype implementation matches the parallelism requirements described in Section 5.2.4 since it recombines a full (124-bit) fresh key in a single clock cycle, the leakage of which we analyze in the rest of this section. Besides, we note that when analyzing the first (green) attack path described in the threat model of Figure 5.2, we are only interested in the value of the recombined fresh key.[3] Concretely,

[3]The first attack path (leveraging the leakages of the shared multiplications) was analyzed in Section 5.2.3 and the arguments of this previous work apply similarly.

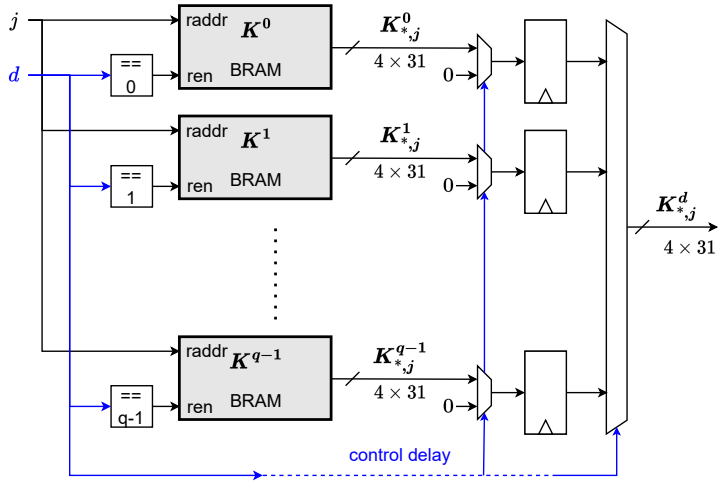


Figure 5.12: Key material memory architecture.

we therefore choose to measure an implementation with a single share (i.e., $q = 1$). This gives the most favorable leakages to the evaluator and therefore puts us as close as possible to the noise-free setting considered in our analyses. This is because the implementation with $q = 1$ is the smallest one and a limited use of resources in turn reduces the noise in the leakages.

The measurements were performed on the Sakura-G evaluation board, which embeds a XC6SLX75-2CSG484C Spartan6 FPGA. The traces were collected using a Picoscope 5244D with a Tektronix CT-1 current probe, following the guidelines given in [BUS21]. In order to take into account the effect of congestion that can take place during the place-and-route step of a dense design, we performed three different syntheses using the ISE14.7 design toolchain. The first one is unconstrained. The second one is constrained in order to achieve 95% of resources' utilization density (using a PBLOCK area constraint). The last one further constraints the design by (artificially) enforcing that higher-capacity routing elements are included in the path of some bits. We next refer to these different syntheses as unconstrained, constrained and amplified. The layout of the unconstrained and constrained syntheses are illustrated in Figure 5.13.

Correlation based security Given a leakage model, the main question of a side-channel security evaluation is to determine the number of attack traces needed to recover a key [SMY09]. In the context of univariate

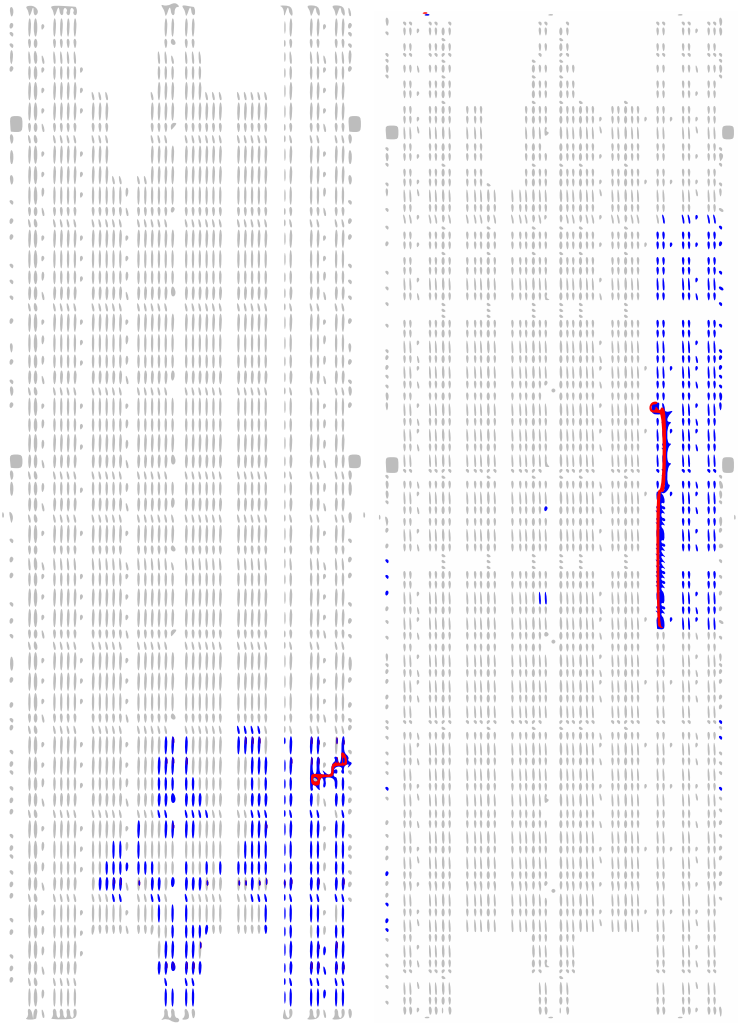


Figure 5.13: Layout of unconstrained (left) and constrained (right) syntheses. Unused resources are in white. Used resources are in blue. Routing is in red.

leakage functions considered in the experiments, Pearson's correlation is among the most popular evaluation tools [BCO04]. Next are described the basic approximations used in the empirical analyses.

Let us first denote a vector of n_a attack values $\overline{\boldsymbol{v}} = [\boldsymbol{v}^1, \boldsymbol{v}^2, \ldots, \boldsymbol{v}^{n_a}]$. From those values, an evaluator can compute the modeled leakage vector:

$$\boldsymbol{m} = [\mathsf{M}_\mathsf{r}(\boldsymbol{v}^1), \mathsf{M}_\mathsf{r}(\boldsymbol{v}^2), \ldots, \mathsf{M}_\mathsf{r}(\boldsymbol{v}^{n_a})].$$

Similarly, she can measure a vector of actual leakage traces $\boldsymbol{l} = [l^1, l^2, \ldots, l^{n_a}]$. Viewing these vectors as n_a samples of random variables M and L, the data complexity of a Correlation Power Analysis (CPA) is given by [Man04]:

$$N = \frac{c}{\hat{\rho}(M, L)^2},$$

where c is a small constant and $\hat{\rho}$ denotes Pearson's correlation coefficient.

In the following experimental evaluations, we are interested in the comparison between different leakage models, with the goal to evaluate whether simplifying a model (e.g., by limiting its degree or quantizing it) results in a significant information loss. Pearson's correlation is convenient for this purpose. For example, let us assume a model M_1 that is a simplification of a model M_2, leading to model errors captured by a random variable E so that $M_1 = M_2 + E$. Then, thanks to the correlation chain rule given in [Sta+06] we have:

$$\hat{\rho}(M_1, L) = \hat{\rho}(M_1, M_2) \cdot \hat{\rho}(M_2, L).$$

As a result, the increase of data complexity that is due to simplifying the model M_2 into M_1 is reflected by the factor $f = \frac{1}{\hat{\rho}(M_1, M_2)^2}$: if $f \approx 1$ the attack using the simplified model is close to the one using the complex model; if $f > 1$ the simpler model misses some leakage features and the attack using this model requires f times more traces than the attack using the more complex model.

5.3.3 Bounded degree of the leakage function

The first physical assumption used in the analysis performed in [Hof+23b] to rule out algebraic attack is that the leakage function has a bounded degree. We next study whether this assumption can be reasonably fulfilled in practice. As detailed in Section 5.3.1, regression-based models are a natural tool for this purpose, since they can be estimated for different, more or less complex, basis functions. A simple case, considered in the previous sections, is to take the bits of the target intermediate value as basis functions. But higher-degree models can be built, culminating with the exhaustive model which corresponds to a (worst-case) profiling where all the mean leakage values are exhaustively estimated [CRR02]. Our goal is to therefore show that a simple linear model does not lead to significant information losses compared to the exhaustive one.

For this purpose, we analyzed the 4 leakage functions f_i of our target implementation. We analyzed them independently to reduce the noise.

They all gave similar results. For the exhaustive model, and since building 2^{31} templates is computationally intensive, we rather estimated a subset of n_a average leakage traces (with $n_a = 1,000$), and stored them in a vector of average leakage traces $\bar{\boldsymbol{l}} = [\overline{l^1}, \overline{l^2}, \ldots, \overline{l^{n_a}}]$. We then computed the vectors corresponding to our regression-based linear models for the three different syntheses, evaluated on the same values. In each case, we first estimated the 31-element vector of coefficients $\hat{\boldsymbol{a}}$ using $q_t^m = 10,000$ profiling traces (meaning $\approx \frac{10,000}{31}$ traces per coefficient, which was sufficient for good estimation given the low noise level of our measurements). As a result, we obtained model predictions:

$$\boldsymbol{m}_{\mathsf{r1}}^u = [\mathsf{M}_{\mathsf{r1}}^u(\boldsymbol{v}^1), \mathsf{M}_{\mathsf{r1}}^u(\boldsymbol{v}^2), \ldots, \mathsf{M}_{\mathsf{r1}}^u(\boldsymbol{v}^{n_a})],$$

for the unconstrained synthesis, and similarly $\boldsymbol{M}_{\mathsf{r1}}^c$ & $\boldsymbol{M}_{\mathsf{r1}}^a$ for the constrained and amplified syntheses. Viewing these vectors as the samples of random variables $\overline{L}$, M_{r1}^u, M_{r1}^c and M_{r1}^a, we finally computed the correlation between the true (averaged) leakages and the models: $\hat{\rho}(M_{\mathsf{r1}}^u, \overline{L})$, $\hat{\rho}(M_{\mathsf{r1}}^c, \overline{L})$ and $\hat{\rho}(M_{\mathsf{r1}}^a, \overline{L})$.

These correlation coefficients are reported in Figure 5.14 in function of the number of traces used to obtain the average traces $\overline{l^i}$. We additionally report the estimates obtained when correlating the real leakage vectors with a Hamming weight leakage model, and each curve in the figure is accompanied by a 95% bootstrap confidence interval. These results confirm that unconstrained syntheses (at the top of the figure) lead to high correlations already with the Hamming weight leakage model. By contrast, for constrained and amplified syntheses reflecting congestioned designs (at the middle and bottom of the figure) the correlation with the Hamming weight leakage model decreases while it remains close to one for the linear models. These experiments show that the generalization we propose indeed captures practically-relevant implementations. Admittedly, reaching such high correlation values does not preclude that higher-degree terms can be characterized and exploited. But it shows that most of the information leaked by our target implementation can be extracted with a simple linear model.

5.3.4 Practical application of the attack

The attacks studied in [Hof+23b] work by interpreting physical leakages as functions in $\mathbb{F}_p$. Next is described how to mount these attacks, which will incidentally lead us to provide good indications that limiting the coefficient's accuracy is also reasonable. Precisely, we show how to move from measured leakages to s-bounded leakages thanks to the linear regression-based model of Section 5.3.1. For this purpose, we re-use

the 31-element vectors of coefficients $\hat{\boldsymbol{a}}$ estimated using $q_t^m = 10,000$ profiling traces, for each of our three syntheses. We next observe that the value of Pearson's correlation is not affected if one of its variables is multiplied by a constant value. As a result, the following simple discretization process for the coefficients $\hat{\boldsymbol{a}}$ can be used:

$$\hat{\boldsymbol{a}}_{\mathsf{d}} = \left\lceil \hat{\boldsymbol{a}} \cdot \frac{s}{\mathsf{max}(\hat{\boldsymbol{a}})} \right\rceil .$$

We finally illustrate the coefficients of the linear models estimated from the unconstrained synthesis measurements together with the coefficients of the corresponding s-bounded functions in Figure 5.15. We can observe that already for $s = 2^8$ the matching between both is quite accurate. We further estimated the correlation $\hat{\rho}(M_{\mathsf{r1}}^s, M_{\mathsf{r1}})$ which was

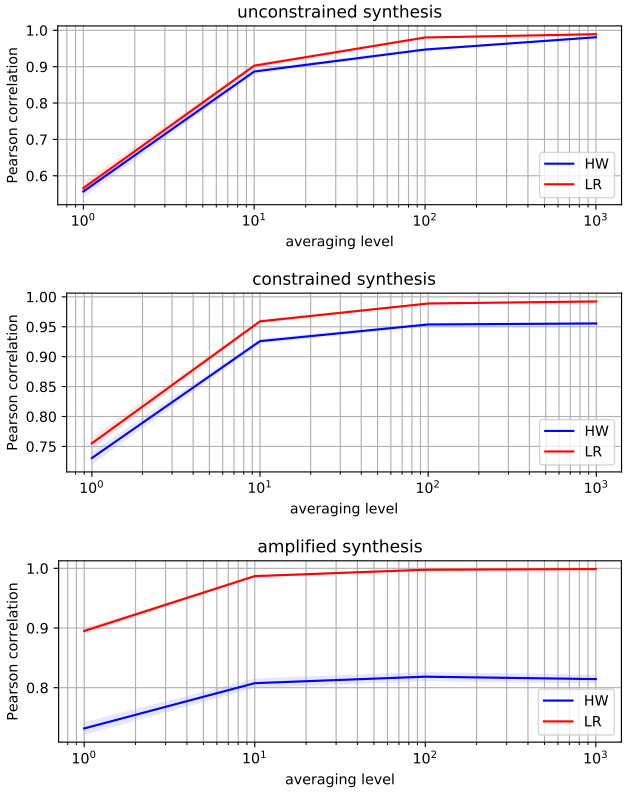


Figure 5.14: Correlation between real traces with different levels of averaging (on the X axis) and Hamming weight vs. regression-based linear models.

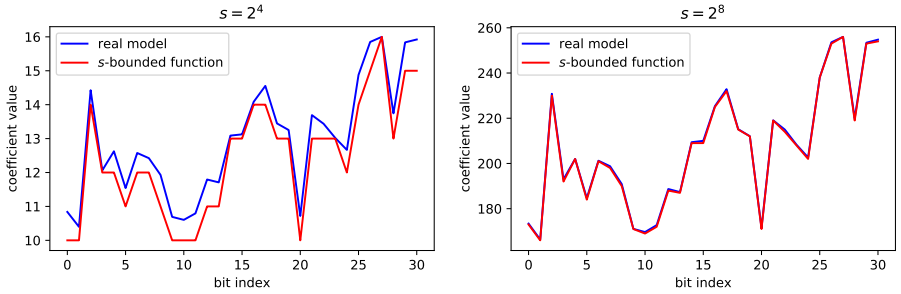


Figure 5.15: Discretization of the linear regression models (unconstrained synth.).

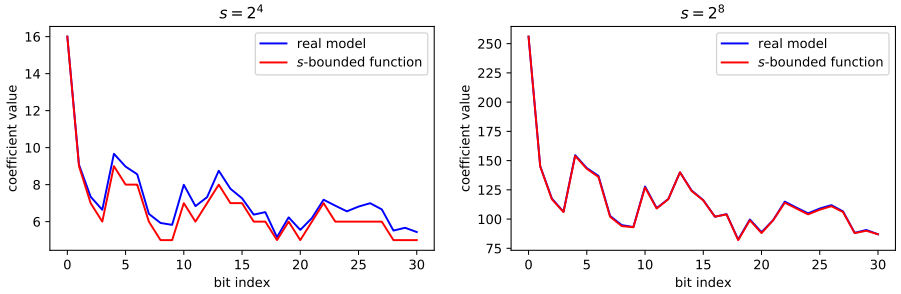


Figure 5.16: Discretization of the linear regression models (constrained synthesis).

worth $1-\phi$ with $\phi < 10^{-6}$ for $s = 2^8$.[4] We conclude that a quantization corresponding to $s = 2^{12}$ offers a sufficient granularity to discretize our linear leakage models nearly perfectly. In other words, while we cannot rule out that other strategies than interpreting the leakages in $\mathbb{F}_p$ as considered in the context of the analysis can lead to better results, this section shows that if this is the best strategy, then the move from measured leakages to s-bounded ones does not imply a significant information loss for the values of s (e.g., 2^{12}) that our theoretical investigations tolerate. The results for the quantization of the models for moderately constrained and amplified syntheses are similar. We illustrate their respective coefficients in Figures 5.16 and 5.17 for completeness.

5.3.5 Bounded measurement accuracy

The previous sections suggested that using linear regression-based models may accurately capture concretely-relevant linear leakage functions

[4] With a relative error of below 1%, which is easy to reach since the estimation is performed from modeled samples (rather than measured ones in Section 5.3.3).

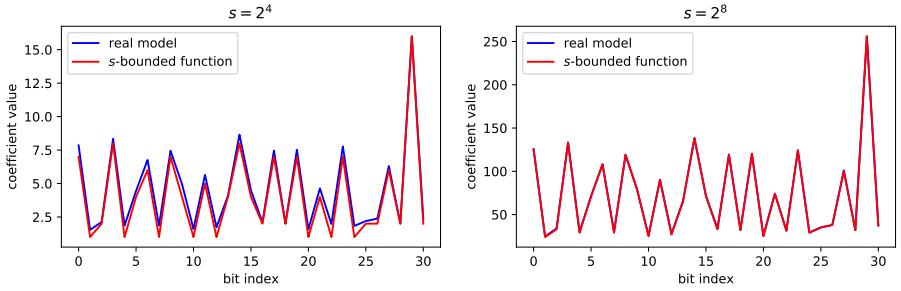


Figure 5.17: Discretization of the linear regression models (amplified synthesis).

and that increasing the coefficient accuracy of the s-bounded pseudo-linear functions beyond 2^{12} does not lead to significant improvements of their informativeness in our case study. Quite naturally, such assessments remain limited by their heuristic nature. For the analysis of the degree, this is unavoidable (since absence of evidence is not evidence of absence) and the only option to make it more robust is to consider higher-degree functions that could theoretically show up, which is done in [Hof+23b]. But for the bounded coefficient accuracy, there is a complementary argument that could be given. For this purpose, the starting observation is that physical leakages are measured by converting an analog signal into a digital one. Concretely, the "vertical resolution" s' of modern oscilloscopes typically ranges from 8 to 12 bits, meaning that the measured leakage can take between 2^8 and 2^{12} values (possibly a bit more thanks to oversampling, at the cost of a reduced "horizontal" resolution). So if the increase of the s parameter from Definition 6 leads to a pseudo-linear function having a codomain with more than s' elements, it implies that the s-bounded leakage function is more informative than the measured one, which contradicts the data processing inequality. So for such leakage functions, the bounded coefficient accuracy can be directly connected to an assumption on the adversary's measurement apparatus.

However, increasing the s parameter does not always increase the cardinality of the leakage functions (e.g., think about the case where $t = 1$ for an increasing s). So one cannot directly bound the coefficient accuracy s based on the resolution of the oscilloscope s'. Yet, we note that the security arguments used in the mathematical analysis depend on an upper bound (implying the value s) on leakage function co-domain cardinality. So if the security analysis could be improved in order to rely only on this cardinality of the leakage functions, independent of the value of s, the assumption on the bounded coefficient accuracy can be

replaced by an assumption on the resolution of the oscilloscope used to mount the attack, which is an interesting open problem.

5.4 Concluding remarks and open problems

The crypto-physical dark matter is presented as a provocative solution to improve the security vs performances trade-off of state-of-the art re-keying schemes. Its main idea is to combine simple computations in a medium size prime field with a physical leakage function that is assumed to operate in a sufficiently different field. It turns out that this combination ensures certain interesting cryptographic properties under the Hamming weight leakage assumption commonly encountered in side-channel literature.

This chapter primarily emphasizes the competitive performance achieved through hardware implementation of the crypto-physical dark matter scheme, both in terms of randomness or key size (directly translating to memory) and latency. These improvements are directly linked to the fact that the proposed solution leverages the leakage function (practically realized by the device's inherent physical behavior) and, therefore, does not require the use of a dedicated circuit to implement the costly nonlinear function it represents. Admittedly, this is only made possible by considering a threat model that is weaker than the one against which the performances are compared (i.e., the adversary has access to the observations of leakage values), which is a notable limitation. However, the latter remains reasonable in the context of a side-channel attack, emphasizing the relevance of research that aims to analyze the hard-physical learning problems.

In this regard, this chapter also supports the validity of the security analysis conducted for the LWPR problem generalized to the s-bounded leakage defined in Section 5.3.1. In particular, it experimentally demonstrates that linear models of degree 1 are already capable of extracting a significant amount of information, and that increasing the maximum value of the weights beyond a certain threshold provides limited additional information. As discussed in Section 5.3.5, these arguments are limited by their heuristic nature, with the primary limitation being the bound on the s value. To address the issue, the analyses could be improved in order to rely only on the degree of the leakage functions and the cardinality of their co-domain. As outlined in Section 5.3.5, this could potentially establish a formal connection between the bounded coefficient accuracy hypothesis and the capabilities of the adversary's measurement equipment.

As described in Section 5.2.4, the security of the proposed solution relies on the existence of a certain degree of parallelism. If the latter is prohibitive, an alternative would be to consider more serial implementations, for example, by leveraging shuffling. However, pursuing this approach would require improving the conducted analyses to take into account multivariate leakage models.

Eventually, from an application point of view, studying the integration of the proposed re-keying could be interesting. A notable first step in this direction is the use of a tweaked version of the LWPR problem (hardness source of the crypto-physical dark matter scheme) in the public key encryption scheme POLKA [Hof+23a] specifically designed in order to enable leveled implementation. The main observation made in that work is that some DPA sensitive operations (namely a product between an ephemeral secret value and a long term secret key) share similarities with the crypto-physical dark matter re-keying scheme presented in this chapter, and can the leverage key-homomorphism properties when masked. Besides, it notes that one of the input of the multiplication is dummied, which has the effect of increasing the practical security order by one (and thus requires one less share for a targeted security order security order) as illustraded in **??**.

6 DPA security without built-in protection

As pointed out by the NIST' lightweight cryptography call for algorithm, the deployment of small computing devices such as RFID tags, sensor nodes, industrial controllers or smart cards is becoming much more common. This trend brings new challenges for security applications, especially when the latter also have to resist side-channel attacks, as the cost of common countermeasures such as masking (discussed in Chapter 4 and Chapter 5) may be prohibitive in a ressource constrained environment. In this context, the question arises of whether it is possible to achieve side-channel security with implementations that do not use built-in protection.

For this purpose, re-keying based solutions may be relevant since they have stronger inherent side-channel security properties, as already discussed in Chapter 3 and Chapter 2.4.2. Indeed, these structures limit the number of key manipulations performed, making efficient leakages exploitation more challenging. From a practical point of view, this translates to the use of implementations protected against SPA only, which can be implemented at an amenable cost by leveraging parallel hardware implementation.

Following this lead, cryptographic coprocessors available in low-end devices appear as relevant targets. Indeed, these hardware accelerators are often implemented in an unprotected manner (i.e., without dedicated countermeasure). Besides, these usually come with an architecture presenting a certain level of parallelism for efficiency purposes. In their work [Unt+20], Unterstein et al. investigate the possibilty to use them as key ingredient in order to build side-channel secure block. In particular, they tackle the issue of secure firmware updates, which are crucial to mitigate software vulnerabilities in application relying on off-the-shelf microcontrollers. To this end, they propose a Leakage-Resilient Authenticated Encryption (LR-AE) mode of operation (which is adapted from the proposals of [DJS19; KS20]) built upon leakage-resilient primitives that leverage algorithmic noise and re-keying to this end, as detailed hereunder.

The first ingredient they rely on is the Leakage-Resilient Pseudo-Random Function (LR-PRF) of Medwed et at. [MSJ12], which is used as a (fresh) key and tag generation function. It follows a tree like architecture specifically designed to reduce the number of different plaintexts that can be encrypted with single a key, which has the effect of reducing DPA security to SPA security. The second is the Leakage-Resilient Pseudo-Random-Generator (LR-PRG) from [SPY13], used to generate a key-stream that is then used in a one-time pad fashion. Both the aformentioned primitives can be implemented based on the AES block cipher which is typically available on common coprocessors.

However, although it seems to be a step in the right direction for achieving security in constrained applications (as well as meeting an industry demand, as reflected by the NXP AN12304 application note[1]), the design of solutions based solely on coprocessors (or more generally, on hardware implemented without built-in protection) faces limitations requiring specific care when integrated into more complex structures such as mode of operations. These limitations arise from the fact that the range of conceivable designs is narrowed, as it forces implementation to rely on the available primitive' circuitry only, which comes with its inherent physical security properties and leaves little room for adaptation. Besides, standard countermeasures like masking mostly depend on a sufficiently high level of noise, which may be uneasy to obtain on low-end devices but easy to evaluate. In contrast, LR-PRFs enable attacks with averaged observations and rather require a sufficiently low side-channel signal in the measurements. Evaluating the latter is a quite sensitive process that can significantly be affected by slight changes in the measurement setup, which can significantly modify the perceived level of security of such primitives.

Taking these considerations into account, this chapter focuses on the implementation aspects of designs based on the LR-PRF from Medwed et al. [MSJ12]. In particular, keeping the same motivation of providing efficient and secure firmware updates, some limitations of the proposal from [Unt+20] are first highlighted. Namely, we first observe that the design does not allow to achieve the main relevant security property (ciphertext integrity with leakage in decryption in the context of firmware updates) and demonstrate that a standard DPA against the tag verification can lead to forgeries. Conceptually, this attack becomes possible due to a weak instantiation of the tag verification, which requires a careful instantiation when no built-in protection is considered.

[1] `https://www.nxp.com/docs/en/application-note/AN12304.pdf`, see also [Med+16].

Second, we perfom an experimental analysis of the LR-PRF heuristic requirements [MSJ12], relying on two hardware coprocessors architectures (with respectively 32-bit and 128-bit). Namely, we illustrate that, although attacking the output of AES may seem challenging, an attack can be mounted against the different S-boxes. Complementarily, we discuss the impact on the performances when trying to mitigate such an attack. More conceptually, these investigations highlight the fact that evaluating LR-PRFs is a delicate task that hardly transfers from one device (or measurement setup) to another, hence suggesting the use of conservative security parameters when instantiating it. We eventually provide implementation results for a new leveled mode of operation that mitigates the aforementioned integrity issue observed in the solution of [Unt+20].

In this chapter, I was involved in the experimental analyses. I mostly worked on the experimental setup even though I also collaborated on the side-channel analyses and attacks against the LR-PRF. Additionally, I contributed to the implementation and performance evaluations of the new mode of operation proposed as an alternative to the solution in [Unt+20] for secure updates.

6.1 Background

In this section, we first recall the specifications of the LR primitive used in the Unterstein et al.' proposal. These being the targets used in the side-channel analyses and attacks presented in this chapter.

6.1.1 Medwed et al.'s leakage-resilient PRF

The LR-PRF introduced in [MSJ12] aims at reducing the number of different plaintexts that can be encrypted with single a key, in effect leading to reducing DPA security to SPA security. As illustrated in Figure 6.1, it processes an input x with a key k using a tree-based construction: at each stage, a plaintext depending on n_b bits of x is encrypted with a key taken as the output (ciphertext) of the previous stage. The last stage of the LR-PRF is a whitening that encrypts a public plaintext. Therefore, the total number of calls to the block cipher (or stages) to run the LR-PRF is worth $1 + n/n_b$ where n is the size of the LR-PRF input. In order to mount a DPA against k, the adversary has access to 2^{n_b} different plaintexts. Since she has control of x, she can observe multiple leakage for the same input and so obtain averaged leakages with minimum noise.

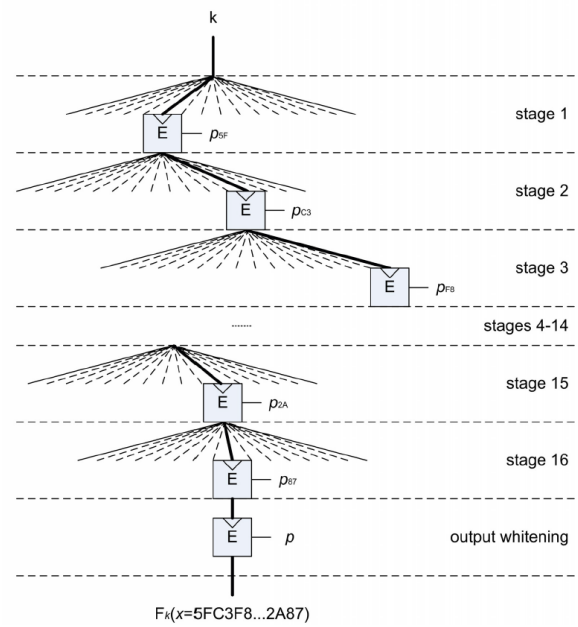


Figure 6.1: LR-PRF construction from [MSJ12] with $n_b = 8$ (illustration from [MSJ12]).

In [MSJ12], the authors format the plaintext based on the n_b bits of x such that all the bytes have the same value (e.g., set to these n_b bits). As a result, for a byte-oriented cipher like the AES, the bytes of the plaintext that enter the key addition plus the S-box in the first cipher round will be equal. This makes the guesses performed by the DPA adversary equal for all the bytes. In a parallel implementation and assuming that the leakage model of each S-box is identical, an adversary will not be able to distinguish the leakage from the S-boxes. In case this attack vector is the best one (i.e., if attacking after the MixColumns operation is hard), the adversary will be forced to enumerate the permutation on all the key bytes. We will further elaborate on these hypotheses in subsection 6.3.1.

6.1.2 Specification of the [Unt+20] proposal

The AEAD mode proposed in [Unt+20] is depicted in Figure 6.2. It is a two passes mode operation built around three main building blocks, namely the LR-PRF described in Section 6.1.1, the Leakage-Resilient Pseudo-Random-Generator (LR-PRG) from [SPY13] and the SHA-256 hash function. The LR-PRG is depicted in Figure 6.3 and leverages

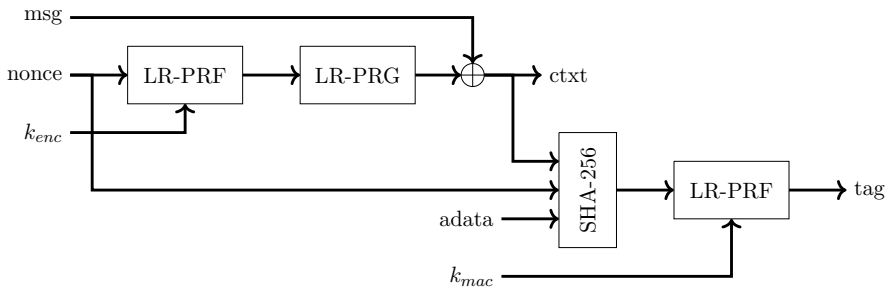


Figure 6.2: Architecture of the mode proposal from [Unt+20]

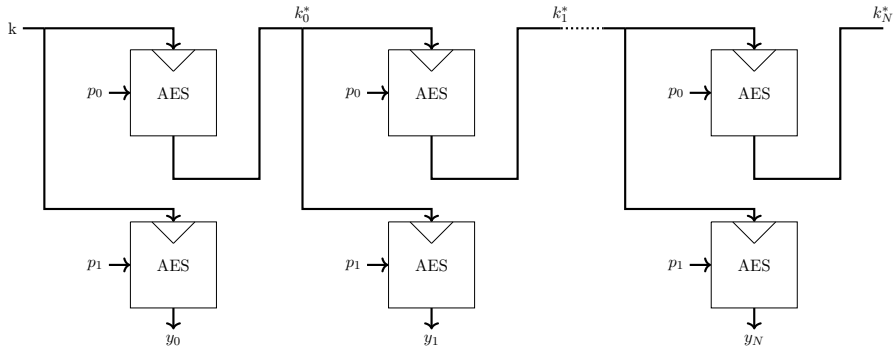


Figure 6.3: Architecture of the LR-PRG primitive from [SPY13]

on AES block cipher instances to generate the pseudo-random string $R = y_0||y_1||\ldots||y_N$ (where $||$ depicts the concatenation operator).

The encryption of a message (depicted msg in Figure 6.2) starts with the computation of a seed by applying the LR-PRF to the nonce using a first key k_{enc}. This seed is then used to generate a pseudo-random stream that is XORed with the message (in a one-time-pad fashion) to obtain the ciphertext (denoted ctxt). Together with the nonce and the associated data, the ciphertext is the hashed and the tag is computed by applying the LR-PRF on the resulting digest using a second key k_{mac}.

6.2 Attack against the integrity of the [Unt+20] LR-AE

6.2.1 Targets and measurement setup

We perform experiments on two commercial MCUs with AES coprocessors. The first one is an ARM Cortex-M4 STM32F439ZI mounted on a

(unmodified) demonstration board Nucleo-144.[2] The coprocessor has a 128-bit architecture meaning that all the S-boxes are processed in parallel. The device is running at its maximum clock frequency of 180[MHz]. The second target is an ARM Cortex-M33 STM32L562QEI6QU with ARM TrustZone and a dual core. The MCU is also mounted on a (unmodified) demonstration board STM32L562E-DK.[3] The coprocessor has a 32-bit architecture meaning that a single AES column is processed in parallel. This target is also running at its maximum clock frequency of 110[MHz]. On both targets, the supply voltage is lowered to 1.8[V] which is the minimal specified value, in order to improve the SNR [BUS21]. The two investigated targets have similar but not identical coprocessors as the ones investigated in [Unt+20], which are respectively based on 32-bit and 128-bit architectures, and are running on ARM Cortex-M4 and ARM Cortex-M3 devices. We evaluate the coprocessor of a more recent ARM Cortex-M33 in place of the ARM Cortex-M3.

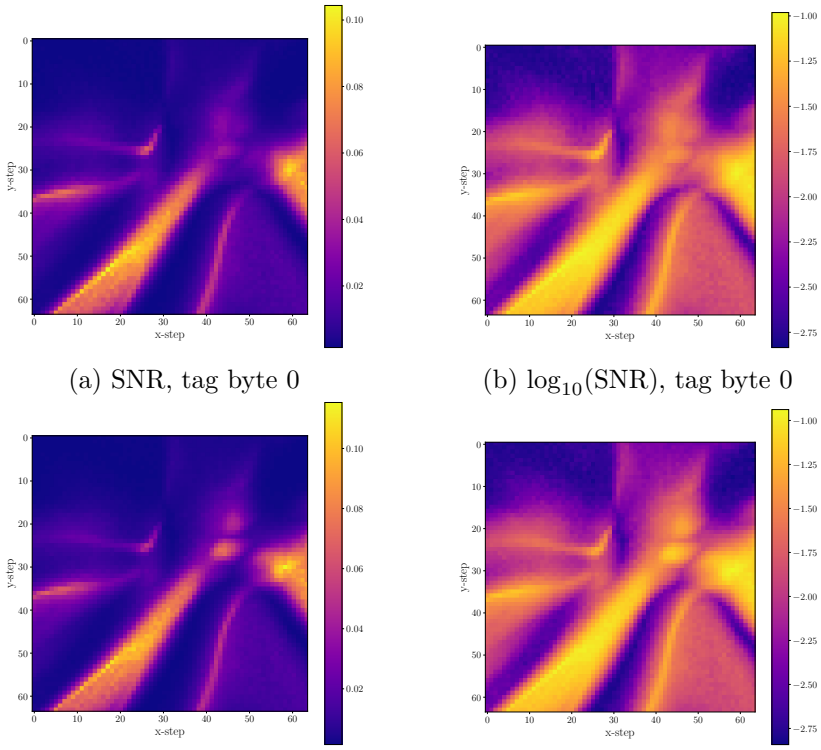
Similarly to [Unt+20], the side-channel leakage is measured with an electromagnetic (EM) probe. The EM signal is sampled with a PicoScope 6424E at 5[GSamples/sec] (with a 500[MHz] analog bandwidth), using a 10-bit resolution. This signal is amplified thanks to a PA 306 from Langer that offers 30[dB] of gain on the bandwidth 100[kHz] to 6[GHz]. For the ARM Cortex-M33, the probe RF-B 0.3-3 from Langer is used while for the ARM-Cortex-M4 the probe RS H2.5-2 from Rohde&Schwarz is selected. The quality of the equipment is similar to the one used in [Unt+20], excepted that we did automatically position the probe instead of manually. For both targets, the probe was positioned thanks to a XYZ table used to scan the top of the (untouched) package. The SNR is estimated for all the S-boxes with 200,000 traces at each point of a grid. For the Cortex-M33, the grid is of size 16x16. For the Cortex-M4, we use a 64x64 grid.[4] The heatmaps obtained after the grid scan of Cortex-M4 are reported in Figure 6.4. For both targets, the heatmap for each S-box suggests placing the probe around the same area. Therefore, the probe is moved to the position maximizing the averaged SNR across the 16 S-boxes.

Overall, we were able to capture the traces at a rate larger than 30,000[measurements/sec] for the two investigated targets. Similarly to [SM16], the AES coprocessor is used as a PRG to generate the random plaintexts and keys, which saves most of the cost of communication with

[2] `https://www.st.com/en/evaluation-tools/nucleo-f439zi.html`

[3] `https://www.st.com/en/evaluation-tools/stm32l562e-dk.html`

[4] The Cortex-M33 package is smaller than the Cortex-M4, which explains the different resolutions.



(a) SNR, tag byte 0

(b) $\log_{10}$(SNR), tag byte 0

(c) SNR, tag byte 1

(d) $\log_{10}$(SNR), tag byte 1

Figure 6.4: Heatmaps on Cortex-M4 (128-bit arch) for various tag bytes.

a control desktop. Namely, it encrypts in CBC mode a zero string with a known random IV. The SNRs are then computed on-the-fly thanks to a one-pass algorithm similar to [SM16], which totally saves the cost of disk writing.

6.2.2 Tag recovery attack

A first observation is that the proposal of [Unt+20] does not achieve the ciphertext integrity with leakage in decryption property (provided by Grade 2 algorithm, as described in Chapter 3), which can lead to ciphertext forgeries. As discussed in [Ber+17], this property can be reached with two calls to a strongly protected block cipher and letting most of the other parts of the implementation leak in an unbounded manner. Yet, it requires a careful instantiation of the tag verification. Namely, either this tag verification is protected at the implementation level (e.g., via masking), which contradicts the goal of relying only on hardware coprocessors, or it has to be designed such that leaking the

```
1 uint32_t tag_verif(uint32_t *S,uint32_t *T){
2     uint32_t flag;
3     flag  = S[0] ^ T[0];
4     flag |= S[1] ^ T[1];
5     flag |= S[2] ^ T[2];
6     flag |= S[3] ^ T[3];
7     return (flag == 0);
8 }
```

Listing 6.1: Tag verification C code.

verified value cannot lead to forgeries. It turns out that the proposal of Unterstein et al. does not include such a careful design. More particularly in [Bel+20a], the authors present a practical side-channel attack against a tag verification implemented on an ARM Cortex-M0. It can directly be used to forge valid messages under adversarial control [Ber+17], and therefore to break the integrity of a software update.

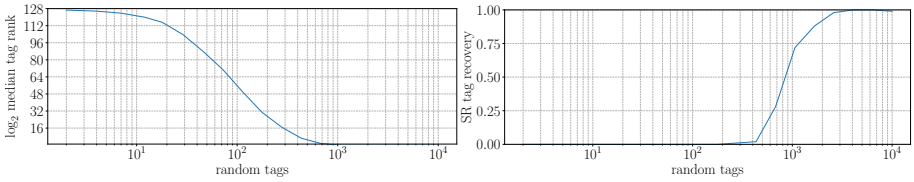
In this section, we repeat that experiment with two differences: *(i)* we use an ARM Cortex-M4 which is more noisy[5], and *(ii)* we do template attacks in a linear subspace instead of directly in the leakage domain (as detailed in Section 2.3). The target code is described in Listing 6.1, where a 128-bit tag candidate S is compared with the correct tag T by using 4 XOR instructions. After using 3 OR and 1 comparison, the returned value is 1 if all the bits of S and T are equal. To be as transparent as possible, the resulting assembly code (obtained with `objdump`) is provided in [Bro+21].

More precisely, considering the scheme from [Unt+20], Algorithm 3, an adversary can try to forge a valid tag T for a ciphertext of her choice and a nonce. By having access to a decryption oracle, she can attempt decryption with multiple random tags S. The oracle will first compute the valid tag T for the provided ciphertext (lines 12 and 13) and then compare it with S (line 14) allowing one to mount a DPA on the XOR instructions in Listing 6.1.[6] In Figure 6.5a, we report the median rank of the 128-bit tag estimated with [PSG16], in function of the number of calls to the decryption oracle. In Figure 6.5b, we similarly report the success probability if no enumeration is possible. After $3,000$ calls, the adversary can recover the valid tag in full without enumeration with a

[5] The maximum SNR on the ARM Cortex-M4 we investigate is approximately 0.15 while it is approximately 0.8 on a similar ARM Cortex-M0 as investigated in [Bel+20a].

[6] We note that by using template attacks, the adversary implicitly exploits all the leakage points that are bijectively related to a byte at the output of the target XOR operation.

probability almost equal to one. After 200 calls, she is already able to reduce the rank of the valid tag below 2^{32}. This would allow her to still succeed in the attack by enumerating these 2^{32} tags.



(a) Median tag ranks (b) Tag recovery success rate

Figure 6.5: Median tag rank & success rate for various number of traces on ARM Cortex-M4. Metrics estimated on 100 independent random keys. $2 \cdot 10^6$ profiling traces used.

In the context of firmware updates investigated in [Unt+20], this tag recovery could be exploited as follows. First, an adversary can anyway select a (garbage) ciphertext C and one nonce of her choice N. By targeting the verification, she is able to obtain the valid tag T and to decrypt the (garbage) plaintext M corresponding to the ciphertext C. This already results in a valid authenticated (garbage) firmware update which can lead to practical issues (e.g., denial-of-service) if no additional protection mechanisms are implemented. Second, if we additionally assume that the adversary can observe the garbage plaintext, she can then also recover the random string R output by the LR-PRG that depends only on N and a secret key, by adding C to M. In this case, she can choose a (malicious) plaintext of her choice M' and compute the corresponding ciphertext $C' = R \oplus M'$, which is valid for the nonce N. Afterwards, she only needs to repeat the attack against the tag verification to recover the corresponding valid tag T' This attack therefore allows the adversary to forge an authenticated firmware update (C', T', N) that she controls without recovering the long-term key. We note that it admittedly assumes that the adversary can observe the garbage plaintext, which may require some additional (possibly side-channel based) reverse engineering. On the one hand, this attack shows that if one single correct plaintext/ciphertext pair is ever recovered, then forging any malicious software update of at most this size becomes possible. On the other hand, the complete investigation presented in [Bro+21] demonstrates that for a well instantiated mode, integrity can hold even with an unbounded leakage of ephemeral secrets (e.g., plaintexts).

6.3 Evaluation of the [MSJ12; Unt+20] LR-PRF

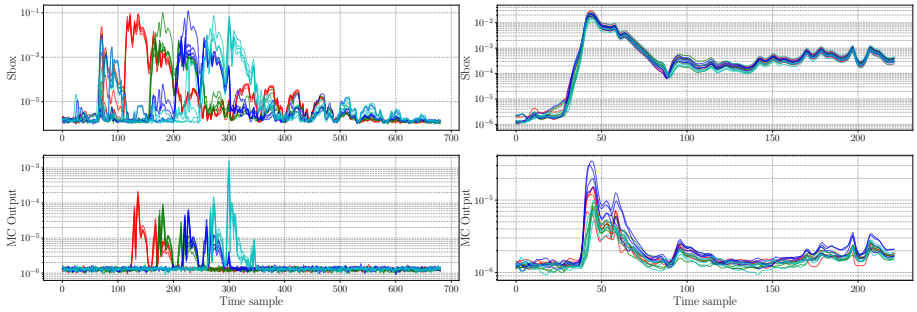
6.3.1 LR-PRF: qualitative leakage analysis

The heuristic security provided by Medwed et al.'s LR-PRF is based on two main working principles. On the one hand, it bounds the amount of average leakage traces that an adversary can exploit to perform a DPA. Namely, she can only access 2^{n_b} traces to attack each stage of the LR-PRF. On the other hand, the plaintexts that are encrypted are not random since they are structured as vectors with 16 times the same value. This aims to generate "key-dependent algorithmic noise" as described in subsection 6.1.1. For this purpose, it is additionally required that the implementation executes the S-boxes in parallel, that their leakage models are sufficiently similar and that targeting the MixColumns operation is hard. If these conditions are satisfied, it implies that the adversary will not be able to distinguish the leakage from the 16 S-boxes (due to their identical input), at least leading to the requirement to enumerate all the possible permutations of key bytes, which has a cost of $16! \approx 2^{44.25}$ for the AES (see [MSJ12], Section 5). As a result, the main qualitative questions when analyzing the physical security of such a primitive are whether attacking MixColumns is indeed hard, and whether the S-boxes' leakage models are sufficiently similar. In the rest of this section, we analyze these two points in detail.

Regarding the first question, Figure 6.6 reports the SNR observed on the S-boxes and MixColumns output bytes, for our two targets. On the ARM Cortex-M33, we first note that the 4 columns are processed serially, confirming that the underlying architecture is 32-bit. For the ARM Cortex-M4, all the SNR peaks occur simultaneously, hinting towards a 128-bit architecture. We observe that the SNR on the MixColumns operation is at least two order of magnitudes lower than the one on the S-boxes. We assume the latter is due to MixColumns's being implemented by harder to target combinational logic and/or its leakages requiring wider key guesses to be exploited. Hence, we conclude from this experiment that Medwed et al.'s first requirement is sufficiently satisfied.[7]

Regarding the second question, Figure 6.7 depicts the leakage distributions in a linear subspace and in the direct leakage space for a constant

[7] Our SNR is admittedly based on 8-bit guesses. However, 32-bit linear models estimated with [SLP05] did not exhibit significantly better results. So given the wide gap with the S-box leakages, we assume that targeting MixColums will remain a worse strategy for our two target devices.



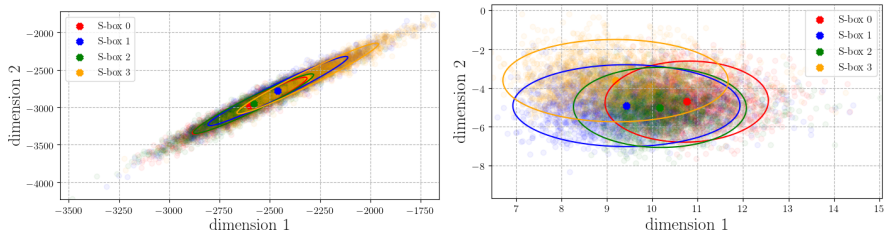
(a) Cortex-M33 (32-bit arch) (b) Cortex-M4 (128-bit arch)

Figure 6.6: SNR on AES S-boxes and MixColumns outputs evaluated on $200 \cdot 10^6$ traces. The color mapping corresponds to bytes within the same AES column.

value at the input of each of the first AES column's S-boxes. The distributions are shown for the two first dimensions and the linear subspace is the one of the first S-box. Looking at the distributions on the ARM Cortex-M33 (resp., Cortex-M4), we observe in Figure 6.7a (resp., Figure 6.7c) that they are already different in the original leakage space. These differences in the first dimensions are enhanced by moving to the linear subspace, as depicted in Figure 6.7b (resp., Figure 6.7c).

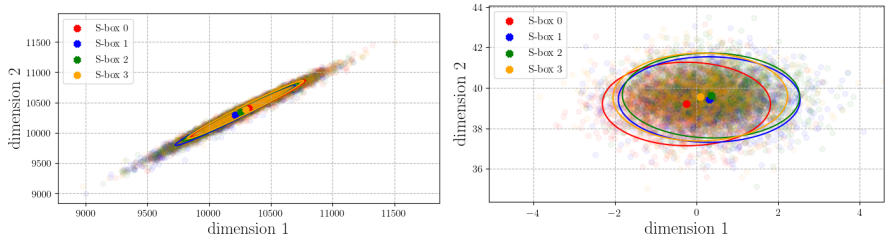
In Figure 6.8, we then study the difference between the leakage distributions of the first vs. the other S-boxes, based on Hotelling's T^2-test [Hot31]. On these plots, the X-axis is the amount of averaging performed by the adversary and the Y-axis is the significance of the difference. First, we observe that by increasing the amount of averaging, the significance increases. This is expected since it reduces the noise on the distributions, making their means easier to estimate. Additionally for the ARM Cortex-M33 (resp., Cortex-M4), by comparing Figure 6.8a (resp., Figure 6.8b) with Figure 6.8c (resp., Figure 6.8d) we observe that the differences are of the same magnitude in the linear subspace and in the direct leakage space but that more dimensions are needed in the second case (200 dimensions instead of 6). We notice that the advantage of using additional dimensions is larger on the ARM Cortex-M33. The difference between the distributions on the ARM Cortex-M33 is also larger than on ARM Cortex-M4, presumably due to a more serial implementation.

While these results remain qualitative, they highlight that the leakage models of different S-boxes are different for our two targets. Hence, they contradict one of Medwed et al.'s assumptions. Informally, our results



(a) Cortex-M33 (32-bit arch), leakage space

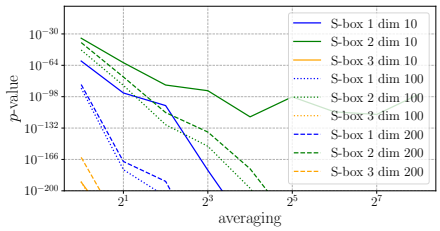
(b) Cortex-M33 (32-bit arch), linear subspace



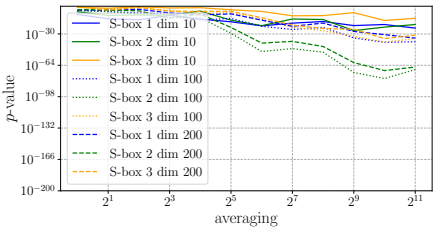
(c) Cortex-M4 (128-bit arch), leakage space

(d) Cortex-M4 (128-bit arch), linear subspace

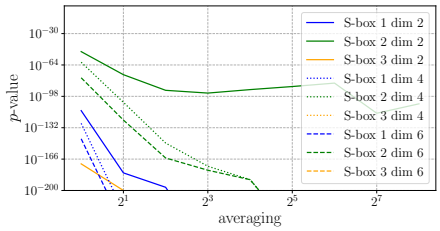
Figure 6.7: Leakage distributions of multiple S-boxes for a constant input value. Distributions after 256 (resp., 2048) averaging on the ARM Cortex-M33 (resp., Cortex-M4). The linear subspace is the one of the first S-box. The two dimensions of the original leakage space are the ones with highest SNR. Curves represent a Gaussian fit on the clusters.



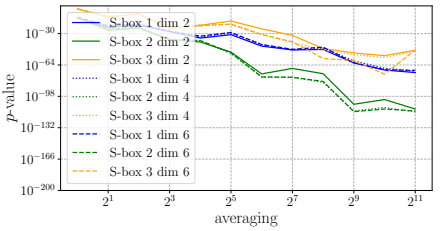
(a) Cortex-M33 (32-bit arch), leakage space



(b) Cortex-M4 (128-bit arch), leakage space



(c) Cortex-M33 (32-bit arch), linear subspace



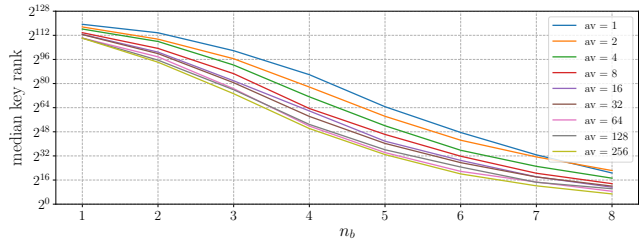
(d) Cortex-M4 (128-bit arch), linear subspace

Figure 6.8: Hotelling's T^2-test between leakage distributions of S-box 0 and other S-boxes for various number of dimensions, computed with (averaged) $2,000$ samples per distribution.

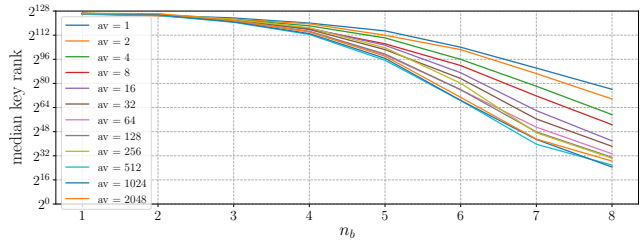
suggest that with a precise enough profiling, it should be possible to exploit these differences, enabling the adversary to escape the enumeration of a 16-permutation as expected from the LR-PRF design. In that context, projecting the leakages in a linear subspace allows "concentrating" these differences in a few dimensions which makes it easier to exploit by an adversary/evaluator. We note that [Unt+17; Unt+18] already pointed out that linear subspaces and spacial resolution can be used to observe different leakage models for different S-boxes. Our experiments show that a similar effect can be obtained without spacial resolution for the targets we look at.

6.3.2 LR-PRF: quantitative attacks

We now report the results of attacks against the LR-PRF. We used template attacks in linear subspace, as described in paragraph 2.3, since they provided the best results. For the ARM Cortex-M33 (resp., Cortex-M4), the profiling is performed on $500 \cdot 10^3$ (resp., $2 \cdot 10^6$) averaged traces. For example, for 256 (resp., 2048) averaging, we collected a total of $256 \cdot 500 \cdot 10^3$ (resp., $2048 \cdot 2 \cdot 10^6$) measurements. We note that the profiling was performed using random keys and plaintexts in order to



(a) Cortex-M33 (32-bit arch)



(b) Cortex-M4 (128-bit arch)

Figure 6.9: Median key rank of the Medwed et al. LR-PRF for various data complexities. Medians are estimated on 100 independent attacks with random keys.

avoid biasing our models with key-dependent features that would not be exploitable during the online attack phase.

The median key rank estimated according to [PSG16] is reported in Figure 6.9, where the X-axis is the LR-PRF parameter n_b (see subsection 6.1.1). We observe that by increasing n_b, and so the number of different averaged traces available to the adversary, the key rank significantly decreases. The fact that the key rank decreases below $16! \approx 2^{44.25}$ confirms that we are able to exploit the different leakage models of the different S-boxes. Averaging also allows decreasing the key rank but its impact saturates at some point, similarly to what can be observed in [Unt+20], Figure 10. On the ARM Cortex-M33 (resp., Cortex-M4) starting from 64 (resp., 512), averaging does not improve the attacks.

Concretely, for the ARM Cortex-M33 our adversary reduces the median key ranks down to 2^9 if $n_b = 8$, and down to 2^{50} if $n_b = 4$. For keeping it larger than 2^{90}, we need $n_b \leq 2$. The ARM Cortex-M4 is more resistant thanks to its parallel architecture: with $n_b = 8$, the median key rank is equal to 2^{25} but with $n_b \leq 5$ it is already above 2^{96}.

6.3.3 LR-PRF: the evaluation challenge

Our analyses imply that smaller n_b values than reported in [Unt+20] should be used for our similar targets and measurement setup. For example, Figure 11 in this reference suggests that on both targets, $n_b = 6$ leads to more than 100 bits of security. In our evaluations, this value is reduced to 20 (resp., 70) on the ARM Cortex-M33 (resp., Cortex-M4).

Technically, these results are not in strong contradiction with previous works. For example, as illustrated in [Unt+17; Unt+18], the resistance of a LR-PRF highly depends on the adversary's measurement capabilities and the exploitable signal within the target. On an FPGA implementation of the LR-PRF, it has been showed that such schemes can be insecure even for $n_b = 1$ depending on the floorplanning. To some extent, this is typical for any side-channel countermeasure: eventually, their security depends on physical assumptions. Fulfilling such assumptions is always a challenge, whether being to ensure a sufficient noise level or a sufficiently low side-channel signal. Yet, one important conclusion of our experiments is that the "low signal" requirement of LR-PRF constructions ensuring low complexity DPA (or even SPA) to be hard is also more challenging to evaluate. The main reason of this fact is that such constructions generally enable averaging the physical noise. In this context, evaluating the signal becomes very dependent on the shape (i.e., the deterministic part) of the leakage function, while "noise amplification" countermeasures such as masking or shuffling only require that the ratio between signal and noise (SNR) is sufficiently low, independently of the signal shape. Since the shape of a leakage function is quite sensitive to (even minor) modifications of the target designs and measurement setups, constructions relying on limited signal (if implemented without additional countermeasures), inherently carry a higher risk of suboptimal (i.e., non worst-case) evaluation.

Such a risk is illustrated by the comparison between our analysis and the one of [Unt+20] where mild variations of the measurement setup and a collection of 1,000 more profiling traces lead to different security parameters for similar targets (i.e., same architecture but not exactly the same MCU). They could be amplified if the adversary obtains a better a-priori knowledge of her target, or performs larger key guesses than the evaluator. Hence, a conclusion is that designers should be conservative when selecting the parameter n_b of a LR-PRF. As will be detailed in Section 6.4.2, low n_b values are anyway not overly detrimental from a performance viewpoint, since they can be amortized for long messages when considering leveled-implementation.

6.4 The LR-BC-2 mode of operation

This section briefly describes the architecture of a new LR-BC-2 operating mode designed to address the integrity issues highlighted in the previous sections [Bro+21]. Keeping a focus on secure firmware updates as its intended application, the later is a Grade-2 algorithm and relies solely on block ciphers, thus leveraging hardware coprocessors for security and performance purposes. Additionally, it relies on a careful instance for the tag verification. This chapter primarily concerns the performance it can achieve and aims to quantify whether a solution as competitive as the one proposed in [Unt+20] can be obtained while ensuring integrity in the presence of decryption leakage. Although this thesis mainly focus on the performance of the implementation, a more detailed description as well as a security analysis of the mode are proposed in [Bro+21].

6.4.1 Specification

The LR-BC-2 mode is represented in Figure 6.10. It follows the general 3-step blueprint suggested described in Chapter 3 for Grade-2 and relies on a block cipher operating on n-bit block (depicted as BC in the Figure). The *initialization* step computes a random $2n$-bit state (K_1, L_1) from a key derivation function $\mathsf{KDF}_K(N)$, where K_1 is an ephemeral encryption key. For this first step and as shown in Figure 6.11b, we borrow the BC-based bit-by-bit PRF (i.e., $n_b = 1$) of the previous section that we apply to the n-bit nonce N to get a seed K_0, allowing producing our initial state (K_1, L_1). As a second step, the *bulk* of the computation is made as a one-time encryption of $M_1 \| \cdots \| M_\ell$ where, at each successive processing of a message block M_i, the corresponding ciphertext block C_i is created and absorbed, and the state is refreshed. Eventually, in the *finalization* step, we use the Tag Generation Function (TGF) shown in Figure 6.12 to authenticate the final state with an appropriate verification mechanism (depicted in blue).

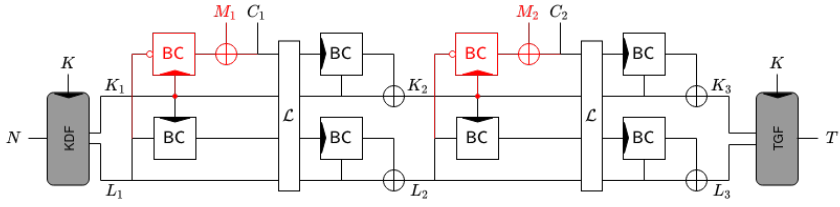
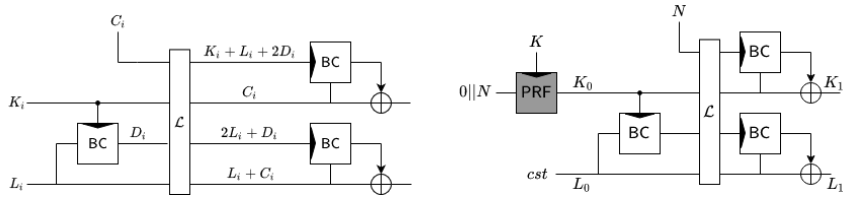


Figure 6.10: The LR-BC-2 mode of operation.



(a) The coefficients of $\mathcal{L}$. (b) KDF for some constant cst.

Figure 6.11: The Key Derivation Function (KDF) and our linear map $\mathcal{L}$.

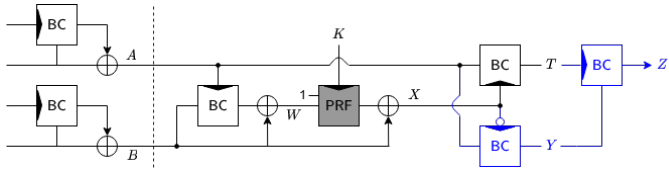


Figure 6.12: The tag generation function with the tag comparison.

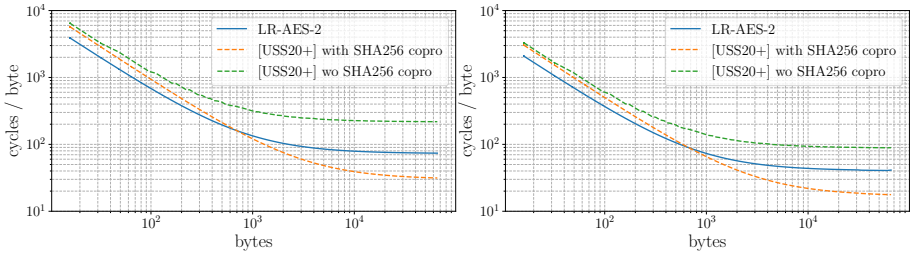
6.4.2 Performance evaluation

BC calls count for LR-BC-2 In Table 6.1, we detail the number of BC calls required to encrypt l-block messages with LR-BC-2 and [Unt+20]. We consider the AES block cipher in the context of our analysis. For the two first modes, decryption requires 3 additional block cipher calls for the tag verification. We separate these figures for the three main steps of the modes, namely the KDF, bulk computation and TGF.

Scheme	Grade	KDF	Bulk	TGF	Total
LR-BC-2	2	$130 + 3$	$4 \cdot l$	$130 + 2$	$260 + 4 \cdot l + 5$
[Unt+20]	?	129	$2 \cdot l +$?	257	$386 + 2 \cdot l +$?

Table 6.1: Number of block cipher calls to encrypt l-block messages.

The main observations from this table are twofold. First, the constant part of the cost of these modes is dominated by the execution of the LR-PRF. By contrast, the construction from [Unt+20] has a larger constant cost due to its more expensive finalization that uses the LR-PRF on 384 bits. Second, our one-pass modes asymptotically require $4 \cdot l$ calls to the block cipher. Evaluating [Unt+20] is more difficult for this part, since it requires $2 \cdot l$ calls to a block cipher and a call to a hash function on all the message (which can be implemented in software or hardware as discussed next).



(a) Cortex-M33 (32-bit arch) (b) Cortex-M4 (128-bit arch)

Figure 6.13: Performances of LR-AES-2 modes -O3 -flto optimizations and $n_b = 1$.

LR-AES encryption throughput We start by describing the two optimizations we used in order to implement the LR-BC mode based on AES coprocessors, that we next denote as LR-AES-2. Note that the proposal from [Unt+20] was instantiated with both an AES and a SHA-256.

Since these schemes heavily rely on calls to the AES, the first guideline we followed is to use the ECB mode (available on most coprocessors) whenever multiple (e.g., 2 in our case) calls to the AES with the same key are needed. This trick allows saving at least the cost of writing the key to the coprocessor, as well as its initialization. Therefore, it significantly speeds up the bulk computation of LR-BC as well as [Unt+20]. In addition, if the coprocessor has a direct memory access (DMA), it will be able to run the two encryptions without relying on any software instruction. The second optimization is to work on explicitly aligned words (i.e., uint32_t): this ensures that the compiler will operate on machine words instead of bytes. Roughly, the combination of these optimizations lead to a $\approx 50\%$ improvement of the throughput reported in [Unt+20].

The curves in Figure 6.13 depict the encryption throughput ([cycles/byte]) for LR-AES-2 and [Unt+20]. By comparing Figure 6.13a with Figure 6.13b, we notice that the Cortex-M4 systematically offers a better throughput than the Cortex-M33 thanks to its faster AES coprocessor (with a 128-bit architecture). For the rest, the comparative results are similar on both targets and follow the trends outlined in Table 6.1.

More into the details, for short messages (e.g., 16 bytes), the cost of the KDF and TGF dominates the cycle count. LR-AES-2 reaches the better throughput. Indeed, it requires $\approx 2 \cdot 130$ calls to the AES for these steps, while [Unt+20] requires ≈ 128 additional calls. For long messages, the cost of the KDF and TGF are amortized and the bulk computation dominates the cycle count. Therefore, the LR-AES-2 gives the best

throughputs. As for the comparison with [Unt+20], it highly depends on the presence of a hardware accelerator for SHA-256. When available, its throughput is ≈ 2.3 better than our Grade-2 designs (but without strong integrity guarantees in the presence of leakage due to the attack in subsection 6.2.2). If there is no hardware accelerator for the hash function, then LR-AES-2 is better with a factor ≈ 2.94 (resp., ≈ 2.17) on the ARM Cortex-M33 (resp., Cortex-M4). We note that because [Unt+20] is two-pass, the entire ciphertext must be stored in memory when decrypting (it is not the case for LR-AES-2 which is online).

Overall, these results suggest that the high cost of the LR-PRF will be quite well amortized for actual software updates, and that hardware coprocessors enable making the $4 \cdot l$ block cipher calls per message block of our modes acceptable in practice. For example, the firmware of our ARM Cortex-M33 has a size of 21,184 bytes and the one of our ARM Cortex-M4 has a size of 19,064 bytes. Even with the most conservative security parameter for the LR-PRF ($n_b = 1$), encrypting this amount of data is sufficient to reach a good amortization (as shown in Figure 6.13). Also, considering the maximum clock frequencies of our two target devices, decrypting their ciphertext with LR-AES-2 will take 0.014 seconds (resp., 0.0043 seconds) on the ARM Cortex M33 (resp., M4).

6.5 Conclusions

This chapter deviates even further from traditional countermeasures and explores the feasibility of achieving DPA security based on devices that rely solely on SPA protections. In the context of hardware, this results in an implementation without low-level protections but with a certain level of parallelism in its architecture (as discussed in Chapter 3). This chapter demonstrates that, even in this context appearing more difficult at first glance, security against DPA can be achieved in practice by limiting the amount of observable leakage.

To this end, the tree architecture relying on AES coprocessor proposed in [MSJ12] offers two architectural advantages. Firstly, it effectively restricts the number of observable leakages by minimizing the use of the long-term key and handling a bounded number of different plaintexts. Secondly, this quantity of plaintext is, in fact, a parameter that can be tweaked to adjust the trade-off between performance and security.

However, as discussed in Section 6.3.1, this LR-PRF solution still requires two heuristic conditions that may prove challenging to satisfy. The stronger condition lies in the assumption that different hardware instances exhibit a similar leakage pattern. Section 6.3.1 demonstrates

that the latter presents first-order leakage differences, contradicting the hypothesis. This observation can be further supported by the leakage model analysis conducted in Chapter 5, Section 5.3.2, which shows that bits stored in registers can have varying influences on the leakage. Furthermore, another hypothesis is that attacking these S-boxes is the best strategy. The assessments in Section 6.3.1 seem to support this idea, indicating that models targeting larger word sizes may be necessary. To this end, the analysis of the attack targeting MixColumn could be extended, especially in light of recent research that has conducted advanced attacks against 32-bit words [Cas+23a].

From a more general perspective, another limitation is associated with the assessment of the security level achieved by such constructions. The latter aims to confirm that there is a sufficiently low signal in the deterministic part of the leakage, as physical noise can be mitigated through averaging. This function is sensitive to minor implementation variations, and its analysis depends on the measurement setup used, which can lead to suboptimal analyses that are challenging to generalize to other devices. Overall, these observations support the choice of conservative parameters when instantiating LR-PRF.

As a complement, this chapter demonstrates that improperly instantiated tag verification can lead to practical attacks, further emphasizing the need for a comprehensive analysis of the security requirements for each component when designing modes of operations. Furthermore, this chapter shows that the performance limitations of the LR-PRF (e.g., when instantiated with the most conservative parameters) can be mitigated by incorporating it into a mode-leveled approach.

7 Conclusion

In this thesis, I explore the design space offered by hardware implementations secure against side-channel attacks and their integration into leakage-resistant modes of operations. They come with the degree of freedom to enable implementations with parallel architecture, which turns out to be beneficial for security against SPA attacks. Taking this consideration into account, I am more particularly interested in a range of DPA protection strategies, ranging from more generic (but expensive and requiring expertise to implement) to more specialized ones that offer better efficiency and ease of implementation.

Parallel hardware implementations have the advantage of inherent resistance to SPA attacks, motivating their use in the context of modes of operations implementations that enable leveled architectures, as explained in Chapter 3. In practical terms, their integration allows for the processing of long messages with the performance of an unprotected implementation, all while ensuring security properties against stronger attacks, such as DPA. However, such solutions require a minimal use of primitives protected against these DPA attacks. In this context, various techniques are evaluated in Chapters 4,5, and 6. They range from a standard approach (using strong low-level countermeasure) to less standard ones (relying on algorithmic strategy empowered by parallel hardware implementation) and face different challenges. Yet, it turns out that they all hold a particular interest in the case of leveled implementations, as illustrated by practical examples.

The first technique evaluated is to rely on masking. The latter is quite common nowadays and has the significant advantage of being the subject of substantial research efforts to assess its practical security. Nonetheless, it comes with a considerable cost (quadratic in terms of the number of shares). This trend is amplified and also affects performances when physical and compositional defaults are considered, as illustrated in Chapter 4 for a masking scheme developed under the PINI framework that is formally proven secure under such conditions. On a more positive note, the chapter also demonstrates that improvements are possible to achieve efficient implementations that offer a good trade-off between area and latency. Such architectures are of particular interest in the

context of leveled implementations (high throughput is not required for the protected primitive in such a case, the majority of the computation being handled by the unprotected primitive), and further developing the work undertaken in [Cas+23b] appears promising in that sense.

Another approach is to rely on re-keying, an alternative to reduce the impact of the limitations inherent to masking of common primitives. In particular, Chapter 5 emphasizes that the inherent physical behavior of the implementation (i.e., the leakage function) can become a source of cryptographic hardness in a side-channel security context. Specifically, assuming it follows a compressive (i.e., not one-to-one) behavior, it can be seen as a rounding function similar to what is used in common hard-learning problems instances (i.e., LWE and LWR). In this context, Chapter 5 introduces the hard-physical learning problem LWPR defined for a key-homomorphic re-keying function. By leveraging key homomorphism properties, the cost of protecting the re-keying function with masking can follow a linear trend with the number of shares (rather than quadratic as observed for more standard masked primitives), thus limiting its impact for high levels of protection. This solution is made possible through a high level of parallelism, and a natural extension is to evaluate the security level that can be achieved for more serial implementations (based, for example, on hardware shuffling). However, such evaluations require extending existing analyses (conducted for a univariate function only) to multivariate leakage functions. Lastly, while cryptographic properties have already been proven for a general class of leakage functions, the analysis of such a solution still requires an assumption about the form of the leakage function, the accuracy of which can be challenging to assess in practice (this conclusion is also observed in Chapter 5). To relax this assumption, a promising exploration track would be to link the formal security to the cardinality of the function, which could be directly related to the level of quantification that measurement apparatus can reach in practice.

Interestingly, Chapter 6 demonstrates that it is possible to achieve DPA security based on primitives protected solely against SPA attacks (i.e., in a context opposed to what is discussed in Chapter 4 and appearing more difficult at first glance). In particular, it shows that LR-PRFs, which limit by design the number of observations that can be obtained when trying to mount an attack, can be used as a re-keying function. While promising, this solution relies on heuristic security arguments that can be challenging to satisfy. Specifically, Chapter 6 demonstrates that the necessity of having multiple instances of the similar subcircuits leaking in the same way is difficult to achieve in practice due to the physical

imperfections of the circuit (this observation is further supported by the analyses conducted in Chapter 5). From a more general perspective, it highlights that assessing the security achieved in practice by such solutions is directly related to the quality of the analysis setup and is difficult to generalize. Given this, relying on such LR-PRFs requires the use of conservative parameters. Nevertheless, this does not prevent solutions based on the latter from achieving competitive performance with limited overheads compared to existing solutions based on low-level countermeasures.

Personal Related Publications

[Bel+20a] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Mode-Level vs. Implementation-Level Physical Security in Symmetric Cryptography - A Practical Guide Through the Leakage-Resistance Jungle." In: *CRYPTO (1)*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12170. Lecture Notes in Computer Science. Springer, 2020, pp. 369–400. DOI: 10.1007/978-3-030-56784-2_13. URL: https://doi.org/10.1007/978-3-030-56784-2_13.

[Bro+21] Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. "Improved Leakage-Resistant Authenticated Encryption based on Hardware AES Coprocessors." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (2021), pp. 641–676. DOI: 10.46586/tches.v2021.i3.641-676. URL: https://doi.org/10.46586/tches.v2021.i3.641-676.

[Duv+21] Sébastien Duval, Pierrick Méaux, Charles Momin, and François-Xavier Standaert. "Exploring Crypto-Physical Dark Matter and Learning with Physical Rounding Towards Secure and Efficient Fresh Re-Keying." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.1 (2021), pp. 373–401.

[Hof+23b] Clément Hoffmann, Pierrick Méaux, Charles Momin, Yann Rotella, François-Xavier Standaert, and Balazs Udvarhelyi. "Learning with Physical Rounding for Linear and Quadratic Leakage Functions." In: *CRYPTO (3)*. Vol. 14083. Lecture Notes in Computer Science. Springer, 2023, pp. 410–439.

[MCS21] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. "Unprotected and Masked Hardware Implementations of Spook v2." In: *SILC Workshop* (2021).

[MCS22] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. "Handcrafting: Improving Automated Masking in Hardware with Manual Optimizations." In: *COSADE*. Ed. by Josep Balasch and Colin O'Flynn. Vol. 13211. Lecture Notes in Computer Science. Springer, 2022, pp. 257–275. DOI: 10.1007/978-3-030-99766-3_12. URL: https://doi.org/10.1007/978-3-030-99766-3_12.

[SC23a] SIMPLE-Crypto. "Masked Hardware AES-128 Encryption with HPC2." In: *SMAesH: technical documentation* (2023).

Personal Others Publications

[Bel+20b] Davide Bellizia et al. “Spook: Sponge-Based Leakage-Resistant Authenticated Encryption with a Masked Tweakable Block Cipher.” In: *IACR Trans. Symmetric Cryptol.* 2020.S1 (2020), pp. 295–349.

[Cas+23b] Gaëtan Cassiers, Barbara Gigerl, Stefan Mangard, Charles Momin, and Rishub Nagpal. “Compress: Reducing Area and Latency of Masked Pipelined.” In: *IACR Cryptol. ePrint Arch.* (2023), p. 1600.

[Cas+23c] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, Amir Moradi, and François-Xavier Standaert. “Randomness Generation for Secure Hardware Masking - Unrolled Trivium to the Rescue.” In: *IACR Cryptol. ePrint Arch.* (2023), p. 1134.

[Hof+23a] Clément Hoffmann, Benoît Libert, Charles Momin, Thomas Peters, and François-Xavier Standaert. “POLKA: Towards Leakage-Resistant Post-quantum CCA-Secure Public Key Encryption.” In: *Public Key Cryptography (1)*. Vol. 13940. Lecture Notes in Computer Science. Springer, 2023, pp. 114–144.

[MBS21] Charles Momin, Olivier Bronchain, and François-Xavier Standaert. “A stealthy Hardware Trojan based on a Statistical Fault Attack.” In: *Cryptogr. Commun.* 13.4 (2021), pp. 587–600.

Bibliography

[ABF13] Michel Abdalla, Sonia Belaïd, and Pierre-Alain Fouque. "Leakage-Resilient Symmetric Encryption via Re-keying." In: *CHES*. Vol. 8086. LNCS. Springer, 2013, pp. 471–488.

[Alf98] Peter Alfke. "Efficient shift registers, LFSR counters, and long pseudo-random sequence generators." In: *http://www. xilinx. com/bvdocs/appnotes/xapp052. pdf* (1998).

[Alw+13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. "Learning with Rounding, Revisited - New Reduction, Properties and Applications." In: *CRYPTO (1)*. Vol. 8042. Lecture Notes in Computer Science. Springer, 2013, pp. 57–74.

[Arc+06] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. "Template Attacks in Principal Subspaces." In: *CHES*. Vol. 4249. Lecture Notes in Computer Science. Springer, 2006, pp. 1–14.

[Bar+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. "Strong Non-Interference and Type-Directed Higher-Order Masking." In: *CCS*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 116–129. DOI: 10.1145/2976749.2978427. URL: https://doi.org/10.1145/2976749.2978427.

[Bar+17a] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. "Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model." In: *EUROCRYPT (1)*. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 535–566.

[Bar+17b] Guy Barwell, Daniel P. Martin, Elisabeth Oswald, and Martijn Stam. "Authenticated Encryption in the Face of Protocol and Side Channel Leakage." In: *ASIACRYPT*

(1). Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 693–723.

[Bat+16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. "Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme." In: *CHES*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Vol. 9813. Lecture Notes in Computer Science. Springer, 2016, pp. 23–39. DOI: 10.1007/978-3-662-53140-2_2. URL: https://doi.org/10.1007/978-3-662-53140-2_2.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation Power Analysis with a Leakage Model." In: *CHES*. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 16–29.

[Bel+15] Sonia Belaïd, Jean-Sébastien Coron, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, and Emmanuel Prouff. "Improved Side-Channel Analysis of Finite-Field Multiplication." In: *CHES*. Vol. 9293. Lecture Notes in Computer Science. Springer, 2015, pp. 395–415.

[Bel+20a] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Mode-Level vs. Implementation-Level Physical Security in Symmetric Cryptography - A Practical Guide Through the Leakage-Resistance Jungle." In: *CRYPTO (1)*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12170. Lecture Notes in Computer Science. Springer, 2020, pp. 369–400. DOI: 10.1007/978-3-030-56784-2_13. URL: https://doi.org/10.1007/978-3-030-56784-2_13.

[Bel+20b] Davide Bellizia et al. "Spook: Sponge-Based Leakage-Resistant Authenticated Encryption with a Masked Tweakable Block Cipher." In: *IACR Trans. Symmetric Cryptol.* 2020.S1 (2020), pp. 295–349.

[Ber+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications." In: *Selected Areas in Cryptography*. Vol. 7118. LNCS. Springer, 2011, pp. 320–337.

[Ber+17] Francesco Berti, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "On Leakage-Resilient Authenticated Encryption with Decryption Leakages." In: *IACR Trans. Symmetric Cryptol.* 2017.3 (2017), pp. 271–293.

[Ber+18] Francesco Berti, François Koeune, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Ciphertext Integrity with Misuse and Leakage: Definition and Efficient Constructions with Symmetric Primitives." In: *AsiaCCS*. ACM, 2018, pp. 37–50.

[Ber+19] Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Strong Authenticity with Leakage Under Weak and Falsifiable Physical Assumptions." In: *Inscrypt*. Vol. 12020. LNCS. Springer, 2019, pp. 517–532.

[Ber+20] Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "TEDT, a Leakage-Resist AEAD Mode for High Physical Security Applications." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.1 (2020), pp. 256–320.

[BFG14] Sonia Belaïd, Pierre-Alain Fouque, and Benoît Gérard. "Side-Channel Analysis of Multiplications in GF(2128) - Application to AES-GCM." In: *ASIACRYPT (2)*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. Lecture Notes in Computer Science. Springer, 2014, pp. 306–325. DOI: 10.1007/978-3-662-45608-8_17. URL: https://doi.org/10.1007/978-3-662-45608-8_17.

[BGS15] Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. "Masking and leakage-resilient primitives: One, the other(s) or both?" In: *Cryptography and Communications* 7.1 (2015), pp. 163–184.

[Bon+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. "Exploring Crypto Dark Matter: - New Simple PRF Candidates and Their Applications." In: *TCC (2)*. Vol. 11240. Lecture Notes in Computer Science. Springer, 2018, pp. 699–729.

[Bos+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM." In: *EuroS&P*. IEEE, 2018, pp. 353–367.

[BP12] Joan Boyar and René Peralta. "A Small Depth-16 Circuit for the AES S-Box." In: *SEC*. Vol. 376. IFIP Advances in Information and Communication Technology. Springer, 2012, pp. 287–298.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. "Pseudorandom Functions and Lattices." In: *EUROCRYPT*. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 719–737.

[Bro+19] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. "Leakage Certification Revisited: Bounding Model Errors in Side-Channel Security Evaluations." In: *CRYPTO (1)*. Vol. 11692. Lecture Notes in Computer Science. Springer, 2019, pp. 713–737.

[Bro+21] Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. "Improved Leakage-Resistant Authenticated Encryption based on Hardware AES Coprocessors." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (2021), pp. 641–676. DOI: 10.46586/tches.v2021.i3.641-676. URL: https://doi.org/10.46586/tches.v2021.i3.641-676.

[BS21] Olivier Bronchain and François-Xavier Standaert. "Breaking Masked Implementations with Many Shares on 32-bit Software Platforms or When the Security Order Does Not Matter." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (2021), pp. 202–234.

[BSS20] Olivier Bronchain, Tobias Schneider, and François-Xavier Standaert. "Reducing Risks Through Simplicity (High Side-Channel Security for Lazy Engineers)." In: *Journal of Cryptographic Engineering*. 2020.

[BUS21] Davide Bellizia, Balazs Udvarhelyi, and François-Xavier Standaert. "Towards a Better Understanding of Side-Channel Analysis Measurements Setups." In: *CARDIS*. Vol. 13173. Lecture Notes in Computer Science. Springer, 2021, pp. 64–79.

[Cas+20a] Gaëtan Cassiers, Benjamin Grégoire, Itamaer Levi, and François-Xavier Standaert. "Hardware Private Circuits: From Trivial Composition to Full Verification (aka Repairing Glitch-Resistant Higher-Order Masking)." In: *IACR ePrint Archive* (2020).

[Cas+20b] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. "Hardware Private Circuits: From Trivial Composition to Full Verification." In: *IEEE Transactions on Computers* (2020 (to appear)).

[Cas+21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. "Hardware Private Circuits: From Trivial Composition to Full Verification." In: *IEEE Trans. Computers* 70.10 (2021), pp. 1677–1690. DOI: 10.1109/TC.2020.3022979. URL: https://doi.org/10.1109/TC.2020.3022979.

[Cas+23a] Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. "Efficient Regression-Based Linear Discriminant Analysis for Side-Channel Security Evaluations Towards Analytical Attacks against 32-bit Implementations." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023.3 (2023), pp. 270–293.

[Cas+23b] Gaëtan Cassiers, Barbara Gigerl, Stefan Mangard, Charles Momin, and Rishub Nagpal. "Compress: Reducing Area and Latency of Masked Pipelined." In: *IACR Cryptol. ePrint Arch.* (2023), p. 1600.

[Cas+23c] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, Amir Moradi, and François-Xavier Standaert. "Randomness Generation for Secure Hardware Masking - Unrolled Trivium to the Rescue." In: *IACR Cryptol. ePrint Arch.* (2023), p. 1134.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. "Differential Power Analysis in the Presence of Hardware Countermeasures." In: *CHES*. Vol. 1965. LNCS. Springer, 2000, pp. 252–263.

[Cha+99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. "Towards Sound Approaches to Counteract Power-Analysis Attacks." In: *CRYPTO*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 398–412. DOI: 10.1007/3-540-48405-1_26. URL: https://doi.org/10.1007/3-540-48405-1_26.

[Ché+19] Eloi de Chérisey, Sylvain Guilley, Olivier Rioul, and Pablo Piantanida. "Best Information is Most Successful Mutual Information and Success Rate in Side-Channel Analysis."

In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 49–79.

[CK13] Omar Choudary and Markus G. Kuhn. "Efficient Template Attacks." In: *CARDIS*. Vol. 8419. Lecture Notes in Computer Science. Springer, 2013, pp. 253–270.

[Cnu+17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. "Does Coupling Affect the Security of Masked Implementations?" In: *COSADE*. Ed. by Sylvain Guilley. Vol. 10348. Lecture Notes in Computer Science. Springer, 2017, pp. 1–18. DOI: 10.1007/978-3-319-64647-3_1. URL: https://doi.org/10.1007/978-3-319-64647-3_1.

[Cor+12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. "Conversion of Security Proofs from One Leakage Model to Another: A New Issue." In: *COSADE*. Ed. by Werner Schindler and Sorin A. Huss. Vol. 7275. Lecture Notes in Computer Science. Springer, 2012, pp. 69–81. DOI: 10.1007/978-3-642-29912-4_6. URL: https://doi.org/10.1007/978-3-642-29912-4_6.

[Cor+13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. "Higher-Order Side Channel Security and Mask Refreshing." In: *FSE*. Vol. 8424. Lecture Notes in Computer Science. Springer, 2013, pp. 410–424.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. "Template Attacks." In: *CHES*. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 13–28.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. "Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference." In: *IEEE Trans. Inf. Forensics Secur.* 15 (2020), pp. 2542–2555. DOI: 10.1109/TIFS.2020.2971153. URL: https://doi.org/10.1109/TIFS.2020.2971153.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. "Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.2 (2021), pp. 136–158.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. "Unifying Leakage Models: From Probing Attacks to Noisy Leakage." In: *EUROCRYPT*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, 2014, pp. 423–440. DOI: 10.1007/978-3-642-55220-5_24. URL: https://doi.org/10.1007/978-3-642-55220-5_24.

[DFS15] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. "Making Masking Security Proofs Concrete - Or How to Evaluate the Security of Any Leaking Device." In: *EUROCRYPT (1)*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 401–429. DOI: 10.1007/978-3-662-46800-5_16. URL: https://doi.org/10.1007/978-3-662-46800-5_16.

[DFS19] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. "Making Masking Security Proofs Concrete (Or How to Evaluate the Security of Any Leaking Device), Extended Version." In: *J. Cryptology* 32.4 (2019), pp. 1263–1297.

[Din+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. "Towards Sound and Optimal Leakage Detection Procedure." In: *CARDIS*. Vol. 10728. Lecture Notes in Computer Science. Springer, 2017, pp. 105–122.

[DJS19] Jean Paul Degabriele, Christian Janson, and Patrick Struck. "Sponges Resist Leakage: The Case of Authenticated Encryption." In: *ASIACRYPT (2)*. Vol. 11922. Lecture Notes in Computer Science. Springer, 2019, pp. 209–240.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. "Leakage-Resilient Cryptography." In: *FOCS*. IEEE Computer Society, 2008, pp. 293–302.

[DP10] Yevgeniy Dodis and Krzysztof Pietrzak. "Leakage-Resilient Pseudorandom Functions and Side-Channel Attacks on Feistel Networks." In: *CRYPTO*. Vol. 6223. Lecture Notes in Computer Science. Springer, 2010, pp. 21–40.

[Duc+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS-Dilithium: A Lattice-Based Digital Signa-

ture Scheme." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.1 (2018), pp. 238–268.

[Duv+21] Sébastien Duval, Pierrick Méaux, Charles Momin, and François-Xavier Standaert. "Exploring Crypto-Physical Dark Matter and Learning with Physical Rounding Towards Secure and Efficient Fresh Re-Keying." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.1 (2021), pp. 373–401.

[Dzi+16] Stefan Dziembowski, Sebastian Faust, Gottfried Herold, Anthony Journault, Daniel Masny, and François-Xavier Standaert. "Towards Sound Fresh Re-keying with Hard (Physical) Learning Problems." In: *CRYPTO (2)*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. Lecture Notes in Computer Science. Springer, 2016, pp. 272–301. DOI: 10.1007/978-3-662-53008-5_10. URL: https://doi.org/10.1007/978-3-662-53008-5_10.

[Fau+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. "Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.3 (2018), pp. 89–120. DOI: 10.13154/tches.v2018.i3.89-120. URL: https://doi.org/10.13154/tches.v2018.i3.89-120.

[FPS12] Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. "Practical Leakage-Resilient Symmetric Cryptography." In: *CHES*. Vol. 7428. Lecture Notes in Computer Science. Springer, 2012, pp. 213–232.

[GFM14] Berndt Gammel, Wieland Fischer, and Stefan Mangard. *Generating a Session Key for Authentication and Secure Data Transfer*. US Patent 8,861,722. 2014.

[GGJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. "A testing methodology for side-channel resistance validation." In: *NIST non-invasive attack testing workshop*. Vol. 7. 2011, pp. 115–136.

[GJ19] Qian Guo and Thomas Johansson. "A new birthday-type algorithm for attacking the fresh re-keying countermeasure." In: *Inf. Process. Lett.* 146 (2019), pp. 30–34.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. "An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order." In: *CT-RSA*. Ed. by Helena Handschuh. Vol. 10159. Lecture Notes in Computer Science. Springer, 2017, pp. 95–112. DOI: 10.1007/978-3-319-52153-4_6. URL: https://doi.org/10.1007/978-3-319-52153-4_6.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. "Electromagnetic Analysis: Concrete Results." In: *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science. Springer, 2001, pp. 251–261. DOI: 10.1007/3-540-44709-1_21. URL: https://doi.org/10.1007/3-540-44709-1_21.

[GP99] Louis Goubin and Jacques Patarin. "DES and Differential Power Analysis (The "Duplication" Method)." In: *CHES*. Vol. 1717. LNCS. Springer, 1999, pp. 158–172.

[GR17] Dahmun Goudarzi and Matthieu Rivain. "How Fast Can Higher-Order Masking Be in Software?" In: *EUROCRYPT (1)*. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 567–597.

[Guo+19] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Authenticated Encryption with Nonce Misuse and Physical Leakage: Definitions, Separation Results and First Construction - (Extended Abstract)." In: *LATINCRYPT*. Vol. 11774. Lecture Notes in Computer Science. Springer, 2019, pp. 150–172.

[Guo+20a] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Towards Low-Energy Leakage-Resistant Authenticated Encryption from the Duplex Sponge Construction." In: *IACR Trans. Symmetric Cryptol.* 2020.1 (2020), pp. 6–42.

[Guo+20b] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Towards Low-Energy Leakage-Resistant Authenticated Encryption from the Duplex Sponge Construction." In: *IACR Trans. Symmetric Cryptol.* 2020.1 (2020), pp. 6–42.

[Hof+23a] Clément Hoffmann, Benoît Libert, Charles Momin, Thomas Peters, and François-Xavier Standaert. "POLKA: Towards Leakage-Resistant Post-quantum CCA-Secure Public Key Encryption." In: *Public Key Cryptography (1)*. Vol. 13940. Lecture Notes in Computer Science. Springer, 2023, pp. 114–144.

[Hof+23b] Clément Hoffmann, Pierrick Méaux, Charles Momin, Yann Rotella, François-Xavier Standaert, and Balazs Udvarhelyi. "Learning with Physical Rounding for Linear and Quadratic Leakage Functions." In: *CRYPTO (3)*. Vol. 14083. Lecture Notes in Computer Science. Springer, 2023, pp. 410–439.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. "An AES Smart Card Implementation Resistant to Power Analysis Attacks." In: *ACNS*. Vol. 3989. Lecture Notes in Computer Science. 2006, pp. 239–252.

[Hot31] Harold Hotelling. "The Generalization of Student's Ratio." In: *The Annals of Mathematical Statistics* 2.3 (1931), pp. 360–378.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. "Private Circuits: Securing Hardware against Probing Attacks." In: *CRYPTO*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 463–481. DOI: 10.1007/978-3-540-45146-4_27. URL: https://doi.org/10.1007/978-3-540-45146-4_27.

[Kap+] J.-P. Kaps, W. Diehl, M. Tempelmeier, E. Homsirikamol, and K. Gaj. "Hardware API for Lightweight Cryptography." In: (). URL: https://cryptography.gmu.edu/athena/LWC/LWC_HW_API.pdf.

[Ker+12] Stéphanie Kerckhof, François Durvaux, Cédric Hocquet, David Bol, and François-Xavier Standaert. "Towards Green Cryptography: A Comparison of Lightweight Ciphers from the Energy Viewpoint." In: *CHES*. Vol. 7428. LNCS. Springer, 2012, pp. 390–407.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis." In: *CRYPTO*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25. URL: https://doi.org/10.1007/3-540-48405-1_25.

[KM22] David Knichel and Amir Moradi. "Low-Latency Hardware Private Circuits." In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM, 2022, pp. 1799–1812. DOI: 10.1145/3548606.3559362. URL: https://doi.org/10.1145/3548606.3559362.

[Kni+22a] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. "Automated Generation of Masked Hardware." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 589–629. DOI: 10.46586/tches.v2022.i1.589-629. URL: https://doi.org/10.46586/tches.v2022.i1.589-629.

[Kni+22b] David Knichel, Amir Moradi, Nicolas Mullër, and Pascal Sasdrich. "Automated Generation of Masked Hardware." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2022).

[Koc03] Paul C. Kocher. *Leak-Resistant Cryptographic Indexed Key Update*. US Patent 6,539,092. 2003.

[Kop+17] Philipp Koppermann, Fabrizio De Santis, Johann Heyszl, and Georg Sigl. "Automatic generation of high-performance modular multipliers for arbitrary mersenne primes on FPGAs." In: *HOST*. IEEE Computer Society, 2017, pp. 35–40.

[KR19] Yael Tauman Kalai and Leonid Reyzin. "A Survey of Leakage-Resilient Cryptography." In: *Providing Sound Foundations for Cryptography*. ACM, 2019, pp. 727–794.

[KS20] Juliane Krämer and Patrick Struck. "Leakage-Resilient Authenticated Encryption from Leakage-Resilient Pseudorandom Functions." In: *COSADE*. Vol. 12244. Lecture Notes in Computer Science. Springer, 2020, pp. 315–337.

[KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. "Generic Hardware Private Circuits Towards Automated Generation of Composable Secure Gadgets." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 323–344. DOI: 10.46586/tches.v2022.i1.323-344. URL: https://doi.org/10.46586/tches.v2022.i1.323-344.

[Man04] Stefan Mangard. "Hardware Countermeasures against DPA ? A Statistical Analysis of Their Effectiveness." In: *CT-RSA*. Vol. 2964. Lecture Notes in Computer Science. Springer, 2004, pp. 222–235.

[MBS21] Charles Momin, Olivier Bronchain, and François-Xavier Standaert. "A stealthy Hardware Trojan based on a Statistical Fault Attack." In: *Cryptogr. Commun.* 13.4 (2021), pp. 587–600.

[MCS21] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. "Unprotected and Masked Hardware Implementations of Spook v2." In: *SILC Workshop* (2021).

[MCS22] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. "Handcrafting: Improving Automated Masking in Hardware with Manual Optimizations." In: *COSADE*. Ed. by Josep Balasch and Colin O'Flynn. Vol. 13211. Lecture Notes in Computer Science. Springer, 2022, pp. 257–275. DOI: 10.1007/978-3-030-99766-3_12. URL: https://doi.org/10.1007/978-3-030-99766-3_12.

[Med+10] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. "Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices." In: *AFRICACRYPT*. Vol. 6055. Lecture Notes in Computer Science. Springer, 2010, pp. 279–296.

[Med+11] Marcel Medwed, Christophe Petit, Francesco Regazzoni, Mathieu Renauld, and François-Xavier Standaert. "Fresh Re-keying II: Securing Multiple Parties against Side-Channel and Fault Attacks." In: *CARDIS*. Vol. 7079. Lecture Notes in Computer Science. Springer, 2011, pp. 115–132.

[Med+16] Marcel Medwed, François-Xavier Standaert, Ventzislav Nikov, and Martin Feldhofer. "Unknown-Input Attacks in the Parallel Setting: Improving the Security of the CHES 2012 Leakage-Resilient PRF." In: *ASIACRYPT (1)*. Vol. 10031. Lecture Notes in Computer Science. 2016, pp. 602–623.

[Moo+19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. "Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 256–292. DOI: 10.13154/tches.v2019.i2.256-292. URL: https://doi.org/10.13154/tches.v2019.i2.256-292.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. "Side-Channel Leakage of Masked CMOS Gates." In: *CT-RSA*. Ed. by Alfred Menezes. Vol. 3376. Lecture Notes in Computer Science. Springer, 2005, pp. 351–365. DOI: 10.1007/978-3-540-30574-3_24. URL: https://doi.org/10.1007/978-3-540-30574-3_24.

[MSJ12] Marcel Medwed, François-Xavier Standaert, and Antoine Joux. "Towards Super-Exponential Side-Channel Security with Efficient Leakage-Resilient PRFs." In: *CHES*. Vol. 7428. Lecture Notes in Computer Science. Springer, 2012, pp. 193–212.

[OC15] Colin O'Flynn and Zhizhang (David) Chen. "Side Channel Power Analysis of an AES-256 Bootloader." In: *CCECE*. IEEE, 2015, pp. 750–755.

[Pet+08] Christophe Petit, François-Xavier Standaert, Olivier Pereira, Tal Malkin, and Moti Yung. "A Block Cipher Based Pseudo Random Number Generator Secure against Side-Channel Key Recovery." In: *AsiaCCS*. Ed. by Masayuki Abe and Virgil D. Gligor. ACM, 2008, pp. 56–65. DOI: 10.1145/1368310.1368322. URL: https://doi.org/10.1145/1368310.1368322.

[Pie09] Krzysztof Pietrzak. "A Leakage-Resilient Mode of Operation." In: *EUROCRYPT*. Vol. 5479. Lecture Notes in Computer Science. Springer, 2009, pp. 462–482.

[PM16] Peter Pessl and Stefan Mangard. "Enhancing Side-Channel Analysis of Binary-Field Multiplication with Bit Reliability." In: *CT-RSA*. Vol. 9610. Lecture Notes in Computer Science. Springer, 2016, pp. 255–270.

[PR13] Emmanuel Prouff and Matthieu Rivain. "Masking against Side-Channel Attacks: A Formal Security Proof." In: *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 142–159. DOI: 10.1007/978-3-642-

38348-9_9. URL: https://doi.org/10.1007/978-3-642-38348-9_9.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. "Simple Key Enumeration (and Rank Estimation) Using Histograms: An Integrated Approach." In: *CHES*. Vol. 9813. Lecture Notes in Computer Science. Springer, 2016, pp. 61–81.

[PSV15] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. "Leakage-Resilient Authentication and Encryption from Symmetric Cryptographic Primitives." In: *CCS*. ACM, 2015, pp. 96–108.

[QS01] Jean-Jacques Quisquater and David Samyde. "Electro-Magnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards." In: *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*. Ed. by Isabelle Attali and Thomas P. Jensen. Vol. 2140. Lecture Notes in Computer Science. Springer, 2001, pp. 200–210. DOI: 10.1007/3-540-45418-7_17. URL: https://doi.org/10.1007/3-540-45418-7_17.

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. "OCB: A Block Cipher Mode of Operation for Efficient Authenticated Encryption." In: *ACM Trans. Inf. Syst. Secur.* 6.3 (2003), pp. 365–403.

[RD] Behnaz Rezvani and William Diehl. "Hardware Implementations of NIST Lightweight Cryptographic Candidates: A First Look." In: (). URL: https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2019/documents/papers/hardware-implementations-of-nist-lwc-candidates-lwc2019.pdf.

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography." In: *STOC*. ACM, 2005, pp. 84–93.

[RP10] Matthieu Rivain and Emmanuel Prouff. "Provably Secure Higher-Order Masking of AES." In: *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer

Science. Springer, 2010, pp. 413–427. DOI: 10.1007/978-3-642-15031-9_28. URL: https://doi.org/10.1007/978-3-642-15031-9_28.

[SC23a] SIMPLE-Crypto. "Masked Hardware AES-128 Encryption with HPC2." In: *SMAesH: technical documentation* (2023).

[SC23b] SIMPLE-Crypto. "SMAesH: preliminary evaluation report." In: *SMAesH: technical documentation* (2023).

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. "A Stochastic Model for Differential Side Channel Cryptanalysis." In: *CHES*. Vol. 3659. Lecture Notes in Computer Science. Springer, 2005, pp. 30–46.

[SM16] Tobias Schneider and Amir Moradi. "Leakage assessment methodology - Extended version." In: *J. Cryptogr. Eng.* 6.2 (2016), pp. 85–99.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. "A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks." In: *EUROCRYPT*. Vol. 5479. Lecture Notes in Computer Science. Springer, 2009, pp. 443–461.

[SN17] National Institute of Standards and Technology (NIST). "Lightweight Cryptography." In: (2017). URL: https://csrc.nist.gov/Projects/lightweight-cryptography.

[SPY13] François-Xavier Standaert, Olivier Pereira, and Yu Yu. "Leakage-Resilient Symmetric Cryptography under Empirically Verifiable Assumptions." In: *CRYPTO (1)*. Vol. 8042. Lecture Notes in Computer Science. Springer, 2013, pp. 335–352.

[Sta+06] François-Xavier Standaert, Eric Peeters, Gaël Rouvroy, and Jean-Jacques Quisquater. "An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays." In: *Proc. IEEE* 94.2 (2006), pp. 383–394.

[Sta+10] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. "Leakage Resilient Cryptography in Practice." In: *Towards Hardware-Intrinsic Security*. Ed. by Ahmad-Reza Sadeghi and David Naccache. Information Security and Cryptography. Springer, 2010, pp. 99–134. DOI: 10.1007/978-3-642-14452-3_5. URL: https://doi.org/10.1007/978-3-642-14452-3_5.

[Sta16] François-Xavier Standaert. "Towards Fair and Efficient Evaluations of Leaking Cryptographic Devices - Overview of the ERC Project CRASH, Part I (Invited Talk)." In: *SPACE*. Vol. 10076. LNCS. Springer, 2016, pp. 353–362.

[Sta19] François-Xavier Standaert. "Towards and Open Approach to Secure Cryptographic Implementations (Invited Talk)." In: *EUROCRYPT I*. Vol. 11476. LNCS. 2019, xv, `https://www.youtube.com/watch?v=KdhrsuJT1sE`.

[TV03] Kris Tiri and Ingrid Verbauwhede. "Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology." In: *CHES*. Vol. 2779. LNCS. Springer, 2003, pp. 125–136.

[TV04] Kris Tiri and Ingrid Verbauwhede. "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation." In: *DATE*. IEEE Computer Society, 2004, pp. 246–251.

[UBS21] Balazs Udvarhelyi, Olivier Bronchain, and François-Xavier Standaert. "Security Analysis of Deterministic Re-keying with Masking and Shuffling: Application to ISAP." In: *COSADE*. Vol. 12910. Lecture Notes in Computer Science. Springer, 2021, pp. 168–183.

[Unt+17] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, and Robert Specht. "Dissecting Leakage Resilient PRFs with Multivariate Localized EM Attacks - A Practical Security Evaluation on FPGA." In: *COSADE*. Vol. 10348. Lecture Notes in Computer Science. Springer, 2017, pp. 34–49.

[Unt+18] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, Robert Specht, and Georg Sigl. "High-Resolution EM Attacks Against Leakage-Resilient PRFs Explained - And an Improved Construction." In: *CT-RSA*. Vol. 10808. Lecture Notes in Computer Science. Springer, 2018, pp. 413–434.

[Unt+20] Florian Unterstein, Marc Schink, Thomas Schamberger, Lars Tebelmann, Manuel Ilg, and Johann Heyszl. "Retrofitting Leakage Resilient Authenticated Encryption to Microcontrollers." In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.4 (2020), pp. 365–388.

[Vey+12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. "Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note." In: *ASIACRYPT*. Vol. 7658. Lecture Notes in Computer Science. Springer, 2012, pp. 740–757.

[YS13] Yu Yu and François-Xavier Standaert. "Practical Leakage-Resilient Pseudorandom Objects with Minimum Public Randomness." In: *CT-RSA*. Vol. 7779. LNCS. Springer, 2013, pp. 223–238.

[Yu+10] Yu Yu, François-Xavier Standaert, Olivier Pereira, and Moti Yung. "Practical leakage-resilient pseudorandom generators." In: *CCS*. Ed. by Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov. ACM, 2010, pp. 141–151. DOI: 10.1145/1866307.1866324. URL: https://doi.org/10.1145/1866307.1866324.