Technische Universität München

Lehrstuhl für Sicherheit in der Informationstechnik
an der Fakultät für Elektrotechnik und Informationstechnik

High Precision Electromagnetic Analysis of Leakage Resilient Cryptographic Constructions

Florian Unterstein

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines **Doktor-Ingenieurs (Dr.-Ing.)** genehmigten Dissertation.

Vorsitzende: Prof. Dr.-Ing Antonia Wachter-Zeh

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Georg Sigl
2. Prof. Dr. rer. nat. Frank Kargl

Die Dissertation wurde am 12.04.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 12.07.2021 angenommen.

Abstract

Information security is the foundation of today's connected digital world. Whether we think of electronic payment solutions or intellectual property and counterfeit protection, all those applications rely on secure and confidential processing of data. While the mathematical side of cryptography is well understood and there is a plethora of secure algorithms available, the implementation security against side-channel attacks is still an arms-race between designers and attackers. Advances in the quality of measurement equipment and increased computing power for post-processing forces manufacturers to invest more and more into adequate countermeasures. Since attack equipment is nowadays available for few thousand USD, such attacks not only threaten high-security smartcards, but also the low-cost devices of the Internet of Things that we interact with every day. Typically, the deployed side-channel countermeasures are designed ad-hoc and try to reduce the information leakage in the side-channel traces. This is supposed to raise the number of traces that an attacker would need to acquire for a successful attack to a number that is not feasible anymore.

Leakage resilient cryptography, in contrast, is a more formalized approach towards side-channel security. Its aim is to design algorithms that are intrinsically resilient against a certain amount of leakage and to proof their security in a certain leakage model. Ideally, this model is generic enough to cover all possible side-channel attacks and thus the established security level is independent of the concrete attack and measurement method that is used. However, such a model naturally has to be based on assumptions about the physical behavior of the device and the emanated information leakage.

The contribution of this thesis in this regard is threefold: First, we analyze if such leakage assumptions hold in practice when facing an attacker capable of high precision electromagnetic (EM) measurements. Specifically, we analyze a leakage resilient pseudo random function (LR-PRF) based on the Advanced Encryption Standard (AES) which depends on two leakage assumptions: limited data complexity and algorithmic noise from equally leaking parallel hardware.

The laboratory analysis of a field programmable gate array (FPGA) implementation of the scheme shows, that the equal leakage assumption does not hold and that the implementation is vulnerable to multivariate template attacks. We identify that in addition to the spatial resolution of the used EM probe, the high temporal resolution is also a contributing factor to the attack's success. It allows exploiting differences in the signal propagation of the different S-boxes which are caused by small imbalances in the the placement and routing. This is a significant result since these effects are hard to control on FPGA devices. It is hence unlikely that a secure implementation of the LR-PRF can be achieved on the analyzed FPGA or similar devices.

As a second contribution, we therefore propose an improved version of the LR-PRF

that uses additional key entropy to retain an acceptable security level even against high-end attacks. The additional key entropy is introduced into the internal state of the LR-PRF in chunks of 128 bits in one or multiple preprocessing steps. The number of required steps depends on the side-channel leakage of the underlying AES implementation and has to be determined through laboratory evaluation. Since the preprocessing re-uses existing hardware, the number of steps can be implemented as a run-time parameter and then be adjusted after the evaluation on the respective platform.

Finally, we demonstrate the practicality of our solution by hardening two applications against side-channel attacks in case studies: bitstream decryption for FPGA system-on-chips (FPGA SoCs) and firmware updates for microcontrollers. FPGA SoCs are powerful systems that combine a CPU and configurable hardware on one die. The hardware configuration (called bitstream) is usually stored in encrypted form in external memory and needs to be authenticated and decrypted for secure boot. Manufacturers do provide dedicated cryptographic cores for this task, but previous publications showed that they are often vulnerable to side-channel attacks. Since those cores cannot be updated, there is no straightforward mitigation available to users once a security issue is identified. By implementing our LR-PRF as part of the configurable logic, we achieve authenticated decryption of bitstreams that is fully updatable and only relies on manufacturer provided cryptography for public key signature verification. We provide a prototype implementation with full security evaluation and give clear instruction on how to port this concept to other platforms.

In the second case study, we explore firmware decryption for microcontrollers. For secure updates to devices in the field, side-channel secured authenticated decryption of firmware is mandatory. In order to generate the algorithmic noise from parallel hardware that is required for the LR-PRF, we use AES co-processors which are already present on many modern microcontrollers. This novel approach allows us to leverage features that are intrinsic to hardware implementations in a software-only implementation. By using hardware accelerators for the security critical operations we thus achieve high security levels and at the same time benefit from improved performance and a small code base.

Kurzfassung

Informationssicherheit ist das Fundament der vernetzten digitalen Welt. Egal ob elektronische Bezahlsysteme oder der (Fälschungs-)Schutz von geistigem Eigentum betrachtet werden, all diese Anwendungen verlangen nach sicherer und vertraulicher Datenverarbeitung. Während die kryptographische Sicherheit bereits gut verstanden ist und eine Vielzahl von sicheren Algorithmen zur Verfügung steht, besteht bei der Implementierungssicherheit gegen Seitenkanalangriffe weiterhin ein Wettrüsten zwischen Entwicklern und Angreifern. Die Fortschritte, die bei Messinstrumenten erzielt wurden, sowie die höhere Rechenleistung für die Nachverarbeitung zwingen die Hersteller dazu, mehr und mehr in Gegenmaßnahmen zu investieren. Da die Ausstattung für Angriffe mittlerweile auch für wenige tausend USD verfügbar ist, betrifft dies nicht mehr nur hochsichere Smartcards, sondern auch die günstigeren Geräte des Internet of Things, mit denen wir jeden Tag interagieren. Üblicherweise werden die eingesetzten Maßnahmen gegen Seitenkanalangriffe ad-hoc entworfen und versuchen, das Informationsleck (*Leakage*) in den Seitenkanalmessungen zu reduzieren. Dies soll die Anzahl der Messungen, die Angreifer für einen erfolgreichen Angriff benötigen, so weit hoch treiben, dass dieser nicht mehr praktisch durchführbar ist.

Im Gegensatz dazu ist *leakage resilient cryptography* ein stärker formalisierter Ansatz um Seitenkanalsicherheit zu erreichen. Das Ziel ist dabei, Algorithmen zu entwerfen, die inhärent robust gegenüber einem gewissen Maß an Leakage sind, und deren Sicherheit in einem Leakage Modell zu beweisen. Idealerweise ist dieses Modell so generisch, dass es alle denkbaren Seitenkanalangriffe einschließt und das erreichte Sicherheitsniveau deshalb unabhängig vom konkreten Angriff und der Messmethode ist. Jedoch muss so ein Modell notwendigerweise auf Annahmen über die physikalischen Eigenschaften des untersuchten Geräts und dessen emittierter Seitenkanalinformation basieren.

Der Beitrag dieser Dissertation umfasst diesbezüglich drei Teile: Zunächst wird untersucht, ob solche Annahmen über das Leakage in der Praxis haltbar sind, wenn Angreifer hochpräzise elektromagnetische-(EM-)Messungen anwenden. Im Detail wird eine auf dem Advanced Encryption Standard (AES) basierende, Leakage resiliente pseudozufällige Funktion (LR-PRF) analysiert, deren Seitenkanalsicherheit auf zwei Annahmen über das Leakage aufbaut: limitierte Datenkomplexität und identisches Leakage paraller Hardware.

Die Laboruntersuchung einer field programmable gate array (FPGA) Implementierung dieses Schemas zeigt, dass die Annahme über identisches Leakage verletzt wird und die Implementierung durch multivariate Template Attacken angreifbar ist. Neben der örtlichen Auflösung identifizieren wir die hohe zeitliche Auflösung der genutzten EM Sonde als wesentlichen Faktor, der zum Erfolg des Angriffs führt. Durch diese können Unterschiede in den Signallaufzeiten der S-boxen ausgenutzt werden, die durch geringfügige

Unausgewogenheiten der Platzierung und des Routings verursacht werden. Dies ist eine bedeutende Erkenntnis, da solche Effekte auf FPGAs schwer kontrollierbar sind. Es ist daher unwahrscheinlich, dass eine sichere Implementierung der LR-PRF auf dem untersuchten FPGA oder ähnlichen Elementen möglich ist.

Als zweiten Beitrag stellen wir deshalb eine verbesserte Version der LR-PRF vor, die zusätzliche Schlüssel-Entropie nutzt, um ein akzeptables Sicherheitsniveau gegenüber High-End Angriffen zu erhalten. Die zusätzliche Entropie wird in den internen Zustand der LR-PRF in Blöcken von 128 Bit und in einem oder mehreren Vorverarbeitungsschritten eingebracht. Die Anzahl der benötigten Schritte hängt von der Seitenkanalsicherheit der zugrunde liegenden AES-Implementierung ab und muss durch Laboruntersuchungen bestimmt werden. Da diese Vorverarbeitung bereits bestehende Hardwareteile nutzt, kann die Anzahl der Schritte als Laufzeitparameter implementiert werden und nach einer Untersuchung der jeweiligen Plattform konfiguriert werden.

Abschließend demonstrieren wir die Praxistauglichkeit unserer Lösung, indem wir zwei Anwendungen in Fallstudien gegen Seitenkanalangriffe härten: Bitstream Entschlüsselung auf FPGA System-on-Chips (FPGA SoCs) und Firmware Updates für Mikrocontroller. FPGA SoCs sind leistungsfähige Systeme, die eine CPU mit konfigurierbarer Hardware auf einem Chip vereinen. Die Hardwarekonfiguration (Bitstream genannt) wird üblicherweise in verschlüsselter Form in externen Speichern abgelegt und muss für Secure Boot authentifiziert und entschlüsselt werden. Die Hersteller bieten zwar dedizierte kryptografische Beschleuniger für diese Aufgabe an, frühere Veröffentlichungen haben aber gezeigt, dass diese oft anfällig für Seitenkanalangriffe sind. Da diese Beschleuniger nicht aktualisiert werden können, gibt es für den Benutzer keine klare Lösung, falls ein Sicherheitsproblem erkannt wird. Durch die Implementierung unserer LR-PRF als Teil der konfigurierbaren Logik erreichen wir eine authentifizierte Entschlüsselung des Bitstreams, die vollständig aktualisierbar ist. Dadurch wird vom Hersteller bereitgestellte Kryptographie lediglich für die Verifikation von Signaturen mit öffentlichen Schlüsseln benötigt. Wir stellen eine prototypische Implementierung mit vollständiger Sicherheitsevaluierung vor und beschreiben, wie dieses Konzept auf andere Plattformen portiert werden kann.

In der zweiten Fallstudie wird die Entschlüsselung von Mikrocontroller Firmware untersucht. Damit sichere Updates für Geräte im Feld möglich sind, ist eine seitenkanalgesicherte, authentifizierte Entschlüsselung der Firmware zwingend erforderlich. Um das für die LR-PRF benötigte algorithmische Rauschen aus paralleler Hardware zu erzeugen, verwenden wir AES-Beschleuniger, die ohnehin auf vielen modernen Mikrocontrollern vorhanden sind. Dieser neue Ansatz erlaubt es, Eigenschaften, die eigentlich intrinsisch für Hardware Implementierungen sind, in einer reinen Softwarelösung auszunutzen. Durch den Einsatz von Hardware-Beschleunigern für die sicherheitskritischen Operationen erreichen wir somit ein hohes Sicherheitsniveau und profitieren gleichzeitig von verbesserter Leistung und einer kleinen Codebasis.

Acknowledgement

First of all I want to thank my supervisor Prof. Dr.-Ing. Georg Sigl for giving me the chance to pursue a PhD at Technische Universität München (TUM).

My special thanks go to Dr.-Ing. Johann Heyszl for the scientific supervision, fruitful discussions and overall support and motivation.

Last but not least, I thank my colleagues at Fraunhofer AISEC, the Chair of Security in Information Technology at TUM and all paper coauthors that contributed to this thesis through their valuable input, friendship and support: Dr. Chongyan Gu, Neil Hanley, Dr.-Ing. Matthias Hiller, Robert Hesselbarth, Stefan Hristozov, Manuel Ilg, Dr.-Ing. Nisha Jacob Kabakci, Dr.-Ing. Philipp Koppermann, Dr.-Ing. Michael Pehl, Carsten Rolfes, Dr.-Ing. Fabrizio De Santis, Thomas Schamberger, Marc Schink, Bodo Selmke, Dr.-Ing. Robert Specht, Silvan Streit, Emanuele Strieder, Martin Striegel, Lars Tebelmann, Alexander Wagner and Andreas Zankl.

List of Abbreviations

AAD	additional authenticated data
AEAD	authenticated encryption with associated data
AES	Advanced Encryption Standard
AES-128	128-bit key AES
ASIC	application specific integrated circuit
AUX	auxiliary input model
BBRAM	battery-backed RAM
BER	bit error rate
BGA	ball grid array
BML	bounded memory leakage
BRAM	block RAM
BRM	bounded retrieval model
CML	continual memory leakage
COTS	commercial off-the-shelf
CPA	correlation power analysis
DPA	differential power analysis
DUT	device under test
DWT	data watchpoint and trace
ECB	electronic codebook
EFM32	EFM32PG12B500F1024
EM	electromagnetic
FIB	focused ion beam
FIFO	first in, first out
FPGA	field programmable gate array
FPGA SoC	FPGA system-on-chip
FSBL	first stage bootloader
GCM	Galois/Counter mode
GMAC	Galois message authentication code
IC	integrated circuit
ICAP	internal configuration access port
IoT	Internet of Things
IP	intellectual property
IQR	interquartile range

IV	initialization vector
LDA	linear discriminant analysis
LOI	location of interest
LR-AEAD	leakage resilient AEAD
LR-PRF	leakage resilient pseudo random function
LR-PRG	leakage resilient pseudorandom generator
MAC	message authentication code
MIA	mutual information analysis
NVM	non-volatile memory
OCL	only computation leaks
OCM	on-chip memory
OFB	output feedback
PCA	principal component analysis
POI	point of interest
PR	partial reconfiguration
PRF	pseudorandom function
PRG	pseudorandom generator
PUF	physical unclonable function
SCA	side-channel analysis
SNR	signal-to-noise ratio
SPA	simple power analysis
STM32	STM32F215RET6
TA	template attack

List of figures

List of tables

Contents

1. Introduction and research questions

The proliferation of smart devices and the ubiquitousness of networked devices in applications like automotive or industry 4.0 make information security more crucial than ever. Considering the track record of such security sensitive devices, the classical "cryptographic security", i.e. the resistance to cryptanalysis, is well studied and usually not the failing point. However, since the scenarios in which these devices are deployed allow attackers direct access to the device, the implementation of these algorithms needs to resist physical attacks. A lack of implementation security, unfortunately, regularly leads to vulnerabilities and hacks of commercial products using side-channel attacks. Side-channel attacks comprise all attacks that extract information about a processed secret over a *side channel*, such as the power consumption, electromagnetic (EM) emanation, and timing behavior. Such attacks still regularly cause security incidents despite two decades of research into side-channel analysis (SCA) and countermeasures. These incidents can be explained by either a negligence of implementation security, e.g. due to budget or time-to-market constraints, or the error-prone implementation of (insufficient) countermeasures.

There are several common approaches to reduce the information that can be extracted over these channels, the so called *side-channel leakage*: Obfuscating sensitive values with random values (masking), randomizing the order of computation during the execution of the algorithm and interleaving it with dummy operations (shuffling/hiding), or balancing the power consumption by processing each sensitive value in parallel to its inverse (dual-rail implementations). These countermeasures require specialized hardware and high quality random numbers, which can be found on security controllers, but are unlikely to be designed into commercial off-the-shelf (COTS) microcontrollers. This leaves a large class of devices without hardware supported countermeasures. Even on devices with configurable hardware like field programmable gate array (FPGA) devices, designers are limited due to the given layout of the internal building blocks of the device. This makes it notoriously difficult to implement secure masked or dual-rail implementations and requires a designer with experience in side-channel analysis. One method to mitigate this problem is to focus on countermeasures that work on an algorithmic level and place very little requirements on the hardware.

Certain concepts from the field of *leakage resilient cryptography* fit that niche as they only require that the underlying hardware does not "excessively" leak information about the secret during a single execution. This assumption does have its caveats which will be thoroughly discussed in this work, but in fact the same requirements apply to any cryptographic implementation: if a given platform leaks the entire secret within one execution, then there is nothing we can do to fix it. Without randomization, we can only prevent attackers from accumulating and combining information over multiple

executions.

Originally, leakage resilient cryptography gained popularity among the cryptographic community as a more formalized approach in contrast to the the rather ad-hoc nature of existing countermeasures. The goal was to anchor implementation security on the basis of a formal model of a device's leakage and to derive security guarantees without making too many assumptions about the hardware. This has the advantage that, given the model is sound, a tight security level can be derived. In contrast, for other countermeasures the achievable security level is often not clear and only defined in the number of traces that an attacker requires to break the device. As this is highly dependent on the measurement equipment and the actual device, drawing general conclusions about the side-channel security of an implementation is hard.

Unfortunately, some of the proposed solutions in the field of leakage resilient cryptography are technically provable secure, but the implementation is cumbersome or based on non-standardized cryptographic primitives that are not common in off-the-shelf hardware. The intent of this work is to bridge the gap between the theory (leakage models and assumptions about the hardware) and practice (the evaluation of the security level of an implementation when facing a state-of-the-art attacker). Therefore it focuses on constructions which can be instantiated with a common cryptographic primitive, namely the Advanced Encryption Standard (AES) block cipher.

The target of the analysis are pseudorandom functions (PRFs), which are an important building block in many cryptographic protocols and enable, e.g., symmetric encryption or message authentication. Specifically we start by analyzing a construction that has first been proposed at the Conference on Cryptographic Hardware and Embedded Systems (CHES) in 2012 [65]. The security of the proposed leakage resilient pseudo random function (LR-PRF) is based on limited data complexity, a widespread principle in leakage resilient cryptography, and a novel method named "carefully chosen inputs". The data complexity denotes the number of *different* operations that a secret key is used in. In general, differential side-channel attacks require the attacker to observe many different operations to succeed, which is prevented when the data complexity is limited. Additionally, side-channel attacks typically divide the secret key in smaller parts which are computationally manageable, e.g., bytes, and attack them separately. This is achieved by exploiting the relation of key bytes to known inputs such as plaintext bytes. Using "carefully chosen inputs" means to construct the known inputs to the critical operation in a way that generates correlated noise from parallel hardware to hinder such divide-and-conquer attacks. In practice, these inputs are constructed such that all bytes have the same value. That makes it impossible for the attacker to link a (secret) key byte to a specific (known) input byte since all input bytes are equal. If in addition all key bytes are processed simultaneously in parallel hardware, then an attacker that observes the power consumption of the device cannot separate the contributions of the individual key bytes and consequently divide-and-conquer is not possible. In this thesis we deal with AES-based constructions and the critical function blocks that need to be carefully implemented in parallel hardware are the S-boxes.

The crux of this concept is that it assumes that the measured leakage of all S-boxes is perfectly synchronous and that the leakage function, i.e., the mapping of the computed

operation to the observed leakage, is identical. If there are differences in the timing behavior or leakage function then this can be exploited to separate the leakage of the S-boxes and once again mount divide-and-conquer attacks. Initially, it is unclear if these assumptions hold in face of high precision EM measurements, which are generally capable of isolating parts of a circuit in a measurement. Also, designing equally leaking S-boxes on configurable hardware is, at least, challenging due to limited control over the internal routing and imbalances in wiring caused by different physical locations on the die. Starting with these observations, the contribution of this thesis is threefold:

1. We evaluate, if the leakage assumptions and consequently the security of such constructions hold, even when facing adversaries with state of the art, high precision EM measurement equipment.

2. After finding that the security is insufficient, we analyze, what effects enable such measurements to break the implementation security and derive an improved construction secure even against such attacks.

3. Finally, we transfer our results into practice and give two case studies where we deploy the improved LR-PRF in applications for FPGA system-on-chips (FPGA SoCs) and on COTS microcontrollers where we re-use existing hardware accelerators.

Outline Chapter 2 gives an overview of the methodology used for SCA and key recovery attacks. It also introduces the electromagnetic side-channel and discusses its properties. The state of the art in leakage resilient cryptography is discussed in Chapter 3 where we give an overview of the different approaches towards leakage resilient cryptography and evaluate their relevance for real applications. It introduces LR-PRFs and the fundamental Goldreich, Goldwasser, Micali construction that is used to build random functions from a pseudorandom generator (PRG). Finally, we discuss the CHES 2012 LR-PRF [65] and its improvement that was presented at the ASIACRYPT 2016 conference [66] which are the starting points for our analysis.

In Chapter 4 we give results of a full EM analysis of the implementation of that scheme on a Xilinx Spartan-6 FPGA. We successfully mount a multi-variate template attack in order to recover the secret key with brute force effort of a level that is computationally feasible with current hardware. This establishes that the security of this LR-PRF implementation is insufficient against high precision EM analysis.

Following up, Chapter 5 analyzes which effects lead to the successful attack. We compare attacks using power measurements and EM measurements and conclude that for the case of power measurements the countermeasure is effective. For EM measurements however, we find that not only the spatial resolution and the isolation of parts of the circuit is enabling the attack, but also the high temporal resolution. This is one of the major findings of this work, because it has drastic consequences for hardware designers. While spatial resolution can be countered with tighter designs and smaller feature sizes, it seems much harder to control timing differences in the internal signal propagation. To derive design recommendations for an improved construction, we also revisit the "equally

leaking S-boxes" assumption originally made in [65]. To quantify the relevance of the S-boxes' leakage functions we run simulated template attacks on a single key byte and vary the leakage functions. We find, that the leakage function is almost irrelevant for cases with very low data complexity which means that in such cases the S-boxes do not have to be designed to leak equally. This enables tighter integration of the hardware circuit and leads to a higher security level.

We make use of these insights in Chapter 6 and propose an improvement of the original construction that withstands even high precision EM attacks. This improved construction incorporates a mechanism to "refill" key entropy which was lost due to side-channel attacks by introducing additional key material in a particular way. We demonstrate the relevance for real applications in Chapter 7 by providing a solution for secure and updatable bitstream encryption for FPGA SoCs. The bitstream is encrypted and authenticated by combining our improved LR-PRF with AES in output feedback (OFB) mode of operation and Galois message authentication code (GMAC) authentication. The key is generated by a physical unclonable function (PUF) during runtime and thus no secure key storage is required. This makes the proposed design almost completely independent from manufacturer provided security mechanisms and has the additional benefit that all parts, even the decryption engine itself, can be updated. Chapter 8 pursues the idea of using cryptographic accelerators which are already present in many COTS microcontrollers, but are not protected against side-channel attacks, to construct an LR-PRF. By means of two example devices, we evaluate the side-channel security of LR-PRF when implemented around existing hardware accelerators. We find that it is feasible to build efficient and secure constructions, which enables side-channel protection for these devices with very little overhead in code size.

Finally, Chapter 9 concludes our findings. Drawing from the experience that was gathered when implementing the two use-cases, we discuss the benefits and difficulties that arise when integrating leakage resilient primitives into actual applications.

Previous publications This thesis is in parts based on research papers that were previously published at peer-reviewed conferences and in collaboration with the respective co-authors. Specifically, Chapters 4,5 and 6 are based on results published in [100] and [101]. Chapter 7 is based on [102] and [103], and Chapter 8 on [104].

2. Background on side-channel analysis

Side-channel attacks are one of the most prevalent threats to information security, especially in the context of embedded systems and the Internet of Things (IoT) where attackers often have physical access. Side-channel attacks comprise all attacks that observe some characteristic of the device under attack and then derive information about an internal secret, either directly or by applying statistical methods. These so called implementation attacks exploit specifics of the implementation of a cryptographic algorithm, rather than cryptographic weaknesses of the algorithm itself. Possible side-channels include the power consumption [52], timing behavior [51], EM emanation [30] or photonic emission [27]. These attack vectors motivated the use of the so called gray-box model for cryptanalysis (Fig. 2.1). In contrast to the traditional black box model, that gives the attacker knowledge over the in- and outputs, the gray-box model additionally provides the attacker with a leakage function that gives information about the internals of the algorithm. This work focuses on power measurements and high precision EM measurements, specifically of CMOS devices, to capture and evaluate the leakage function.

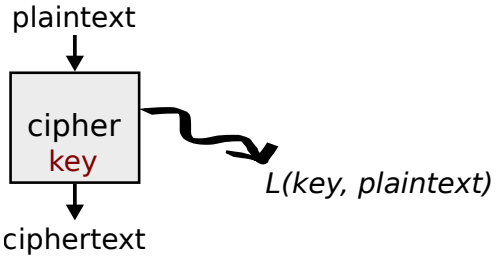


Figure 2.1.: Analysis of a cipher under the gray-box model.

In this chapter, we first provide a categorization of the different types of side-channel attacks and their prerequisites in Section 2.1. We discuss the inherent properties of the EM side-channel in Section 2.2. Typically, a side-channel evaluation consists of three steps:

1. Identifying leaking operations.
2. Mounting the most powerful attack available on intermediate values.
3. Assessing the remaining security level.

We describe the necessary tools and the methodology for these steps in Sections 2.3 through 2.5. Finally, the most common countermeasures are introduced in Section 2.6.

Nomenclature In the following equations, lowercase letters denote vectors or scalars, depending on context. Matrices are written in bold uppercase letters, constants in regular uppercase. Calligraphic uppercase letters denote sets.

2.1. Categorization of side-channel attacks

Passive and active attacks Side-channel attacks can be broadly divided into two categories, namely passive and active attacks. Passive attacks only observe the device while it is operating under normal conditions, whereas active attacks also influence the behavior of the device, e.g. by manipulating its operating frequency or by inducing current into the circuit using laser beams. In this work only passive attacks are considered. However, it is in scope for a passive attacker to remove the package of an integrated circuit (IC) in order to be able to take measurements closer to the die surface. While this requires grinding down the package and removing the residue with acid, modern ICs usually also can be bought in a 'naked' flip-chip package. In that case the backside of the die is directly accessible and no preparation is required. This is why allowing the removal of the package is reasonable when considering a worst-case analysis of the security level, even though the attacker is in general limited to passive attacks.

Simple and differential attacks Attacks can be further classified as simple power analysis (SPA) attacks or differential attacks which can both be either profiled or unprofiled. With SPA attacks, the secret can be directly derived from observing the side-channel [51, 58, 73]. Contrary to SPA attacks, differential attacks recover only a part of the secret with each observation. Usually many observations, so called *traces*, with different inputs are necessary to fully recover the secret. Attacks that fall into this category include the classic differential power analysis (DPA) [52], correlation power analysis (CPA) [10], and mutual information analysis (MIA) [32] attacks. The number of traces with different inputs, but the same secret (e.g. encryptions of different plaintexts under the same key), that an attacker can observe is called the *data complexity*. Simple attacks thus work with data complexity 1, differential attacks with higher data complexity. In both cases an attacker can usually repeat the observation(s) *with the same inputs* to reduce measurement noise.

Profiled and unprofiled attacks Profiled attacks such as Gaussian template attacks [15] and the linear regression based stochastic approach [83] require the attacker to have full control over a device during a profiling stage. During this profiling, the leakage behavior of the device for all realizations of the secret value is characterized. Then this knowledge is used to attack an unknown secret on a different device by comparing the traces to those profiles. Unprofiled attacks require no a-priori classification of the exact leakage behavior. Instead, a leakage model is used that connects the processed value with the side-channel observation. In case of e.g. CPA, the leakage model is chosen explicitly by selecting a leakage function, such as the Hamming weight, for the intermediate value that is attacked. In other attacks, e.g. MIA, only the targeted internal value

is specified and no leakage function is defined. During the attack, an ad-hoc leakage characterization of the traces is performed. Profiled attacks are the most powerful kind of attacks since they most accurately capture the leakage behavior of the device and thus are able to extract the most information from the observed traces. Gaussian templates are the most generic approach and the optimal attack if the noise is actually Gaussian, as observed in most practical cases. The stochastic approach, in comparison, requires the evaluator to define a set of equations that model the leakage and then fits the weight coefficients which determine how much they influence the overall leakage. This reduces the parameter space and consequently the sampling complexity but the quality of the evaluation greatly depends on the evaluators capability to chose an adequate leakage model. Both approaches can either exploit the leakage of a single or multiple points in time, leading to univariate and multivariate attacks, respectively.

To achieve a worst-case analysis of the security level, we use multivariate template attacks in this work for the evaluation of the side-channel security. They are introduced in Section 2.4. The preprocessing steps to identify points of interest (POIs) and to reduce the dimensionality are discussed in Section 2.3. In cases where the key cannot be recovered fully, it is necessary to estimate the remaining brute force effort which is discussed in Section 2.5. This chapter concludes with an overview of countermeasures against SCA attacks in Section 2.6.

2.2. The EM side-channel

The EM side-channel offers some unique properties compared to the power side-channel. To discuss the differences, it is helpful to first understand what causes the leakage on CMOS devices. CMOS logic dissipates the most power when there is switching activity and a cell changes its state. This causes the load capacitances on the output to be charged or discharged (dynamic power consumption). The static power consumption is small in comparison[1] and consists of leakage currents that lead to some power loss of the transistors even when there is no switching activity. This part of the overall power consumption is not data dependent and therefore only contributes to the background noise of the measurements. In digital logic, the design is clocked and consists of registers with combinational logic in between (Fig. 2.2). On a rising clock edge, the registers store the value that is present at their inputs and output the new values. The new value then propagates via multiple paths through the combinational logic which usually consists of multiple layers of logic gates. After the time duration equal to the critical path delay, i.e. the delay of the longest possible path through the combinational logic, all signals settle and the circuit remains idle until the next clock edge. This period of activity is smaller than the time between two clock edges, which means that changes in power consumption and leakage can be observed only directly after a clock edge. The signal propagation and the toggling frequencies of the logic gates is independent from the clock frequency and depends only on the manufacturing technology.

[1] However, with shrinking feature sizes and manufacturing technology going in the deep sub-micron range, the static power consumption becomes substantial.

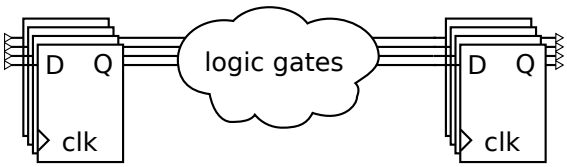


Figure 2.2.: Registers connected with combinational logic.

Power measurements are usually taken using a differential probe and a shunt resistor that is placed between the supply voltage rail and the device under test (DUT). They capture the power consumption of the entire device which can lead to significant background noise. The measurements are also affected by parasitic capacitances, both in the supply lane and within the DUT, which is why the measured power consumption is a low-pass filtered version of the currents that are caused by the chip internal switching activity. Global EM measurements can be taken with large coils that either cover the entire chip or smaller coils that are placed over a decoupling capacitor. Those EM measurement yield results that are comparable to power measurements via shunt resistors. High precision EM measurements, also called localized EM measurements, use near-field probes with coil diameters that are smaller than 1 mm and go down to 100 µm (Fig. 2.3).

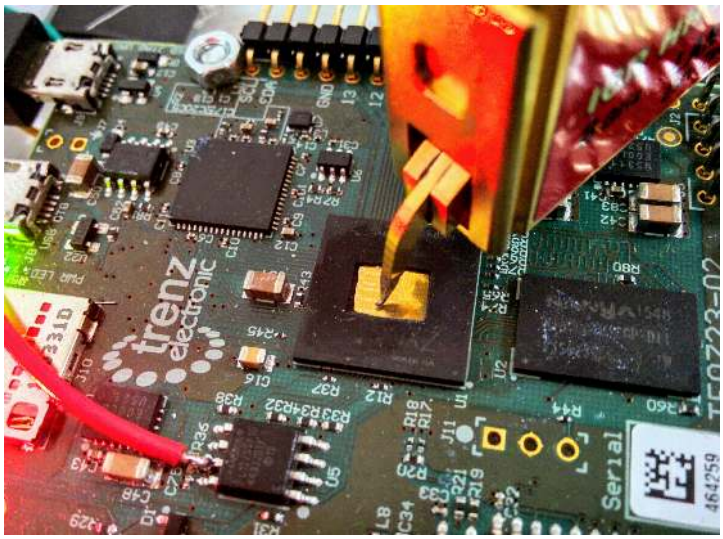


Figure 2.3.: High precision near-field EM probe positioned over a decapsulated chip.

With these probes it is possible to isolate parts of the circuit by placing them over the respective region of the die, this is the *spatial isolation* aspect. As the measurements are less affected by parasitic capacitances and the probes have a frequency range going up to 6 GHz, the temporal resolution is also much higher compared to power measurements. This allows *temporal isolation* of signals that would be interfering with each other when using low bandwidth measurements, a topic that will be discussed in greater detail in Chapter 5. To achieve a high signal-to-noise ratio (SNR), the distance to the target

signal is crucial [88]. If the chip comes in a package, the package therefore has to be removed. Measurements can be taken from the front or backside. On the frontside, the probe is closest to the metal layers on top, on the backside it is closer to the logic gates and the lower metal layers. The substrate can be thinned to further reduce the distance when measuring from the backside. It is hard to predetermine where leaking circuit parts are located and which side works best, but often times the packaging of the chip decides which side is accessible. For ball grid array (BGA) packaging it is e.g. not possible to measure from the frontside as it is blocked by the solder balls. When conducting a high precision EM analysis, the most time consuming part is finding the best probe location over the die. Often the chip layout is unknown, but even when it is known it is not trivial to determine the optimal measurement location. Therefore grid scans and leakage tests are used to find the optimal location for the targeted signal. This is discussed in the next section.

2.3. Leakage detection and of locations/points of interest

In case of EM measurements, the location has a high influence on the quality of the analysis. Therefore, a specific measurement location is selected for each targeted signal. The selection of those locations of interest (LOIs) can be done using leakage tests based on different metrics. This work differentiates between time and location by using the term POIs for time samples and LOIs to refer to the placement of the probe. Technically, leakage detection and the selection of POIs are tasks with different goals. Leakage detection is used in security evaluations and refers to identifying any data-dependent leakage in measurements, disregarding if this leakage can be exploited in an attack. The goal is to provide sufficient proof for the absence of leakage. The selection of POIs on the other hand is a preprocessing step for an actual side-channel attack. In this scenario, the leakage tests should be tailored to be sensitive to the same kind of leakage that will be used in the attack. However, it is common practice to use leakage detection tests as a first step to find LOIs and reduce the search space and then more specific leakage tests for the selection of POIs. For leakage detection, the t-test recently has gained a lot of attention from industry [33] and academia [85, 89, 62] and is becoming the de facto standard tool. Common tests for the task of POI selection are the SNR [60] of the targeted signal and the correlation-based leakage test described by Durvaux and Standaert [23] (in the following called CPOI). Others use a mutual information based distinguisher [32] or are based on linear regression of a leakage model [49].

At the start of an evaluation, little is known about the leakage characteristics of the device. Yet, some assumptions need to be made about the leakage when designing a leakage test. All tests partition the traces in subsets based on the value of some internal state variable. Which variable is chosen for this partitioning determines what kind of leakage is targeted by the test. More specific tests like the SNR and CPOI partition into many sets, e.g. one set for all values of a byte at an S-box output in case of AES.

Broader tests such as the t-test partition only in two sets, e.g. one with fixed inputs and the other with random inputs. The advantage of the t-test is that it has a low sampling complexity as it only needs to estimate parameters for two sets of traces. It also covers more possible leakage sources, thus reducing the risk of not detecting leakage due to an inadequate target signal. A positive t-test however does not always mean that the leakage is exploitable. The SNR and CPOI correlation coefficients, conversely, directly translate into the success rate of an actual attack. The more specific tests also require more traces for a meaningful result which can lead to prohibitive trace acquisition times. A general strategy is therefore to conduct grid scans using tests that require less traces, like the t-test, in order to keep measurement times low and then move towards specific tests in the course of the analysis. After evaluating the leakage of a device and determining the LOIs, the time samples can also be reduced to only include time points with significant leakage. POIs can be selected by cutting out all samples for which the leakage exceeds a certain threshold or by using dimensionality reduction algorithms like principal component analysis (PCA) [75] and linear discriminant analysis (LDA) [28]. These algorithms transform the trace into a subspace with lower dimensionality while retaining the leakage. In the transformed subspace, the leakage is concentrated in the first dimensions and thus the later dimensions can be ignored. Reducing the dimensionality speeds up all following processing steps and reduces the amount of data that needs to be stored. In the following we introduce the techniques which are used in this thesis: the SNR, CPOI, t-test and LDA.

2.3.1. Signal to noise ratio

We use the definition of the SNR used by Mangard et al. [60] to quantify the exploitable signal for an SCA. To compute the SNR over time (called the SNR trace) of a target variable that takes values in $0, \ldots, c-1$, the traces are partitioned according to the values of this variable and the SNR is computed with the estimated mean trace μ_i and variance trace σ_i^2 over all traces with value i:

$$SNR = \frac{Var(Signal)}{Var(Noise)} = \frac{Var(\mu_0, \ldots, \mu_{c-1})}{Mean(\sigma_0^2, \ldots, \sigma_{c-1}^2)}. \tag{2.1}$$

The SNR can be calculated if the evaluator has control over the inputs of the device and knows all relevant internal values. The target value and how the inputs are chosen has an effect on the outcome of the test, as there might be bijections of the target value that lead to the same trace partitioning. Considering AES S-box inputs as example, then the value of the S-box input is the XOR combination of the corresponding plaintext and key byte. Ideally, both the key and the plaintext are randomized during the collection of the traces. If this is not possible, e.g. because the key is immutable, then all possible inputs to the S-box can still be generated by randomizing only the plaintext. In that case, however, there is a bijection between the plaintext byte and the S-box input. The leakage test will then also be sensitive to all operations which are dependent on the plaintext byte alone. These effects can be observed with all similar leakage tests and should be kept in mind in order to correctly interpret the results.

2.3.2. Correlation based leakage test

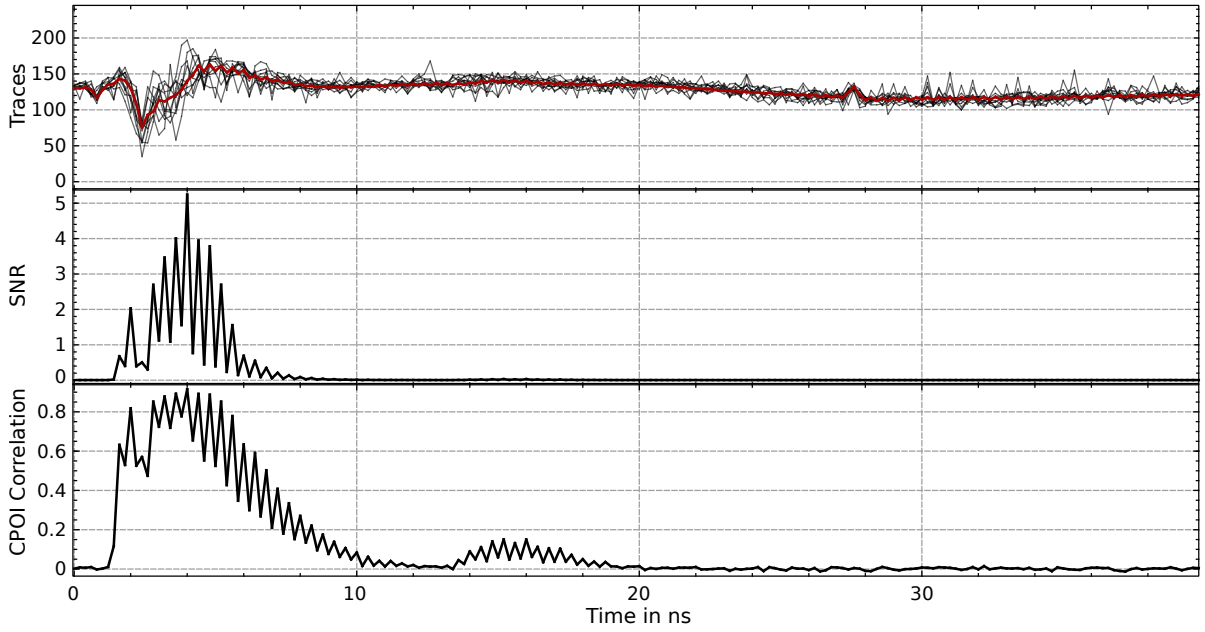


Figure 2.4.: Top: EM Traces for AES round 1, example traces in grey and mean trace in red. Middle and bottom: SNR and CPOI correlation for S-box 0 output.

The correlation based leakage test was put forward by Durvaux et al. [23] and has similar prerequisites and sampling complexity as the SNR test. It was originally published as a black box test that works without knowing the key by exploiting the bijections between plaintext bytes and internal, key-dependent values. Like the SNR test it can, however, be applied to arbitrary internal values if the key is known to the evaluator. The test uses a Pearson correlation based distinguisher with k-fold cross validation. The procedure is very similar to a profiled CPA with the difference that the values of the target variable are known and no key guesses are necessary. The trace set $\mathcal{T}$ is split into k disjoint sets $\mathcal{T}_u$, $u \in [0, k-1]$, of equal size. One set is chosen as the test set, the others form the profiling set:

$$\begin{aligned} \mathcal{T}_{test} &= \mathcal{T}_u \\ \mathcal{T}_{profiling} &= \bigcup_{v|v\in[0,k-1],v\neq u} \mathcal{T}_v \end{aligned} \tag{2.2}$$

This is repeated for all values of u, and for each of these k-possible divisions the following steps are computed:

First, a leakage model is estimated from $\mathcal{T}_{profiling}$. For each possible value of the target variable, the mean trace over all traces with this value in the profiling set is calculated. Let $t_{i,j}$ be all traces with target value $i \in [0, c-1]$ and index $j \in [0, N_i - 1]$ where N_i denotes the number of traces with target value i in $\mathcal{T}_{profiling}$. The leakage model m then

consists of all c mean vectors μ_i and the overall mean $\overline{\mu}$:

$$\begin{aligned} \mu_i &= \frac{1}{N_i} \sum_{j=0}^{N_i-1} t_{i,j} \text{ with } t_{i,j} \in \mathcal{T}_{profiling} \\ \overline{\mu} &= \frac{1}{\sum_{i=0}^{c-1} N_i} \sum_{i,j} t_{i,j} \text{ with } t_{i,j} \in \mathcal{T}_{profiling} \end{aligned} \tag{2.3}$$

Second, the sample Pearson's correlation between the model m and the traces $t \in \mathcal{T}_{test}$ is calculated. We denote the N_t traces in $\mathcal{T}_{test}$ with t^u with $u \in [0, N_t - 1]$ and the mean trace over the test set with $\bar{t}$. Each trace in the test set is correlated with the profiled mean trace of the corresponding target value[2]. We denote these pairs $(t^0, \mu^0), \ldots, (t^{N_t-1}, \mu^{N_t-1})$. If for example trace t^0 was measured with the target value being k, then it is correlated with μ_k and $\mu^0 = \mu_k$.

The correlation trace is defined as:

$$r_{tm} = \frac{\sum_{i=0}^{N_t-1}(t^i - \bar{t})(\mu^i - \overline{\mu})}{\sqrt{\sum_{i=0}^{N_t-1}(t^i - \bar{t})^2}\sqrt{\sum_{i=0}^{N_t-1}(\mu^i - \overline{\mu})^2}} \tag{2.4}$$

Finally, the correlation traces of all k cross validation sets are averaged to get an unbiased result.

Figure 2.4 gives an example of the SNR and CPOI correlation traces for the output of S-box 0 during the first AES round in case of EM measurements. On top, 10 example EM traces are shown in gray and the mean trace over all traces in red. The significant dip around sample 10 corresponds to the rising clock edge that triggers the calculation of the AES round. For around 30 samples activity in the EM traces can be observed, after that the traces settle down and do not show any more significant features. This time period is when the combinational logic is evaluated and wires inside the hardware are switching. The middle and lower subplots show the SNR and CPOI correlation traces, respectively. They are similar in shape and show leakage at the same points in time. While the value of the SNR is unbounded, the correlation is limited to values between -1 and 1 and almost maxes out with value 0.91 at sample 20. The CPOI shows a second region with leakage from sample 70 to 90 which is hard to spot in the SNR plot due to the axis scaling. In fact, the SNR trace also exhibits a small peak of around 0.03 in that region. We conclude that both leakage tests are generally capable of detecting POIs.

2.3.3. t-Test

Welsh's t-test [107] is a statistical test that checks the null hypotheses that two sets (populations) have equal mean (i.e., the difference between them is zero). It was proposed in the context of side-channel leakage detection by Goodwill et al. [33] and rapidly gained

[2]For comparison, in a (profiled) CPA the correlation with the mean traces of all key candidates would be calculated. However, since the key is known during evaluation, in this scenario only the correct mean value is considered.

popularity due to its simple design and low sampling complexity. The test statistic is defined as follows:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{\sigma_0^2}{N_0} + \frac{\sigma_1^2}{N_1}}} \tag{2.5}$$

where N_0 and N_1 are the number of traces for the respective sets. As a rule of thumb, a threshold of $|t| > 4.5$ is used to reject the null hypotheses with a confidence > 0.99999. The actual value of t however has no significance with respect to the quantification of leakage and grows with the number of traces used (just as the confidence grows). This is in contrast to the SNR and CPOI where the results directly translate into success rates of a side-channel attack and can be used to compare implementations. Tests can be conducted as fixed vs. random (non-specific t-test) or fixed vs. fixed tests (specific t-test). In a fixed vs. random test, the target value, e.g. the key, is static in one set and randomized in the other. Fixed vs. fixed compares trace sets that were acquired with two static values. The latter test has been shown to require even less traces [23]. If by chance the two sets exhibit the same leakage behavior (and consequently the same mean trace), e.g. if they have the same Hamming weight, then the t-test would show no leakage. To mitigate this, multiple fixed values should be tested. There can also be false positives when data-dependent leakage is detected, but cannot feasibly be exploited in an attack. One such case arises if leakage is detected that depends on the entire key or, considering AES or similar round-based block ciphers, if leakage is present only after a couple of rounds where the intermediate values are already dependent on too many key bits. These issues were mentioned in the original proposal and are summarized and discussed in detail by Standaert [89]. Nevertheless, its sensitivity to all manifestations of data-dependent first order leakage without being restricted by a chosen leakage model and the low sampling complexity make it a good choice to begin an evaluation.

2.3.4. Dimensionality reduction

A common way to decrease computational effort and simplify the analysis is to reduce the number of time-samples in a trace, i.e. the dimensionality of the trace. A straightforward approach is to drop all time samples for which the detected leakage is below a threshold set by the evaluator. This inevitably leads to some loss of information. Bruneau et al. [12] showed that the ideal compression is a linear combination of the time samples and that both PCA and LDA are close to the optimal solution in practical scenarios. Those methods transform the traces into a lower dimensionality subspace such that the information is compressed in fewer samples, i.e. the sample-wise SNR is higher. This allows for dimensionality reduction without significantly reducing the success probability of side-channel attacks.

PCA is an unsupervised technique that requires no knowledge about any internals of the data. It finds principal components in the directions that explain the most variance in the data. In the case of side-channel measurements, these directions may not always be the desired ones, especially if the traces are noisy and most of the variance is caused by noise.

Fishers's LDA [28] on the other hand is a supervised method and requires a profiling phase where the traces are partitioned according to the value of the target variable. When aiming for a worst-case evaluation of the security of constructions, the focus is therefore on LDA because it is guaranteed to give the optimal transformation with regard to the targeted leakage.

LDA has been proposed for dimensionality reduction in the context of side-channel attacks by Archambeau et al. [4] and for EM measurements in particular by Standaert et al. [90]. It has later been shown by Bruneau et al. [12] that this is in fact the optimal strategy to reduce the dimensionality of leakage traces. LDA also has the advantage that the transformation makes template attacks more robust against measurement campaign-dependent variations caused by temperature or environmental noise [17].

LDA stems from statistical classification and is a linear transformation of a dataset onto a lower-dimensional subspace with good class-separability. It calculates a transformation matrix $\mathbf{W}$, which maximizes the ratio of between-class to within-class scatter. As introduced before, let the target variable have c values, or classes, with range $0, \ldots, c-1$. Let $t_{i,j}$ be all traces with target value i and $j \in [0, N_i - 1]$ where N_i denotes the number of traces with target value i. Further, $\mu_i = \frac{1}{N_i} \sum_{j=0}^{N_i-1} t_{i,j}$ is the estimated class mean vector and $\mu = \frac{1}{c} \sum_{i=0}^{c-1} \mu_i$ the estimated overall mean vector. Then LDA calculates the within-class scatter matrix $\mathbf{S_w}$, between class scatter matrix $\mathbf{S_b}$ and $\mathbf{W}$, such that criterion J is maximized.

$$\mathbf{S_w} = \sum_{i=0}^{c-1} \sum_{j=0}^{N_i-1} (t_{i,j} - \mu_i)(t_{i,j} - \mu_i)^T \tag{2.6}$$

$$\mathbf{S_b} = \sum_{i=0}^{c-1} N_i (\mu_i - \mu)(\mu_i - \mu)^T \tag{2.7}$$

$$J(\mathbf{W}) = \frac{\mathbf{W}^T \mathbf{S_b} \mathbf{W}}{\mathbf{W}^T \mathbf{S_w} \mathbf{W}} \tag{2.8}$$

It can be shown that the solution that maximizes J are the eigenvectors of ${\mathbf{S_w}}^{-1}\mathbf{S_b}$. The columns of $\mathbf{W}$ are the eigenvectors, sorted by descending eigenvalue. The within-class scatter matrix $\mathbf{S_w}$ is asymptotically equal to the *pooled* covariance matrix calculated over all traces. This assumes that all classes share the same covariance matrix (homoscedasticity), which is justified by the intuition that the covariance values are determined by the influence of measurement noise, which should in most practical cases be independent of the inputs. For traces with length n and c classes, the transformation matrix $\mathbf{W}$ has the dimensions $n \times c - 1$. The trace matrix $\mathbf{T}$ with N traces is transformed into a $c-1$ dimensional subspace by multiplying it with the transformation matrix:

$$\underset{N \times c-1}{\tilde{\mathbf{T}}} = \underset{N \times n}{\mathbf{T}} \times \underset{n \times c-1}{\mathbf{W}} \tag{2.9}$$

The dimensions can be further reduced by dropping the last columns of $\mathbf{W}$, i.e. the eigenvectors with the lowest eigenvalues.

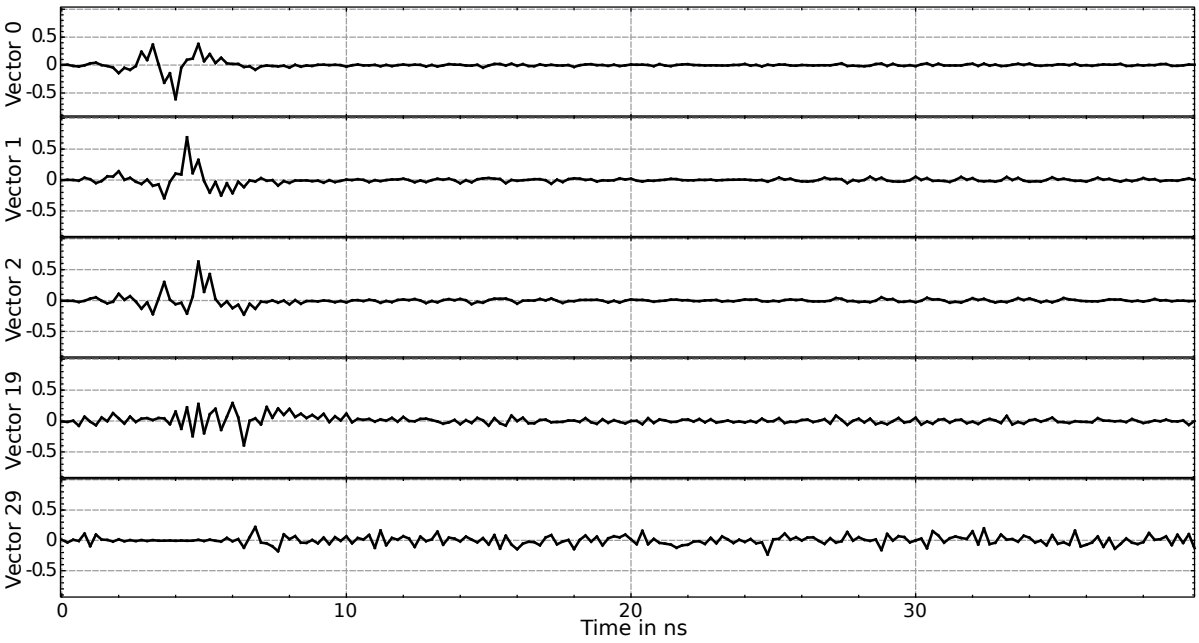


Figure 2.5.: LDA eigenvectors, sorted by eigenvalue. Calculated for AES round 1 S-box 0 output on the traces shown in Fig. 2.4.

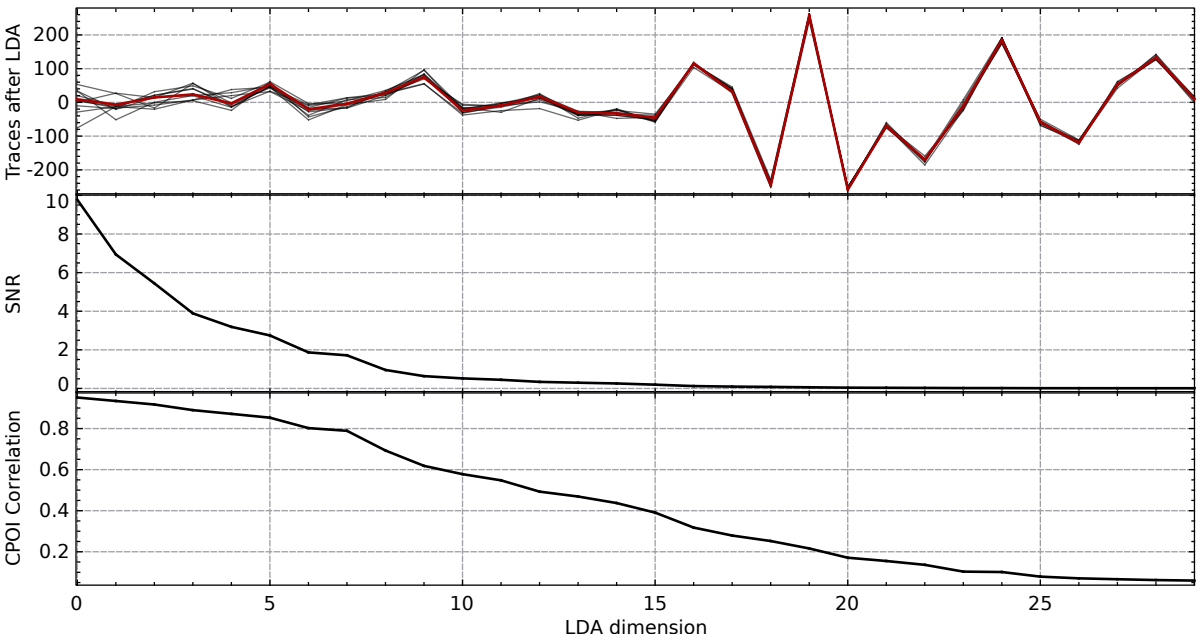


Figure 2.6.: Top: EM Traces after LDA transformation for AES round 1 into subspace with 30 dimensions, example traces in grey and mean trace in red. Middle and bottom: SNR and CPOI correlation for S-box 0 output.

To give an example, we transform the traces shown in Fig. 2.4 and run the SNR and CPOI leakage detection on the transformed traces. Each point in the transformed trace is a linear combination of all samples in the original trace and the weights are given by the aforementioned eigenvectors, i.e. the columns of $\mathbf{W}$. The traces are transformed from 200 samples into a subspace with 30 dimensions and the results are shown on top in Fig. 2.6. Again, the figure shows 10 example traces in the new subspace in grey and the mean trace over all transformed traces in red. We utilize the first (ranked by their eigenvalue) 30 eigenvectors and $\mathbf{W}$ has dimensions $n \times 30$. Figure 2.5 shows the first 3 eigenvectors and eigenvector 19 and 29. The high ranked eigenvectors show the highest weights at samples where we also see leakage in Fig. 2.4. Lower ranked eigenvectors 19 and 29 show smaller weights in general and especially eigenvector 29 does not contain any area with detected leakage. Thus, we expect the leakage to be concentrated in the first dimensions of the LDA transformed trace. This is demonstrated in Fig. 2.6 which shows the SNR and CPOI correlation of the transformed trace. The plots confirm that the leakage is in fact concentrated in the first dimensions and continuously degrades with higher dimensions. Interestingly, many lower dimensions show significantly higher leakage than any single sample in the original trace. This is reasonable since every dimension contains a combination of multiple dimensions from the original trace. The SNR in dimension 0 is 9.77 compared to a maximum of 5.24 in Fig. 2.4, the CPOI reaches 0.95 compared to 0.91. Therefore, the leakage can be exploited more efficiently using less samples.

2.4. Template attacks

Template attacks are multivariate, profiled attacks [15, 17]. They can be mounted using only a single trace in the attack or with multiple traces, then the attack is also called 'template based DPA'. Template attacks are the most powerful side-channel attack, but can be computationally expensive and fragile regarding the trace alignment and inter device portability [17]. In general the analysis and characterization of data sequences like power traces is complex and expensive if no assumption is made about the distribution of the sample points. In case of template attacks, the trace is modeled as a multivariate normal distribution. The trace is viewed as realization of n random variables $X_1, X_2, ..., X_n$ which are assumed to be jointly normal distributed. The probability density function f of a realization vector t is then given as

$$f(t) = \frac{1}{\sqrt{(2\pi)^n \cdot det(\mathbf{\Sigma})}} \cdot exp\left(-\frac{1}{2} \cdot (t-\mu)^T \cdot \mathbf{\Sigma}^{-1} \cdot (t-\mu)\right) \tag{2.10}$$

where $\mathbf{\Sigma}$ is the covariance matrix and μ the mean vector. The distribution is fully defined by the tuple $(\mu, \mathbf{\Sigma})$. The attack consists of two phases: the profiling phase and the attack phase. During the profiling, the attacker has full control over the device and can set all inputs and the key; during the attack, the key is fixed and unknown. The attacker selects a target function $f(p, k)$ which combines (part of) a known input p with (part of) the key k and that takes values in $[0, c-1]$. In case of an attack on AES, p

and k are typically bytes of the plaintext and key, respectively. The target function is e.g. the output of the first round S-box, where the plaintext byte is already combined with one byte of the key. In the profiling phase the attacker builds templates $(\mu_i, \mathbf{\Sigma}_i)$, with $i \in [0, c-1]$ for all c output values of the function $f(p, k)$. In the attack phase, the attacker measures a trace t and, knowing the input v, calculates $f(v, k)$ for all possible key candidates. For each key candidate $\hat{k}$, the likelihood that the trace t is taken out of the respective template, i.e. stems out of the corresponding Gaussian distribution, is calculated:

$$p(t) \mid t \sim \mathcal{N}(\mu_{f(p,\hat{k})}, \mathbf{\Sigma}_{f(p,\hat{k})}) = f(t \mid (\mu_{f(p,\hat{k})}, \mathbf{\Sigma}_{f(p,\hat{k})})) \tag{2.11}$$

If the key candidates do not have equal a-priori probabilities[3], then the result must be multiplied by the a-priori probability. This factor is omitted in the following for the sake of clarity. This procedure is repeated for all subkeys. The result of the attack is a list with candidates and corresponding probabilities for all subkeys. In a successful attack, the correct subkeys are ranked highly in all of the lists. If all correct subkeys are ranked first, then the attack is immediately successful. Otherwise, the attacker has to put in some brute force effort to recover the correct key. How the exact security level is determined in such cases is discussed in the Section 2.5.

As the interest lies within the relative ranking of the candidates, and not the actual probabilities, simplified discriminant functions can be used as long as they preserve the order. The log-likelihood is often used to avoid numerical issues:

$$\log\ f(t \mid (\mu_{f(p,\hat{k})}) = -\frac{1}{2}(\log((2\pi)^n \cdot det(\mathbf{\Sigma}_{f(p,\hat{k})})) + (t - \mu_{f(p,\hat{k})})^T \cdot \mathbf{\Sigma}^{-1}_{f(p,\hat{k})} \cdot (t - \mu_{f(p,\hat{k})})) \tag{2.12}$$

Choudary et al. [16] derive a discriminant function for a key candidate $\hat{k}$ under the observation of a trace t from this:

$$d_{log}(\hat{k} \mid t) = -\frac{1}{2}\log \mid \mathbf{\Sigma}_{f(p,\hat{k})} \mid -\frac{1}{2}(t - \mu_{f(p,\hat{k})})^T \mathbf{\Sigma}^{-1}_{f(p,\hat{k})}(t - \mu_{f(p,\hat{k})}) \tag{2.13}$$

The largest value determines the most probable key candidate. The calculation is further simplified by making use of the homoscedasticity assumption that is also used for LDA, i.e. the assumption that the individual covariance matrices are only dependent on the noise and thus asymptotically equal. Then a single pooled covariance matrix is calculated instead of one covariance matrix per template:

$$\mathbf{\Sigma}_{pooled} = \frac{1}{c} \sum_{i \in [0, c-1]} \mathbf{\Sigma}_i \tag{2.14}$$

In practice, using a pooled covariance matrix also makes the calculation more robust. The covariance matrix $\mathbf{\Sigma}$ grows quadratically with the length of the trace. Typically, the covariance values are very small and the matrix becomes close to being singular, which makes it numerically difficult to invert. A necessary condition for $\mathbf{\Sigma}$ to be non-singular is that the number of traces used in its estimation is larger than the number of samples in

[3]This is for instance the case when the Hamming weight of an intermediate value is targeted.

the trace. However, the traces can be highly correlated so more may be needed. When using the pooled matrix, this requirement is relaxed because all collected traces can be used in the estimation. Another option is to omit the estimation of the covariances altogether and substitute the covariance matrix with the identity matrix. Then the scores only depend on the estimation of the mean trace, but not all available information can be exploited. It is possible to use more than one trace in the attack phase by either averaging the traces beforehand (for all traces where p is equal) or by calculating the joint likelihood. The joint likelihood in case of the logarithmic discriminant function d_{log} is given by the sum of the individual scores of each trace.

2.5. Evaluating the security level after an attack

The result of a template attack (or similar attacks) is a sorted list of candidates for each subkey where each candidate has a probability (or similar score). In a real attack, the attacker then has to assemble and test the full key to see if it is correct. If the correct subkeys are not ranked first in each list, this requires a strategy to test the most probable combinations. A naive approach is to simply test all combinations of the first n candidates of each list. The probability of finding the correct key in this set is called n-th order success rate in literature. The problem is, that the security level is completely unclear if the correct key is not found. For an attacker this is not an issue, since the only goal is key recovery and a failure is a failure. Evaluators, on the contrary, have a strong interest in determining the real security level that remains after an attack. The security level, which is often also called guessing entropy or brute force/enumeration effort, is the determining metric which decides if the device under test is considered secure enough to withstand the attack. Thus, it is important to estimate the guessing entropy after an attack when evaluating a device for which key recovery attacks do not reveal the full key.

During evaluation, the correct key and the ranks of the correct subkeys are known. Lower and upper bounds for the guessing entropy can be directly estimated from the subkey ranks. The lower bound assumes that an attacker has an oracle that is queried with subkey candidates and returns if it is correct or not. Therefore, in each subkey list all candidates up to the the correct one have to be tried. Let $r_0, r_1, \ldots, r_{n-1}$ be the key ranks of all n subkeys. The minimum guessing entropy, in bits, is given as:

$$GE_{min} = \log_2 \prod_{i=0}^{n-1} r_i \tag{2.15}$$

For the upper bound, which is closely related to the n-th order success rate, the attacker has no such oracle available. Instead, all combinations of the first two candidates for each subkey are tried, then all combinations of the first three, four and so on. The upper bound is determined by the highest key rank of all subkeys:

$$GE_{max} = \log_2 max(r_0, r_1, \ldots, r_{n-1})^n \tag{2.16}$$

These estimates can be useful, but neglect the probabilities attached to the key candidates.

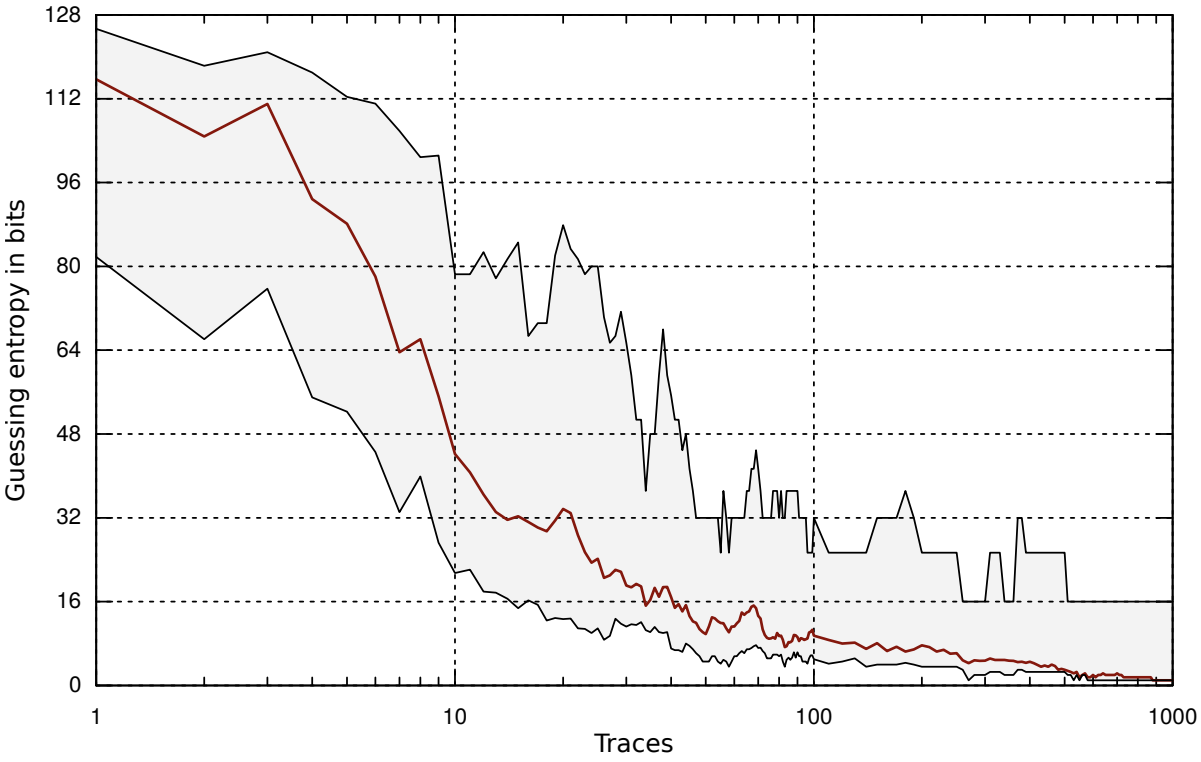


Figure 2.7.: Estimated guessing entropy GE_{kr} after template attack on AES using key rank estimation. Shaded area: region between lower and upper bound.

Key rank enumeration algorithms [105, 78] improve upon that by taking the probabilities into consideration to derive an optimal search strategy. However, this approach is limited by the computational resources of the evaluator. If the correct key lies outside the enumeration capabilities, then it is unknown if one more guess would have led to success or if a million guesses would have been necessary. To estimate the guessing entropy in cases where enumeration is not computationally feasible, key rank estimation algorithms [106, 34, 8] are used.

In this work, the key rank estimator of Glowacz et al. [34] is used. Instead of enumerating all keys from most to least probable until the correct one is reached, the idea is to group keys with similar probabilities and ignore the order within that group. Obviously this adds an estimation error that is determined by the group size, but it saves the computation of exact probabilities of the full keys.

To implement this, we estimate a probability histogram of all (full) keys. Each bin represents a range of probabilities and accounts for the number of keys that fall into that range. Knowing the probability of the correct key (by multiplication of the probabilities of all correct subkeys), the key rank is estimated by counting all keys that are in the same bin as the correct key or in bins with higher probabilities. To avoid numerical issues, logarithmic probabilities are used. Let $LP_i = \log\left(Pr(k_i^* \mid \mathcal{T})\right.$ be the list of

logarithmic probabilities of candidates for subkey i after an attack which observed the set of traces $\mathcal{T}$. In case of, e.g., 128-bit key AES (AES-128) the number of subkeys N_p is 16 and hence $i \in [1, 16]$. The logarithmic probability $lp_{correct} = \log(Pr(k \mid \mathcal{T}))$ denotes the probability of the correct key, i.e. the sum of the logarithmic probabilities of the correct subkeys. For each list LP_i a histogram $H_i = \text{hist}(LP_i, N_{bin})$ with N_{bin} equally sized bins is calculated. A higher number of bins leads to tighter error bounds, but longer runtime. The probability histogram of the full key is estimated by convolution of all subkey histograms. To calculate the estimated rank of the correct key, all keys starting from the bin of the correct key, $\text{bin}(lp_{corr})$, are counted. The whole key rank estimator is specified by Algorithm 1, taken from [34].

Algorithm 1: Key rank estimation$(LP_i, lp_{correct}, N_{bin})$

for $i = 1$ *to* N_p **do**
  $H_i = \text{hist}(LP_i, N_{bin})$;
end
$H_{curr} = H_1$;
for $i = 2$ *to* N_p **do**
  $H_{curr} = \text{convolution}(H_{curr}, H_i)$;
end
$estimated\_rank \approx \sum_{i=\text{bin}(lp_{corr})}^{N_p \cdot N_{bin} - (N_p - 1)} H_{curr}[i]$;
$GE_{kr} = \log_2(estimated\_rank)$;

The guessing entropy expressed in bits is the binary logarithm of the estimated key rank. Figure 2.7 shows the estimated guessing entropy GE_{kr} in case of a template attack on AES in red[4]. The shaded area is bound by the minimum and maximum estimates GE_{min} and GE_{max}. Especially for a lower number of traces, a big discrepancy of around 40 bits between the minimum and maximum estimates can be observed. Asymptotically GE_{kr} approaches GE_{min}, but for cases where few traces are available GE_{min} significantly underestimates the guessing entropy. In these cases the minimum and maximum estimates are insufficient and key rank estimation must be performed to better estimate the real security level.

2.6. Countermeasures

Every successful differential side-channel attack is reliant on three conditions:

1. The attacker needs to predict internal values that depend on the key and some input.
2. The variations of those internal values need to be reflected in the side-channel measurements.

[4]The example is taken from Section 4.2.2, with $m = 4$ and dense placement.

3. It must be possible to combine information over different traces.

Countermeasures against SCA can also be categorized in three categories, according to which of these prerequisites they prevent: Masking, hiding and leakage resilience.

Masking Masking countermeasures break the relation between the intermediate values of the algorithm and the values that are actually processed in the device. The reasoning is that if the attacker cannot predict the intermediate values, then it is also not possible to exploit leakage that they might create. Masking schemes apply random masks to the sensitive values and then perform all calculations on the masked values. In the end, the mask is removed and the correct result retrieved. A boolean masking scheme for DES was first proposed by Goubin et al. [37]. This and similar approaches [3, 36] can suffer from leakage induced by glitches in the hardware [61], thus designers have to take care during the implementation. Threshold implementations [72] are masking schemes that also took inspiration from secret sharing and multi-party computation. The intermediate values and calculations are split up into several (randomized) shares such that the actual values cannot be recovered without knowing all shares. This makes them inherently more robust against hardware glitches. Domain oriented masking [38] is similar to threshold implementations, but can more efficiently be extended to higher protection orders.

Hiding The goal of hiding countermeasures is to disconnect the power consumption from the processed value. This is realized by e.g. introducing additional noise and adding random stall or dummy operations to the execution [96]. The addition of noise and the shuffling of instructions ultimately only reduce the SNR and raise the number of traces which are required for the attack. Therefore they are mostly used in conjunction with other countermeasures like masking [18, 59]. Dual-rail implementations use logic styles that lead to a balanced power consumption regardless of the processed value [82]. However, they are costly to implement and have been shown susceptible to EM attacks when implemented on FPGAs [44].

Leakage resilience Leakage resilience cryptography prevents the attacker from accumulating information about the secret over multiple traces. This is discussed in detail in Chapter 3.

3. Leakage resilient cryptography

In this chapter we first give an overview of the state-of-the-art in leakage resilient cryptography and discuss different approaches in Section 3.1. In Section 3.2, we introduce LR-PRFs as an important building block for many applications and study their foundations. This concludes with a discussion of the two proposals for LR-PRFs by Medwed et al. [65, 66] in Section 3.2.2. These are the constructions which we will analyze, and improve upon, throughout the course of this thesis. Finally, we show how a leakage resilient AEAD (LR-AEAD) scheme is realized using LR-PRFs as building block in Section 3.3. We use this construction in case studies to decrypt FPGA bitstreams and microcontroller firmware updates in Chapters 7 and 8, respectively.

3.1. Overview of research in leakage resilient cryptography

In modern cryptography, the security of an algorithm is based upon assumptions about computational complexity and guaranteed by rigorous security proofs and computational complexity bounds. Typically, these proofs consider the execution of the algorithm inside a black box, i.e. the attacker only sees (and, depending on the security notion, controls) the inputs and outputs to the algorithm. Cryptographic reductions allow to break down systems into collections of well understood primitives and thus to proof the security of complex cryptographic systems against large classes of adversaries. These assumptions and reductions are the foundation for all modern crypto systems that we use on a daily basis.

Motivation With the advent of physical attacks started by Kocher et al. [51, 52], the significance of those security notions got challenged: What good is a provably secure algorithm if the actual implementation can be broken by observing e.g. the power consumption or timing? Reacting to the fact that suddenly most cryptographic implementations were vulnerable against this new class of attacks, several approaches were proposed to break the link between the secret and the leakage of an implementation (see Section 2.6). However, initially those countermeasures were designed ad-hoc to protect a specific implementation against a specific attack. From a cryptographer's point of view, this situation is very unsatisfying compared to the state of the art in theoretic cryptography, because it is not exactly clear, what security guarantees are actually achieved. The natural next step is to come up with new models and assumptions that include the threat of physical attacks against the implementation. If this succeeds, then an

argument can be made that proves a construction secure against *all* attacks of a certain class. It would even cover attacks that are not known at the time of analysis and it would relieve designers from evaluating the security of their design against all known attacks. This is the ultimate goal of "Leakage Resilient Cryptography" as described in the paper of the same name by Dziembowski and Pietrzak [25].

While the appeal of this approach is clear, it is not trivial to define a model that adequately captures the capabilities of all thinkable side-channel adversaries and still leaves room for cryptography. It is intuitively clear, that the black box model has to be augmented in order to allow the attacker insight into the internals of the implementation. This is modeled through a *leakage function*, that gives an attacker information dependent on the internal state of the algorithm. The main challenge is the definition of this leakage function: If it is too powerful, then no provable security can be achieved at all. If it is too weak, then there is a risk that attackers exploit leakage that is not captured by the model, which makes the security proofs pointless.

Different approaches to leakage models To position our research, we categorize the existing work following the structure used by Kalai and Reyzin [48]. This is by no means an exhaustive list, but it should help to get the greater picture of the field and to understand the different research directions. On the highest level of abstraction, the research can be partitioned into two broad fields by considering the source of the leakage in the respective models: memory leakage or computation leakage.

Memory leakage models consider key leakage arising from memory. Typically the amount of leakage available to the attacker, i.e. the length of the output of the leakage function, is bounded. Dziembowski and Di Crescenzo et al. [24, 19] proposed the bounded retrieval model (BRM), which allows the attacker to compute any polynomial-time leakage function on the secret key, as long as the output of that function is limited in the number of bits. Secure constructions can then be achieved by increasing the key length, whereby the computational complexity is only allowed to grow logarithmically with the key length to avoid inefficient constructions. This line of work was extended by Akavia et al. [2] who introduced the bounded memory leakage (BML) model for public key cryptography. Here, the bound for the leakage is not an absolute value but a function of the key length. The bounded leakage models do not translate well into reality, as usually an attacker can measure many bits of information about the secret key, be it by using multiple measurements, various target variables or different measurement setups. Dodis et al. [22] acknowledge this fact in their auxiliary input model (AUX) model and allow an unbounded amount of leakage. The constraint is that the leakage function is hard to invert, or in other words, given the (large) amount of leakage, it should still be hard to calculate the secret key. The continual memory leakage (CML) model by Brakerski et al. [9] and Dodis et al. [21] is similar in that it also allows the total leakage to be unbounded. The idea is that the initial secret key is continuously updated and the amount of leakage of any such temporary key has to be bounded, a concept related to what is also known as fresh re-keying [64, 6].

From a practitioners point of view, the approach that (only) the memory leaks seems

hard to justify. Real world attacks usually target instances where the key is used in some calculation and measure leakage caused by switching activity of combinatorial logic. Although comprehensive results are certainly possible under memory leakage models, it seems more natural to model leakage that arises from computations. These so called *computation leakage* models form the second big research subfield. The computation leakage models introduced by Chari et al. [14] and Ishai et al. [45] both consider leakage from wires of a circuit. The noisy wire mode [14] gives the attacker a noisy view of all activity in the device, whereas the wire-probing model [45] allows the attacker to directly read out a limited amount of wires. This view lends itself naturally to the construction of masking schemes because the addition of random masks makes internal values dependent on more signals. The only computation leaks (OCL) model by Micali and Reyzin [68] has become the most popular model and variations of it have been used to analyze all constructions that are used in this thesis. The premise is, that the algorithm runs in discrete steps and during every step only the active calculation leaks. Consequentially, values that are not touched and reside in memory do not contribute to the leakage. This is modeled by a polynomial-time computable leakage function that depends on three parameters:

1. The current configuration C of the machine. This includes every part of its internals that is measurable and is 'touched' during this step.
2. The settings of the measurement apparatus M, which is a specification of what the attacker chooses to measure.
3. A random string R that represents the noise within the measurements.

In the most powerful attacker model, the measurement setting M can be set by the attacker before every step of the algorithm and in reaction to what was learned in previous steps. The leakage function is then called *adaptive*. It was suggested by Standaert et al. [93] that an adaptive, polynomial-time leakage function might be too powerful and limits the efficiency of the achievable constructions. For example, nothing stops the attacker from choosing a leakage function that leaks about future values, which is clearly unrealistic in a physical setting. Another issue is that the leakage function is in part determined by the measurement apparatus M. In this model, this is an input that is chosen by the attacker. While the measurement setup or the positioning of probes do affect the leakage, it is however mostly defined by the physical properties of the device itself. This leads to an inevitable trade-off, as the authors of [93] remark, between the generality of the leakage function and the connection to the physical world. Generic leakage functions make it hard to achieve positive constructive results. An extremely specific leakage function, on the other extreme, can be tailored to fit an experiment on a device, but has questionable validity outside that experiment.

Towards practical leakage resilient cryptography In an attempt to circumvent the difficulty of formally modeling the leakage function and to make leakage resilience more accessible to practitioners, Standaert et al. [91] in a separate work propose a framework to analyze the side-channel security of implementations and countermeasures. They build

upon the principles introduced by Micali and Reyzin [68], but give up some generality and only consider key recovery adversaries. The idea is to separate the analysis of the implementation and the adversaries' capabilities to allow fair comparisons between countermeasures independently of the attack. Consequentially, they suggest the usage of two different metrics for these subproblems: To answer the question *how much is leaked by the implementation*, an information theoretic metric, the Mutual Information, is used. To evaluate *how much leakage is exploitable by a concrete adversary*, the (n-th order) success rate and the guessing entropy are used. Most importantly, *both* metrics require a practical evaluation. This is in contrast to the other approaches that theoretically model the leakage and find provably secure constructions.

For the estimation of the Mutual Information, they suggest using the "best possible tools" which typically means that the underlying leakage distributions are estimated with template attack style measurements. This step is considered to be the work of an evaluator with deep knowledge of the device and under close to ideal conditions. To assess the security against a concrete adversary, the actual attack is mounted against the implementation. Depending on the defined computational capabilities of the adversary, the result is either given as the n-th order success rate, i.e. the probability that the correct key candidate is within the n highest ranked candidates, or as the remaining guessing entropy, i.e. the number of steps that need to be calculated in a brute force manner in order to reach the correct key. However, it is noted that while the mutual information is a useful tool in many practical scenarios, there exist cases where more information does not lead to better attacks. The authors therefore stress that the information theoretic analysis always should be followed by a security evaluation. This approach is a step away from strong generalization as witnessed in e.g. the work of Micali and Reyzin [68] and is intended to formalize practical evaluations and comparisons of implementations and adversaries. However, the fact that both the evaluation of the implementation and the adversary require taking measurements of the device (potentially using the same equipment) and are subject to the same technical and computational limitations (e.g. in case of very long traces), weakens the separation between the two cases.

Criticism There has also been criticism of the fundamental idea that it is reasonable to model all kinds of side-channel attacks in one theoretical model. For example, Koblitz and Menezes [50] state that leakage resilience cryptography can lead to overly complex constructions that are inefficient and harder to protect which can make them even less secure than traditional constructions. Also, the reliance on (inadequate) leakage models might give the developer a false sense of security, and thus, in their opinion, the only way to reach side-channel security is through extensive ad-hoc testing and constant re-evaluation if new attacks appear. This last point is somewhat in line with the work of Standaert et al. [91], which also demands experimental testing of newly developed schemes. After presenting the results of this thesis, Chapter 9 comes back to this criticism and discusses the merits and difficulties when implementing and evaluating leakage resilient constructions.

Summary and positioning of this work Summarizing, the systematic modeling of security in a setting with side-channel leakage has received a lot of attention of the cryptographic community in recent years. Due to the heterogeneous landscape of side-channel attacks, which ranges from cold boot attacks [40] over remote cache attacks [109, 55] to high precision EM attacks, the modeling of a leakage function that is generic enough to capture all these scenarios and still allow positive deductions is inherently difficult. As of now, this gap between theory and practice still has to be overcome in a generic and meaningful setting. While this problem remains unsolved for now, there exists a line of work such as the examples by Medwed et al. [65, 66] and Standaert et al. [91] that gives up some generality in order to bridge this gap towards easier implementation and evaluation. This work positions itself more on the side of practitioners who are tasked with implementing and evaluating leakage resilient constructions. The goal is to analyze specific constructions and verify, if the underlying leakage model and assumptions hold when facing an adversary with high end equipment. The conclusions drawn from the results are twofold: First, they give a hint about the applicability of leakage resilience and the actual robustness of provably secure constructions. Second, the results can be used to engineer improved constructions. The word *engineer* is used purposefully because the modifications are based on laboratory evaluations of real devices, and not rigorous proofs. For that cause we utilize the framework of [91] to assess the security in terms of guessing entropy. Due to the questionable informativeness of the information theoretic metric and the aforementioned fuzzy separation of implementation and attack, the mutual information is not considered.

3.2. Leakage resilient pseudorandom functions

In this section we first give some context as to why PRFs are important building blocks for cryptographic protocols and define the used primitives. Then we introduce the concrete LR-PRF instantiations that this work builds upon.

3.2.1. Preliminaries

A PRF is defined as follows:

Definition 1. *Pseudorandom function*
A pseudorandom function is a function $F : \{0,1\}^k \times \{0,1\}^p \mapsto \{0,1\}^q$ *that takes a key and a message and outputs a pseudorandom string. The function is indistinguishable (with significant advantage) from a random function for an adversary that is allowed to make queries to it. We denote the execution of* $F(k,x)$ *with* $F_k(x)$*.*

To motivate the practical importance of PRF, we first consider a common initialization problem in stateless communications protocols. Stateless protocols are protocols, for which no session information is stored between the communication parties. All interactions need to be self-contained and include all necessary information. In the context of cryptographic primitives, this means that the primitives are typically initialized with

public inputs. Consider the example of session key derivation for secure (side-channel protected) communication between to parties, e.g. to deploy encrypted updates to the firmware of a microcontroller.

The parties will first exchange an initialization vector (IV) to derive the session key from a pre-shared long-term secret. After this step, the communication is encrypted with the session key. As shown in Fig. 3.1, the natural attack vector for a side-channel adversary is the processing of the IV, since it can be observed or even controlled by an adversary if no further authentication is deployed. This secure initialization problem is

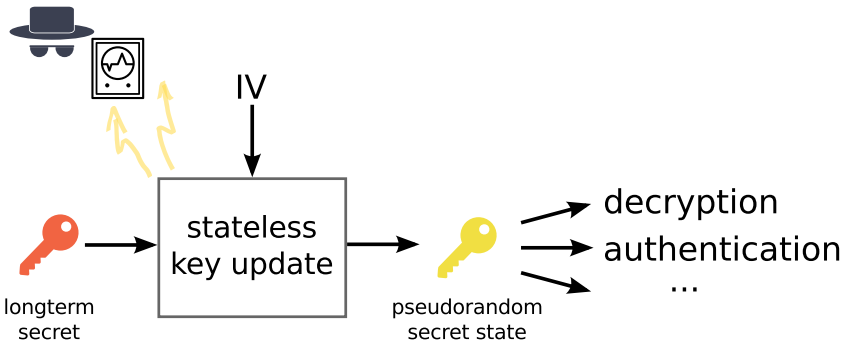


Figure 3.1.: Stateless key update with public IV.

one example that can be solved using LR-PRFs. The LR-PRF takes the IV as input and calculates the session key while protecting the longterm secret from being leaked to an attacker.

Next, we define PRGs in order to be able to understand the PRF construction of Goldreich, Goldwasser and Micali [35] (also called GGM-PRF and GGM tree). This is the foundation for the LR-PRFs by Medwed et al. which are discussed in Section 3.2.2.

Definition 2. *Pseudorandom generator*
A pseudorandom generator is a function $G : \{0,1\}^p \mapsto \{0,1\}^q$ *with* $q > p$ *that takes a seed as input and outputs a longer pseudorandom string such that statistical tests cannot differentiate the output from a string sampled from the uniform distribution (with significant advantage). If the output length* q *is a multiple of* p*, we call it a* q/p*PRG. For* $q = 2p$ *for example, the PRG is called* 2*-PRG or length-doubling PRG.*

The main difference between a PRG and a PRF is that a PRG is stateful. After setting the seed of a PRG, random access to parts of the output string is not possible without computing all bits up to the desired section[5]. In contrast, a PRF allows direct random access by taking a message as a second input. This allows the construction of e.g. symmetric encryption or message authentication code (MAC) schemes: For message authentication, the tag t for message m is calculated as $t = F_k(m)$. To encrypt m, first a random string r is generated. Then the output of the encryption is $Enc_k(m) =$

[5]Specific implementations may allow random access at some granularity, such as the block cipher based PRG introduced in Section 3.2.2. However, this is not reflected in the interface which has no input to request only parts of the output.

$(F_k(r) \oplus m, r)$. PRFs can be constructed from PRGs with the construction by Goldreich, Goldwasser and Micali [35].

Definition 3. *Goldreich, Goldwasser and Micali (GGM) PRF*
Let G be a length-doubling PRG. Denote the two halves of the output as G_0 and G_1: $G(m) = G_0(m)||G_1(m)$. The bits of the input x are denoted as $x_0 \dots x_{i-1}$. Then a PRF is constructed as follows:

$$F_k(x) = F_k(x_0, x_1, \dots, x_{i-1}) = G_{x_{i-1}}(\dots G_{x_1}(G_{x_0}(k)) \dots)$$

The input x is evaluated bit by bit and for each bit, either G_0 or G_1 is selected to seed the PRG for the next iteration. Initially, the PRG is seeded with the secret key k. This is the foundation for the LR-PRFs discussed in the next section.

3.2.2. CHES 2012 and ASIACRYPT 2016 LR-PRFs

At CHES 2012, Medwed et al. presented an LR-PRF which is based on the GGM-PRF [65]. It was later improved at ASIACRYPT 2016 [66]. They are constructed using a block cipher as the primary building block. Specifically, AES-128 is used but any other substitution–permutation network block cipher (e.g. PRESENT) is also possible. Both LR-PRFs have in common that they limit the data complexity (the observable operations of each key) to prevent differential side-channel attacks.

CHES 2012 LR-PRF We first observe in Fig. 3.2, that a 2-PRG can be implemented using a block cipher that encrypts two different inputs under the same key [93]. The encryption key is the seed to the PRG, the two plaintexts p_0 and p_1 are static parameters that can be public. Regarding side-channel attacks, this construction is naturally "2-limiting", which means that the key is only used in two different operations, the encryption of p_0 and p_1. This 2-PRG can be extended to a n-PRG by encrypting n instead of 2 different plaintexts. This is the building block with which the LR-PRF is instantiated.

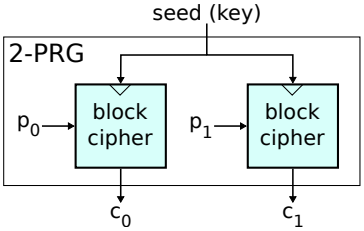


Figure 3.2.: Instantiation of a 2-PRG with a block cipher.

The left side of Fig. 3.3 shows the dataflow graph of the resulting LR-PRF instantiated using this block cipher based 2-PRG. As is the case with the GGM-PRF, the input is processed bit by bit. In the initial stage 1, the long-term key k is used as seed to the 2-PRG, i.e. key for the block cipher. If the first bit of the input is zero, the left part of the

2-PRG output, i.e. $Enc_k(p_0)$ is used as key for the next iteration. Otherwise, $Enc_k(p_1)$ is used. The plaintexts are public and pre-determined, but of a special form which will be explained later. This process continues until all bits of the input are processed. The dataflow can be seen as a walk through a binary tree structure where the result of the LR-PRF is the leaf of the tree. This gives the following equation for the LR-PRF. To make it more readable we write $Enc(k, x)$ for $Enc_k(x)$:

$$F_k(x) = F_k(x_0, x_1, \ldots, x_{i-1}) = Enc(Enc(\ldots Enc(Enc(k, p_{x_0}), p_{x_1} \ldots), p_{x_{i-2}}), p_{x_{i-1}})$$

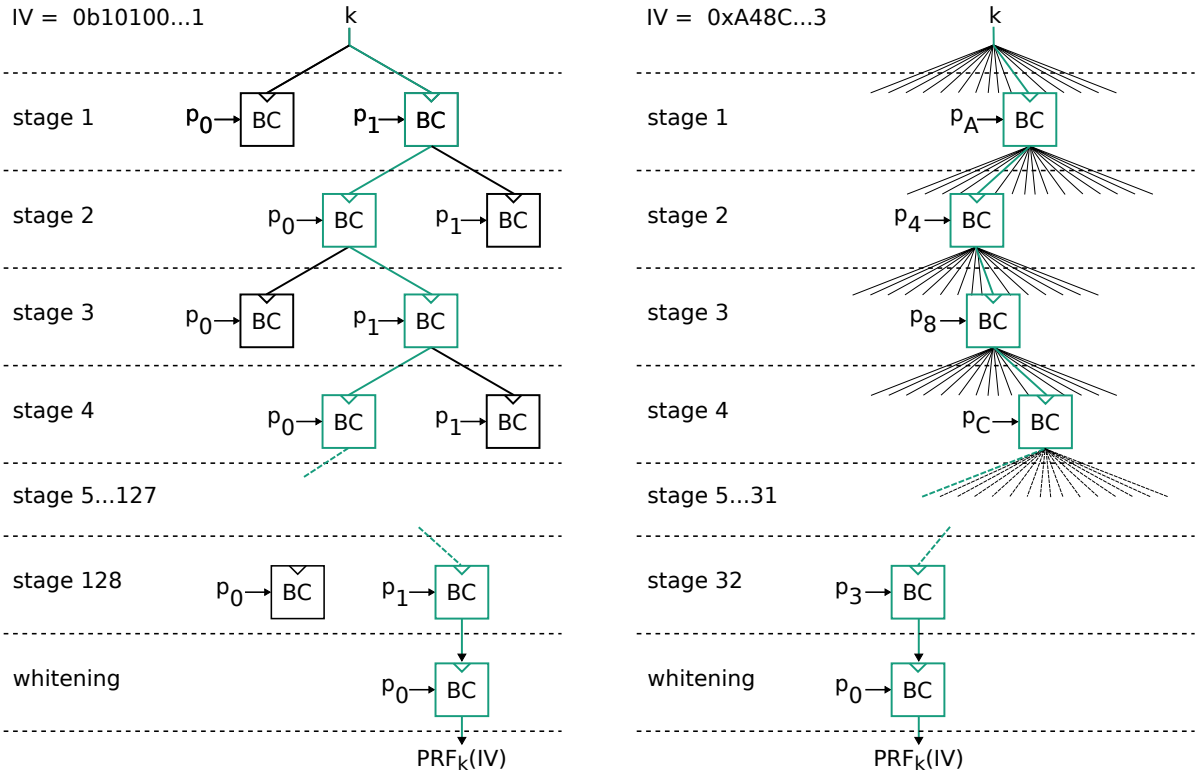


Figure 3.3.: CHES 2012 LR-PRF. Left: data complexity 2 ($n=1$), right: data complexity 16 ($n=4$).

An execution of this LR-PRF requires one AES encryption per bit of the input, which can be a significant overhead for some applications. This is why the authors of [65] proposed an alternative which is more efficient in terms of latency. Instead of evaluating one bit of the input per stage, n bits are evaluated and consequentially one out of 2^n plaintexts is encrypted. To process an input with a length of i bits, the LR-PRF evaluation takes $\lceil i/n \rceil$ block cipher encryptions. The trade-off is that this also increases the data complexity for side-channel attacks. The right side of Fig. 3.3 shows the case of $n=4$, where 32 block cipher executions are required for a 128 bit input and the data complexity for a side-channel attack is 16.

The leakage resilience of this PRF is based on two principles: Limited data complexity and algorithmic noise through parallel S-boxes with chosen inputs. The data complexity

of every key is limited by the PRG that is evaluated on every level of the tree. In case of $n = 1$ (Fig. 3.3, left side), it only allows the encryption of two distinct plaintexts, depending on the corresponding input bit. For $n = 4$ (Fig. 3.3, right side), 16 plaintexts can be encrypted. This is in contrast to regular side-channel attacks, where an attacker typically observes many different inputs under one key. The number of bits n processed per stage thus has a direct impact on the security and the performance of the construction. On the one hand, the data complexity for an attack is determined by n, as each key is used with 2^n different plaintexts. As will be discussed in detail in Section 4.2.2, low data complexity makes differential SCA attacks harder. Therefore, from a security point of view, n has to be kept low. On the other hand, the performance of the LR-PRF deteriorates with lower n because more iterations are necessary to process the input. Hardware designers and evaluators have to find a trade-off in n between security and efficiency.

Limited data compexity as countermeasure against side-channel attacks is only effective if the observation of 2^n distinct encryptions is not enough to successfully recover any key inside the GGM tree. To make key recovery attacks harder, an additional countermeasure leverages parallel hardware and the algorithmic noise it generates. Algorithmic noise is generated by parts of the implementation which are executed at the same time, but are not represented in the attackers target variable. To increase the algorithmic noise in the LR-PRF, the implementation of the AES is required to exhibit parallelism in the S-boxes. In a fully parallel AES hardware implementation as shown in Fig. 3.4, all 16 S-boxes are evaluated in parallel. In the figure $p_0 \dots p_{15}$ and $k_0 \dots k_{15}$ indicate the bytes of the plaintext and key, respectively. Each S-box produces leakage which is a function of the S-box calculation on $p_i \oplus k_i$. The observable leakage is a combination of the leakage from all S-boxes, which gives $L_{total} = \sum_{i=0}^{15} L(Sbox(p_i \oplus k_i))$. This leakage model implicitly assumes that all S-boxes have identical leakage functions; we refer to this as the *equal leakage assumption*. While attacking one byte of the AES key during the S-box operation, all other key bytes generate algorithmic noise in the leakage traces. Unlike electrical noise this form of noise cannot be removed by averaging over multiple traces. Usually, the attacker is able to randomize the S-box operations by changing the plaintext input and is able to remove the algorithmic noise that way. In the LR-PRF construction, this is prevented because the attacker cannot directly chose the plaintexts which are encrypted. Instead, pre-determined plaintexts are selected depending on the input of the LR-PRF. All plaintexts are chosen such that all bytes have the same values, i.e. $p_0 = p_1 = \dots = p_{15}$ (e.g. $p_0 = 0^{128}$ and $p_1 = 1^{128}$ for $n = 2$).

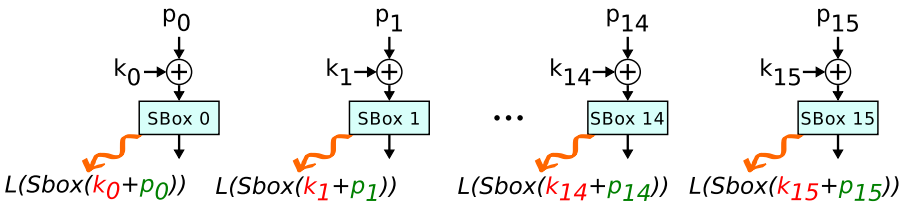


Figure 3.4.: Leakage from parallel S-boxes.

In Fig. 3.4 data which is known to the attacker is printed in green, secret data in red. For the algorithmic noise from parallel S-boxes to be effective, we need to assume that the attacker measures L_{total} and cannot resolve the leakage of the individual S-boxes. As the attacker only knows the plaintext bytes p_i, this is the only way to distinguish between the leakage of the different S-boxes (given a parallel implementation where they are evaluated simultaneously). If all bytes of the plaintext are equal, then it is not possible for an attacker to determine which leakage belongs to which S-box. Consequentially, in a divide-and-conquer attack on the key bytes, an attacker has no means to identify the leakage of a specific key byte k_i in the measured leakage L_{total}. Then, even if all key bytes would be recovered correctly, the attacker still needs to find the correct order of the key bytes. This adds significant brute force effort with super-exponential complexity in the number of S-boxes. In case of AES with its 16 S-boxes, this results in an additional security margin of $16! \approx 2^{44}$ bits. However, this countermeasure is only effective for attacks on the first AES round. The S-box inputs of all following AES rounds and especially the ciphertexts are naturally random and do not exhibit this structure. While the state of intermediate rounds is hidden, the ciphertext of the last stage is also the output of the LR-PRF and is exposed to the attacker. To prevent DPA attacks on the last round of the AES using the ciphertext, the authors of [65] propose an additional output whitening step. This step is realized as an additional stage of the GGM tree which always encrypts the same plaintext. With this whitening step the output of the LR-PRF is (necessarily) still random, but the data complexity for this attack vector is always 1, regardless of the value of n. Thus no differential side-channel attacks on the last round are possible. The only requirement is that the implementation withstands SPA attacks. Due to the fact that the first and last round of the AES lead to comparable leakage, it is reasonable to assume that if an implementation withstands a 2-limited attack on the first round, it should also withstand a 1-limited attack on the last round. The easiest attack vector, and the sufficient object of investigation for a security evaluation, is therefore the first round of the AES.

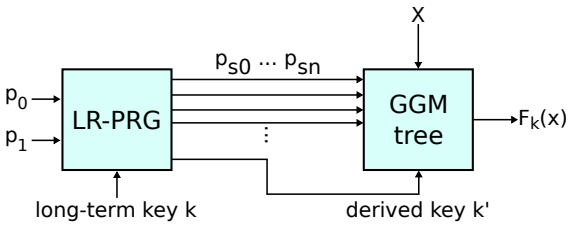


Figure 3.5.: Unknown inputs LR-PRF from ASIACRYPT 2016.

ASIACRYPT 2016 LR-PRF At ASIACRYPT 2016 an improved version of this scheme was proposed [66] which is shown in Fig. 3.5. It is based around the same GGM-PRF instantiated with the AES-based PRG. The difference is how the plaintexts inside the GGM tree are chosen. In the original proposal, they are public, i.e. known to the attacker, but of a special form to hinder key recovery. The ASIACRYPT version of the LR-PRF goes a different way and hides the plaintexts from the attacker. An additional

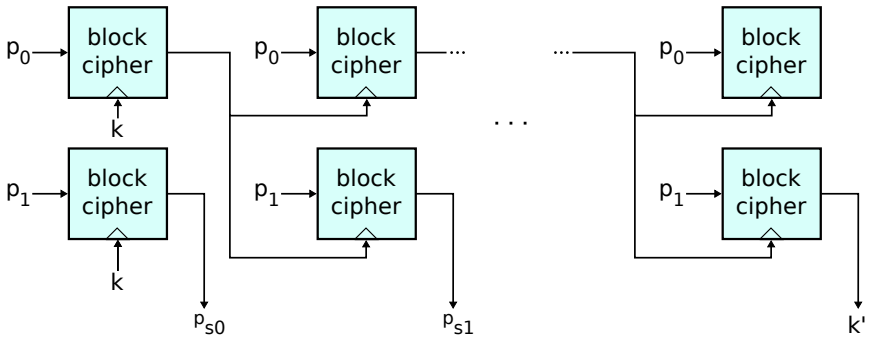


Figure 3.6.: Leakage resilient PRG with variable output length.

preprocessing step using a leakage resilient pseudorandom generator (LR-PRG) [92] derives the secret plaintexts $p_{s0} \dots p_{sn}$ and the key k' for the GGM tree from a long-term key k. The LR-PRG is shown in Fig. 3.6 and uses the same block cipher based 2-PRG as the LR-PRF. Starting with the long-term key k, it encrypts two (public) plaintexts per iteration. One ciphertext is used as key for the next iteration, the other is the output of the LR-PRG. These outputs, i.e. the plaintexts for the GGM tree, are unknown to the attacker. Therefore the data complexity of the tree can be securely increased which enables more efficient implementations which require less stages. The attack vector is shifted to the LR-PRG stage, because this is the starting point where an attacker still knows the inputs to the block cipher. However, the data complexity at this point is always limited to 2 due to the construction of the LR-PRG. When evaluating the side-channel security of this construction, the analysis is equivalent to the evaluation of the original proposal with the lowest possible data complexity $n = 2$.

The overlapping leakage of S-boxes and the plaintexts with equal bytes are novel concepts by the authors of [65] for which the security guarantees are not immediately clear. They provide results for power analysis attacks on a microcontroller implementation as well as results from simulated attacks. In those cases, the security of this construction holds. However, they leave localized EM attacks for future work and only briefly discuss the effects. This work picks up there and provides a laboratory analysis in Chapter 4.

3.3. Leakage resilient authenticated encryption from LR-PRFs

For many applications such as secure boot with encrypted firmware an authenticated encryption with associated data (AEAD) scheme is mandatory. An AEAD scheme is defined as follows:

Definition 4. *Authenticated encryption with associated data (AEAD)*
An AEAD scheme is defined by an encryption algorithm Enc and a decryption algorithm Dec. The encryption algorithm Enc takes a key k, message m, associated data a and nonce n as input and outputs a tag t and ciphertext c. The decryption algorithm Dec

takes k, c, a, n and t as input and outputs either the decrypted message m or the invalid symbol $\perp$ if the tag is incorrect.

$$Enc : k \times m \times a \times n \mapsto (c, t)$$
$$Dec : k \times c \times a \times n \times t \mapsto m \cup \perp$$

AEAD schemes can be implemented using dedicated constructions or build from common primitives such as block ciphers and MACs through generic composition. In order to utilize the presented LR-PRFs, we make use of the results of Krämer and Struck [53]. In their work they revisit the so called $FGHF'$ construction that was proposed by Degabriele et al. [20] in the context of sponge based constructions. The $FGHF'$ construction is a composition LR-AEAD scheme and comprises four building blocks: Two functions F and F', a PRG G and a hash function H. An overview of the $FGHF'$ scheme is shown in Fig. 3.7. It comprises of two parts: (1) encryption using F to generate the seed for the PRG G, which generates the key stream and (2) authentication using the hash H to condense a variable length input into a fixed length input to F' which calculates the tag. Note that the key k in this case consists of two keys k_{enc} and k_{mac} which are used for the stream cipher and MAC part of the scheme, respectively.

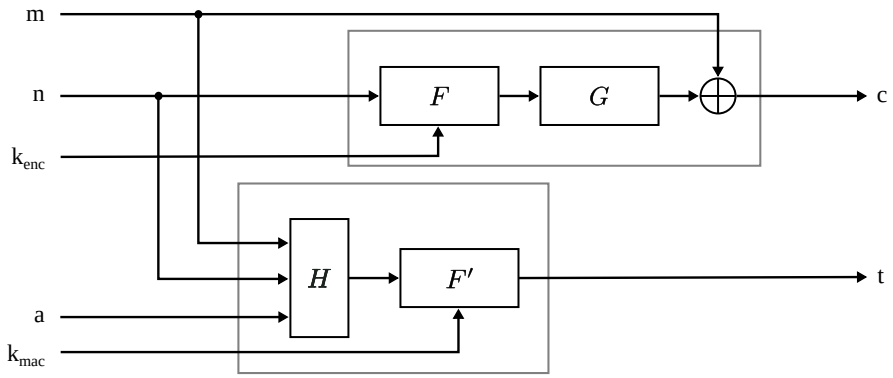


Figure 3.7.: LR-AEAD implementation (adapted from [53]).

In order for the construction to be leakage resilient, the security analysis of Degabriele et al. originally requires both F and F' to be pseudorandom under leakage and F' to be unpredictable under leakage in addition. Note that without considering leakage these two notions imply each other, but under side-channel leakage this is not the case as discussed in e.g. [53]. The hash function and PRG can be instantiated with unprotected primitives. However, the security analysis in [20] implies that the PRG resists SPA attacks. This requirement is intuitively clear since leaking the seed of the PRG would allow an attacker to recreate the key stream and decrypt the message.

Krämer and Struck simplify the security notion and show that for the $FGHF'$ construction unpredictability under leakage of F' is actually not required and that both F and F' can be implemented using LR-PRFs. In the context of our work, this allows us to use one of the AES based LR-PRF that are introduced in this chapter as building block for both F and F'. The PRG G can be implemented using the AES based LR-PRG of

Standaert et al. [92] (Fig. 3.6). We discuss the security implication of using this block cipher based construction for G in the case study presented in Section 7.4.3. The hash function H, e.g. SHA-256, can either be realized using hardware accelerators, if they are present, or implemented in software. Since it does not process sensitive inputs, no side-channel resistance is required.

4. High precision EM Analysis of LR-PRFs

In Section 3.2.2 we discussed the theoretical side-channel security of the CHES 2012 and ASIACRYPT 2016 LR-PRFs [65, 66] and established that the practical security level depends on overlapping leakage of S-boxes and effects caused by limited data complexity. This chapter evaluates, whether these assumptions hold against high precision EM analysis. We implemented the CHES 2012 LR-PRF on a Xilinx Spartan 6 FPGA and provide results from template attacks on the S-boxes of the first AES round and for different configurations of the LR-PRF regarding the data complexity. We describe the implementation and measurement setup in Section 4.1 and the results of the side-channel evaluation in Section 4.2. When comparing results acquired from power traces with results from high precision EM analysis, we find that the countermeasure is effective against power analysis attacks. However, in case of EM analysis the maximum achievable security level is only 20 bits. In an attempt to increase the algorithmic noise of S-boxes, the experiment was repeated with different hardware placement and routing. In this design, the S-boxes are implemented as hard macros with equal internal structure and placed closely together. We find that this increases the security level, but the maximum is still only 48 bits which is in range of brute force attacks and, thus, not acceptable.

4.1. Hardware design and measurement setup

This section describes the hardware design of the LR-PRF and the power and EM measurement setup that was used to acquire the traces.

4.1.1. LR-PRF hardware design

For the implementation of the CHES 2012 LR-PRF test design, we used a Xilinx Spartan 6 XC6SLX9-3TQG144C FPGA manufactured in a 45 nm process technology. A control state machine reads the input in chunks of n bit (depending on the configured data complexity) and generates the plaintext inputs to the block cipher by replicating those bits, thus fulfilling the 'carefully chosen plaintexts' condition (Fig. 4.1). This method saves registers because it is not necessary to pre-compute and store the plaintexts if they are generated on the fly by replication.

At the core of the LR-PRF is an AES block cipher in electronic codebook (ECB) mode. As parallelism in the S-boxes is required by the LR-PRF, we implemented a fully parallel

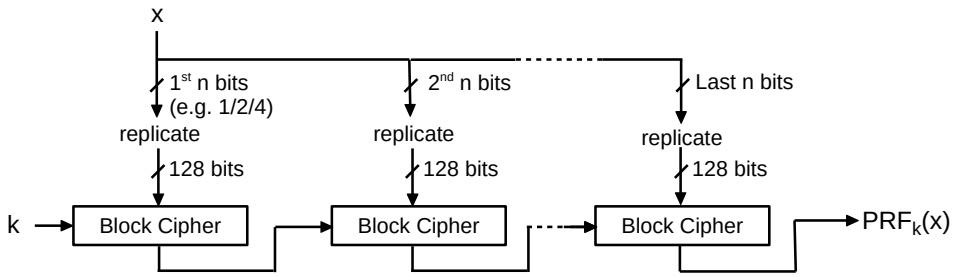


Figure 4.1.: Generation of carefully chosen inputs by replication of input bits.

AES-128 as shown in Fig. 4.2. The design comprises 16 parallel Canright S-boxes [13] in the datapath and 4 additional S-boxes in the key schedule which are all operating at the same time. We used Canright S-boxes because only S-boxes synthesized from logic gates allow the required placement flexibility, contrary to RAM-based S-Box designs. Also, Canright's proposal is state of the art when implementing S-boxes efficiently in combinational logic. Each AES round takes one clock cycle and the intermediate state is stored in a 128 bit wide register.

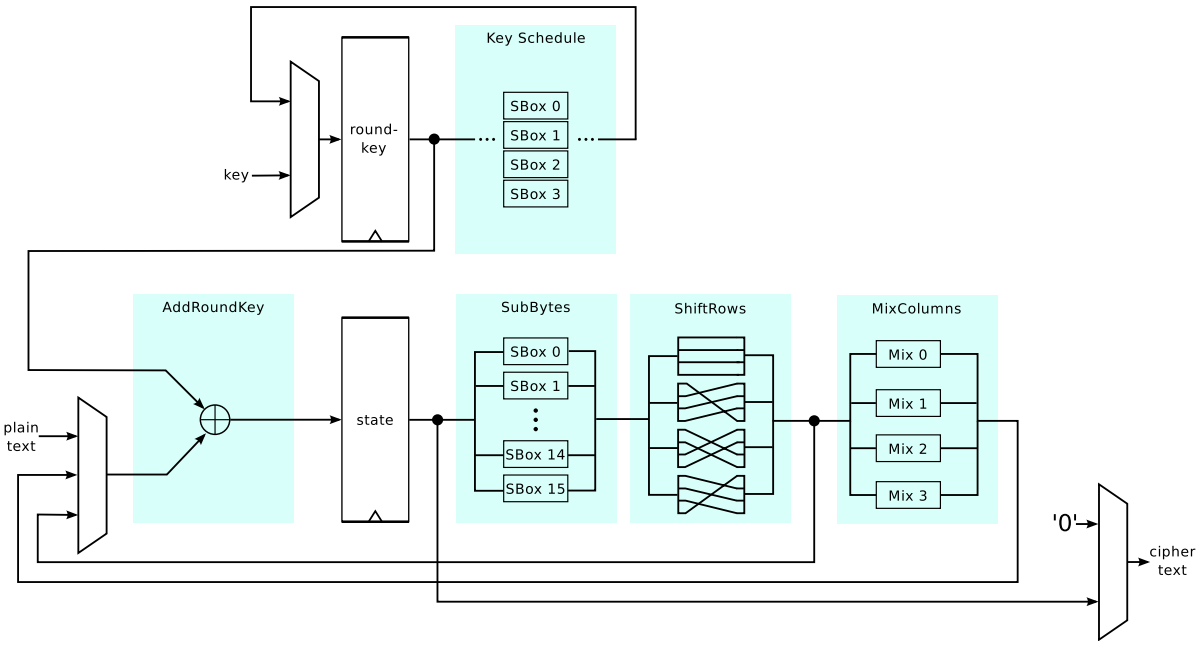


Figure 4.2.: AES hardware design.

We synthesized the design in two ways, (1) without any routing constraints, and (2) with the 16 S-boxes in the datapath implemented as hard-macros and placed as dense as possible. Since the S-boxes in the key schedule are not targeted by our differential attacks, we let them be placed without constraints. The unconstrained design is spread evenly over most of the FPGA die surface and the area covered by the logic is roughly 3 mm by 3 mm large. For the densely placed design, we first placed and routed one S-box as depicted in Fig. 4.3. Then we utilized Xilinx's relative location (RLOC) and

area constraints to clone and place this 'hard-macro' as dense as possible (Fig. 4.4). This is intended to help fulfilling the equal leakage assumption (all S-boxes should have identical leakage functions, c.f. Section 3.2) of the S-boxes as closely as possible because S-boxes are equal to a higher degree (apart from the routing to/from the S-box) and the area is generally smaller, which should make localized EM attacks more difficult. In addition, we constrained the placement of the rest of the AES to a confined area (black box in Fig. 4.4) in an attempt to make the routing to the mix-columns logic as short as possible. Based on the reports of the design tools, the estimated die area occupied by the AES is about $0.5\,\mathrm{mm}^2$. For both placement options, we synthesized designs with data complexity 2 and data complexity 16. To avoid synthesizing logic overhead for the configuration of the data complexity this parameter is first set at synthesis time. As a result, we generated and analyzed four different hardware configurations (i.e. bitstreams).

The design is controlled over a RS232 serial interface and outputs a trigger signal for the oscilloscope before the beginning of every encryption. In addition to being used as part of the LR-PRF, the AES core can also be used alone in OFB mode. This allows the encryption of arbitrary plaintexts regardless of the configured data complexity and is used during the profiling phase of the template attack. We emphasize that the block cipher core remains the same (it shares the same hardware), only the inputs are different between the two modes. OFB mode was chosen over, e.g., ECB mode because the output of one iteration is fed back and XORed with the plaintext as input to the next iteration. In order to observe (pseudo-)random plaintext encryptions it suffices to initialize the mode with one random plaintext and then keep the plaintext of consecutive rounds zero. This is used to significantly speed up trace acquisition during profiling: Only the initial plaintext is sent over the serial interface, all other plaintexts are generated on-chip inherent to the mode of operation. For each encryption, only a start command is sent and the ciphertext is received. To ensure that control PC and the FPGA are still in sync, the ciphertext is compared to the expected result. This additional standalone mode of the AES is only included to speed up trace acquisition for the security evaluation. On a real device, which does not allow direct access to the AES, the profiling step can be conducted by varying the key instead of the plaintext. This also randomizes the inputs to the S-box, but requires that a new key is set for every iteration, which leads to additional communication overhead.

4.1.2. Measurement setup

The parallelism of S-boxes and their desired equal leakage characteristics are crucial to the effectiveness of the original construction. Thus a high precision EM measurement setup is especially relevant. Our assumption is that the localization capability allows a spatial separation of the leakage of the individual S-boxes and the exploitation of even subtle differences in their characteristics. We use a state of the art high-end setup with a Langer ICR HH 100-27 100 µm diameter EM probe which is positioned about 10 µm over the decapsulated die surface (similar to the setup shown in Fig. 2.3). In addition to the built-in 30 dB amplifier of the probe, another Langer PA303 30 dB pre-amplifier

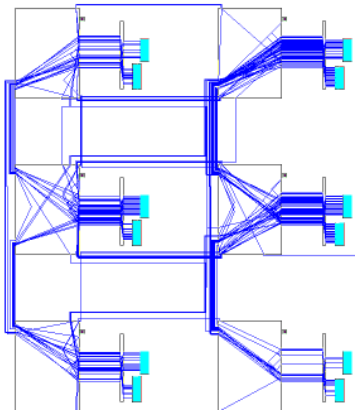


Figure 4.3.: Layout of one S-box in the Xilinx IDE.

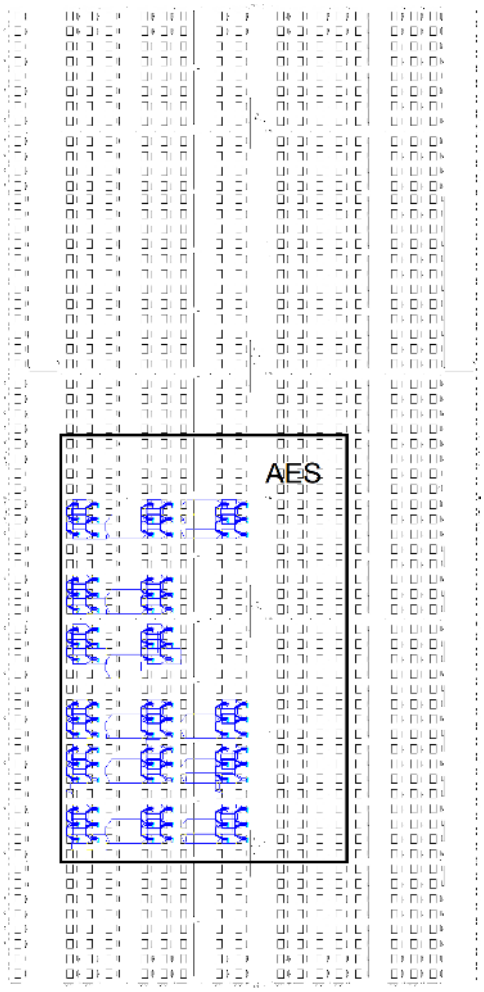


Figure 4.4.: Position of 16 S-boxes on the floorplan of the Xilinx Spartan 6 FPGA. The entire AES is placed within the black box.

is employed. A LeCroy WavePro 725Zi oscilloscope with 2.5 GHz bandwidth and a sampling rate of 5 GS/s records the measurements. The FPGA-based design is clocked at 20 MHz and this clock is synchronized to the oscilloscope. An X-Y-table is used to collect measurements on multiple locations over the die surface. The measurement positions are located within an area of about 2.8 mm by 2.8 mm, which covers most, but not all of the floorplan because the probe movement is limited by the bonding wires. This setup was chosen to examine the effectiveness of the algorithmic noise against a measurement setup with high localization capabilities. Therefore it allows for a worst-case analysis of the achieved security level.

4.2. Side-channel evaluation of CHES 2012 LR-PRF on Xilinx Spartan 6 FPGA

The goal of this section is to analyze the effect of the unique spatial localization capabilities of near-field EM measurements on the security of the LR-PRF. The idea is to isolate the leakage of an individual S-box by placing the probe at the location where this S-box's leakage is high. If this succeeds, then the algorithmic noise and carefully chosen inputs are rendered ineffective and an attack is possible by moving the probe from S-box to S-box and attacking them separately. Our analysis is thus split into three tasks: (1) the localization of the measurement locations with the maximum SNR for each S-box, (2) the profiling phase, and (3), the attack phase.

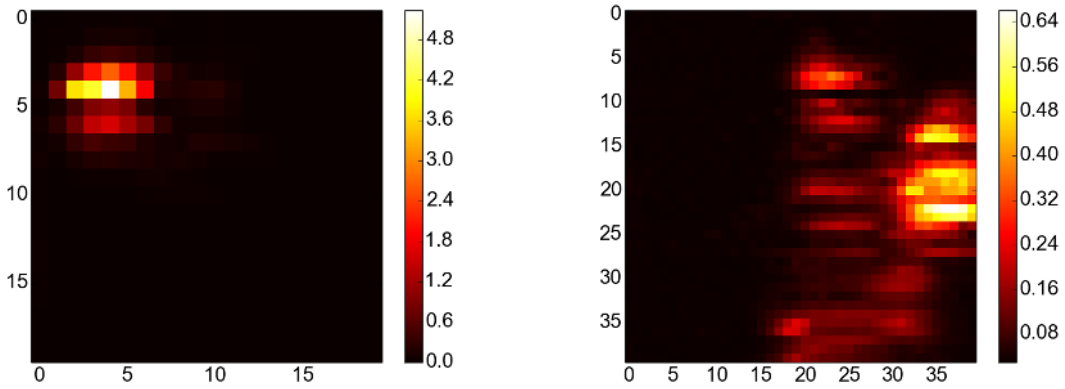
It is common practice to allow profiling for a meaningful implementation security analysis which represents a scenario where adversaries may use their own devices where they could choose keys for profiling. In this profiled setting, the adversary is able to compute *all* internal states of the implementation.

Both calculating the SNR and profiling the templates for individual S-boxes requires control over the inputs to the S-boxes. For evaluation purposes, we achieve this by using the AES in a regular block cipher mode where we can set the plaintext and key. In the real LR-PRF implementation the control over the inputs is very limited. The only way an attacker can observe all values at the S-boxes is by changing (and knowing) the key, which is only possible if a device with unconstrained access is available.

4.2.1. Identifying locations of interest

Finding the location with the maximum leakage for each S-box is the most time consuming task of the attack since it requires a full scan of the die surface. Considering that the measurement time grows quadratically when reducing the step size, we partitioned the measurement area in a grid of 20×20 for the unconstrained design and 40×40 for the dense design, which corresponds to a step size of 140 µm and 70 µm, respectively. We used a larger step size for the unconstrained design since we expect the logic of the S-boxes to be spread over a bigger area. At each location, we acquired 10,000 traces with the AES in OFB mode and random inputs. With our setup, the measurement takes roughly one day for the 20×20 grid and four days for the finer grid. At each location

we calculated the SNR for each of the S-boxes in the data path. The result is a trace of many SNR values (SNR trace) which we evaluated within the timespan where the first AES round is computed. In our case one clock cycle corresponds to 250 samples, and the interesting part, i.e. the part where there is activity after the clock edge, is around 50 samples (10 ns) wide. A detailed discussion of the SNR traces and the leakage behavior is given in Chapter 5, for now we are only concerned with finding the best possible attack. There are several options how to chose locations from this part of the SNR trace. We found that in some cases, the locations with the highest peak SNR value gave the best results, and in others, the locations with the highest mean SNR (calculated over the 50 samples in the interesting region of the SNR trace) performed better. In cases where those metrics gave different locations or were ambiguous due to multiple peaks of similar amplitude, we conducted the rest of the analysis on all such locations for this S-box and kept the best result.



(a) Unconstrained placement.

(b) Dense hard-macro placement.

Figure 4.5.: SNR heat maps for S-box #0 with different placements.

Using the SNR analysis, we were able to localize useful measurement locations for all S-boxes. Figure 4.5 shows one example SNR heat map of S-box 0 on the two different placements of the design with data complexity 2. An overview of the heat maps of all S-boxes is found in Figs. A.1 and A.2 in the appendix. Each colored pixel represents the peak SNR value of the SNR trace at that measurement location for this S-box. In both maps, regions with the highest SNR are clearly distinguishable and most likely correspond to the actual physical location of the logic of S-box 0. An important observation is that the SNR values of the design with the densely placed hard-macro S-boxes are - on average - by a factor of 2 smaller than the ones from the unconstrained placement. The average peak SNR of the S-boxes on the dense placement is 0.87, compared to 1.61 on the unconstrained placement. In the example shown in Fig. 4.5 the difference is even higher, with a SNR over 4.8 on the unconstrained placement and only 0.64 on the dense placement. In the case of dense placement, where SNR values are generally smaller, there are multiple locations which exhibit a relatively high SNR. As described earlier, we evaluated all such locations for the corresponding S-box in the attack instead

of choosing just one, which increased the measurement time of the attack.

4.2.2. Template attack with limited data complexity

Next, we mount template attacks on the S-box calculations of the first AES round at their respective locations. In two separate acquisition campaigns, we collected the profiling and attack traces. We performed profiling using the block cipher in OFB mode of operation instead of the LR-PRF mode. The attack traces, contrarily, were acquired in LR-PRF mode with limited data complexity, i.e., 16 for $n = 4$, and 2 for $n = 1$. For both the profiling and attack phases, traces are acquired at all previously identified locations. Therefore at least $2 \cdot 16$ sets of traces are collected (more if there is ambiguity regarding the best location due to similar SNR). During the attack each S-box is profiled and attacked at its own location. In practice, this requires moving the probe between the locations and restarting the measurement at each location. It is important to note, that this is always possible in case of stateless constructions such as the analyzed LR-PRF. The long-term key does not change, an attacker can always reset the entire device to restart measurements. After acquisition, the traces are cut to contain only the time span where the first AES round is calculated. To further reduce the number of samples included in the templates, we use LDA [28] as dimensionality reduction algorithm.

We compute full estimated Gaussian templates for each S-box and each of their S-box input values. As stated in Section 2.3, for LDA to be applicable, the traces belonging to one S-box are assumed to share a common covariance matrix, regardless of the input value. In this case, it suffices to calculate a single pooled covariance matrix for all templates belonging to one S-box. This gives a better estimate of the actual distribution and drastically reduces the computational effort for the template matching. Our experiments suggest that the assumption holds in our case and gave better results when using the pooled matrix compared to separate covariance matrices. Thus, all our presented attacks were conducted using pooled covariance matrices.

During the attack phase, the traces are matched against the templates in a template based DPA. Since we are using the pooled covariance matrix, we can make use of simplifications detailed in Section 2.4 and calculate the logarithmic score. To combine the score of multiple attack traces, we sum the scores and calculate the average. This results in a list with scores for each subkey candidate. Due to the design of the LR-PRF, the input controlled by the attacker only has a limited effect on the actual inputs to the AES encryption, i.e. few known bits determine the 16 byte inputs. This leads to an inevitable template mismatch, as each template for a byte of the target variable has been profiled while randomizing all other bytes. This is different from a regular template attack where the input bytes can typically be chosen randomly by the attacker. Hence, in a regular template attack the divide-and-conquer approach works better because each byte behaves independently of the others and the attacker may target specific bytes one at a time. When the data complexity is limited, this separation due to uncorrelated values is reduced. As a result, the individual bytes can no longer be attacked independently as they are affected by each others correlated leakage. A way to counteract this is to build larger templates that span multiple bytes, but this approach is not feasible because the

measurement complexity grows quadratically with the number of bits in the template. Due to this limitation in the attack, it is expected that correct key byte candidates are not determined without doubt and that an attacker has to try the most probable combinations until the full key is recovered. In order to calculate the remaining security level, we use key rank estimation as described in Section 2.5.

Table 4.1.: Estimated security levels after the attacks.

S-box placement	**Data complexity**	**Estimated security level in bit**
Unconstrained	2	20
Dense	2	48
Unconstrained	16	0
Dense	16	0

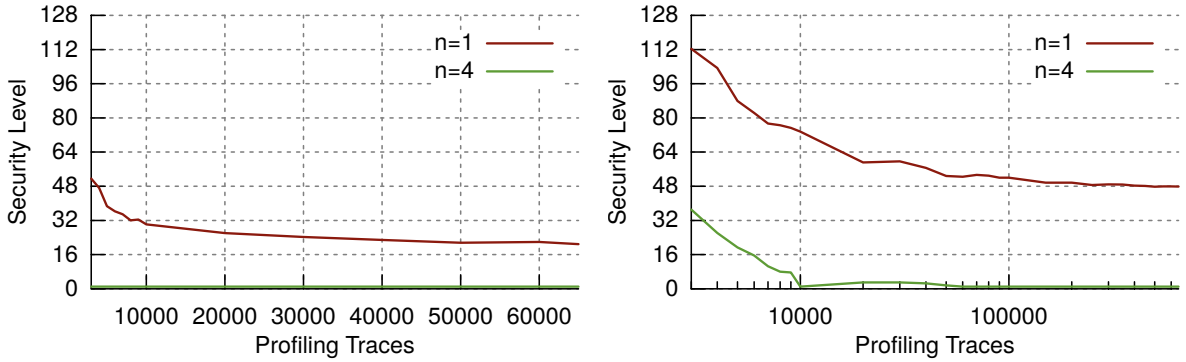
For the profiling phase, we used a maximum of 65,000 traces per location for the unconstrained design and 650,000 traces for the dense design in an effort to compensate for the lower SNR. During the attack, up to 100,000 traces were used per S-box. Table 4.1 summarizes the results of the attacks using all available traces. With a data complexity of 16 during the attack, all key bytes are successfully recovered (i.e. the security level is 0), regardless of the placement. This is a result which is similar to the findings of Belaïd et al. [6].

As expected, a data complexity of 2 leads to more secure results. Several subkeys are not ranked first and consequentially, a higher security level of 20 bit remains for the unconstrained placement case with data complexity 2. However, as an important result, this is an obvious insufficient level of security.

The dense design improves the security significantly and provides a higher security level of 48 bit compared to 20 bit. However, in all cases the achieved security level is insufficient, which is the main result of our investigations. This means that a minimum data complexity of 2 together with parallel S-boxes and carefully chosen inputs is not suited to achieve meaningful leakage-resilient constructions, at least under the present circumstances of a 45 nm feature size FPGA implementation.

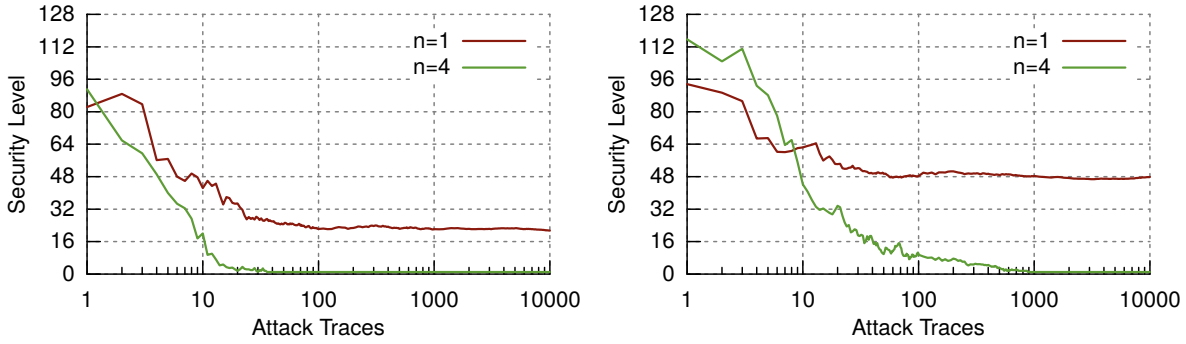
While the security level is established to be insufficient, an interesting question is, whether more profiling traces would further improve the attack, or whether the lower bound is reached. We repeated the attack with different numbers of profiling traces while using all available attack traces. The results for all designs are shown in Fig. 4.6. It can be noted that the gain of using more traces for profiling diminishes and the security levels seem to approach a lower bound at about 20 bit for the unconstrained, and about 48 bit for the dense design. Note that the plot for the dense design has a logarithmic scale since it takes significantly more traces for the security level to settle. We conclude that increasing the number of profiling traces even further seems useless and that the efficiency of the attack is in fact limited by the leakage-resilience, and not by insufficient

profiling. In other words, we expect that other uncorrelated noise sources are averaged out sufficiently.



(a) Unconstrained design. (b) Dense design.

Figure 4.6.: Evolution of security levels with varying number of profiling traces and maximum number of attack traces.



(a) Unconstrained design. (b) Dense design.

Figure 4.7.: Evolution of security levels with varying number of attack traces and maximum number of profiling traces.

In a similar manner, we investigated the number of traces required for the attack. In a real-world scenario, adversaries may have full access to one device for profiling, but limited access to the attacked device. Figure 4.7 shows the influence of the number of attack traces on the security level when using templates built from the maximum number of available profiling traces (65k and 650k, respectively). As an interesting observation, we report that the security level seems to reach its lower bound after only about 100 and $1,000$ attack traces for the unconstrained and dense design, which is a surprisingly low number. Hence we conclude, that the remaining entropy is actually caused by the leakage resilience of the construction, and not by a lack of traces.

To verify the efficiency of the leakage-resilient construction against regular power attacks, we also conducted a template attack where we measured the global power consumption over a resistor in the power line with a differential probe. For increased SNR,

all capacities were removed from the board. Despite using 1,000,000 profiling traces, the attack fails to result in any significant reduction of the security level. Interestingly, the correct subkeys were not even ranked highly but instead were distributed evenly across the subkey list. This is far from optimal for the attacker, ideally the correct subkeys would be ranked in the first 16 locations in all subkey lists and leave only the permutation complexity for the enumeration of the whole key. For the case of *unlimited data complexity*, i.e., when using the AES in a standard mode of operation like OFB mode, we report that an univariate CPA using the Hamming distance leakage model already succeeds with $20,000$ traces. The Hamming distance model was chosen as the implementation does not exhibit leakage for the Hamming weight model. Even though this aspect was not the focus of our research, this discrepancy is an encouraging result when adversaries are limited to global (power) attacks.

4.3. Summary

Our results demonstrate that the achieved security level of the CHES 2012 AES-based LR-PRF employing minimum data complexity and S-box parallelism is insufficient in the high precision EM scenario, at least in cases similar to our FPGA with 45 nm feature size. In particular, we were able to isolate the leakage of individual S-boxes and attack them separately using LDA-based, profiled, multivariate attacks. This proves that with this setup 'equally leaking' and 'correlated algorithmic noise' assumptions no longer hold. We were able to completely recover the correct key for all designs with data complexity 16. A data complexity of 2 proved to be more resilient, but we were still able to reduce the security level to 20 bit and 48 bit for the unconstrained and dense placement, respectively. For power analysis attacks, in contrast, we found that the countermeasure works efficiently.

These results also extend to the ASIACRYPT 2016 LR-PRF if implemented on similar FPGA platforms. As discussed in Section 3.2.2, the plaintexts which are encrypted inside the GGM tree are secret and generated by an LR-PRG. The attack vector is therefore shifted to the LR-PRG. Attacks on the LR-PRG are always 2-limited by construction and thus the outcome corresponds to the results for data complexity 2 in case of the CHES 2012 LR-PRF. In conclusion, both proposals are vulnerable to high precision EM attacks if no further measures are taken.

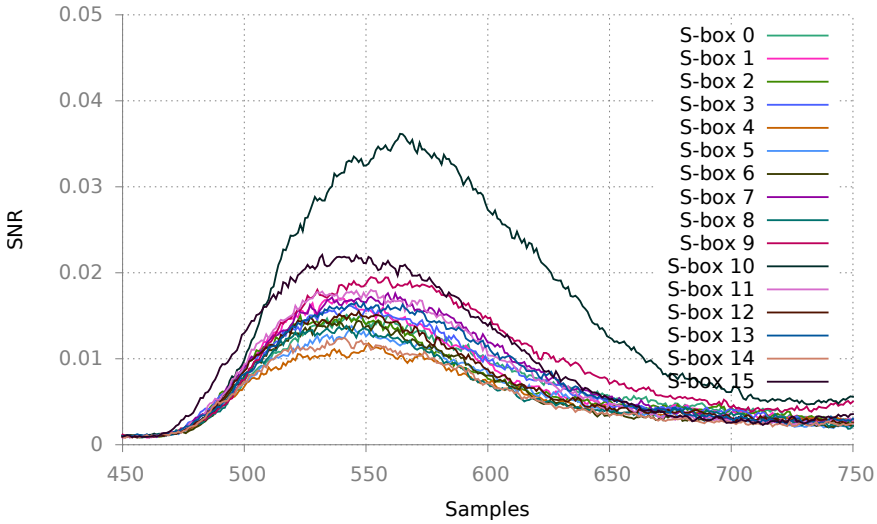
5. Understanding how high precision EM attacks break LR-PRFs

The previous chapter provided results of a profiled attack against the CHES 2012 LR-PRF and by inference the ASIACRYPT 2016 LR-PRF and established that they cannot withstand high precision EM attacks on the analyzed FPGA platform. This chapter takes a deeper look at the effects that enable such a successful attack. We compare the leakage behavior in case of power and EM measurements and the distribution of the leakage across the time samples in Section 5.1 and 5.2. Additionally, we analyze the leakage distribution before and after LDA transformation. The main result is, that EM measurements are capable of separating the leakage of different S-boxes in part due to their high temporal resolution and not only, as previously suspected, due to the spatial positioning of the probe over the chip. This is a challenging result for hardware designers, because tiny timing differences in signal propagation are hard to control, especially on devices with limited control over the hardware layout such as FPGAs. To further explore the effect of parallel S-boxes, we provide results from simulated attacks for different leakage models in Section 5.3. We find that for low data complexities, all evaluated leakage functions of the S-boxes lead to very similar results. In those cases, designers can disregard the 'equal leakage' design paradigm and instead place S-boxes as densely as possible. Dense placement should lead to more temporal (and spatial) overlap of the S-box leakages and to a degree hinder temporal separation through EM probes. However, for strong security guarantees against high precision attacks it seems unavoidable to improve the construction. Thus, we use these insights and propose an improved LR-PRF in Chapter 6 that uses additional key entropy to compensate for losses due to side-channel attacks.

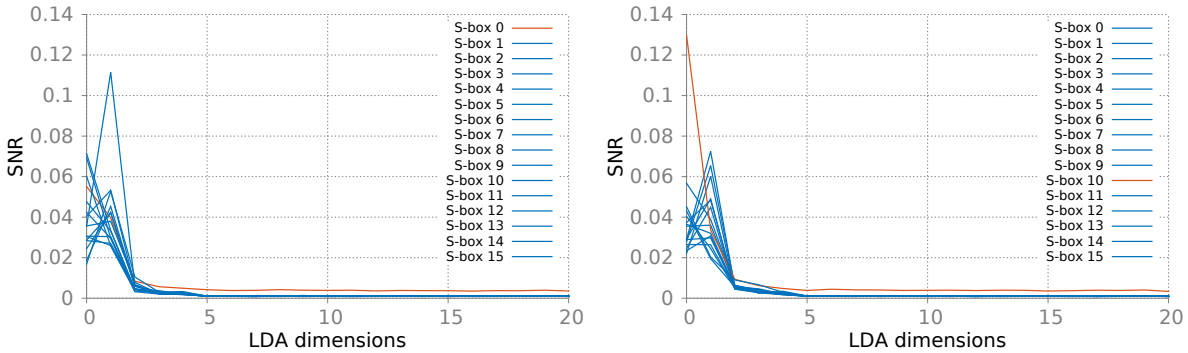
5.1. Leakage resilience holds with current measurements

Before analyzing the reasons why EM measurements break leakage-resilient constructions, we look at the case of current measurements. For such measurements, we reported in Section 4.2.2 that template attacks are not successful and do not lead to a significant reduction in key entropy. This means that the algorithmic noise from the parallel hardware and the limited data complexity work as intended. In other words, the algorithmic noise from the respective other S-boxes makes attacks on individual S-boxes infeasible.

Naturally, with current measurements, there is only one measurement location and set of traces. Figure 5.1a shows the SNR traces, calculated as explained in Section 2.3, of all 16 S-boxes around the time where the S-box function of the first AES round



(a) SNR in current measurement.



(b) SNR after LDA transformation in subspace calculated for S-box 0 in red. Others blue.

(c) SNR after LDA transformation in subspace calculated for S-box 10 in red. Others blue.

Figure 5.1.: SNR of S-boxes before and after LDA transformation.

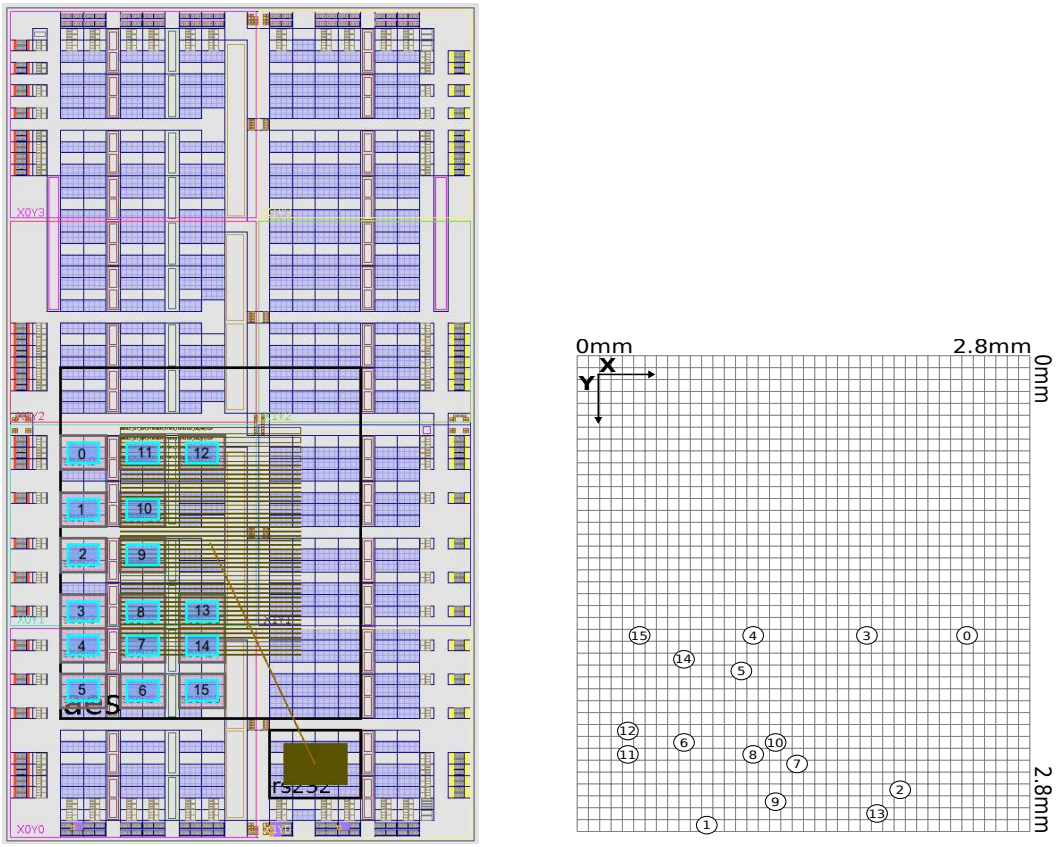
is computed. One clock cycle takes 250 samples and the positive edge of the clock approximately occurs at sample number 460. The signals of all S-boxes stretch over a time period which is almost the entire clock cycle. This is expected with such kinds of measurements due to the high amount of parasitic capacitances and inductivities which low-pass filter the signal. Most importantly, we note that the signals of most S-boxes are uniform in shape and amplitude and that the maximum SNR values of about 0.01 to 0.02 are relatively low. The only outlier is S-box 10 which shows similar shape but slightly higher SNR of 0.035. The reasons for this are unknown and most probably based on specifics of the placement in the FPGA fabric. We did not further investigate since the attack was ultimately not successful despite the increased SNR. The fact that all S-boxes emit their signal at about the same time means that every S-box will effectively produce noise for every other S-box, thus, leading to the generally low SNR. This is exactly what the construction was meant to achieve.

Furthermore, we inspect the SNR after LDA transformation for two cases. One case is S-box number 10 which exhibits the highest SNR values as observed from Fig. 5.1a. The other case is S-box number 0, which is one of the S-boxes from the group that exhibits similar SNR values. Figure 5.1b and 5.1c depict the SNR values after LDA for these two cases. The signal of the targeted S-box is plotted in red, while the signals of the other S-boxes are plotted in blue. It is important to note that for each individual figure, first the LDA transformation that fits the signal of the targeted S-box is calculated and applied to the traces. Then the SNR of all S-boxes is calculated in that subspace. The expectation is that in this subspace the SNR of the S-box for which the LDA transformation was calculated is maximized and other signals are suppressed. The case of S-box 0 in Fig. 5.1b is representative of the most frequent situation and shows that the signals of all S-boxes are in a similar range. Hence, the targeted signal of S-box 0 is similar or even lower than the signals of the other S-boxes which produce noise. Even the single best case of S-box 10 in Fig. 5.1c shows that the signals of the other S-boxes are relatively high (at approximately $\frac{1}{2}$ to $\frac{1}{3}$ of S-box 10) which also leads to significant noise for this best case. With respect to the LDA transformation this means that the subspace fails to separate signal contributions of different S-boxes. The reason can be observed from Fig. 5.1a: All S-boxes leak at the same time samples, thus rendering it impossible for the LDA transformation to find a linear combination that separate them. This explains why attacks in such cases are unsuccessful, i.e. the algorithmic noise works as intended.

5.2. Leakage resilience fails against high-resolution EM measurements

The goal of this section is to explain why the parallelism of S-boxes for leakage resilience fails when using localized EM measurements. A natural assumption is that the high precision setup would lead to measurements where, at the location of a specific S-box, only this S-box exhibits a high SNR while all others exhibit negligible SNR. We show that this is rarely the case, therefore we need another explanation.

We use the EM grid scan measurements from Section 4.2 and select the LOI for each S-box based on the SNR as described previously. Out of the four different designs we evaluated earlier we now restrict analysis to the case of data complexity 2 and dense hard-macro placement. This design configuration is the most resilient to attacks, but still vulnerable. Figure 5.2 shows the physical placement locations of the S-boxes on the FPGA floorplan in Fig. 5.2a and the best measurement locations of the same S-boxes (LOIs) in Fig. 5.2b. The measurement positions are shown as a grid within a quadratic area of about 7.8 mm^2 in between the bonding wires of the decapsulated FPGA. It is hard to match the two areas exactly, but the measurement grid (right) covers most of the floorplan (left). Since the positioning of the probe is limited by the bonding wires, a small margin around the edges cannot be covered. However, we cannot determine exactly which areas of the floorplan are left out and how well the floorplan actually represents



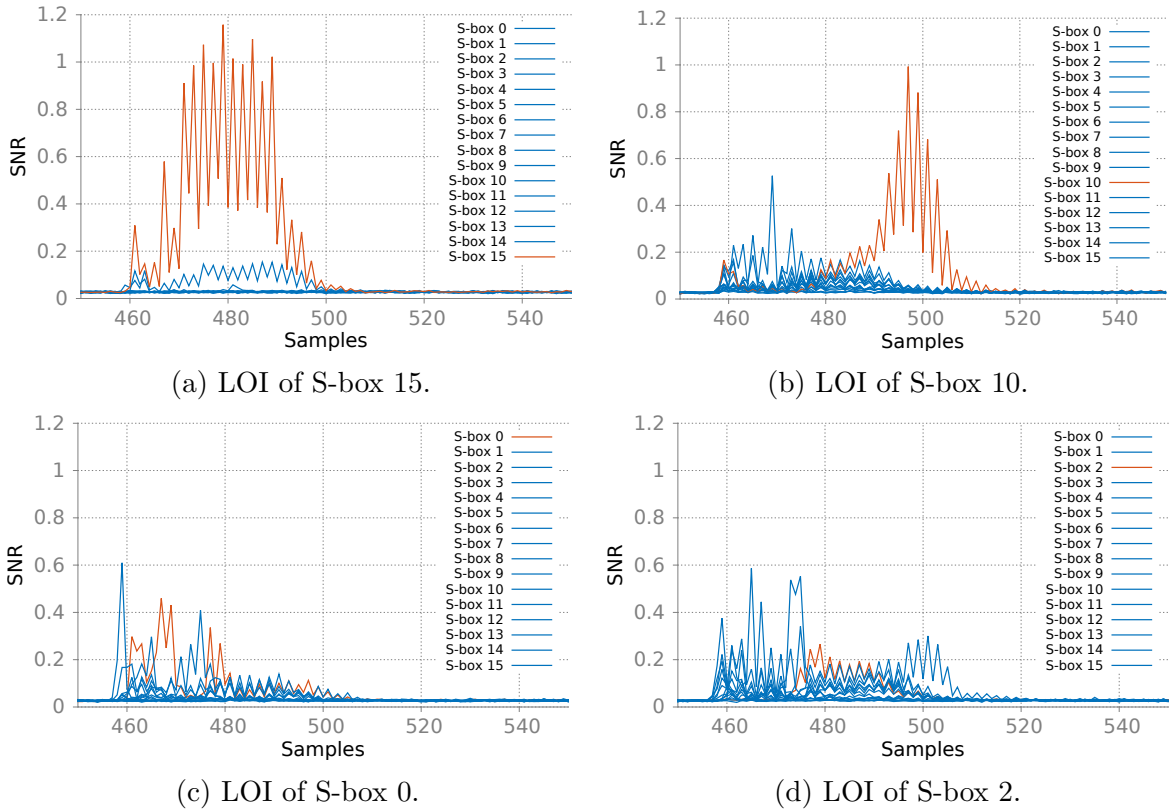
(a) Placement on FPGA. (b) Measurement locations (LOIs).

Figure 5.2.: Placement of S-boxes compared to resulting measurement locations.

the physical layout and hence refrain from overlaying the grid over the floorplan. What is interesting while comparing the two figures is that apart from a general similarity that all S-boxes are situated and measured on the lower left, there is no reasonable placement-to-measurement correspondence. This already hints that we measure signals at the LOIs which have propagated through the circuit from their origin in the S-box, e.g. through the power grid.

Figure 5.3 depicts the SNR of all S-boxes at four LOIs, which have been selected to be best for S-boxes 15, 10, 0, and 2. The four shown cases are representative of the 16 LOIs in total which are given in the appendix in Figs. A.3 and A.4. The figures each show the SNR of the targeted S-box in red and the SNR of all other S-boxes in blue. As a first observation it should be noted that all detectable signals extend over a significantly shorter time period compared to the power analysis. Specifically, they extend over about 50 time samples which corresponds to a time span of 10 ns. This is short compared to the clock cycle duration of 50 ns (250 samples). In fact, it is close to the critical path delay of 15 ns reported by the synthesis tool. This is similar to the findings of Heyszl et al. [43] and confirms that there are only a few parasitics in the measurement chain.

As an important observation, the SNR values in Fig. 5.3 are very high compared



(a) LOI of S-box 15.

(b) LOI of S-box 10.

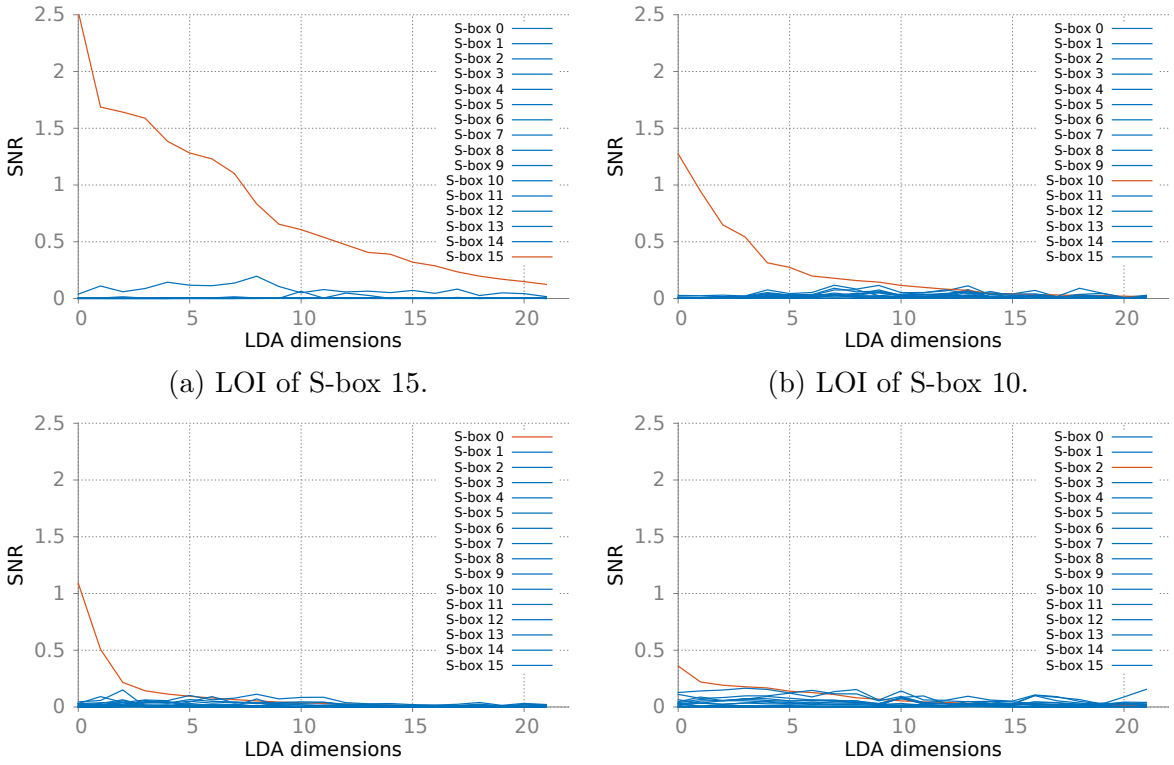
(c) LOI of S-box 0.

(d) LOI of S-box 2.

Figure 5.3.: SNRs at four LOIs of targeted S-boxes (red). Others in blue.

to the results from the current measurement in Fig. 5.1a. The SNR reaches values of 1.2, compared to the maximum of 0.035 which was observed for current measurements. Figure 5.3a depicts the situation of S-box 15 which confirms the assumption, that an isolation of S-box signals can, in cases, be achieved through location-dependence. The SNR of S-box 15 is high while the SNR of the other S-boxes is minimal. The case of S-box 10 in Fig. 5.3b is very different. The SNR of this S-box is again isolated, but only at a certain and precise time. There are times (sample points), where the SNR of other S-boxes is also significant. But at the time samples where the SNR of S-box 10 is highest the others tend to zero. Figure 5.3c and 5.3d depict more cases where there is a strong overlap of signals from different S-boxes. However, again, at certain time-samples the SNR of other S-boxes is small compared to the SNR of the targeted S-box.

In order to make visual inspection easier, we provide the SNR after LDA in Fig. 5.4. It can generally be noted how LDA compresses the available SNR into the highest dimensions. Unsurprisingly, in cases where the separation, in terms of relative proportion of targeted signal to the other signals, before the LDA transformation has already been high, this becomes significantly more visible after LDA. Figure 5.4a depicts S-box 15 and Fig. 5.4b depicts S-box 10. The high SNR values of the targeted S-boxes, 2.5 and 1.3, and very low SNR values of the other S-boxes in the first dimensions are significant and lead to the assumption that attacks on these S-boxes will succeed with very high



(a) LOI of S-box 15.

(b) LOI of S-box 10.

(c) LOI of S-box 0.

(d) LOI of S-box 2.

Figure 5.4.: SNRs after LDA at four LOIs of targeted S-boxes (red). Others in blue.

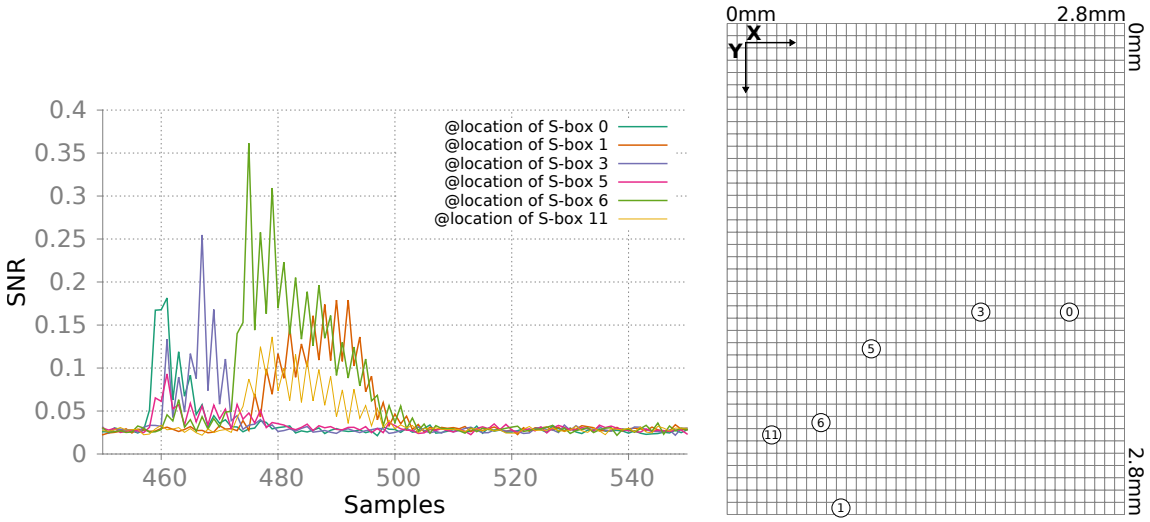
probabilities. However, also for S-box 0 in Fig. 5.4c the proportion of its signal to other signals seems exploitable in this view, despite the overlap in the time domain. Even for the case of S-box 2 in Fig. 5.4d the LDA-transformed SNR hints that there is exploitable SNR.

SNR over different locations and time

In order to examine the leakage behavior of one S-box when observed at different measurement locations, we consider the SNR of S-box 6 as an example. Specifically, we analyze the SNR at several LOIs of other S-boxes. The selected positions are depicted in Fig. 5.5b. The SNR of S-box 6 at those positions is shown in Fig. 5.5a. It can be observed, that the SNR reaches higher values as observed in current measurements at all positions. Depending on the location, it appears in different amplitude and different shape over time. In other words, one S-box exhibits very different leakage behavior when observed from different measurement locations.

Discussion

The most important observation from our analysis is that the leakage signals of different S-boxes are very different when observed with high precision, low-parasitic EM



(a) SNR of S-box 6 at different locations. (b) Measurement locations.

Figure 5.5.: SNR of S-box 6 at different locations.

measurements. This difference is especially remarkable since the S-boxes were carefully designed with equal internal structure and routing. The leakage signal of different S-boxes is in fact detectable at different time samples within a very short time range. To the best of our knowledge, the reasons are within-circuit signal propagation delays, or race-conditions. Hence, depending on circuit differences and depending on the position of the measurement relative to the source of the signals which propagate through the circuit, the timing of different S-boxes is different. As an important insight, we derive that a successful isolation of S-box signals is partly due to the timing of their propagation over the circuit. Hence, the success of attacks on parallel constructions should not be solely attributed to spatial isolation. In fact, a combination of spatial and temporal separation leads to exploitable leakage with distribution over time being dependent on the measurement location. In addition, we found that the leakage behavior of an S-box changes significantly if the measurement probe is moved to another location.

An interesting question is whether these effects are captured by the leakage models that are fundamental to the theoretical security of leakage resilient constructions. If not, then the modeling is obviously incapable of matching the reality and proofs conducted under such assumptions would not bear any meaning. In this matter we recall the OCL model of Micali and Reyzin as an example. The theoretical leakage model has the measurement apparatus as an input to the leakage function. Therefore localized EM attacks can be reflected in the model by having the position of the probe as an input to the leakage function. Then, the result of the leakage function changes according to positioning. We conclude that it is in general possible to capture the observed effects in formal leakage models. Even so, this example shows the dangers of relying on wrongly made assumptions about the leakage[6].

[6] As far as we know there is no formal security proof that takes the equal leakage and correlated noise into consideration, the discussion in [65] is more informal.

However, our results make us question if equally leaking S-boxes can ever be achieved in practice. The hardware design would need to be constructed in a way where at all measurement locations every S-box exhibits the same leakage [7]. Such a design is only imaginable in a technology where the feature size is small enough such that no differences between S-boxes can be resolved by the EM probe. Additionally, all wires to and from the critical logic need to be densely packed and the timing strictly controlled to guarantee that all leakage is in sync and overlaps. This seems hard to achieve with an application specific integrated circuit (ASIC) design and even harder on FPGA devices where the internal routing options are limited by the given function blocks of the configurable logic.

5.3. Effect of different leakage functions in cases with low data complexity

Our EM measurements clearly show that the leakage of the S-boxes is separable because their signals do not overlap enough to generate effective algorithmic noise. One way to increase signal overlap is to place the design closer together. The area occupied by the AES in the evaluated design is dominated by the hard-macro placement of the S-boxes, which was originally used with the intent to achieve similar leakage functions. If the individual S-boxes were placed without this constraint, they could be placed interleaved and packed much tighter. This would inevitably violate the equal-leakage design paradigm, but lead to more signal overlap.

Previous contributions on the CHES 2012 LR-PRF [65] and ASIACRYPT 2016 LR-PRF [66] also argue the security based on this equal-leakage assumption. Hence, our question is, whether equal leakage is really required in this context or if we can sacrifice it in exchange for tighter, interleaved placement. In this section, we show that S-boxes do not necessarily need to have equal leakage characteristics when the data complexity is low.

Medwed et al. [66] simulate a profiled univariate template attack on noise free traces where the leakage of each S-box is exactly the Hamming weight of its output. The leakage trace, i.e. sample, since it has only a single point, is the sum of the leakage of all S-boxes. This setting represents the worst case for an attacker since the signals from all 16 S-boxes perfectly overlap, hence, produce noise. We extend this simulation by considering more realistic leakage models, which consider that due to the hardware design itself and the positioning of the probe relative to the internals some bits contribute more to the leakage signal than others. We modify their simulation by using different probability mass functions for the leakage of the different S-boxes, which we randomize such that they deviate from the Hamming weight leakage. In our model, every S-box possesses its own leakage function which is in contrast to Medwed et al. where all leak equally. Similar to the Hamming weight model, we assume that the total leakage of the S-box is the sum of the contribution of all the bits of the output value. However, for each S-box and each

[7] It is not necessary that an S-box exhibits the same leakage behavior regardless of the location, as long as *at any given location* all S-boxes leak the same.

of its output bits, we draw the value from a discrete normal distribution $\mathcal{N}(100, \sigma^2)$. We increase the leakage's codomain so that all distributions and calculations can remain discrete, otherwise the computational cost becomes prohibitive. This model is realistic in the sense that we expect the leakage to be somehow dependent on the bit values, albeit some bits will have a stronger and different impact than others. As a corner case, we also perform an (unrealistic) simulation where we randomly assign leakage values to S-box output values.

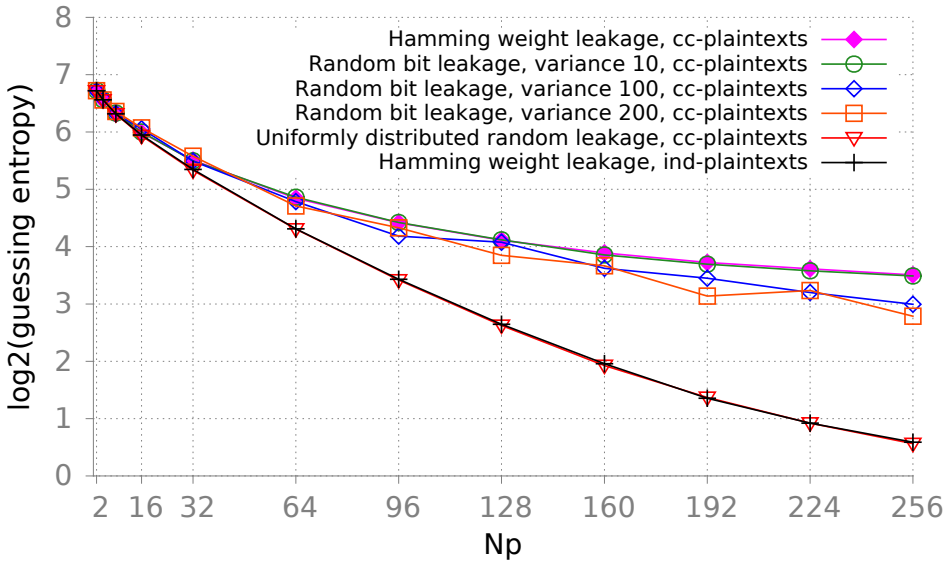


Figure 5.6.: Remaining guessing entropy after simulated attacks on one key byte with different leakage models (cc=carefully-chosen, ind=independent).

We simulate attacks on single key bytes and report the remaining security level in terms of guessing entropy. On average, an uninformed attacker guessing randomly will find the correct key byte after 128 tries, i.e. the maximum guessing entropy is 7 bit. Figure 5.6 depicts the guessing entropy of one key byte after the simulated template attack in relation to the number of observable plaintexts N_p for different simulations. We perform 100,000 simulations using random keys per data point and average the guessing entropy. For comparison, we include the equal-leakage Hamming-weight model with both carefully-chosen plaintexts, where all bytes are equal, and randomly-chosen plaintexts with independent bytes.

The curve of the equal-leakage Hamming-weight model forms the upper boundary of the guessing entropy; this is the best we can expect and equivalent to the experiments of Medwed et al. In general, the guessing entropy goes down with the number of different plaintexts that an attacker can observe. If we randomize the bit leakage, i.e. make the leakage behavior increasingly dissimilar, then the guessing entropy reduces at a faster rate. While the difference for variance 10 is minor compared to the Hamming-weight model, for variance 100 and 200 this effect becomes obvious. The extreme case of this is the uniformly distributed random leakage, which is in line with the curve of the Hamming weight model with randomly-chosen plaintexts. This is expected because if either the

S-box input or the resulting leakage is random, then there can be no correlation between the leakage of S-boxes and, thus, no correlated algorithmic noise. That is the best case for an attacker and leads to the lowest guessing entropy. We can observe and conclude that, for very low data complexities (e.g. 2 or 4), the leakage model does not make a difference on the security of parallel constructions.

5.4. Summary and design recommendations

Our experiments and analyses clearly show that state-of-the art EM measurement equipment is able to separate signal contributions of individual S-boxes from parallel FPGA implementations. We investigated the reason for this and derive that the combination of spatial and temporal separation leads to exploitable leakage.

For parallelism to work in the intended way, the S-boxes' leakage should be small and not separable in the time or space domain to achieve security against localized EM attacks. This seems hard to achieve on certain FPGAs because of the limited influence that the hardware designer has on the immutable internal structure of the building blocks and the routing options. While further investigation of the concrete design and layout of the hardware implementation seems possible, we are pessimistic about its benefit. The reason for this is, that even if S-boxes could manually be placed in a smaller area, and one could argue that a location-dependent isolation may be impossible, the timing of signals of different S-boxes may still be different, allowing an isolation of said signals over time. Moving to platforms with smaller feature sizes and tighter integration could help to realize designs with sufficient overlapping leakage. In fact, we present an analysis on a 28 nm FPGA where the AES implementation does resist attacks with data complexity 2 in Section 7.4.1. Yet, this approach is not applicable when dealing with existing platforms or retrofitting side-channel protection to devices in the field.

On a more optimistic note, we found that with limited data complexity it does not matter if the leakage behavior of the S-boxes is equal. This gives hardware designers more freedom when placing the design since no effort has to be made to craft S-boxes with similar leakage functions. Hence, when implementing an LR-PRF on a Xilinx Spartan 6 FPGA or similar device, as a design recommendation we state:

1. The data complexity should be limited to the minimum of 2.
2. Parallel S-boxes should be concentrated and densely packed, while interleaving the S-boxes with no regard for their individual layout.

In this way the signals of at least a subset of the S-boxes should overlap and cause as much algorithmic noise as possible. This should be sufficient to reach acceptable security levels for this part of the construction so that the improvement presented in the next section can leverage on this to achieve a high overall security level. Nevertheless, it seems unavoidable to perform practical investigations, such as the ones described here, to ensure that the algorithmic noise is effective. We do not expect that any model of the hardware design is able to adequately emulate the EM emanation in order to predict the measured leakage signal.

6. Improving the security of LR-PRFs

In this chapter we use the insights that we gain from the analysis in Chapter 5 and propose an improved version of the LR-PRF scheme that resists high precision EM attacks. Section 6.1 describes the proposed design changes. In Section 6.2 we discuss the side-channel security of the construction by analyzing the relevant attack vectors. We find, that security against key recovery attacks can be achieved by introducing one or more additional keys to 'refill' the entropy. The introduction of the additional key(s) is done using a 2-PRG in a way that does not create new attack vectors nor increase the data complexity for an attack. The only requirement is that the underlying block cipher maintains a certain amount of key entropy after attacks on one iteration of the 2-PRG, i.e. attacks with data complexity 2. Depending on the amount of entropy left, one or more additional stages and keys need to be added.

6.1. Improved design with refilled key entropy

There are two options to improve existing LR-PRFs with respect to high precision EM attacks: trying to prevent the loss of entropy with increased hardware design efforts (placement, routing and timing constraints) or compensating the lost entropy by adding extra key-material. As argued in Chapter 5, it seems hard to design a device in which all S-boxes leak perfectly synchronous and where S-boxes cannot be separated spatially or temporally.

Instead, we propose to modify the ASIACRYPT 2016 LR-PRF [66] (see Fig. 3.5) so that additional key entropy is added to compensate the entropy loss when the construction is subjected to high precision EM attacks. We specifically propose to use their construction with two or more long-term keys instead of one, depending on the amount of entropy loss. Similar to the original construction, the unknown inputs are generated using 2-PRGs. However, we do not use them as LR-PRG (see Fig. 3.6), but concatenate one or more 2-PRG as shown in Fig. 6.1. Multiple stages of the 2-PRG can be used to further increase the entropy. In that case, another new key is introduced with each such stage. An additional key is used for the subsequent GGM tree.

Formally, we construct an LR-PRF $F_k(x)=y$ with $k=(k_0,\ldots,k_i,k_{PRF})$ where $i \geq 0$. Consequently, the minimum required key length with $i=0$ is 256 bits in case of AES-128. Our proposed modified construction is depicted in Fig. 6.1. The initial 2-PRG stage uses *known* inputs p_0 and p_1, valued 0^{128} and 1^{128}, and encrypts them with key k_0. This is the part of the construction, where due to the reasons explained in Chapter 5, parts of the key entropy will be lost in a side-channel attack. Depending on the quality of the implementation, hence, the amount of lost entropy, we then use c_0 and c_1 as either:

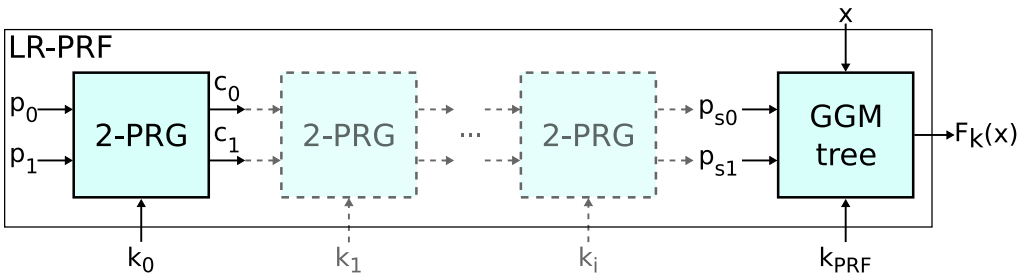


Figure 6.1.: Improved LR-PRF construction, dashed parts are optional.

(1) *unknown* plaintexts in subsequent iterations of the same 2-PRG stage, while each time introducing a new key $k_1, \ldots, k_i$ to further increase the entropy, or (2), as the *unknown* inputs p_{s0} and p_{s1} to the GGM tree. This GGM tree is configured with data complexity 2, i.e. two possible branches in each stage and 128 iterations to process an input x with 128 bits (see left side of Fig. 3.3). In the end, an additional whitening step is carried out where p_{s0} is encrypted regardless of the input. This prevents attacks on the output of the last stage of the GGM tree with data complexity 2 which could directly reveal the GGM tree key.

The idea is that the remaining key entropy of the 2-PRG stages, which is contained in p_{s0} and p_{s1}, carries over to the subsequent unknown-input GGM stage and hinders an attack on k_{PRF}. As argued in [66], attacking the construction with unknown inputs would require second order attacks and there is no straightforward way of testing key candidates. A potential attacker would first need to learn p_{s0} and p_{s1} before being able to launch an attack on k_{PRF} using first order DPA. A similar reasoning applies to all potential 2-PRG stages which use unknown inputs as well. How many stages and keys are needed depends on the leakage of the circuit and has to be evaluated through laboratory analysis. Conveniently, the number of necessary repetitions of the 2-PRG stage can for instance be a matter of configuration after the evaluation of a concrete hardware implementation. We expect that for many designs (as the one we analyzed here) one 2-PRG stage is sufficient because no additional entropy is required and the security level is already raised to an acceptable level. However, the overhead of adding stages lies solely in key memory and execution time. The entire construction can be implemented using only a single AES core for the GGM tree and the 2-PRG stages.

Note that this construction does not allow to increase the data complexity of the GGM tree stages for more efficiency. The reason is that the generation of more than two unknown plaintexts is not possible without losing additional entropy. Consider the LR-PRG used in the unknown-inputs LR-PRF proposal which iterates the 2-PRG multiple times, using c_0 as key for the next iteration and returning c_1 as output (see Fig. 3.2). Since the plaintext inputs are always known, attacks can be launched on *every iteration* and the resulting key candidate lists can be matched across the stages. Because of this, we accept limited efficiency in exchange for improved security.

6.2. Security discussion

The security of the construction is discussed along two major attack paths which connect in the middle:

1. The first attack path targets the 2-PRG with known inputs, which is the first part in Fig. 6.1. This is where we provided an improvement to cope with the loss of entropy due to our findings, and explain how this additional key entropy increases the overall security level.

2. The second attack path targets the GGM tree in Fig. 6.1, or more generally, all 2-PRGs with unknown inputs within this tree, as well as in the optional part in Fig. 6.1. Regarding this part, we revisit the argumentation of Medwed et al. [66], and argue that a recent attack on secret inputs from Unterluggauer et al. [99] can be reduced to the same case.

6.2.1. Part 1: Mitigating the loss of entropy in the 2-PRG

An attack on the first part, i.e. the 2-PRG with known inputs, has been shown to reduce the key entropy of k_0 in Fig. 6.1 to lower levels than originally expected. We denote the remaining key entropy of k_0 in Fig. 6.1 as 2^e. In the attack on an FPGA implementation provided in Chapter 4, this amounts to $\approx 2^{50}$ which is within enumeration levels. Hence, it requires an improvement because k_0 had been the single source of long-term key entropy.

With our tweak, the first 2-PRG with *known inputs* is followed by either the GGM tree or one or more 2-PRG stages with *unknown inputs and additional key entropy*. For this analysis the first stage of the unknown-inputs GGM tree can be seen as a 2-PRG with unknown inputs and key k_{PRF}. The reason is that inside the GGM tree each stage is similar to the 2-PRG because it consists of two possible block cipher encryptions[8]. In that sense the subsequent GGM tree stages can also be seen as concatenation of 2-PRGs with different keys, which are, however, all depending on k_{PRF} and thus add no entropy. Therefore they do not add additional side-channel security and we can ignore them. This simplifies the analysis since we can view the entire construction as a concatenation of two or more 2-PRG where each uses its own key.

With our modifications, an attacker has no way of verifying key candidates resulting from an attack on the known inputs 2-PRG since the outputs are not accessible. Instead, the attack must continue along the chain of 2-PRGs where it encounters new key entropy (at least additional 128 bit). Contrary to the first 2-PRG, all later 2-PRGs operate on unknown inputs. A valid strategy for an attacker is to attack the first 2-PRG and enumerate possible candidates for k_0. Starting with the most probable candidate, the outputs of the stage are calculated. For each candidate, an attack on the next stage is mounted using the calculated outputs as hypothetical inputs.

[8] Note that during an evaluation of the LR-PRF, only one of the plaintexts is actually encrypted depending on the bits of the input. However, if we assume that an attacker can manipulate the input then both encryptions can be observed by flipping the corresponding bit.

The attack on the next iteration has to be repeated for 2^e candidates so that, by expectation, the guess of k_0 is correct in one of the attacks. The attacker has, however, no means of detecting whether the correct k_0 has been used and must continue until the output of the GGM tree to verify key guesses. The attack on the second 2-PRG will, hence, add the same amount of entropy, i.e. 2^e out of the full additional key entropy. As a result, after those two stages, a total entropy, or attack complexity, of $2^e \cdot 2^e = 2^{2e}$ is achieved. This can be generalized over n 2-PRG stages which results in a total remaining entropy of 2^{ne}. Obviously, the entropy of the construction is upper bound by the length of k_{PRF}, i.e. 128 bit. The value of e, and, consequently, the number of required stages, is highly dependent on the exact implementation and can be estimated only by conducting an attack on the final device.

As a note, the attack on the second 2-PRG and, optionally, subsequent 2-PRGs, differs in that the plaintexts are not carefully chosen but random. Hence, there is no correlated noise of S-boxes. But the simulation in Section 5.3 shows, that with data complexity 2, the expected guessing entropy per key byte is practically the same and we can disregard this difference.

6.2.2. Part 2: Security of the unknown-inputs GGM tree

Next we discuss attacks on the GGM tree. Unterluggauer et al. [99] describe, how the Unknown-Plaintext Template Attack [41], which is a second-order profiled DPA, can be modified to fit the case of leakage-resilient constructions with unknown inputs by switching the role of key and plaintext. Their goal was to retrieve unknown plain data from encryptions with frequent key updates. This directly applies to the unknown-inputs construction in [66]. For the attack, they require full control over a device for profiling and build two sets of templates: one for the key bytes and one for the outputs of the S-box transformation. During the attack phase, the key and the outputs of the S-boxes vary while the plaintext is constant. Since the combination of the two templates determines the corresponding plaintext, a differential attack on the (constant) unknown plaintext is possible. They present a practical attack on a microcontroller implementation of AES without parallel noise and succeed with about 2.000 traces. The changing keys are not recovered in this setting which is acceptable for their attack goal.

At first glance, this seems a potential threat also for our construction, specifically to the unknown-inputs GGM tree. However, their attack leads to the recovery of the unknown inputs only which cannot be directly used by an attacker to predict the LR-PRF's output. Hence, a second first-order DPA attack using the resulting guesses for the plaintexts needs to be used to attack the key. This corresponds to an attack on the 2-PRG as discussed in the previous part 1. More importantly, contrary to the setting of Unterluggauer et al., the correlated algorithmic noise from the parallel setting is effective.

To address attacks on unknown inputs and key when such noise from parallelism is present, Medwed et al. [66] use simulations of second-order template attacks on the key using templates for the unknown plaintexts and the S-box outputs (see Fig. 5, right part in [66]). This experiment is equivalent to the attack described by Unterluggauer et al. only with switched roles for plaintexts and keys. The results of Medwed et al. [66]

in Fig. 5, suggest that noise from 2 or 4 "overlapping" S-boxes is sufficient to achieve a guessing entropy per byte greater than 4 and 6 bit, respectively. Considering our practical results, this is equivalent to at least 2, or 4 S-box signals overlapping at every location and point in time. This seems to be a reasonable requirement, as these effects are the same as the ones which are exploited in an attack on the known-inputs 2-PRG and cause the remaining entropy of 2^e. We therefore tend to believe that such attacks are unsuccessful in practice, but leave a thorough analysis for future work.

Finally, note that at the end of the GGM tree a whitening step is computed where p_{s0} is encrypted. Otherwise the last step would be susceptible to an attack with two known ciphertexts, which is equivalent to the known input attack on the initial 2-PRG stage. This step is necessary in the common attacker model where the attacker is assumed to know the in- and outputs of the LR-PRF. In use cases where the output is not exposed to the attacker, this whitening step can be left out to decrease latency.

A Cautionary Note The security of the proposed construction is based on the fundamental assumption that enough entropy remains after an attack on the first 2-PRG stage as shown. This assumption can only be verified empirically by proper laboratory side-channel evaluations. The number of stages can be configured according to the results of this analysis. If no entropy remains after localized EM attacks in the first stage, i.e. k_0 can be recovered without doubt, then our construction only increases the effort of the attacker who has to repeat measurements and attacks on the second and further stages. For a remaining entropy of e bits after an attack on k_0 and a targeted security level of s bits, $\lceil s/e \rceil$ 2-PRG stages and keys are necessary. Whether this is practical is determined by the design constraints of the system.

6.3. Summary

In Chapter 5, we investigated the reasons why state-of-the-art high precision EM attacks are able to successfully isolate the leakage of parallel S-boxes within LR-PRFs. We found that design choices can increase the robustness to side-channel attacks, but on certain devices improved constructions are necessary. Thus, we present an extension to the ASIACRYPT 2016 LR-PRF which introduces additional key entropy to mitigate the entropy loss due to high-resolution EM attacks under verifiable empirical assumptions. It comes at a reasonable overhead and only requires additional key storage and no particularly stringent design constraints, i.e. it can be instantiated on devices with limited control over the underlying process technology, such as FPGAs. With this building block at hand, we can implement leakage resilient applications on devices where earlier proposals failed to establish side-channel security. In Chapters 7 and 8 we explore two case studies where we deploy the LR-PRF to realize bitstream decryption on FPGA SoCs and LR-AEAD on microcontrollers.

7. Case study: Secure and updatable bitstream decryption for FPGA System on Chips

In Chapters 4 through 6 we analyzed the practical side-channel security of LR-PRFs, found weaknesses against high precision EM attacks and proposed an improved construction. In this and the following chapter, we deploy leakage resilience in two concrete use cases and explore the specific difficulties and solutions in case studies.

First, we consider the example of secure configuration of FPGA SoCs in the field. Typically, the configuration bitstream of FPGA SoCs is stored in external non-volatile memory (NVM) and thus has to be encrypted to protect against malicious readouts. As part of the secure boot process the bitstream is decrypted using built-in cryptographic engines. However, reported examples have shown that such cryptographic engines may become insecure against side-channel attacks at any later point in time. Such attacks may allow to recover secret encryption keys and thus compromise the confidentiality of the configuration.

This leaves already deployed systems vulnerable without any mitigation option. To solve this, we propose using an LR-AEAD engine within the FPGA logic instead of the built-in one for the crucial task of bitstream decryption. The actual user intellectual property (IP) cores are then decrypted at runtime and deployed using a feature called dynamic partial reconfiguration (PR). It allows reconfiguration of parts of the FPGA without interrupting the operation of the other parts. Our proposed concept allows to securely update the main FPGA functionality and even allows to update the cryptographic engine itself (i.e. to use a different algorithm later).

We provide two options for the LR-AEAD engine which are both based on an LR-PRF. The first uses the improved LR-PRF presented in Chapter 6 in conjunction with AES in OFB mode for decryption and GMAC for authentication. This option is suitable for devices where the CHES or ASIACRYPT LR-PRF do not provide sufficient practical side-channel security. The second is an implementation of the $FGHF'$ LR-AEAD scheme introduced in Section 3.3. Compared to the AES-OFB based solution it requires less hardware resources and has favorable properties in cases where parts of the plaintext can be guessed by the attacker. However, it is only viable on hardware platforms where the CHES or ASIACRYPT LR-PRF provide a high enough security level after an attack. In other words, in order to use the second option the AES implementation on the target platform must resist attacks with data complexity 2. Depending on a side-channel evaluation of the LR-PRF on the target platform, either the LR-AEAD scheme with

additional side-channel countermeasures or a more efficient $FGHF'$ based variant can be deployed.

As proof of concept, we describe an application to the Xilinx Zynq-7020 FPGA SoC in detail. We provide an analysis and implementation of both options and discuss the differences in terms of side-channel security and hardware resources. For the analyzed Zynq-7020 FPGA SoC we find that both LR-AEAD schemes can be securely implemented and provide high security levels. For complete independence from manufacturer-provided secret key storage, and since many devices do not allow access to a dedicated key storage facility from the FPGA logic, we use a PUF-based secret key storage in the FPGA logic. Note that in the case of the targeted Xilinx Zynq-7020 device, there is no secret key storage which is accessible from FPGA logic. Hence PUF-based key storage is the only option for alternative bitstream decryption engines. PUFs provide a mechanism for key storage by leveraging the manufacturing differences of an integrated circuit to derive a device intrinsic secret. This eliminates the need for a secure NVM.

The initial bitstream consisting of our decryption engine and PUF do not need to be encrypted but have to be authenticated to be protected from tampering. Otherwise, an attacker could modify or replace those initial hardware parts to leak secret keys. This makes using built-in bitstream authentication features unavoidable. However, when using public key signatures (e.g. the Xilinx Zynq-7000 series supports RSA signature authentication), this only requires storing and processing public keys on-chip. No secret private keys are stored on-chip, hence, leading to minimal additional attack surface. In our concept, the integrity of the bootloader and initial FPGA configuration containing the LR-AEAD, PUF, and PR handling are verified in this manner. The actual user logic is dynamically configured during runtime and is decrypted and authenticated using our LR-AEAD core.

Outline In Section 7.1 we introduce FPGA SoCs and motivate the need for side-channel secure cryptography for secure boot and field updates. Section 7.2 describes the required building blocks, specifically a PUF for on-chip key generation and PR to re-configure parts of the FPGA fabric with decrypted IP cores at runtime. In Section 7.3 we describe our approach towards secure and updatable bitstream decryption on FPGA SoCs on a conceptual level and an application on the Xilinx Zynq-7020 FPGA SoC in particular. We discuss two different options for the LR-AEAD core in Section 7.4 and provide the accompanying side-channel analysis and a comparison of the resource utilization. Our design currently does not include software decryption as this poses a different set of challenges than bitstream decryption and we discuss those differences in Section 7.5. We summarize our results in Section 7.6.

7.1. FPGA SoCs require side-channel secure configuration and updatability

FPGA SoCs are powerful computing platforms that combine one or multiple application processor cores with programmable FPGA fabric on a single chip. They offer versatile and flexible hardware acceleration for demanding applications such as image processing or machine learning and are widespread in e.g. the automotive and aviation industry. In these applications the devices are often deployed in locations where adversaries have physical access. Additionally, the lifetime of these devices is often long and updates to the functionality may become necessary. Therefore a secure configuration and update mechanism is required to protect the IP from theft and tampering.

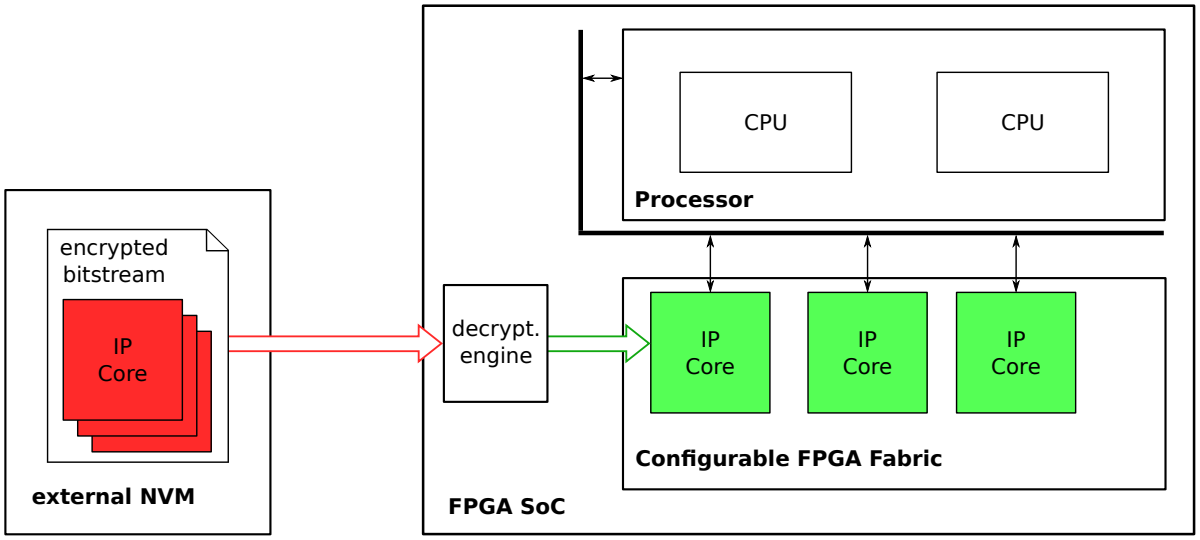


Figure 7.1.: Bitstream decryption on FPGA SoCs using manufacturer provided means. Red: encrypted. Green: decrypted.

FPGAs (i.e. SRAM-based ones) are configured with the hardware implementation at every power-up and the respective bitstream must be stored in a chip-external NVM. To protect IP, FPGA devices usually provide bitstream encryption and authentication features using dedicated built-in hardwired cryptographic engines as shown in Fig. 7.1. During the secure boot process, the encrypted bitstream is loaded from the external NVM and then authenticated and decrypted by the manufacturer provided decryption core.

Unfortunately, it has been shown for several devices from different manufacturers that such cryptographic engines can be attacked using SCA [70, 71, 86, 87, 94]. Another security issue can arise from hardware design flaws, as is evident by the work of Ender et al. [26]. They uncovered a hardware bug in the Xilinx 6 and 7 series FPGAs that allows them to read out user IP after it is decrypted by the built-in hardware cryptographic engine. As with the SCA vulnerabilities of hardwired engines, this bug can not be fixed and requires a new revision of the hardware. Due to this, it is highly advisable

to have a concept for retrofitting and updating the cryptographic engine which is used for bitstream decryption. This applies even if no successful attack is currently known against the cryptographic engine of a certain device.

As a solution, we implement a side-channel protected engine within the FPGA logic instead of using the built-in hardwired engine. To configure the FPGA with the decrypted IP cores, we use dynamic PR. While PR is helpful and available on all major platforms, on-chip key storage for symmetric decryption keys is often not accessible from the FPGA logic. Furthermore, it has been shown that readout of both eFuses and battery-backed RAM (BBRAM), which are commonly used to implement key storage, can be possible with state of the art equipment [97, 56]. For this reason we use a PUF to embed a key into the device. The PUF leverages tiny manufacturing variations to generate entropy as explained in Section 7.2.1.

Related work Xilinx outlines the idea of using custom cores for decryption in [77] and describe a system consisting of an AES core together with a PUF and dynamic PR. They do not provide an implementation or security analysis. An implementation of that scheme was later published by Jacob et al. [46] where they use AES in Galois/Counter mode (GCM) for decryption and authentication with a twisted bistable ring PUF for key storage, but no side-channel protection.

Owen et al. [74] follow a similar approach, but use the PUF to generate a key in a way that is sensitive to all changes within the bitstream to achieve a self-authenticating design. While they therefore do not require the manufacturer provided authentication, they do however need physical access to the chip for the encryption of images since it can only be performed on-chip using a separate bitstream that contains the encryption core. This makes their proposal unsuitable to provide updates for devices in the field as it would entail that said devices need to be brought back to a secure environment for the update.

In contrast to the work of Owen et al., we use the PUF to embed an external secret key, thereby allowing off-chip encryption of updates. We extend the work of Jacob et al. and implement leakage resilience as side-channel countermeasure where required.

7.2. Building blocks: Device intrinsic key generation and partial reconfiguration

Apart from the actual bitstream authentication and decryption core, secure non-volatile key storage is mandatory. In Section 7.2.1 we describe how to use a PUF to embed a user generated key into the device in a way that prevents readout by adversaries. As we use our decryption core in FPGA fabric we need to configure parts of the FPGA with the decrypted user IP cores at runtime. For this task, we use PR which we explain in Section 7.2.2.

7.2.1. Physical unclonable functions (PUFs) for key generation

A PUF is a security primitive that utilizes manufacturing process variations to generate a unique digital fingerprint intrinsic to a physical piece of hardware [31]. As this natural variation between otherwise identical silicon devices is outside the control of even the manufacturer, PUFs are inherently difficult to clone. Constructions can be broadly split into two categories; challenge-response type PUFs which produce a device unique response for a given input challenge, and identity-generator type PUFs which produce few or just a single response for the device. For the use case of bitstream decryption we only require the PUF to generate 256 bits of key material, therefore we use an identity-generator type PUF.

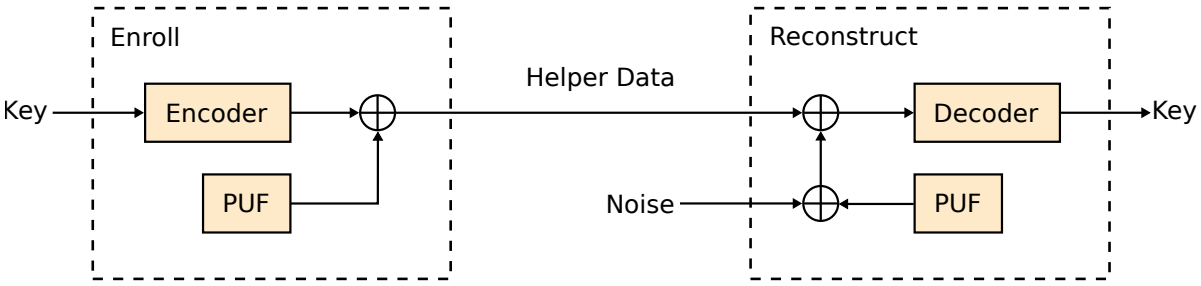


Figure 7.2.: Fuzzy commitment scheme.

The PUF architecture used in this work is based on that in [39] which creates a cross coupled feedback loop contained within a single slice of an FPGA to generate a single PUF bit. The prototype implementation of our system was a joint work with Neil Hanley and Chongyan Gu (Centre for Secure Information Technologies, Queen's University Belfast) who contributed the PUF implementation and evaluation. Hence, this section only gives a brief summary of their results, for full details refer to the original publication [102].

To be able to enroll a key chosen by the user into the PUF we use a fuzzy commitment scheme. As shown in Fig. 7.2, during enrollment a user generated key is first encoded using an error correcting code to add redundancy. This is necessary because there is an intrinsic error probability when reproducing the PUF response. The encoded key is then masked with the PUF response to derive what is called 'helper data'. The key enrollment is done once in a secure environment. Since the helper data is masked by the PUF response, it does not need to be kept secret. Once generated, this helper data is then incorporated as part of the bootloader to enable key reconstruction in the field. Key reconstruction uses a decoder stage to regenerate the key from the helper data and the (noisy) PUF response. We use a $(23, 12, 7)$ Golay linear block code which encodes every 12-bit of the key into 23-bit codewords and is able to correct up to three errors per codeword. A Golay encoder can be implemented very efficiently in hardware using a shift register, with the decoder block utilizing the same encoder block combined with an additional 12-bit look-up table of depth 2048.

Side-channel attacks on PUFs SCA attacks on PUF designs can be categorized into attacks on the core PUF instance, and attacks on the post-processing. While attacks such as directly reading out PUF bits using a focused ion beam (FIB) are outside the scope of the attacker model, there has been work on directly attacking the generation of the PUF output. For example, delay based arbiter designs are attacked using power analysis in [5], while recovering frequencies from ring oscillator PUF designs is investigated in [67]. However, the PUF design used in this work evaluates all bits in parallel in a single clock cycle. Due to the algorithmic noise and the compact design of each cell, which should lead to generally low SNR, we do not expect the core bit generation to be susceptible to (SPA) attacks.

Of greater threat are attacks on the post-processing stage [67, 95]. Typically such attacks are differential attacks and require the attacker to manipulate the helper data. In our proposal, the helper data is authenticated before use and thus protected from tampering. This greatly reduces the number of *different* observable traces that an attacker can obtain, with differences only generated due to PUF response errors. The Golay error correction block consists of linear operations, and a single, constant time, table look-up indirectly based on the PUF response error. Linear operations have been shown to require a large number of side-channel traces to obtain information [79], hence are not considered practical in this scenario. While many attacks against block ciphers target look-up tables, this is as their non-linearity makes them a suitable attack point [79]. In the implementation here, the look-up table is linear, and is used to both speed up and ensure constant time syndrome decoding by mapping the decoding error into memory. These errors are sparse vectors and, similar to the linear operation stage, the reduced number of different observable traces should prevent any power or EM based differential SCA. Under these restrictions, SPA attacks using only a single or few traces, are likely infeasible. Both encoder and decoder architectures run in constant time, hence timing side-channel attacks are also not applicable.

In this work we do not conduct a side-channel analysis of the PUF since the focus is on the leakage resilient decryption and the prototype implementation serves mostly as a proof-of-concept. However, for the reasons explained in this section, we do not think that such attacks have a significant chance of success.

Characterization on the Xilinx Zynq 7000 platform The properties of a PUF are naturally dependent on the platform they are implemented on. To assess the suitability of the PUF architecture for our side-channel protected secure boot design, we therefore characterized the PUF using an array of 20 Xilinx Zynq-7000 devices. The two most important metrics are reliability and bias, i.e. the distribution of zeroes and ones in the PUF response. Ideally, the reliability should be high and the average value of a bit should be 0.5. In addition, the average difference in response bits of different devices should also be 0.5. Hence, a comparison of multiple devices is mandatory to ensure that different responses are generated on different devices.

Analyzing the reliability of each bit, we found that approximately 80% of bits on each device returned a stable value across all evaluations. The average value of a bit

in our evaluation is 0.498 when averaged over all responses and devices. The expected fractional Hamming distance between the outputs of two different devices was empirically estimated to be 0.497. These positive results suggest that the quality of the PUF is sufficient for our purpose and that responses generated on different devices will be sufficiently independent.

Given the configuration of the used Golay encoder, the total bit error rate (BER) of the generated key is 1.4×10^{-3}. The BER could be further lowered by concatenating a repetition code with the Golay encoder or through majority voting of the PUF response. For the prototype implementation we did not further pursue this.

7.2.2. Partial reconfiguration (PR)

PR is a feature of modern FPGAs that allows changing parts of the functionality at runtime. A conventional bitstream contains the complete FPGA configuration and if parts of it need to be changed it entails replacing the bitstream entirely and reconfiguring the whole FPGA after a reset. This is different when PR is used. Then the bitstream is divided into the so called 'static' bitstream and 'partial' bitstreams. The static bitstream is configured at startup and contains the basic functionality and all parts that are not expected to change frequently. It includes slots that are prepared to be configured with partial bitstreams. The partial bitstreams consist of IP cores that can be swapped in and out of the slots in the static bitstream during runtime without interrupting the operation of the rest of the FPGA. This feature is used for updating functionality of IP cores, e.g. to support new standards and protocols, and time sharing FPGA resources.

On Xilinx FPGA SoCs, the reconfiguration is handled either through an interface that is accessible from software or from within the FPGA fabric itself through the internal configuration access port (ICAP) interface. In our use case, the static bitstream contains the PUF, the LR-AEAD engine and a PR controller that interacts with the ICAP interface. The PR controller receives the authenticated and decrypted partial bitstream from the LR-AEAD core and then passes it to the ICAP interface.

7.3. Concept for secure and updatable bitstream decryption

In this section we show how user IP cores in the FPGA design can be protected using leakage resilient cryptographic cores on an FPGA SoC. Custom cores are used for secure key storage and decryption instead of the manufacturer provided built-in key storage and decryption core. To demonstrate the practicality of this approach, we provide a reference design for the Xilinx Zynq-7020 FPGA SoC.

Figure 7.3 provides an overview of the secure configuration on FPGA SoCs using custom cryptographic cores. All modules that participate in the secure configuration are depicted in orange. This includes the bootloader U-boot and the cryptographic cores, namely the PUF, LR-AEAD, and PR controller. These cores are part of the initial static configuration of the FPGA and are independent of the so-called user design

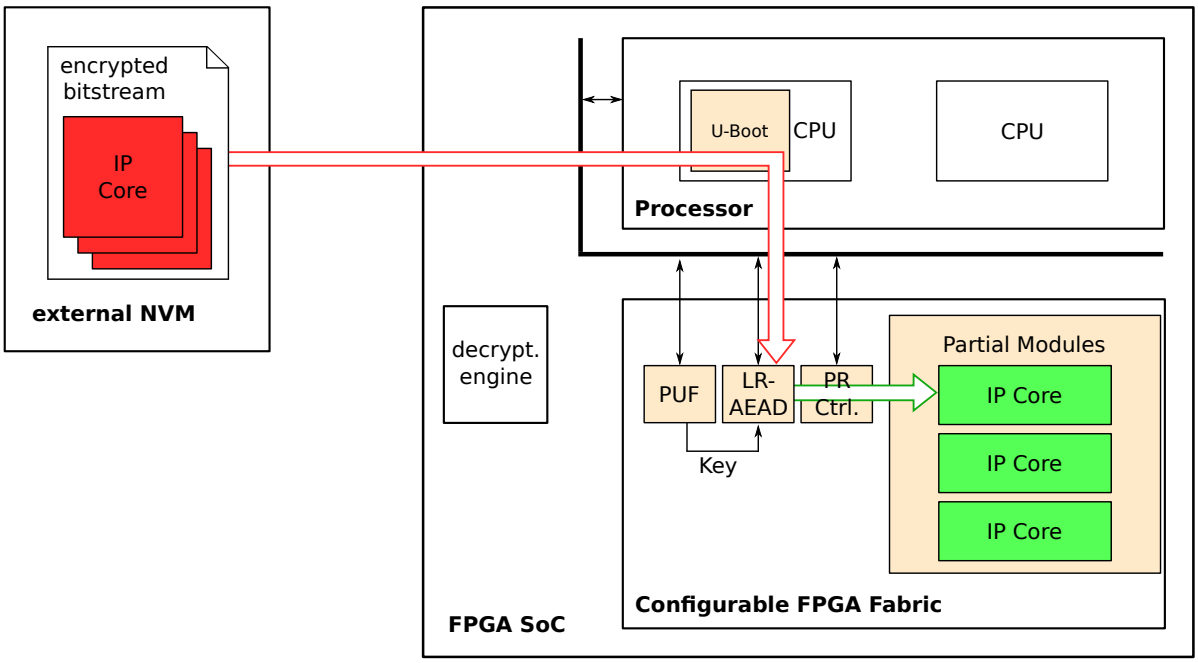


Figure 7.3.: Updatable bitstream decryption on FPGA SoCs using LR-AEAD. Red: encrypted. Green: decrypted.

which contains the IP cores that fulfill the application's purpose. All respective user IP cores (depicted in green) are loaded using dynamic PR only once they have been successfully decrypted and authenticated using the initial static parts.

Device initialization and key enrollment Prior to deployment of the device, a user supplied encryption key is linked to the PUF as a one-time operation. A dedicated software routine is loaded onto the device which generates the PUF helper data for that key. During that process, the PUF internally creates a device intrinsic secret which in combination with that helper data is later used to regenerate the supplied encryption key (this process is illustrated in Fig. 7.2). Neither the encryption key nor the PUF secret are permanently stored on the device which means that the secret cannot be recovered via offline attacks. After embedding the encryption key, an RSA public key is burned into the eFuses of the device and authenticated boot is enforced. This prevents the execution of an unauthentic boot loader and, importantly, also stops adversaries from embedding their own key into the PUF after the device is deployed. If attackers would be able to embed their own key, then they could load malicious (partial) bitstreams and either take over the system or gather information about genuine designs which are already configured. This does not prevent an authentic user who is in possession of the RSA private key from changing the key at a later point in time if required, but that requires a secure environment.

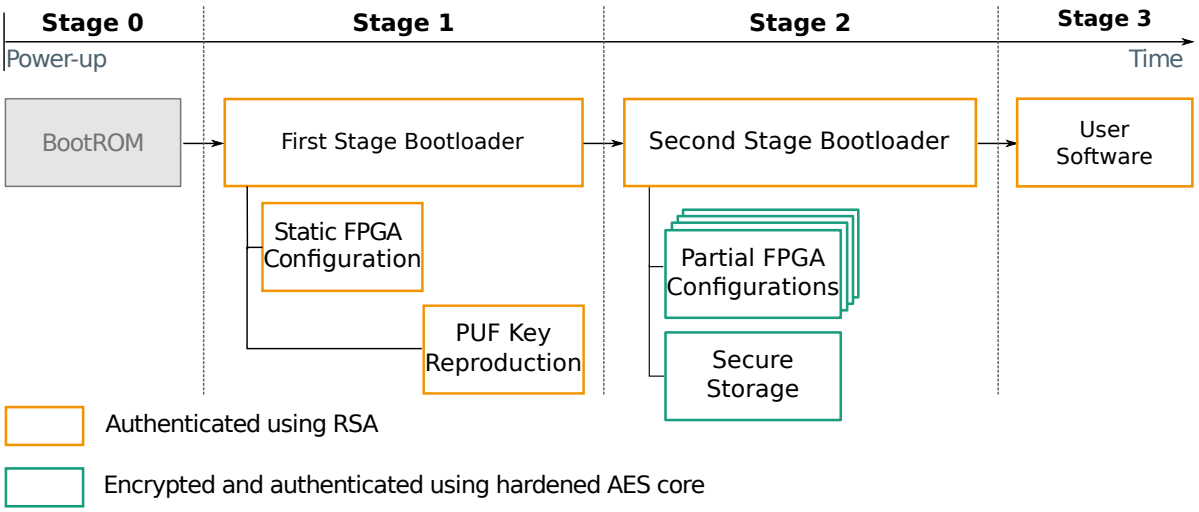


Figure 7.4.: Boot flow of the hardened boot process.

Secure boot in the field Figure 7.4 describes the boot flow of our Xilinx Zynq-7020 implementation once it is deployed. It shows the different stages of the secure boot process starting from power-up until the system has successfully booted and user software is running. After power-up the boot process begins with the hardwired BootROM code in the CPU that is provided by the manufacturer. To establish a chain of trust the first software and logic configuration that is loaded onto the system needs to be protected against manipulation by an adversary. As a result, the BootROM code has to be able to authenticate the first user provided boot code before control is handed over. The BootROM code itself cannot be altered. Hence, all implementations must rely on manufacturer-provided authentication of the first stage bootloader (FSBL), in our case RSA signature verification. However, since the signature check only requires the public part of the RSA key pair, *no secret keys* are stored or processed. Our concept is only based on the assumption that manufacturer-provided storage for the *public key* can be trusted. The FSBL, static bitstream and second stage bootloader (U-boot) are authenticated using this RSA verification routine. After verification, the FSBL is loaded to the on-chip memory and control is handed to it.

The FSBL is a modifiable bootloader that initializes the system, peripherals and FPGA fabric, following which it authenticates and loads the static configuration to the FPGA fabric. We extend the FSBL to transfer the helper data to the PUF module and trigger the regeneration of the encryption key. The helper data is publicly accessible and is stored and authenticated together with the FSBL[9]. Following the successful reproduction of the encryption key, the PUF is locked until the next reset. This prevents any entity from misusing the PUF at a later instance of time in order to attempt to reproduce the key. The encryption key generated by the PUF is directly transferred to the LR-AEAD engine and does not leave the FPGA fabric as can be seen in Fig. 7.3.

[9]This design choice requires the FSBL to be replaced to include updated helper data if the key that is embedded into the PUF is changed.

It remains in the hardware and can be used by the LR-AEAD engine, but never read or accessed by the CPU. The LR-AEAD core is now capable of authenticating and decrypting user hardware IP. The boot process continues with the FSBL authenticating and loading the second stage bootloader, U-boot.

U-boot is an open-source bootloader commonly used for embedded systems whose functions include system initialization and loading the kernel. In addition to this, we use U-boot to securely load the partial bitstreams. After completing the system initialization, U-boot begins sending encrypted partial bitstreams containing user IP to the LR-AEAD core for tag validation and decryption. Upon successful verification, the plain partial bitstreams are transferred from the LR-AEAD core directly to the PR controller, i.e. they never leave the FPGA fabric and are not transferred on any shared resources of the FPGA SoC. The PR controller dynamically reconfigures the relevant part of the FPGA without interrupting other regions and services. Depending on the application, those user cores may for example contain secret data for higher software layers. The last step in the boot process is for U-boot to load the software stack on the CPU. In this concept the software is only authenticated and not encrypted; this is explored in greater detail in Section 7.5.

Remote updates To update a user IP core, the new version is encrypted off-chip using the encryption key that was previously embedded during the enrollment process. The encrypted partial bitstream is then sent to the device as remote update without requiring physical access. Remarkably, the LR-AEAD core for actual bitstream decryption can also be remotely updated by updating the static bitstream. When changes to the core are made, a new version of the entire static bitstream is generated, signed with the RSA private key and then transferred to the device to replace the old one. In contrast, updating the PUF requires a secure environment. Due to its nature, changes to the PUF design will most likely change the intrinsic PUF response and prevent the recovery of the encryption key with existing helper data. In that case, the encryption key enrollment has to be repeated as it was done for a fresh device.

Attack vectors In general, three major attack vectors can be identified:

1. Attacks on the boot process (FSBL, U-boot).
2. Attacks on the PUF.
3. (Side-channel) Attacks on the bitstream decryption.

To mitigate attacks on the boot process, the FSBL and U-boot are authenticated using RSA before being executed. This prevents attackers from running their own modified software which may attempt to bypass the implemented security mechanisms. On the Xilinx Zynq 7020, the authentication is enabled by burning an eFuse and cannot be disabled thereafter.

Attacks on the PUF can be further divided into machine learning attacks, side-channel attacks and attacks that exploit the interface. Machine learning attacks require querying

the PUF to collect different responses and thus do not apply to identity generator type PUFs. As discussed in Section 7.2.1, we expect that side-channel attacks do not have a significant chance of success. An attacker with full access to the PUF interface, however, could exploit this to recover the PUF secret and consequentially the encryption key. As can be seen in Fig. 7.2, the helper data that is generated during key enrollment is an XOR combination of the encoded encryption key and the PUF secret. If an attacker could enroll his own keys, then it is trivial to calculate the PUF secret from the retrieved helper data of two different keys. In our concept, this is prevented by only allowing authorized FSBL code to be executed on the device. This code does not allow key enrollment in the field and additionally access to the PUF is locked after the decryption key is generated.

Another major threat, and the main motivation for this work, are side-channel attacks on the bitstream decryption engine. To thwart such attacks we deploy an LR-AEAD engine. We analyze the side-channel security in Section 7.4. Apart from trying to extract the secret key, an attacker may also aim at damaging the device by trying to configure invalid bitstreams. This is especially relevant if the device offers an interface for remote updates, then this could be exploited to insert bitstreams (without knowing the encryption key) that get decrypted into random data. It was noted by Moradi and Schneider [71] that configuring random data can lead to shorts in the circuit and can potentially damage the device. Thus, we authenticate the bitstream before configuration and, if necessary, buffer the decrypted bitstream until authentication is complete.

7.4. Leakage resilient decryption of bitstreams

In this section, we describe how to build an LR-AEAD scheme based on LR-PRF for bitstream decryption and authentication. As established in Chapter 4, the choice of LR-PRF (and consequentially, the choice of LR-AEAD scheme) depends on the side-channel security of the underlying AES implementation on the target device. If the AES implementation resists side-channel attacks with low data complexity, more efficient constructions in terms of latency and hardware overhead are possible. Otherwise, as is the case with the Spartan 6 analyzed in Chapter 4, constructions with additional measures such as the LR-PRF with added key entropy that is presented in Chapter 6 are necessary to provide a secure construction.

Therefore, as first step, we provide a side-channel analysis of a synthesized AES core on the Xilinx Zynq-7020 device in Section 7.4.1. Contrary to the case of a Spartan 6 FPGA, we find that this implementation resists EM attacks when configured with data complexity 2. Hence no additional entropy is necessary and e.g. the CHES 2012 LR-PRF (configured with data complexity 2) can be used securely and without modifications.

Nevertheless, for the sake of generality we present two alternatives for the LR-AEAD cores in Sections 7.4.2 and 7.4.3: one scheme for the case when additional key entropy is necessary and another for this case where the AES resists attacks with data complexity 2. The first is based on the improved LR-PRF with AES-OFB for decryption and GMAC for authentication. The second is an implementation of the $FGHF'$ scheme (see Section 3.3) and uses the CHES 2012 LR-PRF, an LR-PRG and a SHA-256 hash function. The hash

function is implemented in software, therefore this option uses less hardware resources. We compare the required hardware resources in Section 7.4.4. With these two options at hand, side-channel secure bitstream decryption is applicable to a wide range of platforms.

7.4.1. Side-channel analysis of LR-PRF on Xilinx Zynq-7020

To assess the security level of LR-PRFs on our target device, we analyze the side-channel security of a fully parallel AES core with data complexity 2. We implemented the LR-PRF using the AES core with parallel S-boxes presented in Section 4.1 on the Xilinx Zynq-7020 device. Following the design recommendations that we state in Section 5.4, we did not use hard-macros for the S-boxes and placed the AES as dense as possible as shown in the floorplan in Fig. 7.5.

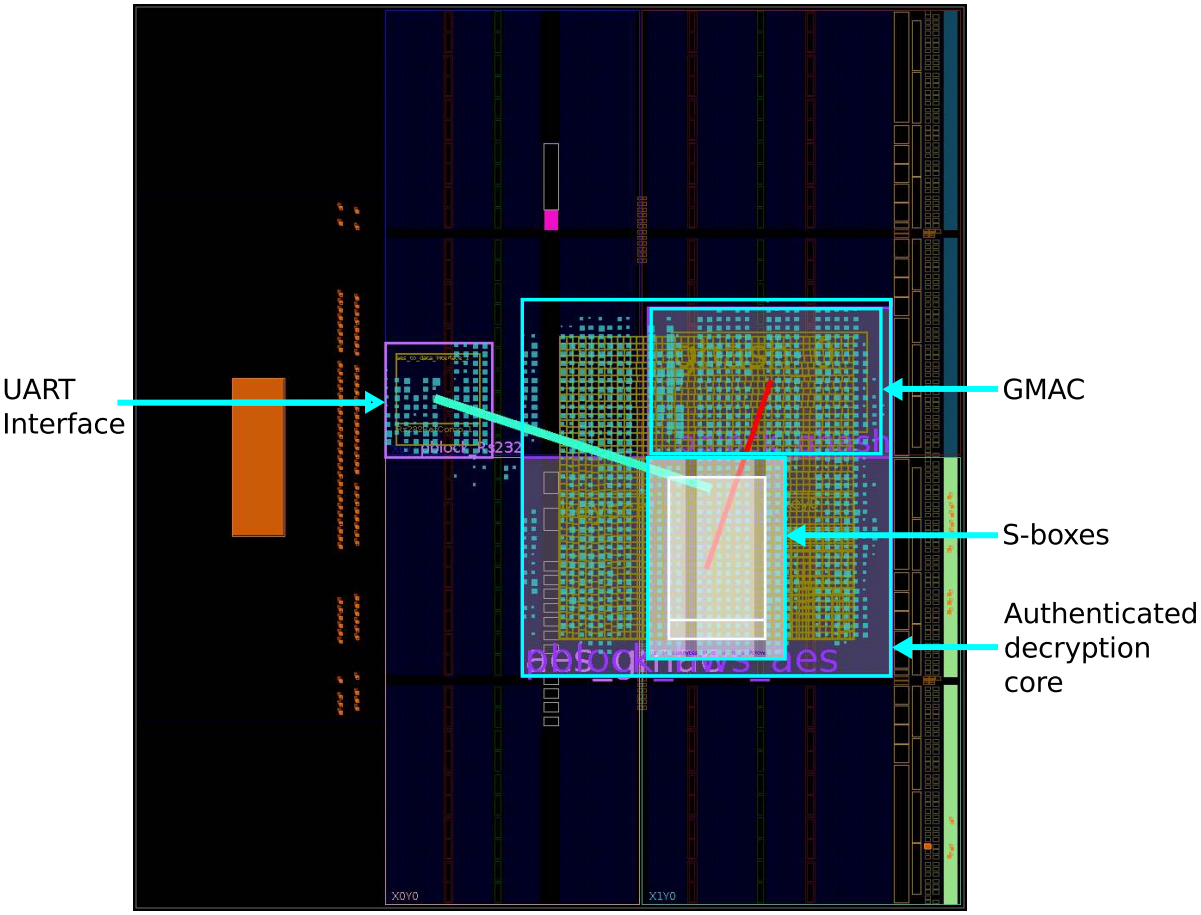


Figure 7.5.: Placement of the AES core for the SCA.

We conduct an EM analysis using a near-field EM probe placed on the decapsulated die using the setup and following the same procedure as described in Chapter 4. We first identify the locations of the S-boxes through a grid scan and then mount template attacks on them using measurements from those locations. For each S-box we collect 400,000 traces for profiling and 100,000 traces for the attack at its respective location.

We empirically found this number to be sufficient and that an increase in either the profiling or the attack set does not improve the attack. We use key rank estimation [34] to determine the security level.

We find that the remaining security level after an attack with data complexity 2 is still 2^{120}, or 120 bits. This is significantly higher than the results reported on the Xilinx Spartan 6 platform in Section 4.2 where the security level is only 2^{48} and we attribute this difference to the smaller feature size (28 nm compared to 45 nm) and the different placement strategy. Note that these results were achieved with a standalone FPGA design that includes only the AES and interface and not the entire system. We expect that the increased noise generated by the full design will only make attacks harder. We conclude that the remaining entropy of the AES core after an attack is sufficient to securely implement the CHES 2012 LR-PRF configured with data complexity 2 or the ASIACRYPT 2016 LR-PRF without additional countermeasures.

7.4.2. AES-OFB based LR-AEAD

First, we present an LR-AEAD scheme that is also applicable had the security level of the AES been insufficient for data complexity 2. We include this option for generality, as we will show in Section 7.4.3 that this platform allows a more efficient alternative. The scheme is based on the improved LR-PRF with added entropy and carefully designed such that in the remainder of the construction keys are only used with secret inputs or data complexity 1. This ensures that the LR-PRF is the 'easiest' attack vector and hence it suffices to analyze attacks on the LR-PRF to determine the overall security level. Figure 7.6 shows a dataflow diagram of the LR-AEAD scheme that consists of three components: a side-channel secure stage with an LR-PRF, decryption using AES in OFB mode and authentication using the GMAC.

We designed this LR-AEAD scheme following the principle described in [76]: As shown in Fig. 7.6, we rely on an initial 'leak-free' stage using an LR-PRF (green box). It establishes a pseudo-random intermediate state (represented in $\hat{\mathrm{IV}}_{ofb}$ and $\hat{\mathrm{IV}}_{gmac}$) that is unknown to the attacker and allows us to use unprotected decryption and authentication for the actual workload. For common SCA attacks the precondition is that the attacker can guess some internal value of the algorithm that depends on the secret and an input (in most cases the ciphertext) that is known to the attacker. The general idea behind our construction, as outlined in [76], is that only the initial LR-PRF stage needs to be side-channel secure (through means of leakage resilient cryptography), because behind it, no public inputs are processed by the block cipher. This gives no surface to mount an SCA attack on the unprotected stages as there are no known inputs and guessing internal values is not possible. The only viable attack vector is thus the LR-PRF stage and the security level of the LR-AEAD scheme is determined by the security level of the LR-PRF.

AES-OFB decryption After the LR-PRF stage, we use AES in OFB mode for the actual decryption of the bitstream. We opted for the OFB mode, instead of, e.g., GCM mode, because the plain- and ciphertext are no direct input to the block cipher. As

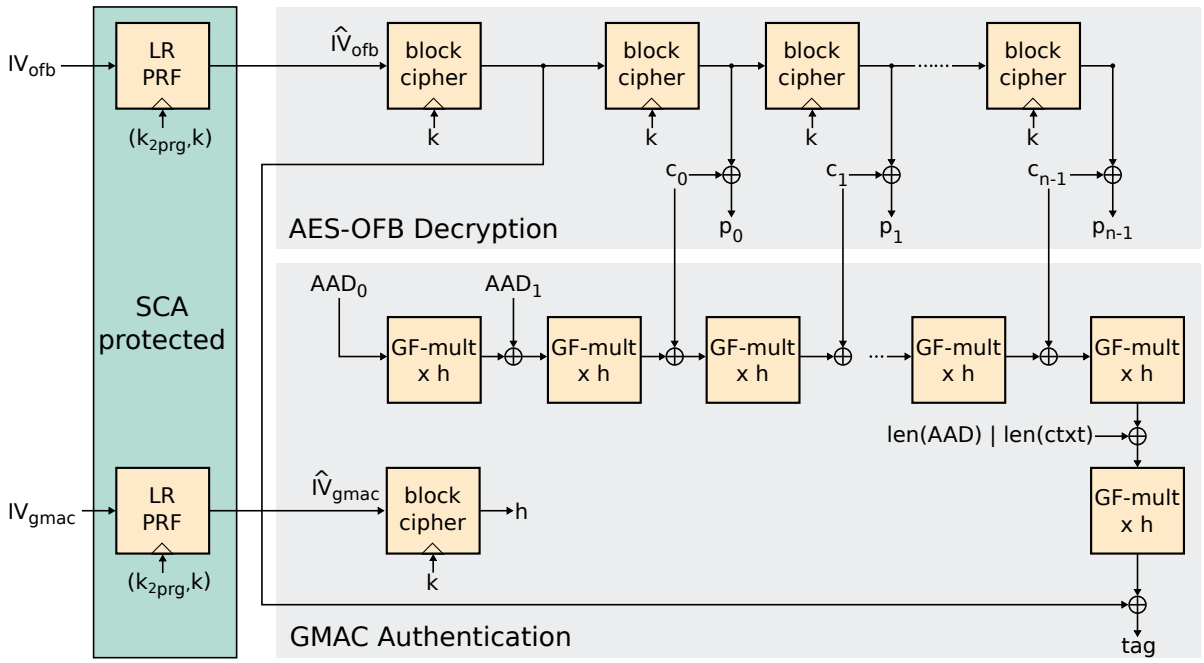


Figure 7.6.: Side-channel secure authenticated decryption.

long as an attacker does not know both plain- and ciphertext, the in- and outputs to the block cipher remain secret and provide no surface for side-channel attacks. This is important because even partial knowledge, as is the case, e.g., in counter modes where the initial counter value may be unknown but the increment of the counter is known, can be sufficient to mount an attack [47]. However, if on any platform portions of the plaintext are known, e.g. because they are all zeros, we propose not encrypting those parts and instead adding them to the authenticated data. An open question remains considering the effect of plaintexts that are unknown, but not uniformly random because they consist of opcodes or addresses that do not utilize the entire value space. We are not aware of any published attacks that exploit this scenario and imagine such an attack to be hard; nevertheless it remains as an interesting topic for future research.

GMAC authentication For message authentication we chose GMAC [63] because it can be efficiently implemented in hardware since it only requires an additional Galois field multiplier. This MAC is typically used with the GCM block cipher mode of operation, but can also be combined with other modes or used standalone. Differing from the specification, where the MAC key h is derived by encrypting a plaintext with all zeros, we derive the MAC key by processing another public IV_{gmac} with the LR-PRF and then encrypt the resulting $\hat{\mathrm{IV}}_{gmac}$ to get h. We divert from the specification here because, as previously explained, we need to prevent any unsecured encryption where the inputs are known to the attacker. This is achieved by processing the public IV_{gmac} with the side-channel protected LR-PRF first. The resulting $\hat{\mathrm{IV}}_{gmac}$ is then unknown to an attacker and can be encrypted with k without any additional countermeasures.

LR-PRF configuration For the initial LR-PRF stage, we use the improved LR-PRF described in Section 6.1 which consists of the GGM tree and one or more length-doubling 2-PRGs (see Fig. 6.1). Both the GGM tree and the (one or more) 2-PRG stages are implemented using the same hardware AES core that is also used in the stream cipher module. This means we only need to instantiate a single AES core that is then time-shared between the different modules.

The number of 2-PRG stages is a design choice that allows to compensate entropy loss after successful side-channel attacks on the very first AES execution which inevitably has to operate on public inputs. The required number of stages depends on the leakage behavior of the device and can be determined through laboratory analysis. Each stage adds a fresh 128-bit key to increase the remaining entropy within the internal secret state.

On the Zynq-7020, we determined that the entropy after an attack is high enough such that no additional 2-PRG stages would be necessary. However, to provide an implementation that remains secure even when implemented on devices which exhibit a low security level similar to the Spartan 6 we opted for one 2-PRG stage. Thus we require two 128-bit keys which we denote k_{2prg} and k[10]. The key k_{2prg} is used only for the initial 2-PRG stage, k for the rest of the construction, i.e. AES-OFB decryption and the generation of the MAC key. Reusing k in this manner is acceptable because all usages of k operate on unknown inputs. Therefore no additional leakage is made available to the attacker.

Buffering of decrypted partial bitstreams A general problem when decrypting and authenticating data in one pass is that the authentication tag can only be checked after processing all the data. Sending unauthenticated data to the PR controller, in our case the Xilinx ICAP, can be dangerous. Configuring the FPGA with unauthentic data could damage the FPGA due to short-circuits caused by false or malicious configurations as stated in [71]. Hence, we need to buffer the decrypted data until it is verified before we pass it on. This buffering has to be done in on-chip memory (OCM), otherwise it would be prone to manipulation, for example by probing of the memory bus. Therefore we implement a first in, first out (FIFO) buffer immediately prior to the PR controller that releases data only after its tag has been verified. To keep the allocated block RAM (BRAM) for the FIFO small, we split the bitstream into segments which are decrypted and authenticated individually. While the FIFO is sending authenticated data to the PR controller, decryption of the next segment is concurrently being performed, thus reducing the latency. To prevent IV re-use, the IV must be unique for every segment. For this purpose, IV_{ofb} is split up into a 96-bit random value and a 32-bit counter value that is incremented with every segment. That requires that at the beginning of every segment, the LR-PRF is evaluated to generate a fresh $\hat{\mathrm{IV}}_{ofb}$. Updating IV_{gmac} in between segments is not necessary because it is only used to generate the GMAC key h which we keep constant for the entire bitstream. Note that in the original GMAC scheme, h

[10]The keys k_{2prg} and k correspond to keys k_0 and k_{prf} in Fig. 6.1. We renamed them in this context to clarify the scope of each key.

is always derived by encrypting an all zero plaintext regardless of the value of the IV. Put differently, in the original scheme h is constant for a given key, in our scheme it is constant for a given bitstream.

U-boot only continues the boot process if the verification of all the segments of the partial bitstream succeeds. If the verification of any of the segments fails, U-boot aborts the boot process and system goes into a secure lockdown mode. A power cycle is necessary to leave this state.

7.4.3. FGHF′ based LR-AEAD

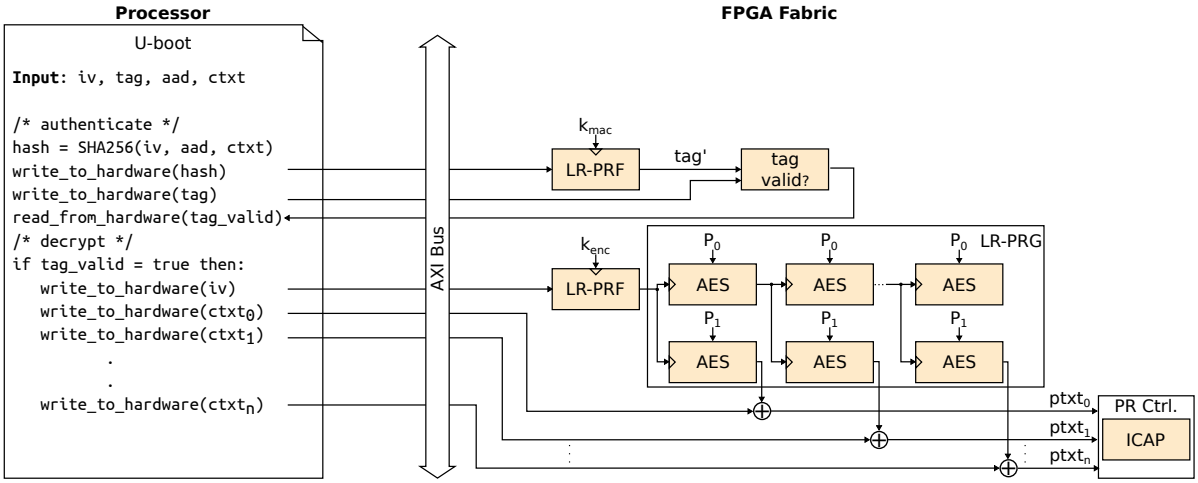


Figure 7.7.: Alternative side-channel hardened authenticated decryption.

The AES-OFB based scheme is motivated by the results in Section 4.2 that demonstrate that on FPGAs, it is possible to recover the key of an AES implementation in a side-channel attack even if only two different operations can be observed (the case of data complexity 2). However, the remaining security level after such attacks is dependent on the layout and manufacturing process of the device, where smaller feature sizes usually make localized EM attacks harder. In the security evaluation on the Xilinx Zynq-7020 device we found, that the AES implementation resists EM attacks in case of data complexity 2 and retains a high security level of 120-bit. This allows us to deploy an alternative construction that is easier to implement and requires less hardware resources.

Architecture In this second implementation option, we drop the 2-PRG stage inside the LR-PRF, therefore we now require only a 128-bit key for the LR-PRF. We also replace the stream cipher part, i.e. the AES in OFB mode, and the GMAC authentication. Instead, we implement an LR-AEAD scheme based on the results of Krämer and Struck [53] who propose that an LR-AEAD scheme can be built using an LR-PRF, an LR-PRG, and a hash function as introduced in Section 3.3. In reference to the introduction of this scheme in [20] we refer to it as $FGHF'$-based LR-AEAD and to our

original scheme as AES-OFB-based LR-AEAD. Figure 7.7 shows the hardware/software partitioning and the building blocks for the proposed $FGHF'$-based LR-AEAD scheme. The left side describes the software implementation in pseudo code and makes references to the hardware on the right whenever data is transferred between the two partitions. The right side describes the *dataflow* in the hardware implementation in a way similar to Fig. 7.6. Note that there is still only one hardware instance of the LR-PRF and that the (one) AES core used in the LR-PRG part is shared with the LR-PRF. The construction uses a single IV and two 128-bit keys, k_{mac} and k_{enc}.

Interestingly, the hash function in this construction does not need to be protected against side-channel attacks as it only processes public inputs and no secrets. Therefore the authentication part of the scheme can be split between hardware and software. First, a hash function, here SHA-256, calculates the hash of the IV, additional authenticated data (AAD) and ciphertext. This is implemented in software as part of U-boot. The hash is truncated to 128 bits and processed by the LR-PRF with key k_{mac} to calculate the tag.

For this side-channel critical operation, the evaluation of the LR-PRF, we still rely on a protected hardware-engine. The tag is now calculated over the entire bitstream at once and checked before decryption is started instead of calculating tags for multiple segments of the bitstream separately. This is different to our previous construction, where authentication and decryption are calculated in parallel hardware and requires a hardware FIFO to buffer the plaintext until the tag is verified. In this new scheme, only the 128-bit hash is sent to the hardware during authentication. As the tag of the entire bitstream is now verified before decryption, the FIFO in front of the ICAP is not necessary and the decrypted bitstream can be directly configured.

Decryption of the bitstream is still implemented in hardware. If the tag is correct, the IV is processed by the LR-PRF using a second key, k_{enc}. The result is used as key for an LR-PRG which produces the key stream to decrypt the ciphertext. The LR-PRG is based on AES and consists of a concatenation of 2-PRG stages (see Fig. 3.6). Within each stage, an all-zero plaintext p_0 and an all-one plaintext p_1 are encrypted with the same key, and the results are used as the key for the next iteration and as part of the key stream, respectively.

Side-channel security In Section 7.4.1 we give results of a side-channel attack on an AES with data complexity 2 implemented on our target platform Xilinx Zynq-7020. For the new construction we identify two attack vectors which we show to be equivalent to the case in Section 7.4.1:

Side-channel attack on the LR-PRF: Keys k_{mac} and k_{enc} can be targeted in an attack on the LR-PRF. Since we are not using the 2-PRG stage in front of the GGM tree PRF, the relevant attack vector is the first iteration of the GGM tree. In the context of an attack, this iteration behaves identically to a 2-PRG stage: it consists of an AES encryption that uses either all-zeros or all-ones as plaintext input.

Side-channel attack on the LR-PRG: According to the security analysis of the LR-AEAD scheme by the authors of [20] the PRG is not required to be secure against

differential side-channel attacks since its key is generated on the fly and is only valid for one message. Thus, if we consider the PRG as a black box, each key is only used for one operation and the only valid attacks are attacks with data complexity 1, i.e. SPA attacks. However, since we use a block cipher based PRG, this single operation in fact consists of multiple sensitive block cipher encryptions which increases the data complexity for an attack. It is also intuitively clear that the seed to the PRG, i.e. the initial key to the AES, must not be leaked in order to protect the confidentiality of the individual message.

Fortunately, the practical side-channel security of the PRG equals the security of the LR-PRF in our case[11] because one iteration of the PRG has the identical side-channel attack surface as one iteration of the LR-PRF in case of data complexity 2. The LR-PRG consists of multiple 2-PRG stages and changes the key between the stages. It is initialized with a key that is derived from the IV of the bitstream. Hence, an attack can at most recover the ephemeral key to decrypt one specific bitstream, and not one of the long-term keys k_{mac} or k_{enc}. An attack on the LR-PRG can be mounted targeting any 2-PRG iteration, i.e. any intermediate key. However, in all cases the attack is limited to data complexity 2 and there is, to the best of our knowledge, no way to combine information gathered from attacks on different iterations.

Both attack vectors are identical to the case investigated in Section 7.4.1. Therefore we can apply the results of the side-channel evaluation there to this construction here. As a result, the remaining entropy after a side-channel attack on k_{mac}, k_{enc} and the ephemeral LR-PRG keys is 120-bits per key. Since there is no meaningful way to combine the results of attacks on the different keys, the security level of the entire construction is thus 120-bits.

7.4.4. Resource utilization and performance

Here, we describe the prototype implementation of both variants for the Xilinx Zynq-7020 FPGA SoC. The resource utilization of the building blocks is provided in Table 7.1. The Zynq-7020 device is among the smaller, low-cost devices of the Xilinx Zynq-7000 product range. Nevertheless, we only use around 22 % of the available slices and 1.5 % of the available BRAM in case of the AES-OFB variant and 18 % of slices and 0.7 % of BRAM in case of the $FGHF'$ based variant. Hence, 78 % and 82 %, respectively, of the slices of this lightweight device are still available for user IP cores.

The AES-OFB LR-AEAD scheme requires a FIFO buffer for the decrypted, but not yet authenticated partial bitstreams. As the BRAM modules on the Xilinx Zynq-7000 series are each 4.5 KB large, we split the partial bitstreams into segments of 4 KB with an additional 16 bytes for the tag appended to each segment. This configuration leads to a storage overhead of less than 0.4 percent. Note that for optimization the BRAM could potentially be shared between the PUF and FIFO, but this requires additional

[11] The *theoretical* side-channel security is not equivalent since the security proof for LR-PRG, contrary to the LR-PRF, requires use of the random oracle model. However this is only to prevent so called future computation attacks and has no practical significance.

Table 7.1.: Resource utilization.

Module	**Slices**		**BRAM**
	LUTs	**Registers**	**RAM36**
AES-OFB based LR-AEAD incl. FIFO	4667	2934	1
$FGHF'$ based LR-AEAD	3137	1779	0
PicoPUF incl. error correction	3917	4179	1
PR Controller	50	53	0

control logic and breaks separation between the modules, thus we did not pursue this option.

The $FGHF'$ LR-AEAD scheme allows moving parts of the authentication, the hash calculation, to software which significantly reduces the hardware overhead. The LR-AEAD hardware requires 3137 LUTs and 1779 slices, compared to a total of 4667 LUTs and 2934 registers for the AES-OFB based scheme and FIFO combined. This is a reduction of 33% and 39%, respectively. The LR-AEAD hardware also does not require any BRAM blocks as no buffering is required (previously the FIFO used one RAM36 block). In addition, the size of the encrypted bitstream is slightly reduced because only one tag for the entire bitstream is stored instead of one tag per segment.

As the performance, i.e., the latency when decrypting one partial bitstream, is dependent on the concrete implementation parameters, we only provide an informal discussion. We give a detailled performance comparison of the two LR-AEAD constructions in a similar setting in Section 8.5. For the AES-OFB based LR-AEAD, decryption of one partial bitstream requires two initial LR-PRF executions and one initial AES encryption. The subsequent decryption of each 128-bit block of data requires one AES encryption. Since the decrypted bitstream has to be buffered in a FIFO, it is potentially split up into multiple segments if it exceeds the size of the buffer. For each new segment, an additional evaluation of the LR-PRF and an AES encryption is required. The duration of one LR-PRF evaluation is significant and depends on the configuration, as explained in Section 3.2. Depending on the hardware platform, it can consist of up to 130 AES encryptions. How many LR-PRF evaluations the AES-OFB based LR-AEAD actually requires depends on the size of the FIFO and length of the partial bitstream.

For the $FGHF'$-based LR-AEAD, initially the hash has to be calculated in software and the LR-PRF is evaluated twice. Then for each 128-bit block of data two AES encryptions are necessary. Hence, the $FGHF'$-based LR-AEAD requires double the AES encryptions per 128-bit data block, but potentially less LR-PRF evaluations. Nevertheless, we expect that reconfiguration on such systems is a task that is most probably performed only once per boot. Therefore we suspect that designers will aim for lower hardware footprint instead of lower latency.

7.5. Towards software encryption and runtime security

Our proposal is currently limited to bitstream decryption. User software is authenticated but not encrypted because this different use-case can lead to new attack vectors as discussed in this section. For many applications this is sufficient, since rarely is the entire software stack confidential. If full confidentiality is desired for software, this usually goes hand-in-hand with runtime integrity protection and encryption of chip external RAM, which are hard problems on their own. However, if our AES-OFB based core is to be used for software decryption, the OFB mode of operation might not be the best choice and could lead to new attack vectors. The side-channel security of the decryption core relies on the fact that an attacker does not know the inputs to the underlying block cipher. In OFB mode this holds, as long as the attacker cannot observe both plain- and ciphertext. Otherwise, the XOR of both reveals the output of the block cipher. With that information a regular DPA on the last round of the cipher becomes possible. In the case of bitstream encryption, the decrypted bitstream is directly sent to the PR controller, i.e. the plaintext never leaves the hardware and is not exposed to the attacker. For software decryption, this is hard to guarantee since the plaintext is transferred back to the CPU and potentially ends up in external RAM.

A straightforward mitigation is to use key whitening to hide the in- and outputs to the cipher from the attacker. XEX [80] and XTS [1] are modes of operation that achieve this. However, there are several published side-channel attacks on those constructions. Luo et al. show an attack on the tweaking function of XTS that exploits the simple structure of the Galois field multiplication by 2 [57]. This attack can be prevented by using XEX which uses multiplications with arbitrary values. In contrast, Unterluggauer et al. attack AES in such a scenario directly by concatenating DPA attacks on the last two rounds [98]. This attack is feasible irrespective of the tweaking function but it requires the attacker to change the input while keeping the IV constant (e.g. writing the same sector multiple times in the case of disk encryption). Whether or not this is a relevant attack vector depends on the application.

In cases where the second, $FGHF'$ based LR-AEAD core is applicable, this attack vector is prevented by the LR-PRG that is used instead of AES-OFB. Similar to AES-OFB, each block of ciphertext is decrypted by XORing with one block of the keystream. However, the main difference is that in case of the LR-PRG the key changes from block to block. This means that the attacker cannot mount DPA attacks even if multiple plain- and ciphertext pairs are known because for every key, only one encryption is observable. This corresponds to an attack with data complexity 1, which is effectively an SPA attack. Since we already require that the AES implementation has to resist attacks with data complexity 2 in a side-channel evaluation, this attack is not feasible as it is even harder for the attacker.

We conclude, that only the LR-AEAD core based on an LR-PRG is capable of protecting firmware decryption where we expect that the attacker can guess large parts of the plaintext. Nevertheless, protecting the confidentiality of software demands much more than only secure decryption. For example, external memory is vulnerable to cold-boot attacks [40] and code can be reverse engineered by observing its behavior [84]. To

provide a sound solution for software encryption, these issues need to be addressed and, depending on the specific use case, additional measures have to be taken which are out of the scope of this thesis.

7.6. Summary

As a step towards a vendor agnostic solution, we present an SCA secure and fully updatable mechanism to securely configure the FPGA logic starting from power-up until the whole system is booted and running. To achieve this, we leverage the PR options of FPGAs, in conjunction with a PUF generated device intrinsic key and a leakage resilient authenticated decryption core to securely load hardware IP cores. While we target a Xilinx Zynq-7020 FPGA SoC for our proof-of-concept, the method is agnostic to a specific FPGA manufacturer/family as long as it provides certain common functionalities. The two key requirements to port this design are that the target provides:

1. Authentication and integrity checking of the static bitstream using public key cryptography

2. Partial reconfiguration

In the initial boot step, an authenticity and integrity check of the static bitstream containing the custom cryptographic cores is necessary. We use signature verification with public key cryptography on the Xilinx Zynq-7020 device to achieve this. Public key cryptography for this purpose is a widely adopted feature and is present in the majority of FPGA SoCs that are currently available. The benefit of using public key cryptography is that no additional *secret* key material is necessary and the operation does not need to be protected against key-recovery attacks. PR is necessary to reconfigure parts of the FPGA fabric at run-time with user IP cores that were decrypted by our protected engine. This feature is supported by the leading manufacturers of FPGA SoCs such as Xilinx and Intel (formerly Altera). Thus this design can be ported to these devices with low effort.

As the Xilinx Zynq-7000 series devices do not provide any user key storage (only 32 bits of general purpose eFuses are available), we opted to use a PUF for key storage. Newer devices, however, often provide user accessible secure key storage. Some devices like the Xilinx Zynq Ultrascale+ and Stratix 10 from Intel even include hard-core PUFs and allow secure boot using these PUFs. In such cases the custom PUF can be omitted if the key storage is trusted to be secure.

For the LR-AEAD core we provide two options which are applicable depending on the side-channel security of an AES core implementation on the target platform. Hence, when porting the design to a different technology, the leakage behavior of the decryption core can change and the side-channel security has to be re-evaluated. On platforms that only provide little side-channel resistance, we deploy the AES-OFB based scheme that uses our improved LR-PRF design. The lack of key entropy is mitigated by adding additional 2-PRG stages and key material. If the AES core resists attacks with data

complexity 2, then the $FGHF'$ scheme can be used. In that case part of the authentication can be implemented in software which reduces the hardware footprint by more than 30 %.

The presented work, to the best of our knowledge, is the first that allows side-channel secure field updates of user IP cores and the decryption engine without relying on any manufacturer provided *secret key* storage. The concept requires very limited trust in the manufacturer and provides the necessary flexibility if demands change or new attacks arise. This approach is orthogonal to upgrading to newer and more expensive feature-rich devices, and is also suited to retro-fit older devices as it uses features that are already widespread in current hardware.

8. Case study: Retrofitting LR-AEAD to off-the-shelf microcontrollers

As a second use case, this chapter considers the side-channel protection of COTS microcontrollers which are increasingly popular in IoT applications. Specifically, we examine the interesting problem of applying methods from leakage resilience to microcontrollers where we have no control over the hardware implementation and can only modify the software.

In general, the security of IoT devices relies on fundamental concepts such as protected firmware updates. In this context attackers usually have physical access to a device and therefore side-channel attacks have to be considered. The microcontrollers that are typically used may have hardware accelerators for cryptographic operations, but usually no built-in countermeasures against side-channel attacks. We demonstrate how the $FGHF'$ based LR-AEAD scheme, which consists of an LR-PRF, PRG and hash function, can be implemented using such unprotected hardware AES engines.

Our concept relies on parallel S-boxes in the AES hardware implementation to achieve the algorithmic noise required for the LR-PRF. Fortunately, this feature is present in many microcontrollers as a measure to increase performance, although not all accelerators are fully parallel and implement 16 S-boxes. To analyze the effect of differing parallelism on the security level, we implement the LR-AEAD scheme for two popular ARM Cortex-M microcontrollers with 16 and 4 parallel S-boxes, respectively. The proposed design uses the AES hardware accelerator to instantiate both the LR-PRF and the PRG, which means that most of the work load is handled efficiently in hardware. Existing hardware accelerators for the hash function can be used where available, otherwise the hash function can be implemented in software.

We evaluate the protection capabilities in realistic IoT attack scenarios, where non-invasive EM measurements are employed by the attacker. We show that the concept provides the side-channel hardening that is required for the long-term security of IoT devices. For comparison, the unprotected AES engines on both tested microcontrollers can be broken (i.e., the security level is reduced to 0 bit) after 2,500 traces when used in a standard mode of operation. After implementing the CHES 2012 LR-PRF on the same hardware with our proposed method, the cryptographic operation withstands similar side-channel attacks using the same measurement setup and results in security levels above 100 bits given our attacker model and measurement equipment.

We also provide an implementation and performance evaluation of the full LR-AEAD scheme for both controllers. On one controller we can make use of an existing SHA-256 accelerator to instantiate the hash function, on the other we use a software implementation. Since all modifications are software-only, our concept can be used to retrofit

existing designs. The fact that the only hardware requirement is an AES with parallel S-boxes makes this solution applicable to a wide range of microcontrollers. It is therefore highly relevant when adding side-channel protection to existing devices, especially when no true random noise sources are available. In such cases, masking or hiding is even impossible and this concept is without alternatives.

Outline First, we discuss the difficulties of protecting COTS microcontrollers and existing approaches towards securing them in Section 8.1. In Section 8.2 we outline the implementation details and trade-offs of LR-PRFs on microcontrollers with hardware acceleration. We also describe the hardware-software partitioning of the LR-AEAD scheme and discuss relevant attack vectors. Subsequently, Section 8.3 gives details about the two microcontrollers that are evaluated.

We present the results of our side-channel evaluation in Section 8.4: After introducing the attacker model and measurement setup in Section 8.4.1, we show in Section 8.4.2 that the key transfer to the hardware accelerators is not vulnerable, which is a necessary requirement for the proposed solution. Next, Section 8.4.3 demonstrates that the AES accelerators are in fact vulnerable to template attacks (TAs) using our setup which illustrates the need for countermeasures. Finally, Section 8.4.4 discusses the side-channel security of the LR-AEAD, which is determined by an analysis of the LR-PRF with different configurations. Section 8.5 assesses the performance overhead of the implementations in terms of runtime and code size. We summarize our findings in Section 8.6.

8.1. Related work

The information security of inexpensive IoT devices is especially important due to their high quantity, prevalence, and as a result the potentially high impact of attacks. Arguably the most important feature for such devices are secure firmware updates. They are needed to mitigate software vulnerabilities which are uncovered while a device is in the field.

Ronen et al. [81] highlight the implications of unprotected update mechanisms by using a side-channel attack to extract an AES master key from a smart light bulb which is used to protect firmware updates for an entire device family. Using this extracted update master key, a worm is created that automatically infects and maliciously replaces the firmware of similar devices within a 100 m radius. Evidently, these low-cost devices also require secure updates and protection against physical attacks.

Secure updates can either be achieved by digital signatures (as suggested by the authors of [81]) or by symmetric AEAD schemes. The advantage of using AEAD schemes is that they also provide confidentiality, not only authenticity as in case of signatures. This is often necessary to, e.g., protect IP or credentials in the firmware or to hide security issues that are fixed in new versions. Otherwise, the vulnerabilities are exposed to adversaries that inspect the differences between the versions and can be exploited in downgrading attacks.

Unfortunately, protecting cryptographic implementations against side-channel attacks is challenging, especially when dealing with existing hardware implementations without built-in protection. So far, the only countermeasure that can be retrofitted without giving up hardware acceleration, and therefore significantly reducing performance, is time-based hiding, i.e., inserting random delays or dummy operations before or after the critical operation. Such countermeasures require true randomness and are limited in their effectiveness because deliberate timing variations can often be filtered by signal processing. Particularly for COTS devices it has been shown that the cryptographic operation can be identified despite hiding countermeasures [42].

An alternative is to use a hardened software implementation instead of the existing cryptographic hardware accelerator and giving up its provided efficiency. The inherent difficulty of this task is evident in the following example. A team from the French ANSSI published an open-source implementation of a side-channel protected AES targeted for COTS microcontrollers [7]. As it is state of the art, they combine masking and shuffling countermeasures to protect against side-channel attacks and provide leakage tests that do not show leakage after 100,000 traces. Despite these seemingly positive results, Bronchain and Standaert [11] published an attack that succeeds with only 2,000 traces, which highlights the issues of combined countermeasures on these devices. In the same paper, they also put forward the general difficulty of securing COTS microcontrollers using masking or shuffling due to the lack of noise when countermeasures are implemented in software.

In this work, we therefore make use of existing hardware accelerators for cryptographic operations and use concepts from leakage resilience that leverage algorithmic noise and limited data complexity. We show the soundness of our proposal through actual side-channel attacks and give concrete security levels. Contrary to the previous example, there is no easy way to circumvent parts of the countermeasure since the source of the algorithmic noise which prevents key recovery is rooted in hardware.

8.2. LR-AEAD on COTS microcontrollers

This section describes how to achieve LR-AEAD for COTS microcontrollers. First, we explain how to implement an LR-PRF and LR-PRG utilizing a hardware AES engine. We specifically describe the partitioning between software running on the CPU and the hardware accelerators. Second, we describe how to use these building blocks together with a hash function in the LR-AEAD scheme of Degabriele et al. [20]. We provide pseudo code for all operations and point out the security critical operations which we analyze in the side-channel evaluation in Section 8.4.

The main aspect of our proposal is to benefit from existing hardware accelerators with parallel S-boxes on microcontrollers to realize an LR-PRF. A typical architecture of a microcontroller with integrated cryptographic coprocessor is outlined in Fig. 8.1. The AES coprocessor is attached to the main CPU via a bus and can run independently and in parallel. It is typically controlled through memory-mapped registers. Commands and data values are exchanged over the bus.

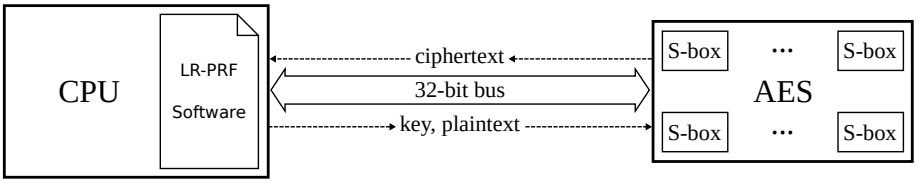


Figure 8.1.: Microcontroller running an LR-PRF using an integrated AES hardware accelerator.

For our analysis we implement the original CHES 2012 LR-PRF since it has the least implementation overhead and is best suited to analyze the security impact of different data complexity configurations. For this LR-PRF, the available data complexity in an attack depends on the configuration and ranges from 2 up to 256. However, we emphasize that the analysis of this LR-PRF with data complexity 2 also covers the ASIACRYPT 2016 LR-PRF and the LR-PRF with added key entropy presented in Chapter 6. As long as the case of data complexity 2 is shown to be secure for a given implementation of the CHES 2012 LR-PRF, all variants of the LR-PRF can be used. The goal of a laboratory evaluation is to find the highest data complexity, i.e., the most efficient configuration, which still achieves high security levels.

The LR-PRF program is executed on the CPU and the hardware accelerator is queried for the necessary block cipher encryptions. The process follows Algorithm 2, where the boxed operations are executed inside the hardware accelerator, while the rest is executed by the CPU. Inputs are the key k, the data input x and the data complexity expressed in the number of bits n that are processed per stage (e.g., for data complexity 4, n equals 2). The expression $(\text{nbits}|\ldots|\text{nbits})^{128}$ denotes a concatenation of bits *nbits* until the string contains 128 bits, 0^{128} is an all zero bitstring with length 128.

We use the 2-PRG based LR-PRG presented in [92] and give the pseudocode description in Algorithm 3. The functionality is split into two functions: An initial seeding of the LR-PRG that sets the key for the first iteration and an iterate function that returns one block of pseudorandom data per call and updates the key state internally. We identify two types of security critical operations in Algorithms 2 and 3. The first type are the AES encryptions (`AES_encrypt`). This is expected from all conceptual considerations.

The second type is implementation-specific, i.e., the bus transfers (`write_to/read_from_accelerator`) of the key. This attack vector is attributed to the use of a non-security controller without builtin countermeasures. In the case of commercial security controller, these bus transfers are masked by random values. In our case of unprotected microcontrollers this attack vector needs to be considered. Wouters et al. [108] demonstrate the effectiveness of such attacks and recover the transponder key of a car immobilizer in a profiled attack on the key transfer to a coprocessor. Therefore, to establish SCA secure LR-PRFs on such microcontrollers we evaluate two attack vectors in this work: *i)* Attacks on the bus transfer of the key in Section 8.4.2 and *ii)* attacks on the AES accelerator with different data complexity in Section 8.4.4.

We give a detailed SCA of both types of operations in Section 8.4. Algorithm 4 puts the building blocks together and describes the encrypt and decrypt operations of the

LR-AEAD. This does not add any additional attack vectors, as all sensitive operations are located within the LR-PRF and LR-PRG. The hash function in particular does not require countermeasures as it does not process any sensitive values.

Side-channel security of the LR-PRG As discussed in Section 7.4.3, the security proof of the LR-AEAD does not require the PRG to be leakage resilient or, in general, resistant to differential side-channel attacks. However, our block cipher based LR-PRG uses the same key twice during a single execution and thus provides the attacker an attack vector with data complexity 2. For all practical purposes this is equivalent to an attack on the LR-PRF, hence a secure LR-PRF with data complexity 2 implies the security of the LR-PRG.

Algorithm 2: LR-PRF

```
1  Function LR-PRF(x, k, n):
       /* Initialize with long term key */
2      key ← k;
       /* Iterate through the PRF stages */
3      for i ← 1 to 128/n do
4          nbits ← read_next_n_bits(x);
5          plaintext ← (nbits|...|nbits)^128;
6          write_to_accelerator(key, plaintext);
7          ciphertext ← AES_encrypt(key, plaintext);
8          read_from_accelerator(ciphertext);
9          key ← ciphertext;
10     end
       /* Whitening step */
11     plaintext ← 0^128;
12     write_to_accelerator(key, plaintext);
13     ciphertext ← AES_encrypt(key, plaintext);
14     read_from_accelerator(ciphertext);
15 return ciphertext;
```

Algorithm 3: LR-PRG

```
1  global key;
2  Function LR-PRG_seed(s):
       /* Initialize with seed */
3      key ← s;
4  return
5  Function LR-PRG_iterate():
       /* Update key */
6      plaintext ← 0^128;
7      write_to_accelerator(key, plaintext);
8      ciphertext ← AES_encrypt(key, plaintext);
9      read_from_accelerator(ciphertext);
10     next_key ← ciphertext;
       /* Generate output block */
11     plaintext ← 1^128;
12     write_to_accelerator(key, plaintext);
13     ciphertext ← AES_encrypt(key, plaintext);
14     read_from_accelerator(ciphertext);
15     key ← next_key;
16 return ciphertext;
```

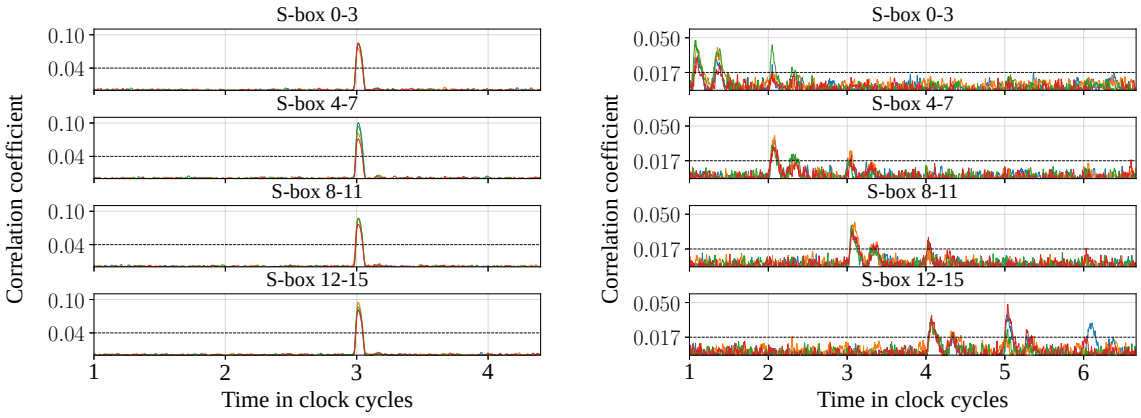
Algorithm 4: LR-AEAD

```
1  Function LR-AEAD_encrypt(msg, adata, nonce, k_enc, k_mac, n):
       /* Encrypt message block by block */
2      seed ← LR-PRF(nonce, k_enc, n);
3      LR-PRG_seed(seed);
4      ctxt ← [];
5      foreach msg_block in msg do
6          ctxt_block ← LR-PRG_iterate() ⊕ msg_block;
7          ctxt ← ctxt|ctxt_block;
8      end
       /* Calculate tag */
9      hash ← SHA-256(nonce, adata, ctxt);
10     tag ← LR-PRF(hash, k_mac, n);
11 return ctxt, tag;
12 Function LR-AEAD_decrypt(ctxt, adata, nonce, tag, k_enc, k_mac, n):
       /* Calculate and compare tag */
13     hash ← SHA-256(nonce, adata, ctxt);
14     tag′ ← LR-PRF(hash, k_mac, n);
15     if tag′ ≠ tag then
16         return ⊥;
17     end
       /* Decrypt message block by block */
18     seed ← LR-PRF(nonce, k_enc, n);
19     LR-PRG_seed(seed);
20     foreach ctxt_block in ctxt do
21         msg_block ← LR-PRG_iterate() ⊕ ctxt_block;
22         msg ← msg|msg_block;
23     end
24 return msg;
```

8.3. Devices under test: STM32 and EFM32

We use two different COTS microcontrollers for our proof of concept, namely the STM32F215RET6 (STM32) and EFM32PG12B500F1024 (EFM32) microcontrollers which are both widely used in IoT applications. Both devices are manufactured in a 90 nm technology and feature a 32-bit ARM Cortex-M4 and Cortex-M3 processor, respectively, and an AES hardware cryptographic accelerator that is not hardened against side-channel attacks. The cryptographic accelerators are different in their level of internal parallelism.

The datasheets lack information about the concrete implementation, but based on the number of clock cycles the cryptographic coprocessors take for a single AES operation we assume that the STM32 implements 16 parallel S-boxes to perform the 10 rounds of an AES-128 and the EFM32 implements four parallel S-boxes. This assumption is confirmed by the results of a correlation-based leakage test shown in Fig. 8.2a for the STM32 and in Fig. 8.2b for the EFM32.



(a) STM32 (b) EFM32

Figure 8.2.: Correlation-based leakage test on the AES S-box input for 100,000 traces with known plaintexts and keys.

In both figures the correlation for the input of the different S-boxes of an AES-128 encryption is depicted for known plaintext and key values. In Fig. 8.2a the maximum correlations for all 16 S-boxes of the STM32 occur at the same point in time indicating a fully parallel design. In Fig. 8.2b it can be observed that groups of 4 S-boxes behave similarly. This confirms that the accelerator implementation of the EFM32 processes 32-bit words of the AES state simultaneously which corresponds to four parallel S-boxes. The words exhibit two distinct peaks in subsequent clock cycles which overlap with the following word. This behavior is consistent with leakage caused by writing or overwriting a shared buffer register. For some S-boxes, e.g., S-box 12-15 at clock cycle 6, additional correlation peaks can be observed. This behavior is also visible several cycles after the computation of the current AES round. A reasonable explanation for these additional

peaks is the complex structure of the cryptographic coprocessor of the EFM32, which contains an ALU with dedicated instruction memory and data registers. The peaks could possibly stem from internal buffers or the switching of multiplexers between register banks.

In order to implement the LR-AEAD as described in Section 3.3 we utilize the SHA-256 hardware accelerator of the EFM32. Unfortunately, the STM32 only provides a SHA-1 accelerator that we opted to ignore, as practical attacks against SHA-1 have already been shown [54]. Instead, we use a software implementation of SHA-256 provided by the open source library *tinycrypt* [69], which is specifically designed for constrained devices.

8.4. Side-channel evaluation

In this section we present the results of a side-channel analysis of LR-AEADs on two microcontrollers. As explained in Section 8.2, the analysis of the LR-AEADs is reduced to an analysis of the LR-PRFs with different data complexity configurations. We are covering two attack vectors on the LR-PRF: Attacks on key transfers from CPU to the hardware accelerator and attacks on the AES accelerator which is used as part of the LR-PRF implementation. In that regard we first demonstrate that the key can be fully recovered if the AES is used in a standard mode where the data complexity for the attack is not limited. Within the LR-PRF, however, the AES is only used with limited data complexity, i.e., with a limited number of different plaintext inputs under one key. Thus, we provide results of attacks with different data complexities and give estimates of the remaining security level:

- We find that attacks on the key transfer do not lead to exploitable security levels.
- For the attacks on the AES, we observe that the security level decreases with rising data complexity.
- However, for both microcontrollers we find configurations that lead to high security levels greater than 100 bit.

We describe our attacker model and measurement setup in Section 8.4.1. In Section 8.4.2 we provide result of the attack on the key transfer, Section 8.4.3 and Section 8.4.4 cover attacks on the AES with unlimited and limited data complexitiy, respectively.

8.4.1. Attacker model and measurement setups

In general, an evaluator uses the best attack and equipment that an attacker is expected to have access to. Therefore the attacker model has a big influence on how the security level is determined. The same concept applies to certification schemes like *common criteria*, where higher certification levels assume attackers with more resources and expertise. In the security evaluations that we presented so far we used the best equipment that is available to us, namely high-end equipment for EM near-field measurements that additionally require decapping of the chip.

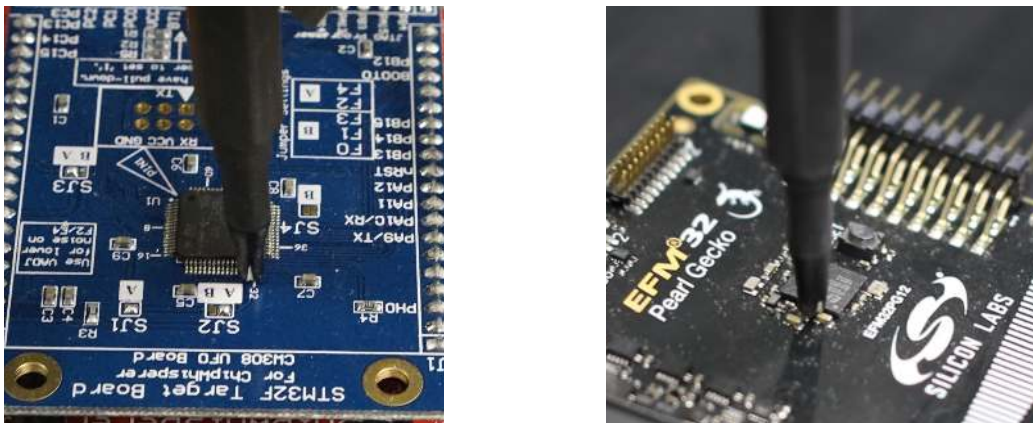
This effort might not be justified in all cases, in particular not when dealing with low-cost unprotected microcontrollers that, by default, provide no side-channel security at all. If we can secure these devices against large classes of side-channel attacks, then this is already a significant success. Applications which are so critical that they need to be secured against the most sophisticated attacks available will hopefully be implemented on dedicated security controllers instead. Hence, we define a more relaxed attacker model and, consequentially, use slightly less powerful equipment, which is cheaper to acquire and easier to use.

Attacker model For the discussed use-case, we do not consider invasive, high precision EM analyses that use equipment worth around 100,000 USD, excluding the equipment for decapping the chips before analyses. Instead, we assume attackers with considerable technical know-how, but moderate capabilities in terms of laboratory equipment. We assume that attackers have access to EM measurement probes allowing measurements close to the packaged chips with manual positioning of the probe. Along with a preamplifier and a USB oscilloscope, such a setup can be built for a few thousand USD.

Since samples of the analyzed devices can be bought without restrictions, attackers may chose to perform profiling with known keys on one or more devices under their control. To reflect this fact in our analysis and in order to avoid inter-device deviations, we perform profiling and attacks on the same device (can be seen as worst case), even though this would not be possible in a real scenario where an attacker has limited control over the attacked device.

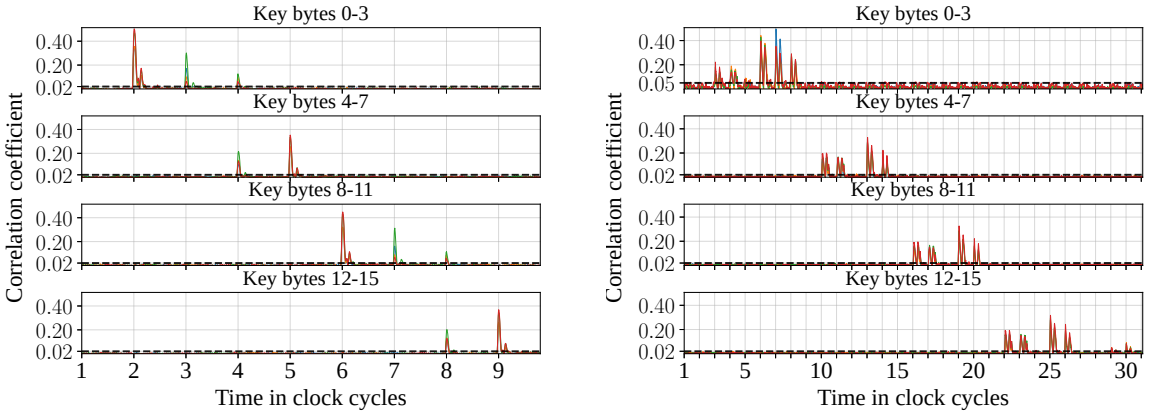
Measurement setups The measurement setup for the STM32 is depicted in Fig. 8.3a and consists of a CW308T-STM32F target board mounted on a CW308 UFO Board running at a clock frequency of 10 MHz. A PicoScope 6402D USB-oscilloscope is used for the data acquisition at a sampling rate of 1.25 GHz. The EM emanations are captured using a passive Langer RF-U 2.5-2 near-field probe that is connected to a Langer PA 303 preamplifier adding a gain of 30 dB. The EFM32 setup consists of an EFM32 Pearl Gecko PG12 Starter Kit running at its default clock frequency of 19 MHz. A LeCroy WavePro 7 Zi-A 2.5 GHz oscilloscope operating at a sampling rate of 5 GHz is used for the data acquisition. We use the same near-field probe and preamplifier as in the STM32 setup.

Consistent with the attacker model the probes are positioned manually. Different positions close to pins, capacitors and on top of the package were tested by inspecting signal amplitudes and qualitative indicators, e.g., if the AES round structure is visible. In case of the CW308T-STM32F target board, the probe is located on Pin 31 (c.f. Fig. 8.3a). For the EFM32 the probe is located in between two decoupling capacitors as shown in Fig. 8.3b. We demonstrate the efficiency of the setups in Section 8.4.3 by successfully attacking the AES core on both devices.



(a) STM32 (b) EFM32

Figure 8.3.: Positioning of the EM probes.



(a) STM32 (b) EFM32

Figure 8.4.: Correlation-based leakage test on the key transfer for 100,000 traces with known random keys.

8.4.2. Template attacks on key transfer

Protecting a cryptographic operation is only feasible if the key is not easily recoverable through SCA of the key transfer from the CPU to the cryptographic accelerator. We perform an evaluation of the key transfer for both microcontrollers and confirm that the leakage cannot be exploited to recover the key.

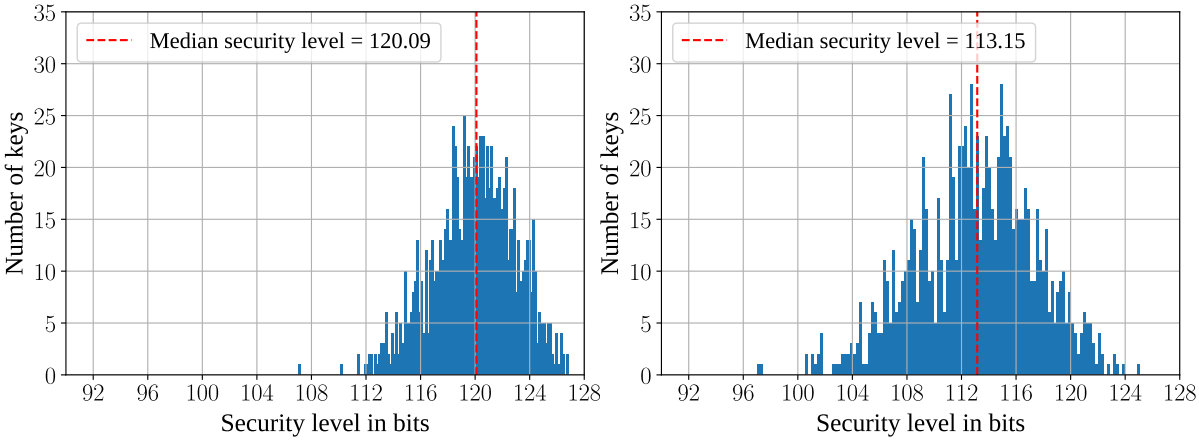
The AES hardware accelerator on both devices is connected to the CPU as a memory-mapped device. During the initialization the key has to be written to a register of the accelerator as four 32-bit words for a key size of 128 bits. The key transfer on the internal bus can be observed by an attacker, enabling a TA on the key bytes with a data complexity of 1. As the key is static, differential attacks are not possible.

Although the key is transferred in words of 32 bits, building templates for 32-bit values is not feasible due to the time which is required to collect a sufficient number of measurements for all 2^{32} templates. With the acquisition rate of our setup of about 50 traces per second, it would take roughly 10,000 years to collect the required traces, even for 16-bit values it takes over 60 days. The TAs carried out in the following sections are therefore based on 8-bit templates.

In a first step, a correlation-based leakage test on the key transfer is conducted to find the POIs for the TA. The correlation for the values of the different key bytes is depicted in Fig. 8.4a for the STM32, and in Fig. 8.4b for the EFM32. For both figures we use 100,000 traces with known random keys. Both devices show relatively high correlation values of approximately 0.4, which is expected for unprotected microcontrollers. Figure 8.4a shows that the duration of the entire key transfer is eight clock cycles and the leakage of the words is partly overlapping. The leakage test for the EFM32 looks similar except for the difference that the four 32-bit transfers do not overlap at all.

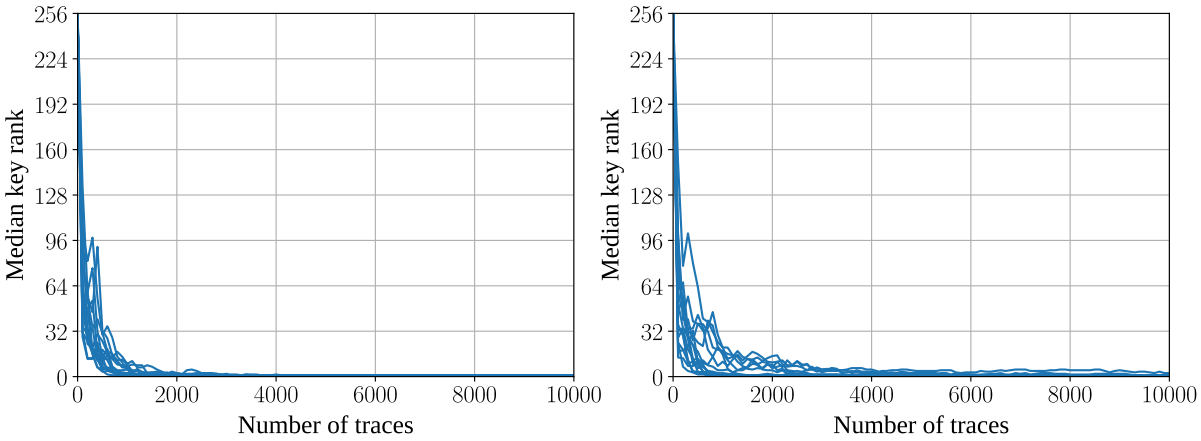
The POIs for the TA on the different words are obtained by using all samples that exhibit a correlation higher than 0.02 for the STM32. For the EFM32 we use a threshold of 0.05 for the first four key bytes and 0.02 for the other key bytes. Both thresholds have been determined visually as being just above the noise floor and are marked with a dashed line. The templates for the 256 possible values are generated from a total of 2,000,000 traces with known random keys in the case of the STM32, and 1,000,000 traces in the case of the EFM32. The difference in the number of traces stems from different acquisition rates on the two setups.

To evaluate the entropy reduction by the TA, 1,000 attacks with different random keys are performed where for each key 1,000 traces are recorded. We found that already for this number of traces per key, the results are stable and more traces per key do not further improve results.



(a) STM32 (b) EFM32

Figure 8.5.: Security level for 1,000 random keys subject to a template attack on the key transfer.



(a) STM32 (b) EFM32

Figure 8.6.: Median key rank of the 16 key bytes subject to a template attack on the AES S-box input for 10 random keys with random plaintexts for varying number of traces.

The results for both devices are depicted in Fig. 8.5. The median security level is 120.09 bits for the STM32 and 113.15 bits for the EFM32. Even the key with the worst security level in our experiments has a remaining entropy of 107 bits and 97 bits for the STM32 and EFM32, respectively. Summing up, while a TA on the key transfer reduces the entropy of the key, it does not compromise the security to an extent where a protection of the cryptographic operation would be pointless.

8.4.3. Template attack on unprotected AES

The idea of retrofitting protection for AES hardware accelerators is motivated by the assumption that the devices are vulnerable to SCA. Hence, we perform TAs on the hardware AES of the EFM32 and the STM32 to verify this assumption. Additionally, a successful attack result serves as a confirmation that the side-channel measurement setup including the manual positioning of the probe is effective. The evaluation also provides a baseline for comparison during the evaluation of the applied protection mechanism in Section 8.4.4.

As an intermediate value we target the S-box input of the first AES round, which is common practice for attacks on AES. The results of a correlation-based leakage test depicted in Figs. 8.2a and 8.2b are used to select suitable POIs. Again, the dashed line marks the threshold for the POI selection. For the STM32 and the EFM32 the threshold is set to 0.04 and 0.017, respectively. In the profiling phase, we acquired 2,000,000 traces (STM32) and 1,000,000 traces (EFM32) with random keys and random plaintexts. Multivariate templates are computed for each of the 256 possible intermediate values of all 8-bit templates. During the attack phase, 10 random keys are evaluated using up to 30,000 traces with random plaintext inputs.

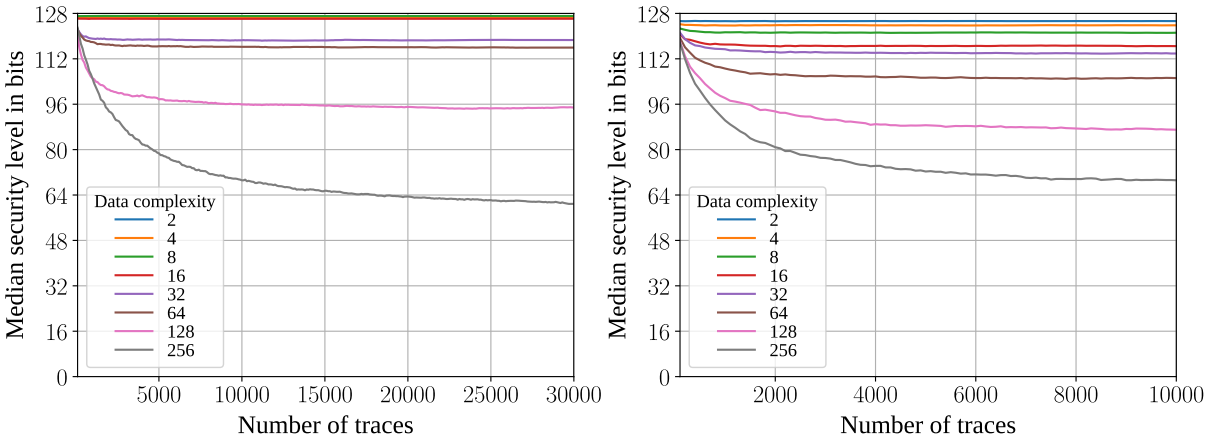
In contrast to the attack on the key transfer in Section 8.4.2, where all processed values

are constant and increasing measurements merely reduce noise effects, variable input values allow the attacker to increase the success probability by increasing the number of traces. Figures 8.6a and 8.6b depict the median key rank based on 10 random keys for an increasing number of traces. Each line represents the median key rank for one of the 16 key bytes (determined by attacking the corresponding S-boxes). The key rank denotes the position of the correct value if the results are sorted according to their likelihoods. A rank of 1 equals a successful recovery, while higher ranks require key enumeration effort by the attacker. For the STM32 the key byte ranks decrease to 1 after about 2,500 traces, i.e., a full key recovery is achieved. In the case of the EFM32, more traces are needed to achieve low key ranks. However, not all key bytes converge to a rank of 1 and therefore a low effort in key rank enumeration is still necessary. Summarizing, both AES engines are clearly vulnerable to SCA and protection mechanisms are required.

8.4.4. Template attacks on LR-PRFs with different data complexities

This section provides the side-channel evaluation of the proposed retrofitted protection mechanism based on LR-PRFs. The security level is determined by a side-channel attack on the initial use of the long-term secret, i.e., an attack on the first stage of the LR-PRF. The number of different plaintexts which are used in the LR-PRF construction is a trade-off and affects the data complexity of a side-channel attack and the runtime. All proposed LR-PRF constructions are able to use a varying number of parallel S-boxes that determine the amount of algorithmic noise. These contributions originally assume dedicated hardware designs where this choice can be made deliberately, whereas in our case the level of parallelism depends on the choice of the controller. Hence, an evaluator's goal is to determine the security of a hardware implementation (with a fixed number of parallel S-boxes) in relation to the data complexity. Then, the LR-PRF construction with the best trade-off between security (security level subject to an attack) and efficiency (implementation cost and runtime) is chosen.

We analyze the security level of the implemented concept for different trade-offs by performing TAs with different data complexities. A side effect of the limited data complexity is that certain combinations of key bytes are easier to attack than others, depending on the leakage behavior of the device. For regular differential attacks with unlimited data complexity, this effect is compensated when observing many traces with different inputs. Hence, we test several keys and estimate the distribution of the resulting security level. The same profiling set as in Section 8.4.3 is used. Data complexities of $\{2, 4, 8, 16, 32, 64, 128, 256\}$ are used and for each 300 different random keys are attacked. All plaintexts are of the form that all 16 bytes are equal as described by Medwed et al. [65]. For each of the 300 attacked keys, we collected a high number of traces to reduce the measurement noise. Specifically, we recorded 30,000 traces in case of the STM32 and 10,000 traces in case of the EFM32. We found that the attack does not achieve further significant improvement with higher numbers of traces.



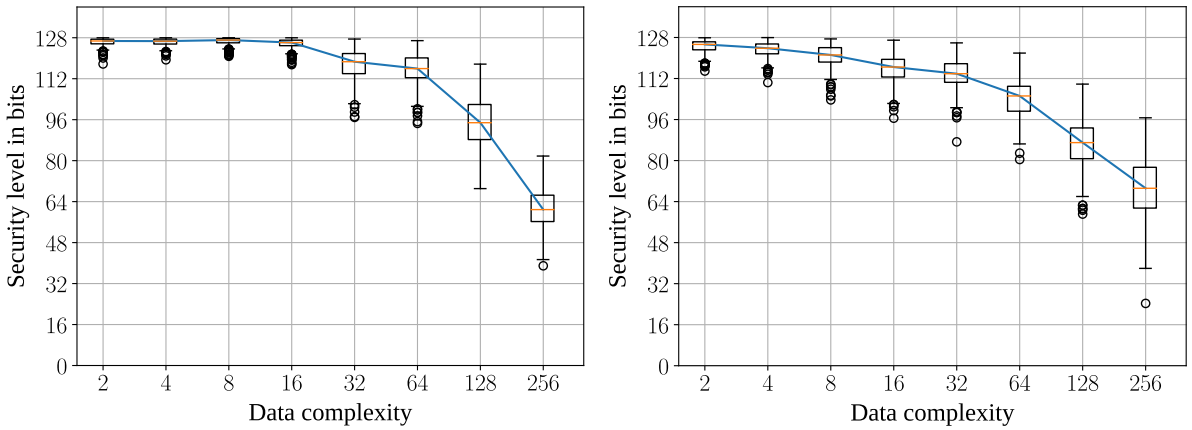
(a) STM32 (b) EFM32

Figure 8.7.: Median security levels from 300 random keys subject to a template attack on the AES S-box input for varying number of traces and data complexities.

Figures 8.7a and 8.7b present the attack results as a median security level[12] of the full key in bits (after key rank estimation) over the number of traces and for different data complexities. As expected, the security level generally decreases with increased numbers of traces. However, importantly, the security levels do not approach 0 bit security because of the protection mechanism. Contrarily, the security levels stagnate and do not decrease further after a certain number of traces, which proves that the protection mechanism is effective. Note that a data complexity of 256 means that all values are used per plaintext byte. However, all plaintext bytes are still equal for the parallel S-boxes. This is the important difference to a regular attack scenario as described in Section 8.4.3 and the prevalent reason for the working protection in these cases.

The security level of individual keys depends on their concrete value in the case of limited data complexity. Keys are chosen at random and the resulting security levels after attacks vary accordingly. It is therefore important to not only consider the median security level but the whole distribution as the outliers determine the worst-case security level. We show this variance in Figs. 8.8a and 8.8b. The figures present attack results after the maximum number of traces, i.e., 30,000 traces for the STM32 and 10,000 traces for the EFM32. Hence, this focuses on the rightmost verticals of Fig. 8.7. Each vertical on the x-axis contains the security levels from 300 attacks for this data complexity and the fixed number of attack traces.

[12]The median is used instead of the mean as it denotes the security level that an attacker achieves in 50% of the cases. However, as the security levels are nearly normally distributed depicting the mean would result in similar values.



(a) STM32 (b) EFM32

Figure 8.8.: Security levels of 300 random keys subject to a template attack on the AES S-box input for 30,000 (STM32) respectively 10,000 (EFM32) traces and different data complexities.

The distribution of the 300 attack results is visualized as a box plot. The red lines in the center of the boxes denote the median for the respective data complexity (this corresponds to the data shown previously on the rightmost vertical). The boxes include 50 % of the values within the quartiles Q1 and Q3[13]. All whiskers of the boxplots are drawn at 1.5 IQR or at the extrema. Outliers that diverge more than 1.5 IQR from the box edges are denoted as circles.

As most important observation we note that for data complexities up to 16, security levels including outliers are higher than 96 bits, which is a very positive result. In comparison, for this number of attack traces, the attack on the unprotected AES results in a security level close to 0 bit. As expected and observed in the previous results, the security level decreases with increasing data complexity, i.e., if the number of observable plaintexts is extended. Interestingly, the distribution is rather broad with high differences between the median security level and the worst case of individual keys. Considering the STM32 implementation configured with data complexity 128 as an example, a median security level above 90 bits is achieved but individual cases are as low as 70 bits. The variance is similar for both microcontrollers which reinforces the assumption that the choice of key values, not the measurement setup or hardware implementation, is the main reason for this observation.

The two AES engines have different numbers of parallel hardware S-boxes. The EFM32 includes four parallel S-boxes while the STM32 is fully parallel with 16 S-boxes. This means that the desired key dependent (algorithmic) noise from parallel structures is higher for the STM32. This explains the observation of higher resulting security levels at lower data complexities (2 to 64) as can be observed when comparing Figs. 8.7a and 8.7b. The results confirm that the higher parallelism provides better protection. For

[13]Q1 and Q3 are defined as the points where 25 % of all values are below Q1 and 25 % above Q3. The interquartile range (IQR) is the distance between Q1 and Q3.

the EFM32 with four parallel S-boxes the algorithmic noise provides less protection. Nevertheless, the security level still remains above 100 bits for data complexities smaller than 16.

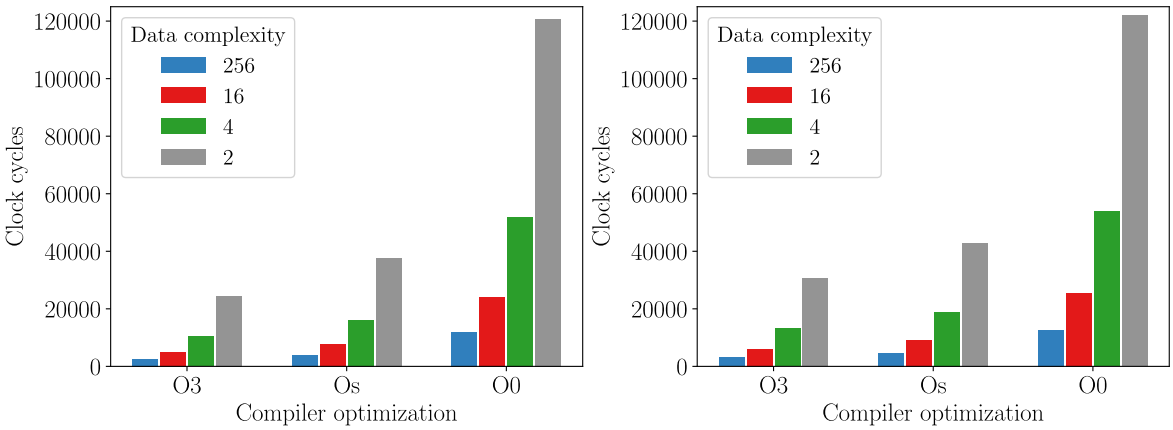
Note that the median security level for the data complexity of 256 is lower for the STM32. This is in contrast to the described reasoning and can, in our opinion, be explained using Fig. 8.6, which shows that the TA on the unprotected EFM32 does not converge to a key rank of one for single bytes. For higher data complexities, attacks generally become more comparable to attacks on the unprotected AES. The results in Fig. 8.6 for the unprotected case show that the EFM32 is harder to attack judging by the required number of traces and the partially imperfect key recovery. This could be the reason why the security level of the EFM32 does not decrease at the same rate for high data complexities. Thus we note that the device which was easier to attack without any countermeasures, the STM32, proves to be the more secure platform to implement LR-PRFs due to the higher level of parallelism of the hardware accelerator. In other words, the higher parallelism clearly leads to comparably better protection with the LR-PRF concept despite the fact that the same device is less secure when unprotected.

In summary this evaluation shows that secure LR-PRFs, and consequentially secure LR-AEAD, can be achieved on both devices. High security levels above 100 bit are achieved for all experiments with data complexities up to 16 on the STM32 and up to 8 on the EFM32. This shows that for protection in IoT scenarios, it is sufficient to use the CHES 2012 LR-PRF as long as the data complexity is within the discussed boundaries for the targeted security level. Comparing the two devices at hand, the fully parallel AES implementation of the STM32 allows for more efficient constructions while retaining a higher security level. Naturally, the unknown-inputs and the improved LR-PRF shown in Chapter 6 can also be used since they are limited to data complexity 2 by design and hence their side-channel security is equivalent to those cases shown here for the CHES 2012 LR-PRF. However, they come with the downside of additional preprocessing steps and the requirement to temporarily store secret plaintexts.

8.5. Performance analysis

In this section we evaluate the performance of our LR-PRF and LR-AEAD implementation with respect to execution time and code size on both microcontrollers. We analyze the impact of different data complexities, however, we only consider LR-PRF configurations that process a number of bits per stage that is a divisor of 128 (i.e., $n = 1, 2, 4, 8$ corresponding to data complexities of $2, 4, 16, 256$). This avoids having a last LR-PRF iteration that does not use the full data complexity.

In order to measure the execution time of the LR-PRF and LR-AEAD, we use the data watchpoint and trace (DWT) debug component of the ARM Cortex-M processor. This feature allows for non-invasive and cycle accurate execution time measurements. Non-invasive in this case means that it is not necessary to modify the code under test to perform the timing measurements.



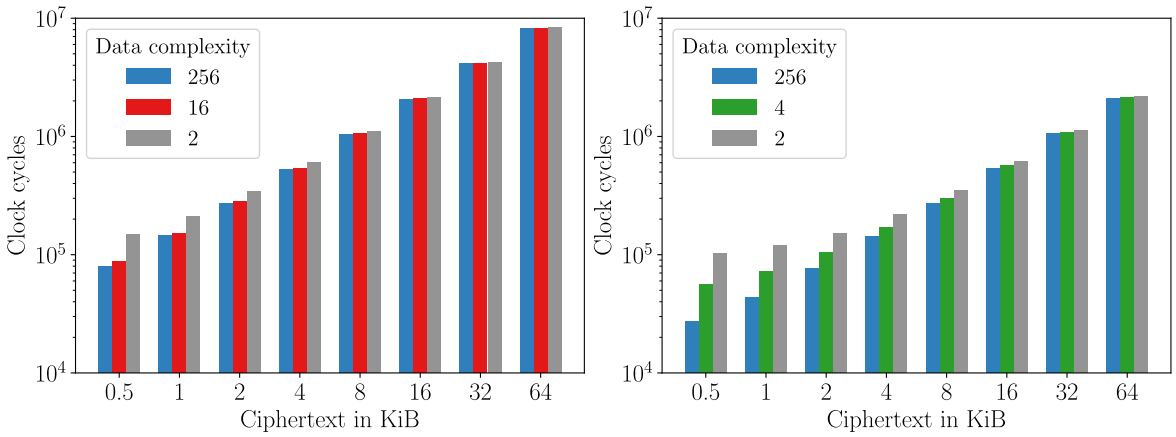
(a) STM32 (b) EFM32

Figure 8.9.: Performance evaluation of the LR-PRF implementation for different optimization levels and varying data complexities.

Single LR-PRF execution Figure 8.9 depicts the number of clock cycles required for a single LR-PRF execution with varying data complexities and different compiler optimization levels. We use three different optimization levels: no optimization (O0), optimization for size (Os) and optimization performance and size (O3). The results are also included in tabular format in Table B.1 in Appendix B. Note that a particular optimization level choice does not have an impact on the side-channel security, as the AES hardware accelerator is not influenced by the compiler. The same holds for the key transfer over the bus. The diagram shows that the number of clock cycles grows logarithmically with increasing data complexity (i.e., linearly with the number of input bits processed per iteration). This is expected since an increasing data complexity leads to a decreasing number of required iterations in the LR-PRF tree.

Contrary to our expectation, Fig. 8.9 does not reflect the performance difference between the AES coprocessors of the devices. Even though the STM32 has a fully parallel AES core, the LR-PRF implementation is only slightly faster than the one on the EFM32. We would expect a factor of about four because a fully parallel implementation is capable of calculating an AES round in one cycle whereas an implementation with four S-boxes requires four cycles. We assume that the difference results from the different low-level software libraries used for accessing the accelerators and differences in the interfacing protocols. For both microcontrollers, the optimizations Os and O3 result in a 2 and 3 times faster execution time, respectively, in comparison to the baseline without optimization (O0). Given the fact that the difference in performance is only marginal for practical applications, we suggest using the optimization Os as it comes with the additional benefit of a reduced code size. Therefore, we evaluate the LR-AEAD performance with optimization level Os.

Complete LR-AEAD execution For the evaluation of the complete LR-AEAD, we measure its execution time for different data complexities and varying ciphertext sizes



(a) STM32 (b) EFM32

Figure 8.10.: Performance evaluation of the LR-AEAD implementation for different data complexities and varying ciphertext sizes.

on both microcontrollers. We use the Os optimization level for the evaluation depicted in Fig. 8.10. We evaluate the LR-PRF with the minimum and maximum data complexity of 2 and 256 on both controllers. Additionally, we evaluate the LR-PRF with a data complexity of 4 and 16 for the EFM32 and STM32, respectively. These values turned out to be a suitable trade off between execution time and security in Section 8.4.4 (disregarding configurations for n that are not a divisor of 128). These results and additionally the required clock cycles to process a single 16-Byte block of data are also listed in Table B.2. The necessary clock cycles for the basic function calls, `AES_encrypt()`, `LR-PRG_seed()` and `LR-PRG_iterate()`, are given in Table B.3.

On the EFM32 the implementation makes use of the SHA-256 hardware accelerator whereas on the STM32 the hash is implemented in software [69]. This leads to a decreased performance of the STM32 in the LR-AEAD case despite the fact that it has a slightly faster AES core. For smaller ciphertexts one can clearly see the advantage of a higher data complexity, however, the difference vanishes with increasing ciphertext lengths. The reason is that the LR-PRF is only evaluated twice, regardless of the length of the ciphertext, and thus the overhead amortizes with increased length. In the context of firmware updates, we usually deal with encrypted firmware images larger than 16 KiB, hence, the performance penalty from decreased data complexity is low. Assuming a core clock frequency of 4 MHz, a data complexity of two and a firmware update size of 64 KiB, the decryption process takes around two seconds on the STM32 and less than a second on the EFM32. These results are quite practical for a secure firmware update in IoT applications.

Code size Besides the execution time of the LR-PRF and the LR-AEAD, their code size is an important parameter, especially for constrained embedded devices. In order to determine the code size required to retrofit the LR-AEAD to an existing application, we look at the additional code size of both functions when added to a simple baseline

application. This application consists of only a main function with an endless while loop together with the necessary initialization routines such as stack initialization. We use the example code from the microcontroller manufacturers as a template for the baseline application and implement both the LR-PRF and LR-AEAD on top. The additional code size, i.e., the code size required for the LR-PRF and the full LR-AEAD (including the LR-PRF) is listed in Table 8.1.

Table 8.1.: Code size in bytes of the LR-PRF and LR-AEAD implementations.

	STM32			EFM32		
Code	0s	03	O0	Os	O3	O0
LR-PRF	544	604	976	576	668	1,144
LR-AEAD	2,236	3,236	3,992	1,720	2,872	3,076

The LR-AEAD implementation occupies between 0.43 % and 0.76 % of the STM32's 512 KiB flash memory, depending on the optimization level. For the EFM32, the implementation needs between 0.16 % and 0.29 % of the microcontroller's 1024 KiB flash memory.

Comparison to a protected software implementation As a comparison, we measure the performance of a side-channel protected software implementation of AES developed by the ANSSI [7]. They implement affine masking as described in [29] in combination with several hiding countermeasures. We measured around 108,000 cycles for one call to the protected `aes()` function on an ARM Cortex-M4 microcontroller similar to their reference platform (optimization Os). For the example of 64 KiB firmware updates, we can give a rough estimate of the runtime by considering only the AES calls that are required to decrypt the firmware. This significantly underestimates the real runtime because it neglects the overhead that arises when implementing a block cipher mode of operation and the MAC calculation. It does also not include the collection of the randomness that is required for the countermeasures. Nevertheless, with an estimate of more than 442 Mio. clock cycles this alone is a factor 208 and 45 slower compared to the complete LR-AEAD with data complexity 2 on the EFM32 and STM32, respectively. This means that a secured firmware update would take in the range of minutes instead of single seconds which is significant. The code size of 6,392 bytes is also about three times larger.

Note that none of our implementations were optimized for performance and our aim was not to compare to the fastest possible implementation. The numbers given only serve the purpose of estimating the overhead of our solution compared to software-only countermeasures. There is significant potential for performance optimization for our implementation, e.g., by replacing the manufacturer provided API calls to the AES with optimized low level register accesses. However, this kind of optimization was not the focus of this work and we show that even a basic implementation of the provided scheme performs well in comparison.

In summary, the runtime overhead and the flash memory footprint of the LR-PRF and the LR-AEAD implementation are low in comparison. Thus our solution is applicable for secure firmware updates in resource constrained scenarios such as IoT applications where alternative protection exhibits prohibitive runtime. The overhead of using lower data complexities to achieve higher security levels becomes less significant if the length of the payload increases.

8.6. Summary

In this chapter we explored how to use concepts from leakage resilient cryptography to tackle the difficult problem of securing COTS microcontrollers against side-channel attacks. We propose to implement an LR-AEAD scheme using a block cipher based LR-PRF as the underlying side-channel hardened primitive. Specifically, we implement the LR-PRF in software and use existing hardware accelerators to leverage the algorithmic noise of parallel implementations to protect against side-channel attacks. In a case study on two ARM Cortex-M controllers with AES accelerators we analyze the side-channel security of our construction and find that it resists profiled attacks and retains security levels above 100 bits. We give concrete results for a configuration parameter that allows a trade-off between security level and performance. The overhead in code size is small and occupies only about 1 percent of the available memory on the two controllers. Compared to an exemplary side-channel protected software AES implementation, the runtime of our proposal is up to 200 times faster with a memory footprint of only one third. Our solution is applicable to any microcontroller that has an AES accelerator with parallel S-boxes. Therefore, it enables retrofitting side-channel protection to a wide range of devices. This will help to realize root of trust security mechanisms such as secure firmware updates for low-cost IoT devices.

9. Conclusions

This thesis started by revisiting the LR-PRF by Medwed et al. [65]. Its resilience to side-channel attacks relies on novel assumptions: limited data complexity and equal leakage of parallel hardware. We evaluated, whether these leakage assumptions hold against high precision EM analysis and showed that one of the crucial requirements for the side-channel resistance, the "equal leakage of S-boxes", is violated in the examined FPGA designs. As an important finding, our laboratory analysis showed that this can not only be attributed to the high spatial but also to the high temporal resolution of the utilized equipment. It is unlikely that this can be mitigated on FPGA devices by routing or placement changes alone because, unlike on ASIC devices, the actual physical hardware layout cannot be changed. Indeed, despite significant effort to equalize placement and routing of the individual S-boxes we were unable to achieve equal leakage. Thus we conclude, that on the examined FPGA device no secure implementation of this scheme is possible with the means available to the designer. Not even the most secure LR-PRF configuration with minimum data complexity is able to provide an acceptable security level.

Based on the insights of this laboratory analysis, we proposed a modification of the LR-PRF that allows the secure introduction of additional key material to raise the overall security level. This improved version of the LR-PRF also resists high-end attacks on devices that previously failed to achieve high security levels, all at the cost of longer keys and increased latency.

We demonstrated that an application of this LR-PRF is practical by means of two case studies: bitstream decryption for FPGA SoCs and firmware updates for microcontrollers. In case of bitstream decryption, we achieve authenticated decryption that is fully updatable and requires almost no trust in manufacturer provided cryptographic cores. In case of microcontroller firmware decryption, we proposed a novel implementation approach that uses AES co-processors which are already present on many modern microcontrollers. This has the unique benefit that it allows leveraging algorithmic noise from the parallel hardware of the AES core while requiring only software modifications. Typically, software-only countermeasures like masking suffer from the fact that the noise level of the microcontroller's CPU is low. By using hardware accelerators for the security critical operations we circumvent this problem and at the same time benefit from improved performance. These examples show that practical applications can be secured against side-channel attacks through the LR-PRF and with very limited requirements on the hardware platform.

Leakage resilient constructions require practical evaluation However, the results presented in this thesis also highlight the main difficulty of leakage resilient cryptography

in practice: the modeling of a leakage function for the theoretic analysis that translates into empirically verifiable assumptions in practice. In the provided security evaluations we resorted to the most powerful attacks using high-end equipment and extensive chip preparation to determine the security level. This requires expensive equipment and expert knowledge, yet, it is still difficult to achieve general results as the quality of the attack is dependent on many factors such as the time allocated for the measurements (i.e., the number of tested measurement locations) and the noise level in the traces after signal processing. The expertise of the hardware designer also has a strong impact on the achieved security level. We observed that for different design strategies the security levels vary by a large margin and that the best strategy is not immediately clear.

These practical findings contrast the idea of leakage resilience as coined by Dziembowski and Pietrzak [25], which is defined by a move away from ad-hoc countermeasures and evaluations towards a more thorough theoretical treatment resulting in tight security bounds. However, this observation is also consistent with the current state of research which comprises two major strains: There is the work coming out of the cryptology community which directly follows Dziembowski and Pietrzak and introduces new modes and provides the theoretical treatment and proofs. Typically, this comes down to some construction with limited data complexity and the security bounds often rely on the bounded leakage per execution. Then there is the practitioners track of research, coming from people with a background in practical side-channel analysis. This is where the work of Medwed et al. and also this thesis is situated. Here, the theoretical treatment is often neglected in favor of more heuristic security notions.

The results presented in this thesis showcase that this second, more practical strain of research is equally as important as the theoretical foundation. As of now, the leakage of integrated circuits still seems to be too complex to be fully captured by a mathematical model without it becoming too generic and thus allowing unrealistic attack scenarios. Considering, e.g., the equal leakage of S-boxes that we analyzed on FPGA designs. For the same hardware description, we observed varying leakage behavior with different placement strategies. We saw that this behavior is rooted in minuscule changes in the timing behavior of signals relative to each other and requires sophisticated measurement equipment to be made visible. An evaluator using simpler equipment might not be able to pick up these differences and could fail to attack the design, hence overestimating the achievable security level. Of course attackers face the same problems, but from an evaluators point of view this all-or-nothing scenario is undesirable because it does not allow extrapolating results from a given setup. As including all these factors in a leakage model seems to be an intractable problem, determining the practical security level requires a skilled evaluator using adequate equipment.

Comparison to masking countermeasures The most prominent and widely deployed countermeasure against side-channel attacks is currently masking. In a direct comparison, the analyzed LR-PRF offers certain benefits: It does not require high quality random numbers which might not be available on a given platform or can be expensive to generate. Also it does not require specialized hardware; as shown in the two case

studies a standard AES accelerator is sufficient to implement the scheme. That makes it applicable to a wide range of (legacy) devices.

Finally, the achievable security level is independent of the order of the attack. Masking schemes are usually categorized by the order of protection, e.g. a first-order secure scheme processes data that is split up into a masked value and the mask. This is secure against attacks that exploit leakage of any *one* of the two shares, but can always be broken by combining the leakage of the two shares. Exploiting more than one intermediate variable at once in a side-channel trace is more prone to noise and hence requires a lot more traces for a successful attack. Yet, in practice a combination with noise amplification methods and hiding countermeasures is often necessary to meet security requirements, which are often defined as the number of traces that an attacker is allowed to evaluate.

This is fundamentally different for leakage resilient constructions that rely on limited data complexity such as the discussed LR-PRF. By design, the information about the secret that is exposed to an attacker is limited to what is contained in the very few different traces that are available. Increasing the number of traces only reduces the amount of noise for those traces, but does not expose new information to the attacker. The attacker's goal is then to extract as much of this information as possible by deploying the most powerful measurement and analysis methods. There is, however, an intrinsic limit on the amount of information that can be extracted which is dependent on the device and the measurement equipment. Contrary to the case of masking, there is no further gain when acquiring more traces beyond the removal of (measurement) noise. There are also no straightforward higher-order attacks that break the construction and to the best of our knowledge multivariate first-order attacks are the most efficient tool in this scenario. The security level is therefore fully determined by the extractable information of the few unique traces and independent of the order of the applied attack, which seems a more generic approach.

Future work A factor that could hinder the adoption of leakage resilience as a countermeasure is the high effort that is required for a sound practical evaluation. As conducted in this thesis, the evaluation includes profiling and conducting a full template attack including key rank estimation to determine the security levels. Further, this has to be repeated for a sufficient number of different keys since in the low data complexity scenario the actual values of the key bytes have a big influence on the attack efficiency. Therefore research into "shortcuts" that speed up evaluation would be beneficial. For masking countermeasures, e.g., a (first-order) leakage test is sufficient to prove the effectiveness of the measure and for standard differential attacks the success rate can be estimated from the correlation coefficient. Finding a similar, easy to determine, metric (maybe based on the multivariate SNR) would help to formalize the evaluation.

Another interesting research direction is to further explore how to use existing hardware accelerators on common microcontrollers for side-channel hardened hardware/software-codesigns. We made a first step by utilizing an AES accelerator to benefit from its algorithmic noise. Similarly, leakage resilient constructions could be designed that use, e.g., hash accelerators as building blocks.

Bibliography

[1] IEEE standard for cryptographic protection of data on block-oriented storage devices. *IEEE Std 1619-2007*, pages 1–32, 2008.

[2] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–495. Springer, 2009.

[3] M. Akkar and C. Giraud. An implementation of DES and AES, secure against some attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

[4] C. Archambeau, E. Peeters, F. Standaert, and J. Quisquater. Template attacks in principal subspaces. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[5] G. T. Becker and R. Kumar. Active and passive side-channel attacks on delay based PUF designs. *IACR Cryptology ePrint Archive*, 2014:287, 2014.

[6] S. Belaïd, F. De Santis, J. Heyszl, S. Mangard, M. Medwed, J.-M. Schmidt, F.-X. Standaert, and S. Tillich. Towards fresh re-keying with leakage-resilient PRFs: cipher design principles and analysis. *Journal of Cryptographic Engineering*, 4(3):157–171, 2014.

[7] R. Benadjila, L. Khati, E. Prouff, and A. Thillard. Hardened library for AES-128 encryption/decryption on ARM Cortex M4 achitecture. https://github.com/ANSSI-FR/SecAESSTM32, Commit: 39af47f.

[8] D. J. Bernstein, T. Lange, and C. van Vredendaal. Tighter, faster, simpler side-channel security evaluations beyond computing power. *IACR Cryptology ePrint Archive*, 2015:221, 2015.

[9] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010,*

October 23-26, 2010, Las Vegas, Nevada, USA, pages 501–510. IEEE Computer Society, 2010.

[10] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

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

[12] N. Bruneau, S. Guilley, A. Heuser, D. Marion, and O. Rioul. Less is more - dimensionality reduction from a theoretical perspective. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2015.

[13] D. Canright. A very compact S-box for AES. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[14] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[15] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[16] O. Choudary and M. G. Kuhn. Efficient template attacks. In *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2013.

[17] O. Choudary and M. G. Kuhn. Template attacks on different devices. In *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 179–198. Springer, 2014.

[18] C. Clavier, J. Coron, and N. Dabbous. Differential power analysis in the presence of hardware countermeasures. In *Cryptographic Hardware and Embedded Systems*

- CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000.

[19] G. D. Crescenzo, R. J. Lipton, and S. Walfish. Perfectly secure password protocols in the bounded retrieval model. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 225–244. Springer, 2006.

[20] J. P. Degabriele, C. Janson, and P. Struck. Sponges resist leakage: The case of authenticated encryption. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 209–240. Springer, 2019.

[21] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 511–520. IEEE Computer Society, 2010.

[22] Y. Dodis, Y. T. Kalai, and S. Lovett. On cryptography with auxiliary input. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 621–630. ACM, 2009.

[23] F. Durvaux and F. Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 240–262. Springer, 2016.

[24] S. Dziembowski. Intrusion-resilience via the bounded-storage model. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2006.

[25] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 293–302. IEEE Computer Society, 2008.

[26] M. Ender, A. Moradi, and C. Paar. The unpatchable silicon: A full break of the bitstream encryption of Xilinx 7-series FPGAs. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1803–1819. USENIX Association, 2020.

[27] J. Ferrigno and M. Hlavác. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008.

[28] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[29] G. Fumaroli, A. Martinelli, E. Prouff, and M. Rivain. Affine masking against higher-order side channel analysis. In *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010.

[30] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[31] B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security - CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 148–160. ACM, 2002.

[32] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[33] B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST Non-invasive attack testing workshop*, 2011.

[34] C. Glowacz, V. Grosso, R. Poussier, J. Schüth, and F. Standaert. Simpler and more efficient rank estimation for side-channel security assessment. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2015.

[35] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 464–479. IEEE Computer Society, 1984.

[36] J. D. Golic and C. Tymen. Multiplicative masking and power analysis of AES. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.

[37] L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[38] H. Groß, S. Mangard, and T. Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.

[39] C. Gu and M. O'Neill. Ultra-compact and robust FPGA-based PUF identification generator. In *2015 IEEE International Symposium on Circuits and Systems, ISCAS2015, Lisbon, Portugal, May 24-27, 2015*, pages 934–937. IEEE, 2015.

[40] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60. USENIX Association, 2008.

[41] N. Hanley, M. Tunstall, and W. P. Marnane. Unknown plaintext template attacks. In *Information Security Applications, 10th International Workshop, WISA 2009, Busan, Korea, August 25-27, 2009, Revised Selected Papers*, volume 5932 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2009.

[42] B. Heinz, J. Heyszl, and F. Stumpf. Side-channel analysis of a high-throughput AES peripheral with countermeasures. In *2014 International Symposium on Integrated Circuits (ISIC), Singapore, December 10-12, 2014*, pages 25–29. IEEE, 2014.

[43] J. Heyszl, D. Merli, B. Heinz, F. D. Santis, and G. Sigl. Strengths and limitations of high-resolution electromagnetic field measurements for side-channel analysis. In *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, volume 7771 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2012.

[44] V. Immler, R. Specht, and F. Unterstein. Your rails cannot hide from localized EM: how dual-rail logic fails on FPGAs - extended version. *Journal of Cryptographic Engineering*, 8(2):125–139, 2018.

[45] Y. Ishai, A. Sahai, and D. A. Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[46] N. Jacob, J. Wittmann, J. Heyszl, R. Hesselbarth, F. Wilde, M. Pehl, G. Sigl, and K. Fischer. Securing FPGA SoC configurations independent of their manufacturers. In *30th IEEE International System-on-Chip Conference, SOCC 2017, Munich, Germany, September 5-8, 2017*, pages 114–119. IEEE, 2017.

[47] J. Jaffe. A first-order DPA attack against AES in counter mode with unknown initial counter. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2007.

[48] Y. T. Kalai and L. Reyzin. A survey of leakage-resilient cryptography. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 727–794. ACM, 2019.

[49] M. Kasper, W. Schindler, and M. Stöttinger. A stochastic method for security evaluation of cryptographic FPGA implementations. In *Proceedings of the International Conference on Field-Programmable Technology, FPT 2010, 8-10 December 2010, Tsinghua University, Beijing, China*, pages 146–153, 2010.

[50] N. Koblitz and A. Menezes. Another look at security definitions. *Adv. in Math. of Comm.*, 7(1):1–38, 2013.

[51] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[52] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[53] J. Krämer and P. Struck. Leakage-resilient authenticated encryption from leakage-resilient pseudorandom functions. In *Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020*, 2020.

[54] G. Leurent and T. Peyrin. From collisions to chosen-prefix collisions application to full SHA-1. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 527–555. Springer, 2019.

[55] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 605–622. IEEE Computer Society, 2015.

[56] H. Lohrke, S. Tajik, T. Krachenfels, C. Boit, and J. Seifert. Key extraction using thermal laser stimulation: A case study on Xilinx Ultrascale FPGAs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):573–595, 2018.

[57] C. Luo, Y. Fei, and A. A. Ding. Side-channel power analysis of XTS-AES. In *Design, Automation & Test in Europe Conference & Exhibition,DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, pages 1330–1335. IEEE, 2017.

[58] S. Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2002.

[59] S. Mangard. Hardware countermeasures against DPA? A statistical analysis of their effectiveness. In *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

[60] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks*. Springer, 2008.

[61] S. Mangard, T. Popp, and B. M. Gammel. Side-channel leakage of masked CMOS gates. In *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[62] L. Mather, E. Oswald, J. Bandenburg, and M. Wójcik. Does my device leak information? An a priori statistical power analysis of leakage detection tests. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 486–505. Springer, 2013.

[63] D. A. McGrew and J. Viega. *The Galois/Counter Mode of Operation (GCM)*. NIST.

[64] M. Medwed, F. Standaert, J. Großschädl, and F. Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In *Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2010. Proceedings*, volume 6055 of *Lecture Notes in Computer Science*, pages 279–296. Springer, 2010.

[65] M. Medwed, F. Standaert, and A. Joux. Towards super-exponential side-channel security with efficient leakage-resilient PRFs. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium,*

September 9-12, 2012. Proceedings, volume 7428 of *Lecture Notes in Computer Science*, pages 193–212. Springer, 2012.

[66] M. Medwed, F. Standaert, V. Nikov, and M. Feldhofer. Unknown-input attacks in the parallel setting: Improving the security of the CHES 2012 leakage-resilient PRF. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 602–623, 2016.

[67] D. Merli, D. Schuster, F. Stumpf, and G. Sigl. Side-channel analysis of PUFs and fuzzy extractors. In *Trust and Trustworthy Computing - 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings*, volume 6740 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2011.

[68] S. Micali and L. Reyzin. Physically observable cryptography (extended abstract). In M. Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004.

[69] R. Misoczki. Tinycrypt cryptographic library. https://github.com/intel/tinycrypt/, Commit: 5969b0e.

[70] A. Moradi, A. Barenghi, T. Kasper, and C. Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-II FPGAs. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 111–124. ACM, 2011.

[71] A. Moradi and T. Schneider. Improved side-channel analysis attacks on Xilinx bitstream encryption of 5, 6, and 7 series. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*, volume 9689 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2016.

[72] S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[73] R. Novak. SPA-based adaptive chosen-ciphertext attack on RSA implementation. In *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, volume 2274 of *Lecture Notes in Computer Science*, pages 252–262. Springer, 2002.

[74] D. Owen Jr., D. Heeger, C. Chan, W. Che, F. Saqib, M. Areno, and J. Plusquellic. An autonomous, self-authenticating, and self-contained secure boot process for field-programmable gate arrays. *Cryptography*, 2(3):15, 2018.

[75] K. Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[76] O. Pereira, F. Standaert, and S. Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 96–108, 2015.

[77] E. Peterson. *Leveraging asymmetric authentication to enhance security-critical applications using Zynq-7000 all programmable SoCs.* Xilinx, 2015.

[78] R. Poussier, F. Standaert, and V. Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2016.

[79] E. Prouff. DPA attacks and S-boxes. In *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 2005.

[80] P. Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conferenceon the Theory and Application of Cryptology and Information Security,Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[81] E. Ronen, A. Shamir, A. Weingarten, and C. O'Flynn. IoT goes nuclear: Creating a ZigBee chain reaction. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 195–212. IEEE Computer Society, 2017.

[82] H. Saputra, N. Vijaykrishnan, M. T. Kandemir, M. J. Irwin, R. R. Brooks, S. Kim, and W. Zhang. Masking the energy behavior of DES encryption. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10084–10089. IEEE Computer Society, 2003.

[83] W. Schindler, K. Lemke, and C. Paar. A stochastic model for differential side channel cryptanalysis. In *Cryptographic Hardware and Embedded Systems - CHES*

2005, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[84] M. Schink and J. Obermaier. Taking a look into execute-only memory. In A. Gantman and C. Maurice, editors, *13th USENIX Workshop on Offensive Technologies, WOOT 2019, Santa Clara, CA, USA, August 12-13, 2019*. USENIX Association, 2019.

[85] T. Schneider and A. Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.

[86] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 23–40. Springer, 2012.

[87] S. Skorobogatov and C. Woods. In the blink of an eye: There goes your AES key. *IACR Cryptology ePrint Archive*, 2012:296, 2012.

[88] R. Specht, J. Heyszl, and G. Sigl. Investigating measurement methods for high-resolution electromagnetic field side-channel analysis. In *2014 International Symposium on Integrated Circuits (ISIC), Singapore, December 10-12, 2014*, pages 21–24. IEEE, 2014.

[89] F. Standaert. How (not) to use welch's t-test in side-channel security evaluations. In *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers.*, volume 11389 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2018.

[90] F. Standaert and C. Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008.

[91] F. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[92] F. Standaert, O. Pereira, and Y. Yu. Leakage-resilient symmetric cryptography under empirically verifiable assumptions. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 335–352. Springer, 2013.

[93] F. Standaert, O. Pereira, Y. Yu, J. Quisquater, M. Yung, and E. Oswald. Leakage resilient cryptography in practice. In *Towards Hardware-Intrinsic Security - Foundations and Practice*, Information Security and Cryptography, pages 99–134. Springer, 2010.

[94] P. Swierczynski, A. Moradi, D. Oswald, and C. Paar. Physical security evaluation of the bitstream encryption mechanism of Altera Stratix II and Stratix III FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 7(4):34:1–34:23, 2015.

[95] L. Tebelmann, M. Pehl, and G. Sigl. EM side-channel analysis of BCH-based error correction for PUF-based key generation. In *Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security, ASHES@CCS 2017, Dallas, TX, USA, November 3, 2017*, pages 43–52. ACM, 2017.

[96] K. Tiri, M. Akmal, and I. Verbauwhede. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Proceedings of the 28th European Solid-State Circuits Conference*, pages 403–406, 2002.

[97] S. Trimberger and J. Moore. FPGA security: Motivations, features, and applications. *Proc. IEEE*, 102(8):1248–1265, 2014.

[98] T. Unterluggauer and S. Mangard. Exploiting the physical disparity: Side-channel attacks on memory encryption. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*, volume 9689 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2016.

[99] T. Unterluggauer, M. Werner, and S. Mangard. Side-channel plaintext-recovery attacks on leakage-resilient encryption. In D. Atienza and G. D. Natale, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, pages 1318–1323. IEEE, 2017.

[100] F. Unterstein, J. Heyszl, F. De Santis, and R. Specht. Dissecting leakage resilient PRFs with multivariate localized EM attacks - A practical security evaluation on FPGA. In *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2017.

[101] F. Unterstein, J. Heyszl, F. De Santis, R. Specht, and G. Sigl. High-resolution EM attacks against leakage-resilient PRFs explained - and an improved construction. In *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, volume 10808 of *Lecture Notes in Computer Science*, pages 413–434. Springer, 2018.

[102] F. Unterstein, N. Jacob, N. Hanley, C. Gu, and J. Heyszl. SCA secure and updatable crypto engines for FPGA SoC bitstream decryption. In *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2019, London, UK, November 15, 2019*, pages 43–53. ACM, 2019.

[103] F. Unterstein, N. Jacob, N. Hanley, C. Gu, and J. Heyszl. SCA secure and updatable crypto engines for FPGA SoC bitstream decryption: extended version. *Journal of Cryptographic Engineering*, pages 1–16, 2020.

[104] F. Unterstein, M. Schink, T. Schamberger, L. Tebelmann, M. Ilg, and J. Heyszl. Retrofitting leakage resilient authenticated encryption to microcontrollers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):365–388, 2020.

[105] N. Veyrat-Charvillon, B. Gérard, M. Renauld, and F. Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2012.

[106] N. Veyrat-Charvillon, B. Gérard, and F. Standaert. Security evaluations beyond computing power. 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*, volume 7881 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2013.

[107] B. L. Welch. The generalization of 'student's' problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[108] L. Wouters, J. V. den Herrewegen, F. D. Garcia, D. Oswald, B. Gierlichs, and B. Preneel. Dismantling DST80-based immobiliser systems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):99–127, 2020.

[109] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu. One bit flips, one cloud flops: Cross-VM row hammer attacks and privilege escalation. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 19–35. USENIX Association, 2016.

A. EM analysis of LR-PRF on FPGA

A.1. SNR heat maps of all S-boxes

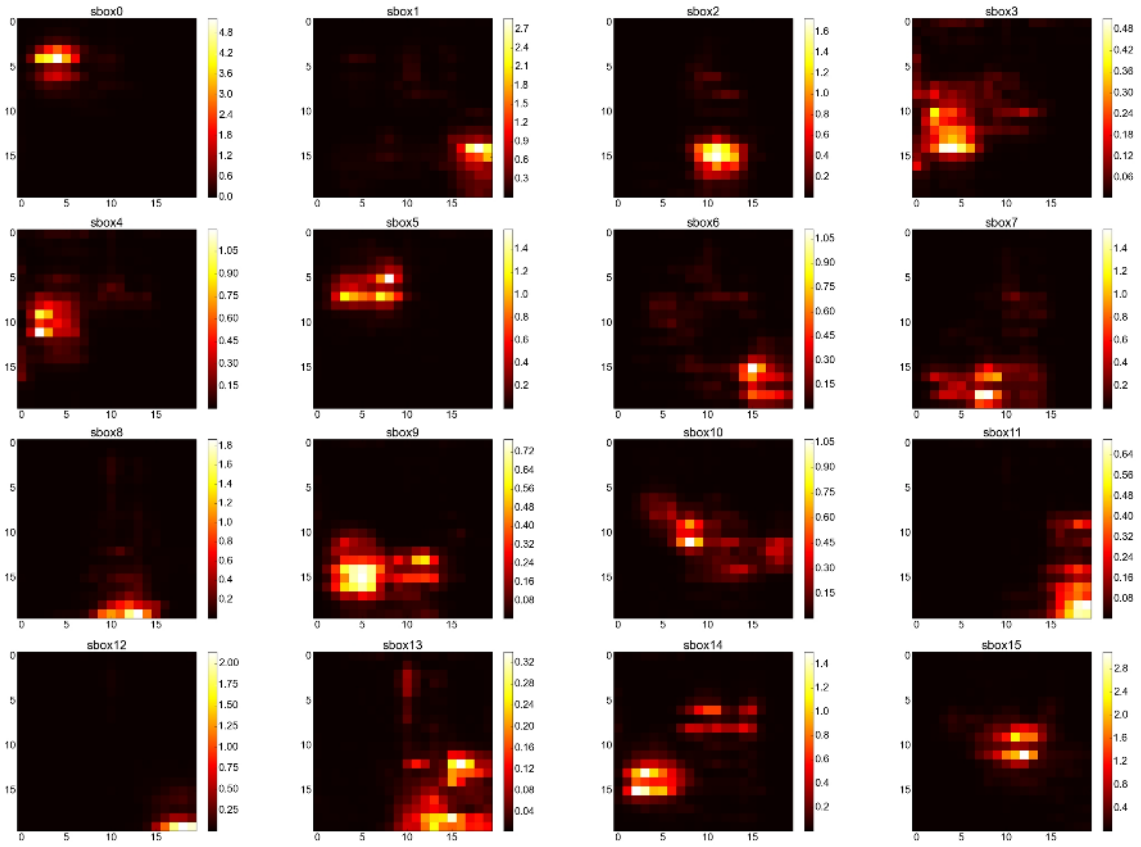


Figure A.1.: SNR heat maps of unconstrained placement.

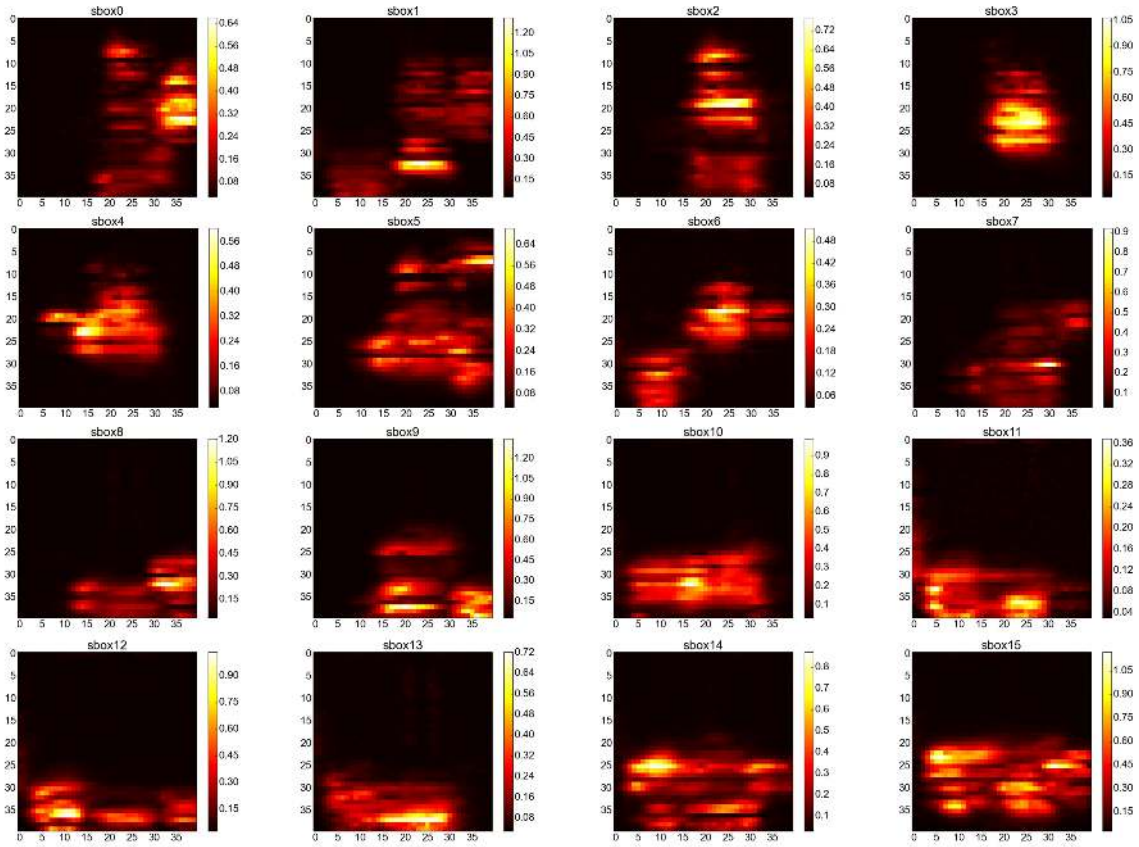


Figure A.2.: SNR heat maps of dense hard-macro placement.

A.2. SNR traces of all S-boxes

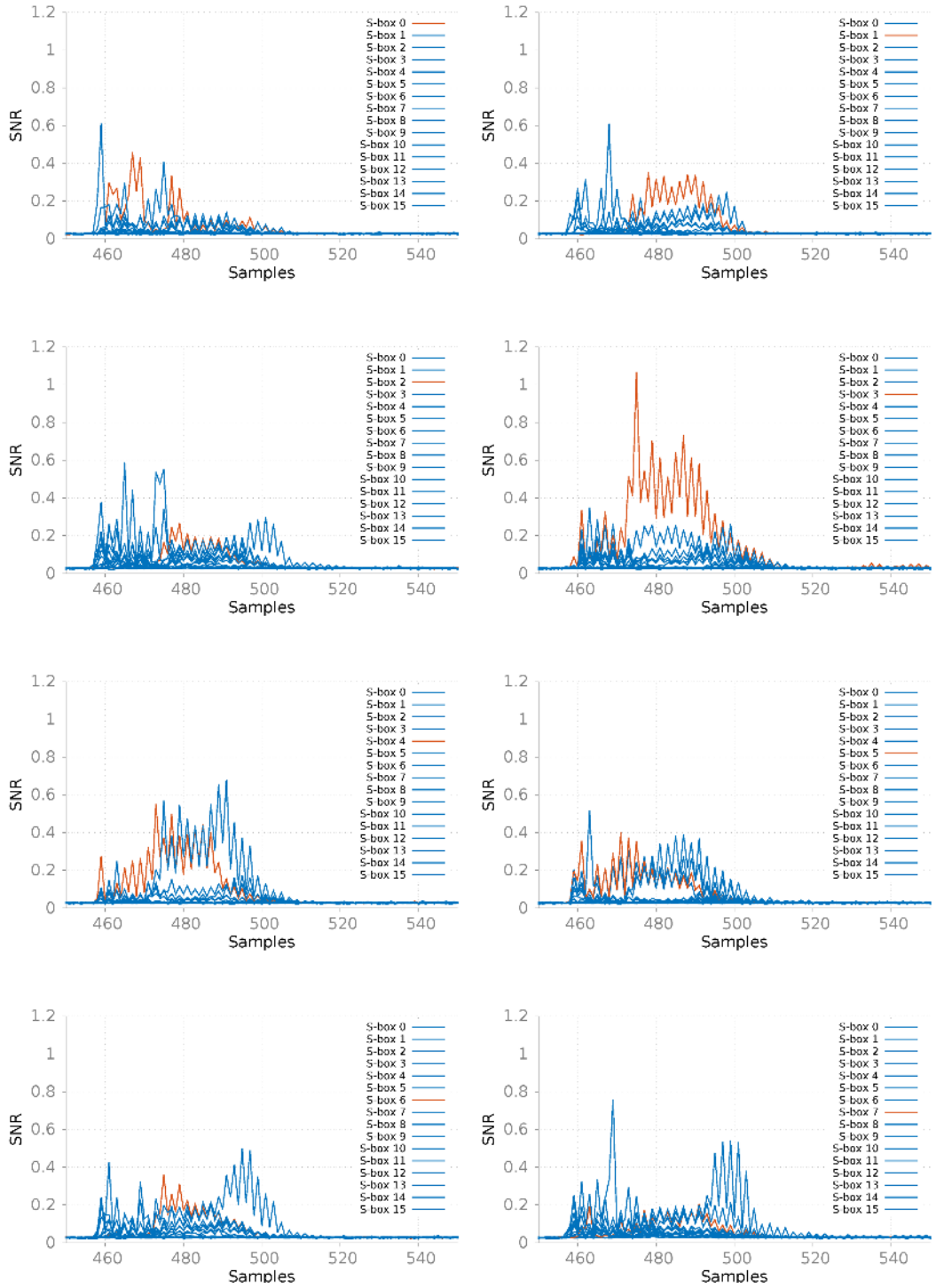


Figure A.3.: SNR for S-boxes 0 to 7.

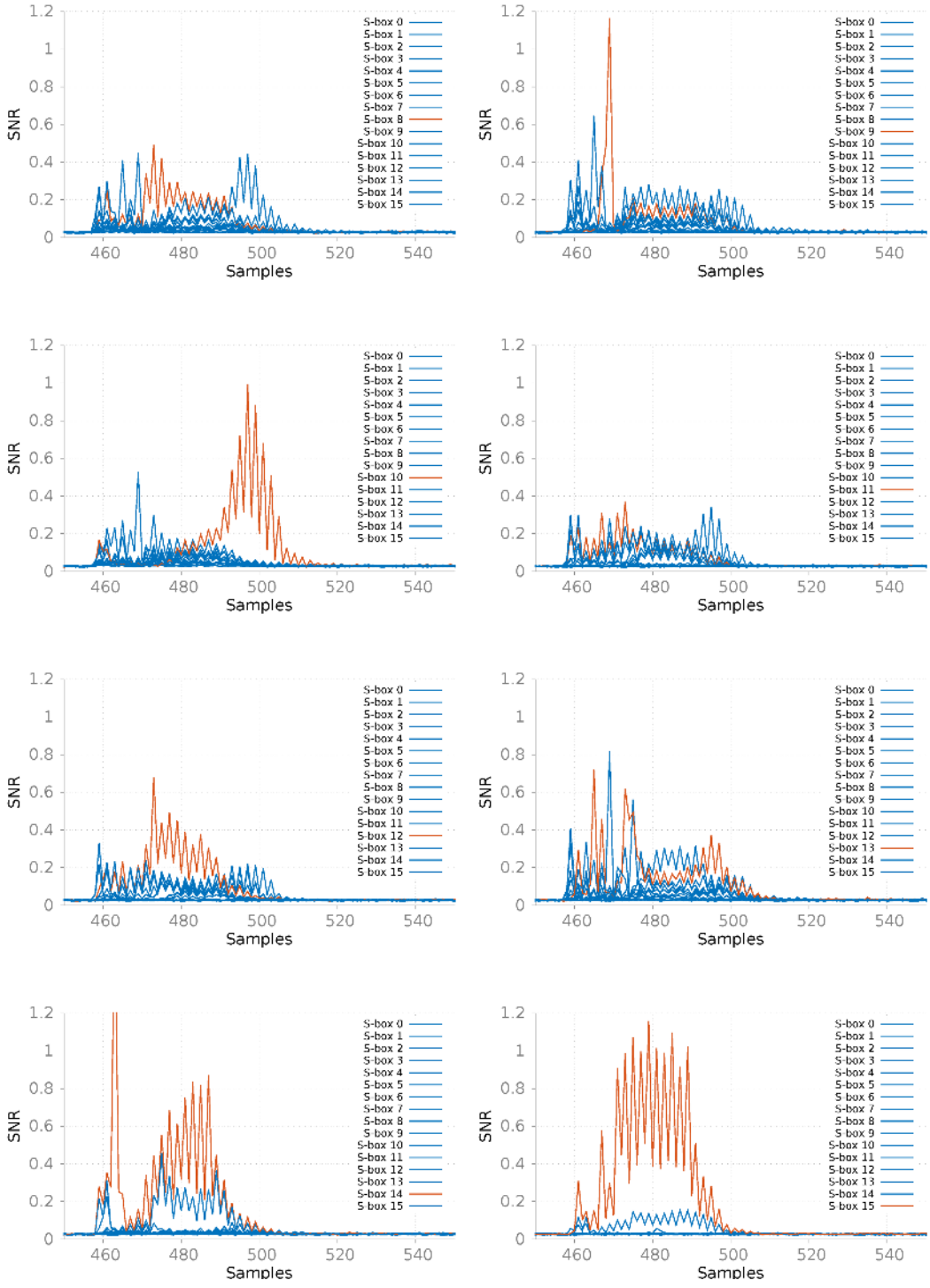


Figure A.4.: SNR for S-boxes 8 to 15.

B. Performance of LR-PRF and LR-AEAD on microcontrollers

Table B.1.: Execution time in clock cycles of the LR-PRF implementation for different optimization levels and varying data complexities.

	STM32			EFM32		
Data complexity	O3	Os	O0	O3	Os	O0
256	2,491	3,908	11,973	3,094	4,636	12,750
16	4,875	7,668	24,149	6,134	9,036	25,469
4	10,411	16,244	51,861	13,142	18,892	53,790
2	24,558	37,620	120,725	30,573	42,847	121,972

Table B.2.: Execution time in clock cycles of the LR-AEAD implementation (optimization level Os) for different data complexities (DCs) and varying ciphertext sizes.

	STM32			EFM32		
Ciphertext size	DC 256	DC 16	DC 2	DC 256	DC 4	DC 2
16 B	15,194	22,746	82,842	10,429	19,214	21,165
0.5 KiB	80,777	88,329	148,425	27,578	55,898	103,514
1 KiB	145,441	152,993	213,089	44,008	72,328	119,944
2 KiB	274,769	282,321	342,417	76,968	105,288	152,904
4 KiB	533,425	540,977	601,073	142,888	171,208	218,824
8 KiB	1,050,737	1,058,289	1,118,385	274,728	303,048	350,664
16 KiB	2,085,361	2,092,913	2,153,009	538,408	566,728	614,344
32 KiB	4,154,609	4,162,161	4,222,257	1,065,768	1,094,088	1,141,704
64 KiB	8,293,105	8,300,657	8,360,753	2,120,488	2,148,808	2,196,424

Table B.3.: Execution time in clock cycles of the function calls used by the LR-AEAD, including input/output.

	STM32			EFM32		
Function call	O3	Os	O0	O3	Os	O0
`AES_encrypt()`	87	87	207	126	132	319
`LR-PRG_seed()`	20	19	49	18	17	44
`LR-PRG_iterate()`	217	227	479	309	327	716