Side-Channel Resistance and Pairing-Based Cryptography for the Internet of Things

by

Thomas Unterluggauer

A PhD Thesis
Presented to the Faculty of Computer Science in Partial Fulfillment of the Requirements for the PhD Degree

Assessors

Prof. Stefan Mangard (TU Graz, Austria)
Prof. Tim Güneysu (Ruhr-University Bochum, Germany)

December 2017

Institute for Applied Information Processing and Communications (IAIK)
Faculty of Computer Science
Graz University of Technology, Austria

Abstract

Internet-of-Things (IoT) devices increasingly process sensitive data in a distributed computing environment and hence require appropriate security mechanisms. One major challenge in developing such mechanisms is that attackers often have direct physical access to IoT devices. This physical access allows them to perform very powerful attacks, such as accessing and tampering with data in external memory. In addition, physical access to running IoT devices allows to infer information about the data these devices process from observing their physical properties, e.g., the power consumption, by using side-channel attacks.

In the first part of this thesis, we focus on the side-channel security of IoT devices and present a method to model bounded side-channel leakage in permutation-based cryptographic schemes. This method allows to assess these schemes' security when their implementation leaks a certain amount of bits about their secret state via side channels. To concretely quantify an implementation's leakage, we also derive a bound on the side-channel leakage from communication theory. We further design two permutation-based key derivation functions to securely perform frequent re-keying, which prevents key recovery via Differential Power Analysis (DPA). However, we also raise awareness of possible side effects from side-channel countermeasures by showing that frequent re-keying can induce a DPA vulnerability that enables recovery of constant plaintext parts.

The second part of this thesis concerns the side-channel security of schemes protecting external memory from physical access. We show that state-of-the-art memory encryption and authentication schemes are vulnerable to DPA and differential fault analysis attacks and we practically apply DPA to extract the secret key used for encrypting the ext4 file system from a modern system on chip. As a countermeasure, we present the first memory encryption and authentication scheme that is secure against DPA. The scheme combines ideas from fresh re-keying and authentication trees to offer both DPA security and performance comparable to contemporary memory authentication techniques.

The last part of this thesis relates to privacy concerns in the IoT. While there are cryptographic schemes that can guarantee user privacy, most of these schemes involve cryptographic primitives that are costly compute, which makes their deployment to low-resource devices difficult. We hence present three lightweight hard- and software implementations of bilinear pairings with side-channel protection to make privacy-preserving protocols ready for the IoT. In addition, we investigate the side-channel security of pairing-based cryptography and use correlation power analysis to recover the secret key of an identity-based encryption scheme from an unprotected implementation of bilinear pairings.

Acknowledgements

The successful accomplishment of a PhD thesis depends on the support of many different people, whom in the following I would like to give thanks. First, I would like to thank Stefan Mangard for supervising my thesis, his guidance, and giving me the opportunity to conduct this research, and I would like to thank Tim Güneysu for valuable feedback, assessing my thesis, and coming to Graz for my PhD defense.

Besides, I would like to acknowledge all the people, who I had the chance to work with over the years. In particular, I want to thank Erich Wenger for reinforcing my decision to start a PhD program and his helpful guidance in the early days as a PhD student, and Mario Werner for many helpful discussions, regularly providing a different view, and plenty of joint works. As well, I am grateful for cryptographic advice by and successful collaborations with Christoph Dobraunig, Maria Eichlseder, and Florian Mendel, and for constructive conversations and worthwhile cooperations with Hannes Groß, Thomas Korak, Robert Schilling, Raphael Spreitzer, Christian Hanser, Daniel Slamanig, Karl-Christian Posch, Manuel Jelinek, and David Schaffenrath. However, I also feel blessed that I came to know many of my colleagues better over the years, and greatly enjoyed our casual discussions and joint morning coffee in addition to professional work.

Finally, I would like to express my deep appreciation to both friends and family for their encouragement, especially my parents Renate and Franz for giving me the chance to follow my interests over the years and always having an open ear for me, and my girlfriend Raphaela, who had a lot of patience with me, never lost her belief in my abilities, and always found the right words to cheer me up.

Kind regards,
Thomas

Table of Contents

Abstract iii

Acknowledgements iv

List of Tables ix

List of Figures x

Acronyms xii

1 Introduction 2
- 1.1 Contribution and Outline 4
  - 1.1.1 Bounded Side-Channel Leakage 5
  - 1.1.2 Side-Channel Security for Memory Encryption 5
  - 1.1.3 Bilinear Pairings for Embedded Devices 6

2 Side-Channel Attacks 7
- 2.1 Attacks and Definitions 7
  - 2.1.1 Simple Power Analysis 8
  - 2.1.2 Differential Power Analysis 9
  - 2.1.3 Profiled Attacks 9
- 2.2 DPA Countermeasures 10
  - 2.2.1 Masking 10
  - 2.2.2 Frequent Re-Keying 11
- 2.3 Leakage-Resilient Cryptography 11
  - 2.3.1 Leakage-Resilient Encryption 11
  - 2.3.2 Leakage-Resilient MAC 12
- 2.4 Secure Re-keying Functions 13
- 2.5 Conclusion 14

I Bounded Side-Channel Leakage 15

3 Re-Keying and Leakage Model from Cryptographic Sponges 17
- 3.1 Leakage in Permutation-based Designs 18
  - 3.1.1 Sponge Leakage Model 18
- 3.2 Secure Re-Keying Functions 20

3.2.1 Re-Keying with Inherent DPA Security 20
3.2.2 More Efficient Re-keying 21
3.3 Implementation 22
3.3.1 Instantiation Keccak[400] 22
3.3.2 Security with State Leakage 23
3.3.3 Results and Comparison 24
3.4 Conclusion 24

4 Leakage Bounds for Gaussian Side Channels 26
4.1 Background 27
4.2 Modeling Side-Channel Leakage as a Communication Channel . . 28
4.2.1 Attack Model 28
4.2.2 Mutual Information 28
4.2.3 Linear Channel Model 29
4.2.4 Leakage Bound for Gaussian Side Channels 30
4.2.5 Description of Common Leakage Models 32
4.3 Complexity of State Recovery 33
4.3.1 Attack Model 34
4.3.2 Averaging Attacker 34
4.3.3 Expected Minimum Attack Complexity 35
4.3.4 Divide-and-Conquer Attacks 36
4.4 Experimental Verification and Security Analysis 36
4.4.1 Evaluation Hardware: Fulmine 37
4.4.2 Soundness of Model and Bounds 38
4.4.3 Estimating Security Bounds for ISAP 40
4.5 Conclusion 42

5 Side-Channel Attacks on Leakage-Resilient Encryption 44
5.1 Side-Channel Plaintext-Recovery Attack 45
5.1.1 Stream Cipher Mode 45
5.1.2 Block Cipher Mode 46
5.2 Implications and Applicability 47
5.2.1 Communication 47
5.2.2 Memory Encryption 48
5.2.3 Remarks and Countermeasures 48
5.3 Practical Evaluation 49
5.3.1 Stream Cipher Mode 49
5.3.2 Block Cipher Mode 49
5.4 Conclusion 51

II Side-Channel Security for Memory Encryption 53

6 Side-Channel Attacks on Memory Encryption 55
6.1 Memory Encryption 56
6.1.1 Definition 56

6.1.2 Memory Encryption in Practice 57
6.1.3 State-of-the-Art Implementations 58
6.2 Physical Attacks on Memory Encryption 59
6.2.1 Differential Power Analysis 60
6.2.2 Differential Fault Analysis 62
6.3 EM Attack on Ext4 Encryption 64
6.3.1 Analysis of Ext4 Disk Encryption 65
6.3.2 General Attack Flow 65
6.3.3 Experimental Setup and Results 66
6.4 Conclusion 67

7 Side-Channel Secure Memory Encryption and Authentication 69
7.1 Memory Encryption and Authentication 71
7.1.1 Threat Model and Requirements 71
7.1.2 Memory Encryption 72
7.1.3 Memory Authentication 72
7.2 Re-Keying for Memory Encryption 74
7.2.1 The Re-Keying Operation 74
7.2.2 Re-Keying and Plaintext Confidentiality 75
7.3 DPA-Secure Memory Encryption and Authentication 75
7.3.1 Construction 76
7.3.2 Authenticity 79
7.3.3 Side-Channel Discussion 80
7.4 Higher-Order DPA Security 82
7.4.1 Concept 82
7.4.2 Masking Details 82
7.4.3 Side-Channel Discussion 83
7.4.4 Implementation Aspects 84
7.5 Instantiation 84
7.5.1 Meas-v1 85
7.5.2 Meas-v2 87
7.6 Implementation 88
7.6.1 Platform 88
7.6.2 Memory Layout 89
7.6.3 Address Translation 89
7.6.4 MEAS Pipeline 89
7.7 Evaluation 91
7.7.1 Security Properties 92
7.7.2 Parallelizability 92
7.7.3 Memory Overhead 93
7.7.4 Memory Overhead with Masking 94
7.7.5 Randomness 95
7.7.6 Implementation Results 96
7.8 Conclusion 99

III Bilinear Pairings for Embedded Devices 101

8 Efficient Pairings and ECC for Embedded Devices 103

8.1 Background on Pairings . . . 104
8.2 High-Level Arithmetic . . . 105
8.2.1 Implementation Details . . . 106
8.2.2 Optimized Final Exponentiation . . . 107
8.2.3 Optimized Prime-Field Inversion . . . 108
8.3 Hardware Architectures . . . 108
8.3.1 The Used Microprocessor . . . 109
8.3.2 The Software Framework . . . 110
8.3.3 Assembly-Optimized Software Implementation *(a)* . . . 110
8.3.4 Multiply-Accumulate Hardware Extensions *(b)* . . . 111
8.3.5 The Drop-in Module *(d)* . . . 111
8.4 Implementation Results . . . 113
8.5 Comparison with Related Work . . . 116
8.6 Re-usability of our Drop-in Architecture . . . 117
8.6.1 Using the Drop-in Module for Pairing-based Protocols . . . 118
8.6.2 Using the Drop-in Module for Elliptic Curve Cryptography (ECC) . . . 118
8.7 Conclusion . . . 119

9 Side-Channel Attacks on Bilinear Pairings 120

9.1 Related Work . . . 121
9.2 Identity-based Encryption . . . 122
9.2.1 Vulnerability . . . 123
9.3 General Attack . . . 125
9.4 Practical Setup and Results . . . 127
9.5 Countermeasures . . . 131
9.6 Conclusion . . . 132

10 Conclusions 133

Bibliography 137

About the Author 156

List of Tables

3.1 Security level (in bits) for the two re-keying functions. 22
3.2 Parameters for IsapRk1 and IsapRk2 with 128-bit security using the Keccak[400] permutation ($|K| = |\overline{K}_0| = 128$, $s_p = 400$). . . . 23
3.3 Implementation results for secure re-keying functions (130 nm). . 24

7.1 Comparison of Meas with other constructions for scalable authentic and/or confidential memory which offer block wise random access. 92

8.1 Propagation of data within the pipelined drop-in module 112
8.2 Performance of various operations on architectures *(a)*, *(b)*, and *(d)*. 113
8.3 Implementation characteristics for 130 nm and 90 nm process technologies. 115
8.4 Related software implementations of Ate pairings over BN curves. 115
8.5 Related hardware platforms (130 nm). 116
8.6 Performance of pairing-based protocols on the drop-in platform. . 118

List of Figures

2.1 General principle of side-channel attacks. 8
2.2 Leakage-resilient stream cipher. 12
2.3 Leakage-resilient block cipher encryption. 13
2.4 Leakage-resilient Message Authentication Code (MAC). 14

3.1 Leakage of information in sponge-based constructions. 19
3.2 Re-keying inherently secure against DPA attacks: IsapRk1. . . 20
3.3 Sponge construction for re-keying: IsapRk2. 21

4.1 Expected side-channel capacity given the number of averaged leakage traces for different numbers of Points of Interest (POIs) and $SNR_m = 0.01$. 36
4.2 Expected minimum attack complexity over the number of independent leakages for different state sizes and $SNR_1 = 0.1$. . . . 37
4.3 FULMINE SoC and HWCRYPT architecture. 38
4.4 Side-channel capacity, mutual information and success rate for the Keccak[400] permutation given the number of averaged traces and different numbers of POIs and number of classes. The remaining state was held constant. 39
4.5 Mutual information of Keccak[400] on FULMINE and side-channel capacity of different channel models (256 classes, 10 POIs). . . . 41
4.6 Minimum attack complexity as the number of measurements needed to average to recover (parts) of the ISAP state from FULMINE. As a security margin we set $\gamma = 100$. 42

5.1 Unknown Plaintext Template Attack (UPTA)-like plaintext recovery attack on one plaintext byte in the first AES round. Two template sets, one on the whitening key $\overline{K}_0$, and one on the S-box output $\mathcal{S}(P_0 \oplus \overline{K}_0)$, have to be trained. 47
5.2 Single plaintext byte result of a 1st-order DPA on the leakage-resilient stream cipher in Figure 2.2. The correct plaintext byte is highlighted in bold. 50
5.3 POI detection for the S-box and the key template. The main key leakage is located at sample 470. The S-box output leaks the most at sample 1782. 51

5.4 Plaintext probabilities of an UPTA-like attack on one plaintext byte. The correct plaintext value is highlighted in bold black. . 52

6.1 Generic model of memory encryption. 57
6.2 Tweakable ciphers for disk encryption. 58
6.3 Disk encryption via CBC and ESSIV. 59
6.4 Counter mode memory encryption. 60
6.5 AES round function (left) and its alternative representation (right). 63
6.6 Distribution of t-test results on the chip surface. 66
6.7 Single-byte correlation results for ext4 key derivation. 67

7.1 MEAS' tree construction for $n_b = 8$ data blocks and with an arity of $a = 2$. 77
7.2 Schematic overview of ENC in MEAS-v1. 84
7.3 Schematic overview of AE in MEAS-v1. 85
7.4 Zynq platform with MEAS pipeline. 88
7.5 Memory layout for 4-ary MEAS. 89
7.6 Data node requests for 4-ary MEAS. 90
7.7 MEAS encryption and authentication pipeline. 91
7.8 Memory overhead comparison for 4-ary trees depending on protection order and block size with a security level of 128 bits ($a = 4$, $s_{nonce} = s_{tag} = s_{key} = 128$, $s_{hash} = 256$). 94
7.9 Memory overhead of MEAS depending on arity and protection order (1024-bit blocks, 128-bit security). 96
7.10 Read performance determined with tinymembench (`NEON read prefetched (64 bytes step)`). 97
7.11 Write performance determined with tinymembench (`NEON fill`). 98
7.12 Memory latency determined with LMBENCH (`lat_mem_rd 8M`). 99
7.13 Field Programmable Gate Array (FPGA) utilization on XC7Z020 for 8-ary trees. 100

8.1 Arithmetic required for pairings over Barreto-Naehrig curves. . 106
8.2 Architectural options for fast and flexible pairing designs. 109
8.3 High-level representation of architecture *(d)* (without program memory). Note that the sizes of the blocks are not proportional to their respective hardware footprints. 111
8.4 5×5-word zig-zag product scanning multi-precision multiplication method. 112
8.5 Group operations at 48 MHz. 114
8.6 Characteristics of related hardware. 117

9.1 Correlation of multiplication result for 16-bit hypotheses. 127
9.2 Correlation of word multiplication results. 128
9.3 Correlation of accumulation register. 131

Glossary

AREA	Added Rendundancy Explicit Authenticity.
ASIC	Application Specific Integrated Circuit.
ASIP	Application-Specific Instruction-set Processor.
AXI	Advanced Extensible Interfaces.
BN	Barreto-Naehrig.
BRAM	Block RAM.
CBC	Cipher Block Chaining.
CICO	Constrained-Input Constrained-Output.
COTS	Common Off-The-Shelf.
CPA	Correlation Power Analysis.
CPU	Central Processing Unit.
DEK	Data Encryption Key.
DFA	Differential Fault Analysis.
DKEK	Derived Key Encryption Key.
DLP	Discrete Logarithm Problem.
DMA	Direct Memory Access.
DPA	Differential Power Analysis.
ECB	Electronic Code Book.
ECC	Elliptic Curve Cryptography.
ECDSA	Elliptic Curve Digital Signature Algorithm.
EM	Electromagnetic Emanation.
ESSIV	Encrypted Salt-Sector IV.
FIPS	Finely Integrated Product Scanning.
FPGA	Field Programmable Gate Array.
GGM	Goldreich-Goldwasser-Micali.
HD	Hamming Distance.
HDD	Hard Disk Drive.
HW	Hamming Weight.

IEEE	Institute of Electrical and Electronics Engineers.
IoT	Internet-of-Things.
IV	Initial Vector.
KDF	Key Derivation Function.
KEK	Key Encryption Key.
KEM	Key Encapsulation Mechanism.
LR-PRF	Leakage-Resilient Pseudo Random Function.
LSB	Least Significant Bit.
LUKS	Linux Unified Key Setup.
LUT	Look-Up Table.
MAC	Message Authentication Code.
MCU	Microcontroller Unit.
MI	Mutual Information.
MIMO	Multi-Input Multi-Output.
NFC	Near-Field Communication.
NIST	National Institute of Standards and Technology.
NVM	Non-Volatile Memory.
PAT	Parallelizable Authentication Trees.
PCB	Printed Circuit Board.
PL	Programmable Logic.
POI	Point of Interest.
PRNG	Pseudo-Random Number Generator.
PS	Processing System.
RAM	Random Access Memory.
RFID	Radio-Frequency Identification.
RISC	Reduced Instruction Set Computer.
ROM	Read-Only Memory.
RPC	Randomized Projective Coordinates.
SIMO	Single-Input Multi-Output.
SNR	Signal-to-Noise Ratio.
SoC	System on Chip.
SPA	Simple Power Analysis.
SPS	Separate Product Scanning.
TCDM	Tightly-Coupled Data Memory.
TEC	Tamper Evident Counter.
TEE	Trusted Execution Environment.
TMTO	Time-Memory Trade-Off.

TPM	Trusted Platform Module.
UPTA	Unknown Plaintext Template Attack.
XEX	Xor-Encrypt-Xor.
XTS	XEX-based Tweaked codebook mode with ciphertext Stealing.

1

Introduction

The ever-growing networking of electronic devices has made the Internet-of-Things (IoT) reach the masses. The IoT connects a heterogeneous field of components, ranging from industrial parts, such as sensor nodes, actuators, production machines, robots, and their automation logic, to consumer products, such as contactless smart cards, control systems, smart phones, and computers in, e.g., a smart home environment. This trend towards ubiquitous networking of devices gives rise to many new applications, but also demands for security services to protect sensitive data, privacy, intellectual property, communication, and, due to the nature of the IoT, real-world things.

From a security point of view, the pervasive nature of embedded IoT devices is particularly challenging as it causes many of these devices to operate in hostile environments. This means that the running device is in possession of an entity that may use certain services of the device, but that also has a strong interest in learning confidential information (or tampering with information) that is processed inside the device and that the entity should not have access to. For example, a corporate customer buying a production machine is interested in intellectual property inside the machine's control device, i.e., control parameters and source code, which its vendor wishes to be protected from unauthorized access and proliferation. Similarly, malicious modification of data, such as billing information in a pay-per-use business model, is conceivable.

In scenarios like these, malicious device owners, or attackers, can perform a wide range of attacks in order to learn or modify information as desired. For example, they may access and tamper with external memory such as Random Access Memory (RAM) and flash devices, probe and force buses on a Printed Circuit Board (PCB), exchange peripherals, inject code, and look for exploits on any external interface. One prominent example in this setting are cold boot attacks [Hal+09], where RAM is operated at low temperatures to learn the information stored therein after the device has been shut off. Performing such an attack in the right moment can leak sensitive data, key material or tokens. This particular risk of physical attackers is omnipresent in IoT applications and demands for appropriate mechanisms to secure these platforms.

In this respect, a common approach to achieve secure IoT platforms is cryptography. In particular, cryptography prevents many of the aforementioned physical attacks on IoT devices, e.g., by providing memory encryption [Fru05; Rog04; IEE08b] and memory authentication [Mer80; HJ05; Elb+07]. However, physical access inherently bears the risk of side-channel attacks as well. In such side-channel attacks, side-channel information such as the power consumption or the Electromagnetic Emanation (EM) of an IoT device is recorded and then used to learn about the data processed inside the device [KJJ99; Mor+11a; Eis+08], e.g., the secret key used for encryption. These side-channel attacks are realistic in many IoT applications, because an attacker who is able to probe buses on a PCB or to conduct cold boot attacks is typically also capable of performing power or EM measurements using an oscilloscope. In this regard, a particularly strong variant of side-channel attacks is Differential Power Analysis (DPA). DPA efficiently accumulates side-channel information about secret data from multiple executions using the same secret, but different inputs. For instance, DPA is often capable of recovering the secret key from unprotected software implementations of cryptographic algorithms using less than 100 en-/decryptions with the same key [MOP07]. In addition, DPA has recently been shown to be a serious threat to more complex, state-of-the-art desktop systems as well [SRH16]. This powerful class of attacks must hence be considered during both the implementation of cryptography and the design of secure IoT platforms.

From another perspective, the increasing number of devices in IoT networks is a massive concern for privacy too. Namely, the distributed processing of user-related, sensitive data strongly increases the risk of leaking critical information and tracking users. This kind of threat can however be prevented by using techniques from cryptography as well. In particular, to protect the privacy of users in applications running on IoT networks, there is the ongoing process of designing novel cryptographic protocols in the field of modern cryptography. Prominent examples of such privacy-preserving protocols are group signature schemes [CH91; BBS04] and attribute-based credentials [CL02]. However, modern cryptography also aims to overcome shortcomings of current cryptographic schemes in IoT scenarios. For example, identity-based encryption [Sha84; BB04] and one-round multi-party key agreement [Jou04] aid IoT applications by reducing communication. Consequently, there is a clear need to make such protocols and schemes available to the IoT.

Many of these new protocols are most efficiently constructed from asymmetric cryptographic primitives such as elliptic curves and bilinear pairings. However, the evaluation of such primitives is yet complex and often involves large computational effort. While this is not a problem on modern smart phones or desktop computers, there are also many embedded devices in the IoT lacking computational power and resources, often solely consisting of a single microchip and an antenna. Such devices have many, often contradictory constraints, making it difficult to provide embedded IoT devices with modern cryptography. For example, performance must be practical to suit applications with user interaction, but on the contrary, embedded implementations must also limit their demand for memory and additional hardware components as these directly affect chip area, implementation cost, and power consumption. The latter parameters are particularly relevant for mass production and contactless IoT devices. Apart from that, side-channel resistance must be considered for implementations of modern cryptography as well. As a result, optimized implementations tailored to the need of embedded IoT devices are required to facilitate the widespread deployment of modern cryptography.

Summarizing, two current challenges in the IoT are efficient implementations of modern cryptographic primitives as well as IoT platforms that can guarantee security in hostile environments where physical attackers have direct access to the IoT device. While progress with respect to the first challenge can serve as the foundation for new privacy-preserving technologies in a wide field of applications, advances in terms of secure IoT platforms are indispensable to reliably provide applications dealing with sensitive information in an ubiquitous computing environment. In terms of both challenges, side-channel attacks are a highly relevant threat that needs to be considered and practically evaluated with respect to any new protection mechanism. The challenge hereby is to prevent such attacks by adding appropriate side-channel countermeasures to implementations or developing cryptography and platform security measures that prohibit side-channel attacks by design.

1.1 Contribution and Outline

In the course of this thesis, we make progress towards both, the construction of secure IoT platforms and the secure and efficient implementation of modern cryptographic primitives for improving IoT privacy, and hereby put a strong emphasis on side-channel attacks and corresponding countermeasures. In more detail, our contributions fall into three main categories: (1) to prevent side-channel attacks on IoT devices, we advance in the topic of bounding side-channel leakage, (2) in the context of memory encryption and authentication, we investigate side-channel attacks and design suitable countermeasures, and (3) to make privacy-preserving protocols available to the IoT, we provide efficient bilinear pairings and analyze their side-channel security. According to these main categories, the detailed contributions and outline are as follows.

Chapter 2 provides background information on side-channel attacks and state-of-the-art countermeasures, such as masking, frequent re-keying, and leakage-resilient encryption.

1.1.1 Bounded Side-Channel Leakage

Chapter 3 introduces a novel method to model side-channel leakage in permutation-based cryptographic designs such as sponges. This model allows to scale implementations for different leakage bounds faced in practice. We further use this model to design two permutation-based key derivation functions that allow secure re-keying of cryptographic primitives to prevent DPA. This work has been published in [Dob+17].

Chapter 4 presents an approach to determine a bound for the side-channel leakage from cryptographic implementations under a single data input. More concretely, we use results on the channel capacity of Multi-Input Multi-Output (MIMO) channels from information theory to give bounds for Gaussian side channels and further investigate the effect of signal averaging. While this gives a tool to estimate the complexity of side-channel attacks, the bounds determined using this new approach are also well suited to scale implementations according to the leakage modeling techniques from Chapter 3. This work has been published in [Unt+17].

Chapter 5 analyzes the security of re-keying based side-channel countermeasures and leakage-resilient encryption. While these countermeasures prevent the leakage of sensitive keys, our analysis reveals that they induce another DPA vulnerability which allows for plaintext recovery. In particular, plaintext recovery is possible whenever re-keying causes constant data to be encrypted multiple times using different keys. This examplary attack shall raise awareness of possible side effects of side-channel countermeasures depending on the actual use case. The work has been published in [UWM17c].

1.1.2 Side-Channel Security for Memory Encryption

Chapter 6 investigates current memory encryption schemes in terms of side-channel and physical fault attacks. The results show that both DPA and Differential Fault Analysis (DFA) break all memory and disk encryption schemes used in practice and that these attacks are practical on state-of-the-art devices by attacking ext4 disk encryption on a Zynq-7010 System on Chip (SoC). The attacks have been published in [UM16] and received the best paper award at COSADE 2016.

Chapter 7 combines ideas from frequent re-keying and authentication trees to introduce the first memory encryption and authentication scheme that is secure against DPA attacks. The scheme is applicable to both Non-Volatile Memory (NVM) and RAM and can further be used with cryptographic accelerators provided in Common Off-The-Shelf (COTS) systems. Our scheme's evaluation on a Zynq-7020 SoC shows that its memory and performance overhead is comparable to state-of-the-art memory authentication techniques without DPA protection.

This work was in part published in [UWM17b] and the extended results in this chapter are currently in submission to [UWM17a].

1.1.3 Bilinear Pairings for Embedded Devices

Chapter 8 presents three side-channel protected hard- and software implementations of bilinear pairings that can be used to realize privacy-preserving protocols. Most prominently, the placement of a drop-in hardware accelerator between the ARM Cortex-M0+ CPU and the data memory yields a lightweight design with practical runtime that is suitable for resource-constrained applications. This hardware design has been published in [UW14a].

Chapter 9 studies the side-channel security of bilinear pairing algorithms. In particular, we present a Correlation Power Analysis (CPA) attack that extracts the secret key from an implementation of identity-based encryption and raise awareness of randomization techniques as suitable countermeasures in this context. The results of this chapter have been published in [UW14b].

Chapter 10 finally concludes this thesis.

2

Side-Channel Attacks

Many embedded devices nowadays contain and process sensitive data and hence demand for appropriate security architectures. However, attackers often get direct physical access to these embedded devices when they are running in the field. This physical access allows attackers to perform a wide range of attacks, such as to observe and to analyze a device's physical behavior as well as to tamper with a device.

In this thesis, we strongly focus on attackers, who have physical access to a running Internet-of-Things (IoT) device and who analyze the device's physical properties to learn about the data it processes in passive side-channel attacks. To provide the required background information, this chapter gives a non-exhaustive introduction to passive side-channel attacks and their countermeasures. In particular, it introduces frequent re-keying and leakage-resilient encryption which are referred to throughout this thesis. Parts of this chapter have been taken from the publications [UWM17b; Dob+17; UWM17c].

This chapter is organized as follows. Section 2.1 gives a general introduction to side-channel attacks and its variants, and Section 2.2 presents possible countermeasures. Section 2.3 introduces leakage-resilient encryption and Section 2.4 gives an overview on secure re-keying functions.

2.1 Attacks and Definitions

An implementation in general performs certain operations on its inputs to create its output. For example, a cryptographic device transforms its input plaintext using a secret key K to give the output ciphertext. The respective in- and output interfaces embody the intended behavior of the implementation and form its main

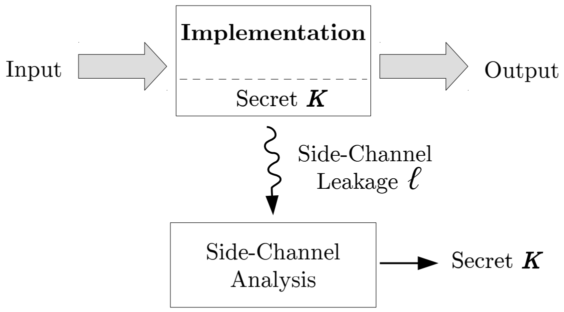


Figure 2.1: General principle of side-channel attacks.

channel of communication. However, in practice and as shown in Figure 2.1, the execution of an implementation also leaks information on the internally processed data via its physical properties, such as power, timing, and Electromagnetic Emanation (EM). For example, a cryptographic implementation leaks information about the secret key K when performing en-/decryption. As this information leakage does not occur on its designated interfaces, we denote it as side-channel leakage. Attackers can exploit this side-channel leakage in so-called passive side-channel attacks in order to learn secret information such as the key K of a cryptographic implementation. However, while the topic of side-channel attacks emerged in the field of cryptography, note that the side-channel leakage of any sensitive data processed by an implementation is a concern.

Independent of the concrete source of side-channel leakage, there are two basic types of side-channel attacks [KJJ99]: Simple Power Analysis (SPA) and Differential Power Analysis (DPA). Originally, Simple Power Analysis (SPA) and Differential Power Analysis (DPA) have been introduced for the power side-channel, but their basic principle is applicable to all kinds of side channels such as power, EM, and timing. Therefore, we will use the terms SPA and DPA throughout this thesis, but note that our elaborations apply to all kinds of side channels unless stated otherwise.

2.1.1 Simple Power Analysis

In SPA attacks, the adversary tries to learn the secret value processed inside a device from observing side channels during a single processing of the secret value to be revealed, e.g., the adversary tries to learn the encryption key from a power trace observed during a single encryption. However, the adversary is allowed to observe the same encryption multiple times to reduce measurement noise. Clearly, an implementation that cannot keep a key secret for a single encryption is worthless. Therefore, security against SPA attacks is a typical precondition for an implementation.

2.1.2 Differential Power Analysis

Quite naturally, the amount of information learned about a secret value from a side channel increases with the number of different inputs processed under the respective secret. This is exploited in DPA attacks, which use the observation of several different processings of a secret value in a device to learn its value, e.g., the adversary tries to learn the secret key from power traces observed during the en-/decryption of multiple (public) input values. However, note that depending on the statistical techniques being used, there are several variants of DPA, such as Correlation Power Analysis (CPA) [BCO04].

Procedure. One typical procedure to perform a DPA attack is to measure the power of n_t different en-/decryptions for known plain- or ciphertexts, to compute certain intermediate values within the en-/decryption based on the n_t different plain-/ciphertexts and the possible keys, and to map the intermediate values to power consumptions according to a hypothetical leakage model. Common leakage models assume the power consumption of a device to relate to the Hamming Weight (HW) of, or the Hamming Distance (HD) between contents of a device-internal register. Applying a statistical distinguisher, e.g., correlation, to the power traces of the n_t en-/decryptions and the respective hypothetical power consumptions for the different, possible keys reduces the key space or determines the key uniquely.

Attack Order. While the described attack procedure forms a first-order DPA attack, there is also DPA of higher order. Higher-order DPA attacks [KJJ99; Mes00; Gie+10] exploit joint statistical properties of multiple aspects of the side-channel signal. For a DPA attack of order d, this typically means to jointly analyze the leakage of d different internal values of the executed algorithms, which is often measured in d different Points of Interest (POIs) in the side-channel leakage trace. The attack complexity of DPA grows exponentially with its order [Cha+99].

2.1.3 Profiled Attacks

Independently of whether SPA or DPA is performed, side-channel attacks can make use of profiling. Profiling of a side-channel means to construct templates [CRR02] that classify the side-channel information of a target device with respect to a certain value processed inside the device. For example, the power consumption in m POIs of a power trace that is observed when the device processes a certain value is typically characterized by using a multivariate Gaussian distribution. In the actual attack, the templates are matched with the side-channel trace to gain some information on the value processed inside the device. The information learned from template matching can then be exploited in either SPA or DPA manner, as for example shown in [OM07]. Note however that conducting profiled attacks requires much more effort than performing non-profiled attacks.

Further note that in many applications it is impossible to perform the required profiling step at all.

Unknown Plaintext Template Attacks. One special form of profiled attacks which we refer to in this thesis are Unknown Plaintext Template Attacks (UPTAs) [HTM09]. An UPTA recovers the secret key of a block cipher without having access to both the plain- and the ciphertext. Hereby, two sets of templates are used to recover the required information instead. Combining the leakage from multiple block cipher invocations in a DPA-like manner then permits to recover the secret key.

In more detail, the original UPTA attacks the constant key K of a block cipher E by observing the encryption of several unknown plaintexts with the help of power templates. Hereby, the power templates are used to learn information on the unknown plaintexts $P_0, P_1, ...$ and intermediate values $V_0, V_1, ...$ in the respective encryption processes $E(K; P_0), E(K; P_1), ...$. Exploiting the relation between the information learned on $P_0, P_1, ...$ and $V_0, V_1, ...$, the key K is recovered. As the attack combines side-channel information from both the unknown plaintexts $P_0, P_1, ...$ and the intermediate values $V_0, V_1, ...$, the order of this attack is two.

2.2 DPA Countermeasures

The effectiveness of DPA attacks has caused a lot of effort to be put into the development of countermeasures. Two basic approaches have evolved to counteract DPA, namely, (1) to protect the cryptographic implementation by using mechanisms like masking, and (2) the frequent re-keying of unprotected cryptographic primitives.

2.2.1 Masking

Masking [Cha+99; GP99; PR13], also called secret sharing, is a technique that can hinder DPA attacks up to certain orders. The idea behind masking is to prevent DPA by making the side-channel leakage independent from the processed data. In a masked cryptographic implementation, every secret value V is split into $d + 1$ shares $V_0, ..., V_d$ in order to protect against d-th order DPA attacks. Hereby, d shares are chosen uniformly at random and the $(d + 1)$-th share is chosen such that the combination of all $d + 1$ shares gives the actual secret value V. As a result, an adversary is required to combine the side-channel leakage of all $d + 1$ shares to be able to exploit the side channel, i.e., to perform a $(d + 1)$-th order DPA.

While the masking operation itself is usually cheap, e.g., XOR, cryptographic primitives typically contain several operations that become more complex in the masked representation. This eventually results in significant implementation overheads. For example, the 1st-order DPA secure threshold implementations of AES in [Bil+14; Mor+11b] add an area-time overhead of a factor of four.

2.2.2 Frequent Re-Keying

The probability for key recovery via DPA to be successful rises with the number of side-channel observations for different inputs. Therefore, one approach to counteract DPA is frequent re-keying [Koc03; Med+10]. Frequent re-keying tries to limit the number of different processed inputs per key, i.e., the data complexity.

In more detail, the countermeasure constrains a cryptographic scheme to use a certain key K only for q different public inputs (q-limiting [Sta+10]). When the limit of q different inputs is reached, another key K' is chosen. Thus, for a certain key K, an adversary can only obtain side-channel leakage for q different inputs, which limits the feasibility of DPA to recover K.

As a result, designing cryptographic schemes and protocols with small data complexity q is one measure to prohibit DPA against unprotected cryptographic implementations. In more detail, it is widely accepted that very small data complexities, i.e., $q = 1$ and $q = 2$, have sufficiently small side-channel leakage and do not allow for successful key recovery from DPA attacks [Bel+14; Pie09; Sta+10; TS15]. However, when re-keying is applied, the implementation must still be resistant to SPA attacks.

2.3 Leakage-Resilient Cryptography

Frequent re-keying can be applied to any cryptographic scheme, e.g., an encryption scheme ENC or an authenticated encryption scheme AE, by choosing a new key whenever a new message has to be encrypted and authenticated, respectively. However, in such a re-keying approach, side-channel resistance is also affected by the concrete instance of the cryptographic scheme. In practice, the cryptographic scheme must be able to process arbitrarily long messages using a standard primitive, e.g., AES with 128-bit block size. This situation facilitates DPA in certain modes, such as Cipher Block Chaining (CBC). Therefore, the cryptographic scheme must be designed with special care.

In this respect, leakage-resilient cryptography comprises modes that are designed to process arbitrarily long messages without DPA vulnerability and to resist a certain amount of side-channel leakage. The latter typically means that if every invocation of the underlying primitive leaks, e.g., λ bits of information, leakage-resilient schemes guarantee that their overall leakage on the key stays within predefined bounds.

2.3.1 Leakage-Resilient Encryption

Two prominent examples of leakage-resilient encryption are depicted in Figure 2.2 and Figure 2.3. Both figures show two basic components: (1) a secure re-keying function g, and (2) a streaming mode performing leakage-resilient encryption. The re-keying function $g : (K, N) \mapsto \overline{K}_0$ securely derives a fresh initial session key $\overline{K}_0$ from a pre-shared (symmetric) master secret K and a fresh nonce N and must be implemented such as to resist both SPA and DPA attacks. We discuss possible constructions for g in Section 2.4.

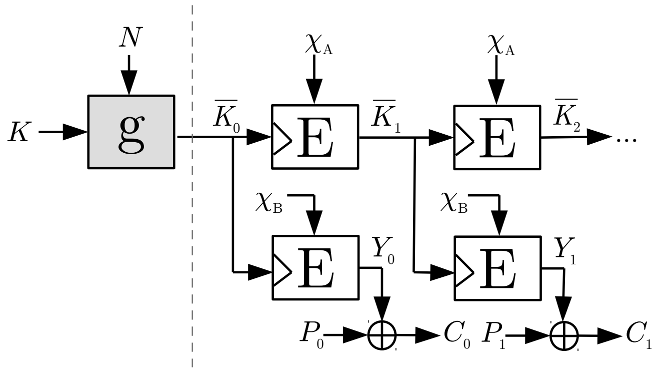


Figure 2.2: Leakage-resilient stream cipher.

In terms of leakage-resilient encryption, both schemes continuously perform key updates to securely process an arbitrary number of message blocks. In particular, when a certain plaintext block P_i has been processed with key $\overline{K}_i$, the next block's key $\overline{K}_{i+1}$ is obtained from encrypting a constant value χ_A using an encryption primitive E, e.g., a block cipher, with the current block's key $\overline{K}_i$. This key update mechanism ensures the use of a different key for the encryption of each plaintext block P_i and bounds the data complexity to $q = 2$ per $\overline{K}_i$. However, this iterative key update mechanism also results in random accesses to individual blocks to become quite expensive.

To perform the encryption of the single plaintext blocks P_i, current leakage-resilient encryption modes employ two basic variants. The majority of modes uses a stream cipher approach [Pie09; PSV15; Sta+10] similar to Figure 2.2. These modes use the current block's key $\overline{K}_i$ and a constant χ_B to compute a value $Y_i = E(\overline{K}_i; \chi_B)$ that is used to pad the plaintext P_i as $C_i = P_i \oplus Y_i$. On the other hand, there are also proposals for a block-cipher based encryption [TS15] such as in Figure 2.3, which directly use the current block's key $\overline{K}_i$ to encrypt P_i as $C_i = E(\overline{K}_i; P_i)$. Both variants clearly limit each key's data complexity during encryption by $q = 2$ to make DPA on the keys $\overline{K}_i$ infeasible. Yet, both variants must be implemented such that they resist SPA-like attacks. Therefore, such schemes must have bounded leakage of the key material, i.e., it must be hard to usefully combine the leakages of the single $\overline{K}_i$ in order to learn the key stream. Streaming modes such as in Figure 2.2 thus often come with a proof for bounded leakage of the key material given bounded leakage of the primitive E. Contrary to that, schemes such as in Figure 2.3 currently lack this feature.

2.3.2 Leakage-Resilient MAC

In terms of authentication, Pereira et al. [PSV15] presented a leakage-resilient Message Authentication Code (MAC), which is depicted in Figure 2.4. As leakage-resilient encryption, this leakage-resilient MAC makes use of a secure re-keying function g to derive a unique session key $\overline{K}_0$ from a pre-shared master key K

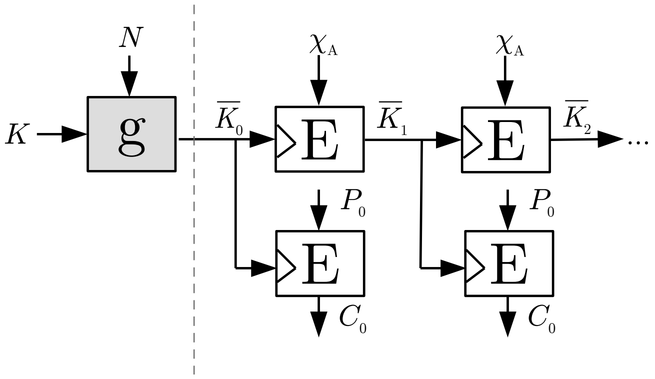


Figure 2.3: Leakage-resilient block cipher encryption.

and a nonce N. This session key $\overline{K}_0$ is then used to encrypt the hash h of the message M to form the tag T. The use of a fresh nonce N for each authentication ensures that without collisions no two messages M are authenticated using the same session key $\overline{K}_0$.

2.4 Secure Re-keying Functions

The leakage-resilient schemes presented before make use of a re-keying function $g : (K, N) \mapsto \overline{K}_0$ to derive a fresh session key $\overline{K}_0$ for initialization. This re-keying function g is $2^{|N|}$-limiting, where $|N|$ denotes the length of the nonce N, and gives a DPA setting that naturally results from combining a fixed key K with a random nonce N. As this DPA setting cannot trivially be prevented, the implementation of g does not only require bounded SPA leakage, but also needs dedicated countermeasures to protect against DPA attacks or a carefully chosen design to limit the feasibility of DPA on its internal components. We denote a re-keying function g fulfilling these requirements a *secure re-keying function*.

Today, several proposals to implement a secure re-keying function exist. The first approach is to build g in such a way that it is easy to secure by classical countermeasures like masking. This is the basic idea of fresh re-keying [Med+10; Med+11], which uses a polynomial multiplication of K and N over a finite field to implement g. While such multiplications can be masked easily, [Med+11; BFG14; Dob+14; Bel+15; GJ16; PM16] point out that the algebraic structure of a multiplication opens the door to combined attacks on g and the encryption. As a result, Dziembowski et al. [Dzi+16] recently proposed two new schemes for cryptographically strong re-keying that are suitable for masking and based on the learning parity with leakage and the learning with rounding problem, respectively.

A second approach presented in [Sta+10] is based on the classical Goldreich-Goldwasser-Micali (GGM) construction [GGM86]. The GGM construction can be used to mix a secret K with a public N in a tree-like approach, where on each

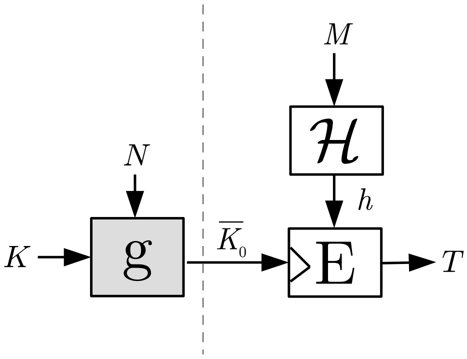


Figure 2.4: Leakage-resilient MAC.

tree level, exactly one bit of the public N is absorbed. Starting with $S_0 = K$, the state S_{i+1} is computed by encrypting one of two predefined plaintexts P_0, P_1 using the state S_i as the key, depending on the i-th bit of N. The output of the last level, $S_{|N|}$, is then, after postprocessing, used as the session key $\overline{K}_0$. This construction is 2-limiting, because the attacker only obtains the leakage for two inputs P_0 and P_1 to collect information about each S_i, and is thus typically considered secure against DPA attacks. A variant of this GGM construction was presented in [FPS12] and could be proven secure in the presence of side-channel leakage.

A third approach presented in [MSJ12; Bel+14] also originates from the classical GGM construction. It follows the idea of [Sta+10] by extending the number of observable measurements per key and deriving a Leakage-Resilient Pseudo Random Function (LR-PRF) from common block cipher designs to achieve secure re-keying. The core assumption of this approach is that the attacker is not able to distinguish the leakage of different hardware components on a chip.

2.5 Conclusion

IoT devices often operate in the presence of attackers with direct physical access to these devices, which allows these attackers to recover internally processed data by using side-channel attacks. In this chapter, we hence discussed the principles of side-channel attacks and their variants SPA and DPA. As the latter is particularly effective, we further explained two prominent DPA countermeasures, namely, secret sharing (or masking) and frequent re-keying. While masking tries to make the side-channel information independent from the processed data, frequent re-keying bounds the data complexity for a certain secret to make side-channel attacks that exploit multiple, different processings of the respective secret infeasible. We finally discussed the application of frequent re-keying in leakage-resilient cryptography to prove bounded side-channel leakage and re-keying functions to securely initialize leakage-resilient schemes.

Part I

Bounded Side-Channel Leakage

Side-channel attacks are an inherent threat to Internet-of-Things (IoT) devices and demand for appropriate countermeasures. Leakage-resilient cryptographic schemes are hence designed to resist a limited amount of side-channel leakage from the used cryptographic primitive and guarantee that their overall leakage on the key stays bounded. In this respect, it is of ongoing interest to find viable leakage models to argue on the side-channel security of cryptographic schemes given a certain amount of side-channel leakage from the implementation as well as to determine leakage bounds from physical properties of a device. In this part, we advance in terms of these questions and make the following main contributions:

- We introduce a model for side-channel leakage in permutation-based designs, which allows to argue on these schemes' leakage resilience by using the security properties of cryptographic sponges. Using our model, we present two novel, permutation-based re-keying functions secure against side-channel attacks to securely initialize leakage-resilient modes.

- We present an approach to determine the amount of side-channel leakage on a secret value in bits that can be learned via Gaussian side channels under a single data input and when attackers use signal averaging to remove noise. This method to quantify leakage can be used together with our leakage model in Chapter 3 to determine the security level of a permutation-based cryptographic scheme in the presence of side-channel leakage.

- To raise awareness of side effects from side-channel countermeasures, we show that in certain scenarios side-channel leakage obtained from leakage-resilient encryption results in a loss of plaintext confidentiality even though the leakage on the key is bounded.

3

Re-Keying and Leakage Model from Cryptographic Sponges

Leakage-resilient schemes are designed to resist a certain amount of side-channel leakage from their implementation. However, for a device that is expected to give a certain amount of side-channel leakage, it is unclear how to generically design and adapt cryptographic schemes to attain a specific security level. As well, it is an ongoing topic of research how to securely initialize such leakage-resilient schemes. While there are re-keying functions designed to give sufficiently low side-channel leakage and to perform secure initialization from a pre-shared secret and a nonce (cf. Chapter 2), many of the current re-keying functions suffer from algebraic weaknesses, lack efficiency, or rely on strong leakage assumptions. Together with Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, and Florian Mendel, we hence investigated new methods to realize secure re-keying and to model side-channel leakage when designing cryptographic modes, and we presented our results in [Dob+17]. In this chapter, we use the respective results from [Dob+17], but note that [Dob+17] additionally presents a leakage-resilient authenticated encryption scheme. However, regarding the results from [Dob+17] we use in this chapter, I contributed the idea, the hardware implementations, and most of the text, while Christoph Dobraunig, Maria Eichlseder, and Florian Mendel contributed the specific Keccak[400] instances. The detailed scientific contributions of this chapter are as follows.

Contribution. We first present a model for side-channel leakage in permutation-based designs, such as cryptographic sponges [Ber+11b], and show that these designs are particularly suitable in the face of side-channel leakage. In particular, we model the side-channel leakage as a part of the public output of the permutation. This allows to adjust the maximum tolerable leakage by varying the permutation parameters and to scale implementations for different leakage bounds. We emphasize that our leakage model gives a very powerful design tool since sponges are highly versatile and allow the design of many different cryptographic primitives, such as (authenticated) encryption, Message Authentication Codes (MACs), and hash functions.

Second, using this flexible tool as a basis, we propose two novel re-keying functions that are based on permutations and designed to completely prevent Differential Power Analysis (DPA) attacks. We instantiate both re-keying functions using Keccak[400] and present their hardware implementations. Our results indicate that the two re-keying functions guarantee 128-bit security in the presence of up to 135 bits leakage per permutation call and yet offer lower runtime and better efficiency than state-of-the-art re-keying functions.

Outline. Section 3.1 introduces the sponge leakage model and we design two novel re-keying functions in Section 3.2. Section 3.3 presents concrete Keccak[400]-based instances and the respective hardware implementations and Section 3.4 concludes this chapter.

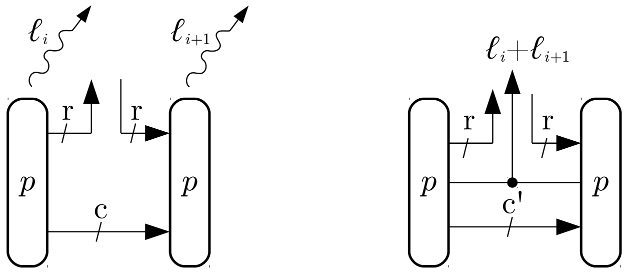
3.1 Leakage in Permutation-based Designs

As implementations are expected to leak a limited and defined amount of information via side channels, it is a challenging task for cryptographers to design schemes that provide a certain security level. On the other hand, permutation-based designs, such as the sponge construction [Ber+11b], have turned out to be highly flexible as they can be used to build a wide range of different cryptographic primitives, e.g, (authenticated) encryption [Dob+16; Ber+12], MACs [Ber+12] and hash functions [Ber+11c].

In this section, we make use of this flexibility to provide a model of side-channel leakage in keyed permutation-based designs. This model gives cryptographers a tool to consider side-channel leakage already in the design phase and to adapt the security parameters according to the expected amount of side-channel leakage of an implementation.

3.1.1 Sponge Leakage Model

The sponge parameters provide a convenient tool to argue on the side-channel security of keyed sponge constructions given bounded side-channel leakage of the single permutation. For illustration, we model the leakage from a permutation p by allowing an adversary to learn a certain amount of the state between subsequent permutation calls as depicted in Figure 3.1. Hereby, we use ℓ to denote the amount of information (in bits) that an attacker can learn about the state from the collected side-channel information. We do not care how and where the leakage is created within p, but let the adversary account the learned



(a) Leaking permutations **(b)** Sponge model

Figure 3.1: Leakage of information in sponge-based constructions.

information to either the input or the output state of p. Therefore, given two consecutive permutations p with leakages ℓ_i and ℓ_{i+1}, respectively, the maximum an adversary might learn about the state is $\ell_i + \ell_{i+1}$. This means that if each leakage ℓ_i, ℓ_{i+1} is bounded by λ bits and the adversary can optimally combine these two leakages, the adversary will learn at most 2λ bits of the state between the respective two permutation calls.

The basic idea now is to use the sponge parameters to express a construction's capability to cope with the leakage generated by the permutation. In particular, the sponge parameters are adjusted according to the amount of information an adversary learned about the secret state. This means that if the adversary learns 2λ bits of the internal, secret state, the leaked bits can be considered as an increase of the rate, i.e., $r' = r+2\lambda$, which results in a smaller capacity $c' = c-2\lambda$ and thus reduced security. However, a reduced security level corresponding to a capacity of $c - 2\lambda$ bits is still guaranteed by the cryptographic properties of the permutation and the associated Constrained-Input Constrained-Output (CICO) problem [Ber+11b]. Sponge-based constructions can thus be considered to have bounded security loss for bounded leakage of the permutation.

Bounded Side-Channel Leakage. Clearly, the challenge in practice is to build an implementation that bounds the leakage of p. While frequent rekeying, which implicitly occurs in many permutation-based designs, thwarts DPA, determining a concrete Simple Power Analysis (SPA) leakage bound is difficult. Especially if a large class of different devices have to use the same cryptographic algorithm it might be infeasible to make any realistic assumptions about the leakage of p. Nevertheless, the advantage of the sponge-based construction is that besides standard SPA countermeasures, like hiding and masking, the capacity is an additional and very natural security parameter that helps to increase the ability of a design to withstand side-channel attacks in practice. For example, the leakage assumption in Figure 3.1 allows to scale a sponge-based scheme to maintain its desired security level in the presence of λ-bit leakages ℓ_i, ℓ_{i+1} by increasing the capacity by 2λ bits (and either decreasing the rate r accordingly, or increasing the size of the permutation p).

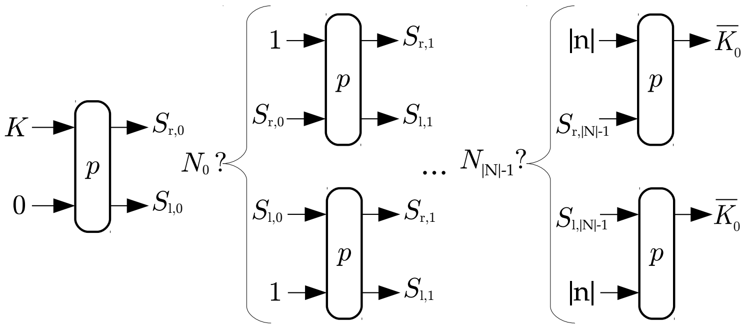


Figure 3.2: Re-keying inherently secure against DPA attacks: IsapRk1.

3.2 Secure Re-Keying Functions

As shown in Chapter 2, there are several secure re-keying functions $g : (K, N) \mapsto \overline{K}_0$ to initialize leakage-resilient schemes with a session key $\overline{K}_0$ from a pre-shared master secret K and a nonce N. However, many of them have severe drawbacks, such as being vulnerable to combined attacks or relying on strong leakage assumptions. On the other hand, permutation-based designs are particularly suitable to cope with side-channel leakage. In this section, we hence present two novel re-keying functions, IsapRk1 and IsapRk2, based on cryptographic permutations and secure against passive side-channel attacks. IsapRk1 and IsapRk2 are 1- and 2-limiting, respectively, to prevent DPA and both allow to control their resilience to SPA as shown in Section 3.1.

3.2.1 Re-Keying with Inherent DPA Security

In our first design, we use a variation of the classical Goldreich-Goldwasser-Micali (GGM) construction [GGM86]. The respective re-keying function IsapRk1 is shown in Figure 3.2 and works as follows. The state is first initialized with the padded master key K, followed by an application of the permutation p. In each iteration, one bit of the nonce N is processed by either choosing the left or right half of the permutation output, padding it to the permutation size, and again applying the permutation p. Hereby, the padding incorporates information on which half was chosen and on the index of the nonce bit being processed. After all nonce bits have been processed, the session key $\overline{K}_0$ is generated from the last permutation output.

The approach to re-keying used in IsapRk1 inherently protects against DPA attacks, since the same secret (i.e., right or left part of the permutation output) is never combined with more than one public input. In this respect, IsapRk1 has a lower data complexity bound than present GGM-based re-keying functions [FPS12; Sta+10] which are 2-limiting when instantiated using common block ciphers [Pie09]. In terms of SPA leakage, even though IsapRk1 is not a sponge construction, our modeling from Section 3.1 can be applied with only

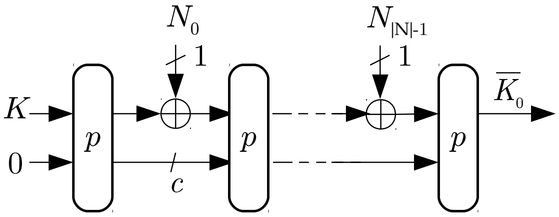


Figure 3.3: Sponge construction for re-keying: IsapRk2.

little adaptations: we append the 2λ bits learned about the intermediate state to the known part of the state that without leakage only consists of padding bits. As a result, the secret part, which—depending on the i-th nonce bit N_i—either is $S_{r,i}$ or $S_{l,i}$, is reduced by 2λ bits. Thus, the size of the permutation p used in IsapRk1 has to be chosen accordingly to obtain a sufficiently large secret part to maintain the desired security level in the presence of λ-bit leakage of p.

3.2.2 More Efficient Re-keying

A more efficient re-keying function than IsapRk1 can be obtained from sponges directly, potentially reducing the required state and permutation size. However, the presented re-keying function uses a stronger security assumption than IsapRk1, namely, that DPA is impossible on a 2-limiting primitive, i.e., given the leakages from two different public inputs.

The basic idea is to make DPA infeasible by reducing the input data complexity accordingly. For this purpose, a secret state is constantly updated with small portions of public data by repeating two phases, (1) modifying the secret state according to the public data, and (2) updating the state such that predictions on the future state based on the absorbed public data become infeasible.

Sponges are an ideal choice to implement this basic idea as the rate directly influences the input data complexity for each permutation. Choosing the smallest possible rate ($r = 1$) results in the design IsapRk2 shown in Figure 3.3. IsapRk2 first initializes the sponge state by applying the initial permutation p to the padded master key K. Then, IsapRk2 repeatedly injects single nonce bits into the state, each separated by a permutation call. After full absorption of the nonce and a final permutation call, the session key $\overline{K}_0$ is output. This working principle is similar to sponge instances of a prefix MAC. While for general MAC computations the absorption rate can be as big as the state size [Ber+12], IsapRk2 uses a small absorption rate in order to limit the data complexity exploitable in a DPA.

In terms of DPA security, IsapRk2 uses a different assumption than IsapRk1. For each secret state in IsapRk2, a permutation p will produce the leakages for two different public inputs. Thus, IsapRk2 is not inherently secure to DPA attacks, but 2-limiting. This results in IsapRk2 being a secure re-keying function under the assumption that the combined leakage resulting from the processing of two different public inputs is bounded such that DPA on the secret state is infeasible. In this case, our leakage considerations from Section 3.1 can be straightforwardly applied to include appropriate security margins for the side-channel leakage from the processing of two different inputs. However, note that this construction for a secure re-keying function is again related to the classical GGM construction [GGM86] and can be seen as their sponge equivalent, similar to [TS14]. ISAPRK2 is similar to it in the sense that the exploitable data complexity is equal for ISAPRK2 and the block-cipher based instantiations of both [FPS12] and the 2PRG primitive used in [Sta+10].

3.3 Implementation

The two re-keying functions were designed to be secure against passive side-channel attacks. For practical use, we will now specify instance parameters for both ISAPRK1 and ISAPRK2 based on the permutation Keccak[400]. We implemented both re-keying functions in hardware to show that ISAPRK1 and ISAPRK2 are both smaller and more efficient than state-of-the-art re-keying functions while at the same time allowing a side-channel leakage of up to 36 bits and 135 bits per permutation call, respectively.

3.3.1 Instantiation Keccak[400]

Besides side-channel attacks, the concrete choice of the parameters for instances of ISAPRK1 and ISAPRK2 depends on the desired security against cryptographic attacks. In this respect, we state the security level as the intended number of bits of security. The *cryptographic* security level of the two re-keying functions ISAPRK1 and ISAPRK2 is summarized for a s_p-bit permutation p, a capacity c, a master key K of length $|K|$, and a session key $\overline{K}_0$ of length $|\overline{K}_0|$, in Table 3.1.

Based on Table 3.1, we use the permutation Keccak[400] to target a 128-bit security level. In particular, Keccak[400] uses a permutation size of $s_p = 400$ and is hence suitable to meet the security requirements of the two re-keying functions ISAPRK1 and ISAPRK2. In terms of the number of permutation rounds, we base our choice on Keccak[400], which uses 20 [Ber+11c] rounds, and on the CAESAR candidate Keyak [Ber+14b], which uses 12 rounds for both the 800 and 1600-bit Keccak permutation. For ISAPRK2, injecting the secret key before the first permutation call significantly restricts possible attack vectors. We hence followed the proposal of the similar construction Keyak [Ber+14b] and use 12 rounds for ISAPRK2. On the other hand, ISAPRK1 is permutation-based, but not a sponge.

Table 3.1: Security level (in bits) for the two re-keying functions.

Function	Security (bits)	References
ISAPRK1	$\min(\lvert K\rvert, s_p/2, \lvert\overline{K}_0\rvert)$	
ISAPRK2	$\min(\lvert K\rvert, s_p/2, c, \lvert\overline{K}_0\rvert)$	[And+15]

As a result, we cannot rely on third-party analysis for IsapRk1 as before and aim for a very strong permutation like for the hash function Keccak [Ber+11c]. We hence decided to use a very conservative number of 20 rounds for IsapRk1. Table 3.2 summarizes the recommended capacity and the number of permutation rounds for both IsapRk1 and IsapRk2.

While we chose the number of permutation rounds conservatively, note that IsapRk2 can easily be optimized by using a smaller number of rounds similar to the CAESAR candidate Ketje [Ber+14a]. In particular, while the initial and last permutations in IsapRk2 should have a large number of rounds to ensure cryptographic security via good diffusion of the in- and output keys, the permutation calls between the injection of the nonce bits only serve the purpose of breaking the link between side-channel leakages. As a result, a reduced number of rounds for the inner permutations can securely yield higher performance if the side-channel leakages remain hard to combine.

Table 3.2: Parameters for IsapRk1 and IsapRk2 with 128-bit security using the Keccak[400] permutation ($|K| = |\overline{K}_0| = 128$, $s_p = 400$).

Function	Capacity c	max. State Leakage	Rounds	
IsapRk1	–	72	20	[Ber+11c]
IsapRk2	399	271	12	[Ber+14b]

3.3.2 Security with State Leakage

Based on the approach in Section 3.1 and the security properties in Table 3.1, Table 3.2 gives the maximum tolerated state leakage for IsapRk1 and IsapRk2 and 128-bit security. For IsapRk1, up to $2\lambda = 72$ bits of the internal state can be leaked for $s_p/2 - 2\lambda \geq 128$ bits to hold true. This is equivalent to the permutation leaking up to $\lambda = 36$ bits per invocation. For IsapRk2 and an 1-bit rate, $2\lambda = 271$ bits of the internal state can be leaked for $c - 2\lambda$ not to fall below 128 bits, i.e., roughly $\lambda = 135$ bits per invocation of p. However, IsapRk2 is a 2-limiting design and therefore allowing such larger amounts of leakage will be necessary in practice.

For both IsapRk1 and IsapRk2, we recommend implementers to store on a device the 400-bit expanded key, which is initially used after the first invocation of p, instead of the 128-bit master key. Directly using the 400-bit expanded key avoids leakage of master-key bits in the initial permutation and increases the overall security due to the larger secret state that is used within the re-keying functions. Besides, note that load-time leakage of the master key can be considered as state leakage before the initial permutation.

3.3.3 Results and Comparison

We implemented the re-keying functions IsapRk1 and IsapRk2 according to the instance parameters in Section 3.3.1. Both designs compute one round of the Keccak[400] permutation per cycle and were synthesized in an UMC 130 nm technology. For comparison, we also implemented and synthesized the GGM-based re-keying function [Sta+10] based on an AES capable of one round per cycle. The respective results shown in Table 3.3 are supplemented with synthesis results stated in the literature for re-keying with polynomial multiplication [Med+10].

Table 3.3: Implementation results for secure re-keying functions (130 nm).

Function	Area [kGE]	Frequency [MHz]	Runtime[a]	
			[cycles]	[μs]
IsapRk1	8.5	172	2 709	15.8
IsapRk2	7.7	212	1 677	7.9
AES-GGM [Sta+10]	11.2	101	1 536	15.2
PolyMult [Med+10]	10.2	–	1 160	–

---

[a] Runtime is given for 128-bit nonces.

The results in Table 3.3 suggest that both IsapRk1 and IsapRk2 give a smaller re-keying function than using an AES-based GGM tree. While IsapRk2 is twice as fast as AES-GGM, IsapRk1 is similarly fast. Note however that IsapRk1 provides inherent security to DPA attacks while AES-GGM and IsapRk2 are 2-limiting. Moreover, both IsapRk1 and IsapRk2 provide a large security margin to cope with SPA leakage.

Compared to the polynomial multiplication with 3rd order masking in [Med+10], the re-keying functions IsapRk1 and IsapRk2 have a smaller area footprint and do not allow for Time-Memory Trade-Off (TMTO) attacks [Men+12; Bog+14; Dob+14]. Besides, the Isap re-keying functions do not require dedicated DPA countermeasures, but were designed in view of preventing DPA scenarios at all. Re-keying functions based on Leakage-Resilient Pseudo Random Functions (LR-PRFs) [Bel+14] are omitted in Table 3.3 as their security can hardly be compared to the ones of IsapRk1 and IsapRk2 as it is mainly based on the assumption that locality information is physically not observable. However, the 7.3 GE implementation of LR-PRFs performs re-keying in 0.5 μs at 338 MHz and is thus faster than both Isap re-keying functions. Yet, concerning applications requiring bulk en-/decryption of, e.g., Field Programmable Gate Array (FPGA) bit files, re-keying will typically not be the bottleneck.

3.4 Conclusion

In this chapter, we proposed a leakage model for permutation-based designs, like sponges, that allows to design cryptographic schemes that reach a certain security level in presence of a specific amount of side-channel leakage per permutation call. In particular, we modeled side-channel leakage in cryptographic sponges as part of the public rate of the state, which reduces the capacity and thus security accordingly. In order to reach a certain security level in the presence of leakage, designers thus can choose between a smaller rate or a larger permutation to compensate for this security reduction. We further presented two novel re-keying functions based on permutations that can be used to securely initialize leakage-resilient schemes in the presence of side-channel leakage. In particular, the two re-keying functions IsapRk1 and IsapRk2 are designed to prevent DPA by limiting the number of different inputs to the permutation by one and two, respectively. Finally, we implemented the two re-keying functions in hardware to show that both require less area and are more efficient than state-of-the-art re-keying functions while resisting up to 135 bits of side-channel leakage per permutation call.

4

Leakage Bounds for Gaussian Side Channels

Both in our leakage model from Chapter 3 and in leakage-resilient schemes, the side-channel information from the underlying primitive is assumed to be bounded by λ bits. It is however an ongoing topic of research to specify concrete leakage bounds λ based on the implementation and its physical properties. To address this question, together with Thomas Korak, Stefan Mangard and Robert Schilling, we developed a new method to specify the leakage from a side channel in bits, we verified the method through practical measurements on the chip FULMINE, which was developed by Robert Schilling in collaboration with Luca Benini, Frank Gürkaynak and Michael Muehlberghuber at ETH Zurich, and we published both the method and results in [Unt+17]. In this publication, I was the main author contributing ideas, experiments and most parts of the text, while Thomas Korak and Robert Schilling performed all measurements. In this chapter, we use text and results from [Unt+17] and make the following detailed contributions.

Contribution. We present a new approach to give reliable upper bounds for the leakage from side channels of cryptographic implementations under a single data input. For this purpose, we map results from communication theory to the side-channel domain. In particular, we show that the channel capacity of n-to-m communication channels is the natural upper bound for the Mutual Information (MI) in multivariate side-channel leakages with Gaussian noise. Without any further leakage assumptions, we show that this bound depends on a device- and measurement-specific Signal-to-Noise Ratio (SNR) that is uniquely determined by the device's statistical leakage behavior in the m Points of Interest (POIs) in the leakage trace. In a second step, we investigate the effect of signal averaging on this device- and measurement-specific SNR and show that averaging $\mathcal{N}$ leakage traces increases the SNR by a factor $\mathcal{N}^m$. Our results provide both attackers and implementers with a tool for computing the expected minimum attack complexity, i.e., the number of leakage traces required to learn a certain amount of the processed state from side-channel information. We then show that our model and results fit the reality by evaluating the MI in multivariate Gaussian templates. For this purpose, we used power measurements from a real System on Chip (SoC) that features a Keccak[400] engine that computes three rounds per cycle. Last, we use our model to demonstrate the security of the scheme Isap implemented on this SoC in terms of power analysis attacks.

Outline. This chapter is organized as follows. We briefly describe open questions and current approaches to determine leakage bounds in Section 4.1. Section 4.2 gives bounds for the information leakage via multivariate side channels with Gaussian noise. We analyze the case of signal averaging and provide a tool to compute the expected minimum attack complexity for side-channel attackers in Section 4.3. The soundness of our leakage model is shown in Section 4.4 based on power measurements of an Application Specific Integrated Circuit (ASIC), and we finally conclude in Section 4.5.

4.1 Background

For assessing the leakage behavior of cryptographic implementations, one possible starting point are practical security evaluations. For example, Medwed et al. [Med+16] evaluated a set of practical Differential Power Analysis (DPA) attacks on simulated leakages from parallel implementations with unknown in- and outputs. Their resulting success probabilities indicate that even for identity leakage of the secret state, its exploitation is practically hard once enough processes happen in parallel. While their specific results also suggest security and bounded leakage for limited data complexities as they occur in leakage-resilient schemes, it is hard to derive a concrete leakage bound λ in bits.

As an alternative approach, Standaert et al. [SMY09] suggested using the Mutual Information (MI) from information theory as a general tool to concretely state the amount of information learned from side-channel leakage in bits. This MI can be exactly computed once the actual leakage distribution of an implementation is known. However, this leakage distribution is typically hard to determine in practice. On the other hand, Duc et al. [DFS15] mention an upper bound for the MI for univariate leakages that solely depends on the device- and measurement-specific SNR. It, however, so far is an open question how this bound scales for multivariate leakages that are exploited in practice. The method presented in this chapter is the first to assess the information in multivariate leakages from a (multivariate) device- and measurement-specific SNR.

In the view of physical constraints such as the SNR, the MI is typically bounded for a single measurement of the side-channel leakage and hence suits leakage-resilient schemes. While most of these schemes indeed confine the attacker to a single measurement by requiring a fresh initial state on every invocation, there are also schemes allowing attackers to observe the same execution using the same data multiple times, e.g., as for multiple decryptions in ISAP [Dob+17]. However, multiple measurements of the same decryption process allow an attacker to perform signal averaging to increase the SNR. This can allow unbounded side-channel attackers to distinguish tiny variances in the signal to learn the complete secret state. However, in practice, side-channel attackers are bounded by physical and computational resources. This gives the interesting question of the actual attack complexity when the side-channel attacker is capable of observing the same execution multiple times and performing signal averaging. In this respect, the approach presented in this chapter also gives a tool to estimate the attack complexity when an attacker is capable of signal averaging to remove noise.

4.2 Modeling Side-Channel Leakage as a Communication Channel

In this section, we consider the case of leakage-resilient cryptography where an attacker can use the side-channel information in a single leakage trace to learn the secret state of a device. In particular, we adapt the results from communication theory to fit side-channel leakages and use the channel capacity of n-to-m wireless channels to give a leakage upper bound for multivariate side channels with Gaussian noise independent of the underlying leakage function.

4.2.1 Attack Model

We consider an attacker trying to recover the secret state x from a single leakage trace l_x generated by an implementation $\mathcal{I}$ with input complexity $q = 1$. This implies that the attacker is unable to perform multi-input attacks such as DPA. Moreover, attackers are allowed to observe the operation using the secret state x only a single time, i.e., they are not allowed to average traces to improve their SNR. However, we will discuss the scenario of trace averaging later in Section 4.3. Besides, we consider a profiled attack setting, i.e., the attacker has the opportunity to build templates before performing the actual attack.

4.2.2 Mutual Information

A common metric to assess the amount of information about a secret x contained in the leakage l_x is the Mutual Information (MI) [SMY09; DFS15]. We therefore introduce the random variables X and L_x to denote the distributions of x and l_x, respectively. The mutual information is then defined as

$$MI(X; L_x) = H[X] - H[X|L_x]. \tag{4.1}$$

Hereby, $H[X]$ and $H[X|L_x]$ denote the entropy of the random variable X and the conditional entropy of X given the leakage L_x, respectively. Note however that the (conditional) entropy (and thus the MI) is an average metric depending on the actual distribution of values $x_i \in X$ and $l_x \in L_x$. This means that the actual information learned from a side-channel leakage depends on the actually processed value and might thus for some events even be higher than the MI. Yet, the MI is a good metric to give bounds on the expected leakage behavior.

4.2.3 Linear Channel Model

For giving bounds on the MI of side channels, we consider an implementation that transmits the single bits of a secret state to the attacker via a side channel. Hereby, the physical leakage behavior and measurement effects define the mapping of the single bits to the output samples of the side channel. We model this multivariate side channel as an n-to-m linear communication channel with Gaussian noise, i.e., it transfers linear combinations of the bits of the secret state. While this linear channel model allows to adapt results from communication theory, the resulting bounds are yet independent from the concrete leakage behavior and Gaussian noise is the sole assumption. Namely, our final bounds will only depend on the side-channel signal observed by the attacker. Further note that non-linear mappings can easily be added to this model similar as for regression techniques [SLP05].

In our linear channel model, the attacker observes an $m \times 1$ leakage trace $\mathbf{l}_x$ from the processing of the secret state x in the implementation $\mathcal{I}$. Let $\mathbf{x}$ denote the $n \times 1$ vector consisting of the n bits of the secret state x. We then model the leakage trace $\mathbf{l}_x$ as the multiplication of the secret state vector $\mathbf{x}$ with an $m \times n$ side-channel matrix $\mathbf{H}$ plus an $m \times 1$ noise vector $\boldsymbol{\nu}$:

$$\mathbf{l}_x = \mathbf{H}\mathbf{x} + \boldsymbol{\nu}. \tag{4.2}$$

The i-th row of $\mathbf{H}$ specifies how the n bits of the secret state x map to the i-th point of the measured leakage $\mathbf{l}_\mathbf{x}$. The maximum MI that an attacker can learn from the side-channel leakage according to Equation 4.2 depends on the maximum number of states that are distinguishable at the receiver of this channel. This upper bound on the MI is typically called the channel capacity. In particular, Telatar [Tel99] states the channel capacity $\mathcal{C}$ as the maximum average mutual information between in- and output over the choice of the input distribution, i.e.,

$$\mathcal{C} = \max_{p(X)} MI(X, L_x). \tag{4.3}$$

We observe that the side-channel leakage given by Equation 4.2 bears some familiarity with the notion of Multi-Input Multi-Output (MIMO) channels as used in wireless communication. For a constant, known channel $\mathbf{H}$, Goldsmith et al. [Gol+03] state the channel capacity for signals in the domain of complex numbers as follows:

$$\mathcal{C} = \max_{\Sigma_{\mathbf{x}}:tr(\Sigma_{\mathbf{x}})=P} \log_2 |\mathbf{I}_m + \mathbf{H}\Sigma_{\mathbf{x}}\mathbf{H}^H| \tag{4.4}$$

Hereby, $\mathbf{I}_m$ and $\Sigma_{\mathbf{x}}$ denote the $m \times m$ identity matrix and $n \times n$ signal covariance matrix, respectively. P is the total power constraint of the transmitter, $\mathbf{H}^H$ the complex conjugate of $\mathbf{H}$, $|\cdot|$ the determinant, and $tr(\cdot)$ the trace of a matrix. For Equation 4.2 to hold true, the noise vector $\boldsymbol{\nu}$ must consist of independent samples of Gaussian white noise with variance $\sigma_\nu^2 = 1$, i.e., the $m \times m$ noise covariance matrix $\Sigma_{\boldsymbol{\nu}}$ is the identity matrix $\mathbf{I}_m$.

We can use the channel capacity of MIMO channels as an upper bound for the MI in side-channel leakages according to Equation 4.2. However, there are different constraints for side channels than in wireless communication, requiring some modifications of Equation 4.4. For example, an attacker cannot influence the signal covariance $\Sigma_{\mathbf{x}}$ such as to optimize the capacity $\mathcal{C}$. Moreover, the capacity of communication channels as in Equation 4.4 is given for the transmission of symbols in the domain of complex numbers, which is twice the capacity of channels transferring symbols in the domain of real numbers. On the other hand, side-channel attacks typically exploit real-valued information like the power consumption. As a result, the capacity in Equation 4.4 effectively halves for the side-channel case. In practice, we also observe that the samples in the noise vector $\boldsymbol{\nu}$ are not necessarily independent and have different variances. According to [Gol+03], dependent samples in the noise $\boldsymbol{\nu}$ can be modeled via a modified channel matrix $\tilde{\mathbf{H}} = \Sigma_{\boldsymbol{\nu}}^{-1/2}\mathbf{H}$ given the noise covariance matrix $\Sigma_{\boldsymbol{\nu}}$. By adapting Equation 4.4 according to these considerations, we extract the special case of linear side channels as in Equation 4.2 and state their leakage upper bound:

$$\mathcal{C} = \max_{p(X)} MI(X, L_x) = \frac{1}{2} \log_2 |\mathbf{I}_m + \Sigma_{\boldsymbol{\nu}}^{-1}\mathbf{H}\Sigma_{\mathbf{x}}\mathbf{H}^H|. \tag{4.5}$$

4.2.4 Leakage Bound for Gaussian Side Channels

The side-channel matrix $\mathbf{H}$ is typically unknown but fixed. An interesting question thus is how to determine the channel capacity if $\mathbf{H}$ is unknown. A common approach to characterize a side channel are multivariate Gaussian templates. Hereby, for each secret state $\mathbf{x}$, the respective side-channel leakage $\mathbf{l}_{\mathbf{x}}$ is described as a multivariate Gaussian distribution. This characterization gives a set of templates $(\mu_i, \Sigma_{\boldsymbol{\nu},i})$ with mean μ_i and noise covariance $\Sigma_{\boldsymbol{\nu},i}$ for all states $\mathbf{x}_i$. The means μ_i give an estimation of the $n \times n$ covariance matrix $\Sigma_{\mathbf{y}}$ of the side-channel signal $\mathbf{y} = \mathbf{H}\mathbf{x}$. This covariance matrix $\Sigma_{\mathbf{y}}$ equals $\mathbf{H}\Sigma_{\mathbf{x}}\mathbf{H}^H$ from Equation 4.5. Similarly, assuming that the noise is independent from the signal and thus has constant covariance (as in [Riv08]), the single noise covariances $\Sigma_{\boldsymbol{\nu},i}$ give an estimation of $\Sigma_{\boldsymbol{\nu}}$.[1] Putting this together, we adapt Equation 4.5 to derive our main result. Namely, we use the signal and noise covariance matrices $\Sigma_{\mathbf{y}}, \Sigma_{\boldsymbol{\nu}}$ to state the capacity of a side channel characterized via multivariate Gaussian templates, or more generally, of multivariate leakages with Gaussian noise.

Main Result (Leakage Bound of a Gaussian Side Channel)**.** *The mutual information of a multivariate side channel with signal covariance* $\Sigma_{\mathbf{y}}$ *and Gaussian noise* $\Sigma_{\boldsymbol{\nu}}$ *is bounded by*

$$\mathcal{C} = \frac{1}{2}\log_2 |\mathbf{I}_m + \Sigma_{\boldsymbol{\nu}}^{-1}\Sigma_{\mathbf{y}}|. \tag{4.6}$$

Interestingly, the term $\Sigma_{\boldsymbol{\nu}}^{-1}\Sigma_{\mathbf{y}}$ is an SNR taking noise and signal covariances between the POIs into account. The capacity of the side channel is thus determined by the actual power of signal and noise, and correlations in the samples of $\boldsymbol{\nu}$ and $\mathbf{y}$. Such correlations typically mark redundancies that effectively reduce the side-channel capacity. Moreover, note that the side-channel capacity given here depends on the side-channel signal $\mathbf{y}$ only. This means that our result applies to any leakage function/model having the properties given by $\Sigma_{\mathbf{y}}$.

For univariate leakages or when the same leakage is observed in multiple POIs within the leakage trace, the leakage bound in Equation 4.6 can further be simplified.

Univariate Leakage.

An attacker exploiting univariate leakage is confined to the leakage in a single point of the execution of an implementation $\mathcal{I}$. This means that the side channel degenerates to

$$l_{\mathbf{x}} = \mathbf{h}\mathbf{x} + \nu, \tag{4.7}$$

where $l_{\mathbf{x}}$ and ν are scalars and the $1 \times n$ channel vector $\mathbf{h}$ specifies the leakage of the single bits of the state $\mathbf{x}$. Let us now assume that the channel vector $\mathbf{h}$ maps the n bits in $\mathbf{x}$ to y according to the identity of the respective state variable x. Intuitively, the MI between the secret state $\mathbf{x}$ and its leakage $l_{\mathbf{x}}$ is here bounded by the number of different states that an attacker can distinguish in the single leakage point $l_{\mathbf{x}}$. This number depends both on the distance between the different states along the measured signal range and the noise. When adapting Equation 4.6 for univariate leakage, we can observe exactly this dependence:

$$\mathcal{C} = \frac{1}{2}\log_2\left(1 + \frac{\sigma_y^2}{\sigma_\nu^2}\right) = \frac{1}{2}\log_2\left(1 + \mathit{SNR}\right), \tag{4.8}$$

---

[1] The constant covariance assumption is invalid in case the covariance carries information as, e.g., in masked implementations. However, leakage-resilient cryptography aims to bound the leakage without the use of countermeasures like masking, and thus noise will typically be independent from the signal.

where σ_y^2 is the variance of the signal $y = \mathbf{h}\mathbf{x}$ and σ_ν^2 is the variance of the noise ν. As also noted in [DFS15; Miz+13], this upper bound for the MI in univariate leakages solely depends on the SNR and is better known as the Shannon-Hartley theorem [CT12].

Identical Leakage in Multiple Points.

In many cases, an attacker will try to exploit the leakage in multiple points of the execution to increase their success rate. If these points are chosen to be in close vicinity within the leakage trace, these POIs will often carry highly redundant information. An example where this case occurs are attackers sampling the side channel at a very high rate and using several consecutive sampling points in their attack. In such situation, one can assume the leakage to be the same for all points of the leakage trace. This case is equivalent to Single-Input Multi-Output (SIMO) channels in wireless communication. The side-channel matrix is then expressed as the vector multiplication $\mathbf{H} = \mathbf{h}_{gain} \cdot \mathbf{h}_l$, where $\mathbf{h}_l$ states the $1 \times n$ side-channel vector mapping the n bits of x to a scalar value and $\mathbf{h}_{gain}$ is the $m \times 1$ gain vector over the m POIs used by the attacker. The capacity formula in Equation 4.5 degenerates for such leakage behavior, but can simply be expressed using the vector $\mathbf{h}_{gain}$ only [Gol05]:

$$\mathcal{C} = \frac{1}{2} \log_2 \left(1 + \sigma_z^2 \mathbf{h}_{gain}^H \Sigma_{\boldsymbol{\nu}}^{-1} \mathbf{h}_{gain}\right), \tag{4.9}$$

where σ_z^2 is the variance of the signal $z = \mathbf{h}_l\mathbf{x}$ such that $\mathbf{l_x} = \mathbf{h}_{gain} z + \boldsymbol{\nu}$.

4.2.5 Description of Common Leakage Models

Our leakage model in Equation 4.2 allows to easily describe linear side-channel leakages. We now give several examples on how to map existing power models to Equation 4.2. Note that we give these examples without consideration of the effective signal range in the leakage $\mathbf{l_x}$.

Identity Leakage. In identity leakage, the n-bit secret state $\mathbf{x}$ leaks linear to the value x it represents. If $\mathbf{x}$ leaks the identity in the i-th sample of $\mathbf{l_x}$, the i-th row in the side-channel matrix $\mathbf{H}$ takes the form $\mathbf{h} = \begin{pmatrix} 2^0 & 2^1 & 2^2 & \dots & 2^{n-2} & 2^{n-1} \end{pmatrix}$.

Hamming Weight Leakage. In Hamming Weight (HW) leakage, the secret state $\mathbf{x}$ leaks the number of bits set to one. HW leakage in the i-th sample of $\mathbf{l_x}$ results in the i-th row of $\mathbf{H}$ to take the form $\mathbf{h} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 1 \end{pmatrix}$. Hamming Distance (HD) leakage is modeled in the same way by setting the secret x to be the XOR of the leaking state before and after it toggles.

Time-Serialized Leakage. In time-serialized implementations, an attacker collecting the side-channel leakage at different points in time will be able to learn different information in the different POIs. One prominent example are byte-oriented cryptographic implementations, where in each clock cycle a different byte of the n-bit state $\mathbf{x}$ is processed and leaks. For simplicity, let us assume an 8-bit state and HW leakage of a 2-bit chunk processed in the respective clock cycle. This will give a side-channel matrix of the form

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Localized Leakage. Localized Electromagnetic Emanation (EM) attacks are a powerful way to extract information from parts of the secret state. Such localized EM leakage can in principal be modeled the same way as time-varying leakage. For example, consider the leakages $\mathbf{l}_{\mathbf{x},1}$ and $\mathbf{l}_{\mathbf{x},2}$ observed in two different EM positions. Moreover, assume that $\mathbf{l}_{\mathbf{x},1}, \mathbf{l}_{\mathbf{x},2}$ consist each of two samples leaking the identity of the first or second half of a 4-bit state, respectively. Concatenating the two leakages $\mathbf{l}_{\mathbf{x}}^T = (\mathbf{l}_{\mathbf{x},1}^T \mathbf{l}_{\mathbf{x},2}^T)$ means concatenating the respective channel matrices $\mathbf{H}_1, \mathbf{H}_2$ to a combined side-channel matrix of the form

$$\mathbf{H} = \begin{pmatrix} 2^0 & 2^1 & 0 & 0 \\ 2^0 & 2^1 & 0 & 0 \\ 0 & 0 & 2^0 & 2^1 \\ 0 & 0 & 2^0 & 2^1 \end{pmatrix}.$$

This model underlines the intuition that gathering additional leakage from observing a parallel implementation in different locations and measuring a serial implementation at different times is equivalent. In particular, it shows that side-channel leakage becomes optimal if the leakages in the side-channel signal $\mathbf{y} = \mathbf{H}\mathbf{x}$ are independent. In the best case, the signal covariance matrix becomes a diagonal matrix, i.e, $\Sigma_{\mathbf{y}} = diag(\sigma_{y_1}^2, \sigma_{y_2}^2, ..., \sigma_{y_m}^2)$. In the same way, noise effects are canceled out the best if the noise samples in $\boldsymbol{\nu}$ are independent, i.e., $\Sigma_{\boldsymbol{\nu}}$ is a diagonal matrix as well.

4.3 Complexity of State Recovery

The side-channel capacity is an upper bound on the MI to be learned via a side channel. This bound essentially depends on the implementation's SNR. While in most leakage-resilient schemes an attacker is restricted to a single leakage trace for a specific state, there are schemes, e.g., ISAP [Dob+17], that allow attackers to observe the execution of an implementation processing the same data multiple times. This gives attackers the option to perform signal averaging, which improves the side-channel SNR and thus side-channel capacity.

In this section, we therefore consider an attacker capable of averaging multiple leakage traces. We show how averaging improves the side-channel capacity in multivariate attacks and provide attackers and implementers with a tool to compute the expected minimum complexity to learn the secret state of a device.

4.3.1 Attack Model

As in Section 4.2, we assume an attacker trying to recover a secret state x from side-channel leakages l_x generated by an implementation $\mathcal{I}$ with input complexity $q = 1$ and thus preclude multi-input attacks. However, the attacker is capable of observing the same execution of $\mathcal{I}$ multiple times. This attack setting is observed when a ciphertext, e.g., a firmware image, must be decrypted multiple times using a leakage-resilient scheme like Isap.

4.3.2 Averaging Attacker

An attacker that observes the same processing of the secret state x multiple times is capable of averaging the side-channel leakage l_x to yield a better SNR and thus higher side-channel capacity. In general, averaging $\mathcal{N}$ observations gives the averaged noise covariance matrix

$$\overline{\Sigma}_{\boldsymbol{\nu}} = \frac{1}{\mathcal{N}}\Sigma_{\boldsymbol{\nu}}, \tag{4.10}$$

where $\Sigma_{\boldsymbol{\nu}}$ is the noise covariance matrix valid for a single leakage trace. This means that the noise (co-)variances reduce linearly with the number of averaged traces. Note here that for the univariate case Equation 4.10 simplifies to the well-known relation $\overline{\sigma}_{\nu}^2 = \frac{\sigma_{\nu}^2}{\mathcal{N}}$. Given the noise covariance matrix after averaging $\overline{\Sigma}_{\boldsymbol{\nu}}$, we can now investigate the effect of averaging on the side-channel capacity. Inserting Equation 4.10 into the generic side-channel capacity given in Equation 4.6 yields

$$\mathcal{C} = \frac{1}{2}\log_2 \left|\mathbf{I}_m + \mathcal{N} \cdot \Sigma_{\boldsymbol{\nu}}^{-1}\Sigma_{\mathbf{y}}\right|. \tag{4.11}$$

Note that the SNR term $\mathcal{N} \cdot \Sigma_{\boldsymbol{\nu}}^{-1}\Sigma_{\mathbf{y}}$ is an $m \times m$ matrix and its determinant behaves proportionally to $\mathcal{N}^m$. This means that the side-channel capacity increases stronger with the number of averaged traces the more POIs are used in an attack. This is because each POI can potentially transfer completely independent data as, e.g., for time-serialized and localized EM leakages.

Identical Leakage in Multiple Points. For identical leakage in all POIs, the side-channel capacity behaves differently. Inserting Equation 4.10 into the SIMO channel capacity given in Equation 4.9 yields

$$\mathcal{C} = \frac{1}{2}\log_2 \left(1 + \mathcal{N} \cdot \sigma_z^2 \cdot \mathbf{h}_{gain}^H \Sigma_{\boldsymbol{\nu}}^{-1}\mathbf{h}_{gain}\right). \tag{4.12}$$

It shows that the number of traces $\mathcal{N}$ used for averaging has a linear influence on the SNR and is independent of the number of POIs m.

4.3.3 Expected Minimum Attack Complexity

In the worst case, physical attackers have unbounded complexity. This means that they can measure and average an unlimited number of leakage traces $\mathcal{N} \to \infty$, leading to zero noise and virtually unlimited channel capacity and MI. This can be thought of state differences causing vanishingly small differences in the side-channel signal being distinguishable if the noise is eliminated completely. It thus seems reasonable to set the side-channel capacity in relation with the actual attack complexity, i.e., the number of leakage traces $\mathcal{N}$, to learn a certain amount of bits. This is also the common approach when assessing the security of masked implementations.

It is yet difficult to determine such attack complexity since it is strongly influenced by the implementation's leakage behavior, which is commonly unknown. For example, it is unknown to what extent information and noise in the single points of a leakage trace correlate, and as shown in Section 4.2, these effects strongly influence the side-channel capacity. However, the device-specific multivariate $SNR = \Sigma_{\mathbf{y}} \cdot \Sigma_{\boldsymbol{\nu}}^{-1}$ takes exactly these effects into account and can thus be used to generically express the expected minimum complexity for a side-channel attacker without any concrete leakage assumptions. In particular, we can rewrite the multivariate channel capacity for averaging attackers in Equation 4.11 as follows:

$$\mathcal{C} = \frac{1}{2} \log_2 \mathcal{N}^m \left| \frac{1}{\mathcal{N}} \mathbf{I}_m + \Sigma_{\boldsymbol{\nu}}^{-1} \Sigma_{\mathbf{y}} \right| . \tag{4.13}$$

For a large number of averaged traces $\mathcal{N}$, Equation 4.13 can be further approximated to give the side-channel capacity in dependence of a scalar device SNR.

$$\mathcal{C} \approx \frac{1}{2} \log_2 \left(1 + \mathcal{N}^m \left|\Sigma_{\boldsymbol{\nu}}^{-1} \Sigma_{\mathbf{y}}\right|\right) = \frac{1}{2} \log_2 (1 + \mathcal{N}^m \cdot SNR_m) \tag{4.14}$$

An implementation will in practice give some side-channel $SNR_m = |\Sigma_{\mathbf{y}} \cdot \Sigma_{\boldsymbol{\nu}}^{-1}|$ that is observed in m POIs in the leakage traces. This SNR takes into account all kinds of correlations in both noise and side-channel leakage. For an implementation that is expected to give a certain SNR_m, designers and implementers can thus compute the expected minimum attack complexity in terms of traces to measure and average.

Figure 4.1 gives an overview on the expected side-channel capacity for $m = 1, 5, 10$ POIs given the number of averaged traces. It shows that the side-channel capacity rises quickly with the number of averaged traces for multivariate leakages. In particular, it shows that if SNR_m is not sufficiently low, a state of virtually any size can theoretically be recovered with practical complexity. However, this effect is also limited by the available POIs with sufficiently low signal correlations.

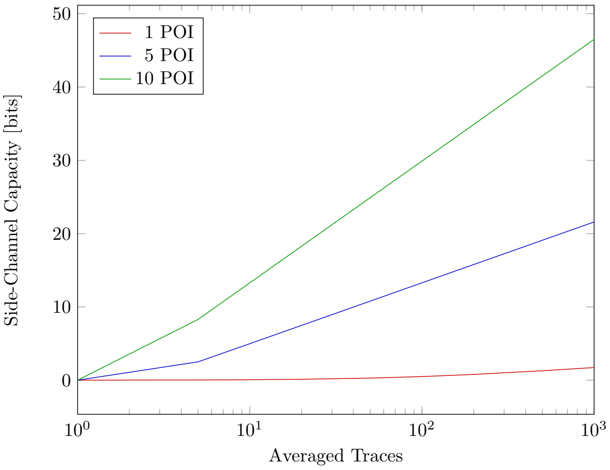


Figure 4.1: Expected side-channel capacity given the number of averaged leakage traces for different numbers of POIs and $SNR_m = 0.01$.

4.3.4 Divide-and-Conquer Attacks

Divide-and-conquer attacks such as localized EM attacks and attacks on time-serialized implementations allow an attacker to learn specific parts of the secret state. In particular, in terms of our model, such divide-and-conquer setting gives multiple, parallel side-channels with independent leakages observed in the POIs of a leakage trace. This increases the total side-channel capacity proportional to the number of independent leakages t, i.e., $\mathcal{C}_{total} \sim t \cdot \mathcal{C}$, when $\mathcal{C}$ is assumed equal for all independent leakages.

Figure 4.2 gives an overview on the expected minimum attack complexity for different numbers t of univariate, independent leakages and sizes of the secret state x. In this example, we computed the expected minimum attack complexity by multiplying the number of independent leakages t with the number of leakage traces to average for each of the t independent leakages. Moreover, the single-trace SNR is set to 0.1 for each of the t independent leakages. Clearly, the minimum attack complexity drops rapidly with the number of independent leakages observed by an attacker. We notice however that splitting an 128-bit state into 16 chunks as, e.g., for the byte-oriented AES S-box, still gives an expected minimum attack complexity of 10^7.

4.4 Experimental Verification and Security Analysis

The previous sections introduced theoretical leakage upper bounds for multivariate side channels with Gaussian noise. In this section, we show that these bounds match the real leakage behavior by evaluating the MI on a hardware

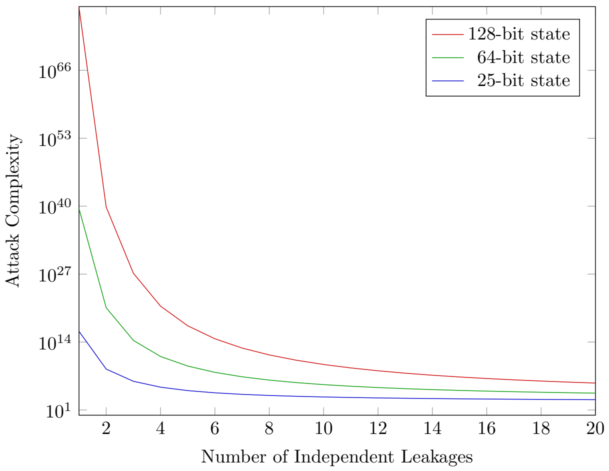


Figure 4.2: Expected minimum attack complexity over the number of independent leakages for different state sizes and $SNR_1 = 0.1$.

implementation of the Keccak[400]-based scheme Isap [Dob+17] on the real system on chip Fulmine. Our experiments further show the security of this implementation of Isap in terms of power analysis attacks.

4.4.1 Evaluation Hardware: Fulmine

At FSE 2017, the sponge-based authenticated encryption scheme Isap [Dob+17] was presented next to our sponge leakage model and re-keying functions from Chapter 3. Isap inherently prevents DPA during both en-/decryption by limiting the number of inputs processed under a single key by one, and uses our sponge leakage model from Chapter 3 to express its capability to cope with side-channel leakage from a single data input. However, in the view of Isap allowing for the multiple decryption of the same ciphertext and tag, it is an open question how much information an attacker can learn when averaging multiple leakage traces.

To verify the soundness of our leakage bounds and to evaluate the side-channel resistance of Isap, we developed and fabricated the multi-core SoC Fulmine, a prototype ASIC in the UMC 65 nm LL 1P8M technology. Fulmine, as shown in Figure 4.3, is based on the PULP platform [Ros+15] including four general purpose processing elements (enhanced OpenRISC cores with DSP extensions [Lam+03; Gau+17]) and two dedicated hardware accelerators: the Hardware Cryptography Engine (hwcrypt) and the Hardware Convolution Engine (hwce). All processing elements share the same 64 kB level-1 Tightly-Coupled Data Memory (TCDM) to support a fast and efficient communication and to avoid single point-to-point channels.

hwcrypt is a flexible, software-programmable hardware accelerator supporting various cryptographic primitive functions such as the Keccak[400] permutation [Ber+09]. Moreover, the accelerator supports high-level encryption schemes

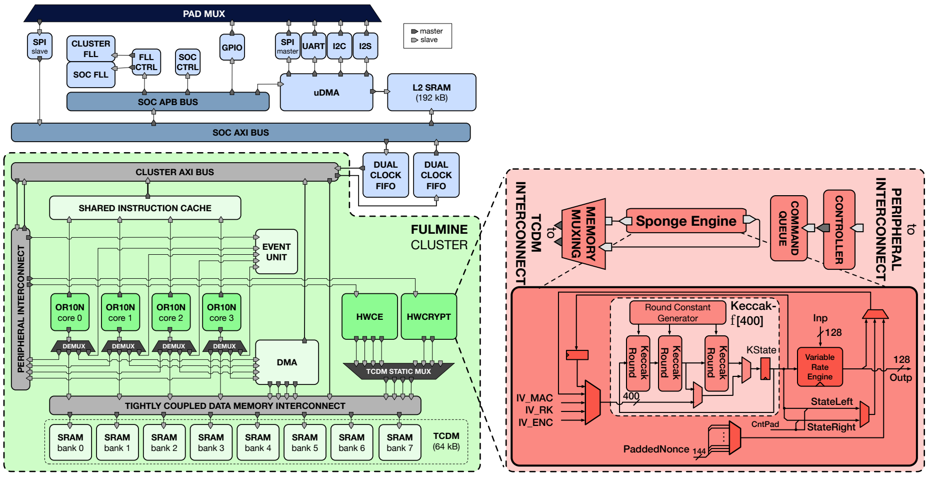
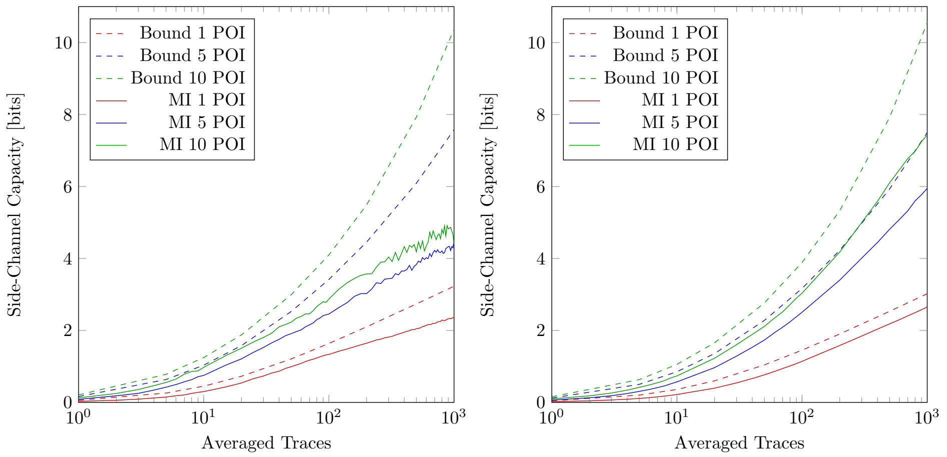


Figure 4.3: FULMINE SoC and HWCRYPT architecture.

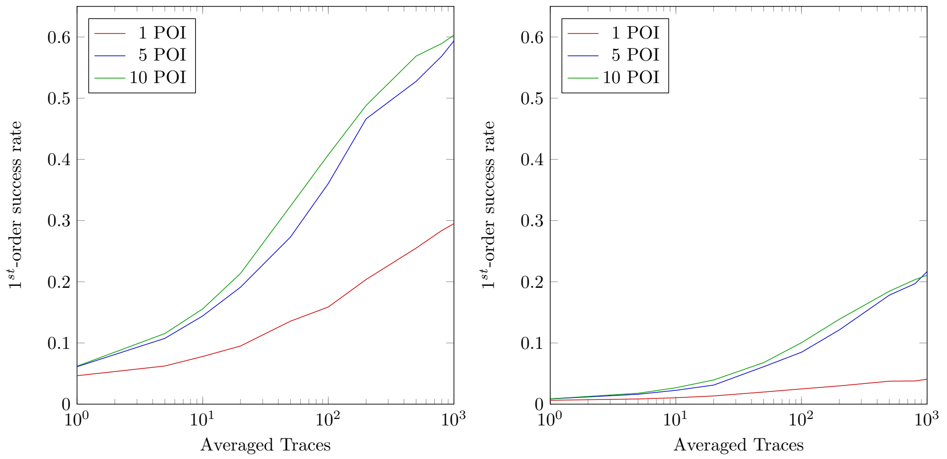
such as ISAP. The accelerator is designed to achieve maximum throughput. To achieve that goal, the Keccak[400] permutation utilizes three fully parallel round instances to maximize the throughput but to also match the length of the critical path of other parts of the accelerator. When using ISAP, HWCRYPT supports a flexible configuration of the rate (from 1 bit to 128 bits in powers of two) and the number of permutation rounds in multiples of three including 20 to flexibly trade-off between throughput and security. HWCRYPT is configured and monitored via a set of status registers. A flexible event and interrupt system indicates other processing elements when an operation has finished.

4.4.2 Soundness of Model and Bounds

To verify the soundness of our model and the bounds in Section 4.2, we analyzed the leakage behavior of the Keccak[400] permutation on FULMINE. For this purpose, we constructed multivariate Gaussian templates for the power consumption of FULMINE for 5- and 8-bit parts of the 400-bit state of Keccak[400]. More concretely, we target the intermediate state *KState* of Keccak[400], depicted in Figure 4.3, such that FULMINE computes three rounds of the permutation before and after the target state to preclude load-time leakages. The remaining state not covered by our templates, i.e., 395 and 392 bits respectively, was held constant. For each class, we used 1400 power measurements in the training phase. The POIs were chosen as the points of highest variance fulfilling a certain minimum distance within the leakage trace and include both register and combinatorial activity. Based on these templates, we computed the side-channel capacity and evaluated both the MI and the 1st-order success rate of classification. The evaluations were done as a function of the number of leakage traces used for signal averaging.



(a) 32 classes. **(b)** 256 classes.



(c) 32 classes. **(d)** 256 classes.

Figure 4.4: Side-channel capacity, mutual information and success rate for the Keccak[400] permutation given the number of averaged traces and different numbers of POIs and number of classes. The remaining state was held constant.

Our evaluation results in Figure 4.4 suggest that the channel model used to compute the side-channel capacity of multivariate leakages is sound. In particular, for both 5-bit and 8-bit templates the MI between leakage and secret state stays within the bounds given by the side-channel capacity. While there is a gap between the MI and the channel capacity, the MI follows the shape of the side-channel bound well. Moreover, the first-order classification rate rises accordingly. However, Figure 4.4a also shows that for higher numbers of averaged traces the MI goes into saturation, and thus the gap between capacity and the learned information gets bigger. In particular, it shows that once the MI converges to the maximum number of bits that could be recovered using the trained template set, i.e., 5 or 8 bits respectively, the increase in learned information for additional numbers of averaged traces gets successively smaller. This indicates that the side-channel information is not distributed to perfectly use the channel.

We further investigated how different channel models suit the actual leakage behavior. In particular, it seems interesting how the side-channel capacity behaves depending on the underlying channel model and relative to the measured mutual information. We therefore compared the MIMO channel model used in the previous evaluation with the SIMO channel model, which assumes identical leakages in the POIs of a leakage trace, e.g., within a clock cycle. For the SIMO channel model, we analyzed two cases: one taking noise correlations into account, and one assuming independent noise. The channel capacities of the different channel models were computed based on the 8-bit templates constructed in the previous evaluation. In particular, for the SIMO model we used the signal variance in each POI, but neglected signal covariances. The results of our evaluations are shown in Figure 4.5. These suggest that the leakages in the different POIs are not identical and thus the MIMO channel model suits the leakage behavior clearly better than the SIMO channel model. Moreover, from the plots using the SIMO model one can observe that there is some noise correlation that lowers the channel capacity.

4.4.3 Estimating Security Bounds for ISAP

In most situations, designers and implementers want to assess the security of a complete cryptographic implementation. This can be done by determining the SNR of the implementation as the SNR gives a bound on the implementation's side-channel leakage. For many implementations, however, it is impossible to determine the SNR exactly. For example, it is impossible to construct templates for the 400-bit ISAP state on FULMINE that would allow to compute the SNR and the leakage bound exactly. On the other hand, the previous sections showed that it is feasible to compute an SNR from a small number of templates that only match a part of a large state. Using this SNR, we can hence try to estimate the side-channel security of implementations such as ISAP on FULMINE. Since the templates used for computing this SNR, compared to templates considering the full state, may reflect the activity of smaller hardware regions, we however recommend adding a suitable security margin.

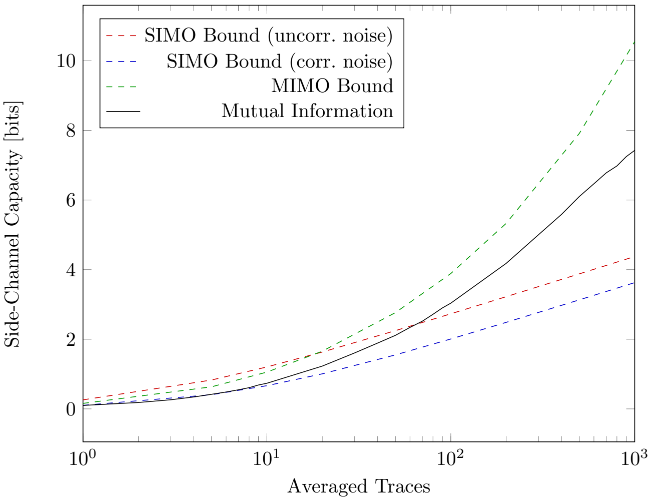


Figure 4.5: Mutual information of Keccak[400] on FULMINE and side-channel capacity of different channel models (256 classes, 10 POIs).

In the following, we estimate the security of ISAP on our chip FULMINE using our previous experiments on the Keccak[400] permutation. According to our experimental results from Figure 4.4, the channel capacity and the SNR are practically the same for the 5- and the 8-bit templates. We hence assume the measured noise not to massively change if the same measurement setup was used to construct 400-bit templates. Similarly, we do not expect the range of the side-channel signal to rise by orders of magnitude using the same setup, especially since the diffusion of three rounds of Keccak[400] on FULMINE already cause large parts of the logic to become active within the profiled clock cycle.

For our security estimate of ISAP on FULMINE, we hence scale the SNR with a factor $\gamma = 100$ to have a generous security margin. Note hereby that the side-channel leakage from a single power measurement of FULMINE is very low and the channel capacity hardly rises for a channel SNR that is 100 times higher. Using the SNR_m of the m-variate leakage from the 8-bit templates, we approximate the minimum number of traces needed to learn the state of size s:

$$\mathcal{N} = \left(\frac{2^{2s} - 1}{\gamma \cdot SNR_m} \right)^{1/m} . \tag{4.15}$$

[Dob+17] states concrete leakage bounds for the authenticated encryption scheme ISAP and our re-keying function ISAPRK2 from Chapter 3 to still provide 128-bit security. We thus evaluated Equation 4.15 on FULMINE for three different state sizes: the full state of Keccak[400], the leakage bound for our ISAPRK2 re-keying function (271 bits), and the leakage bound for the ISAP encryption itself (128 bits). The results in Figure 4.6 indicate that the minimum attack complexity in terms of measurement traces is impracticable for less than 20 POIs

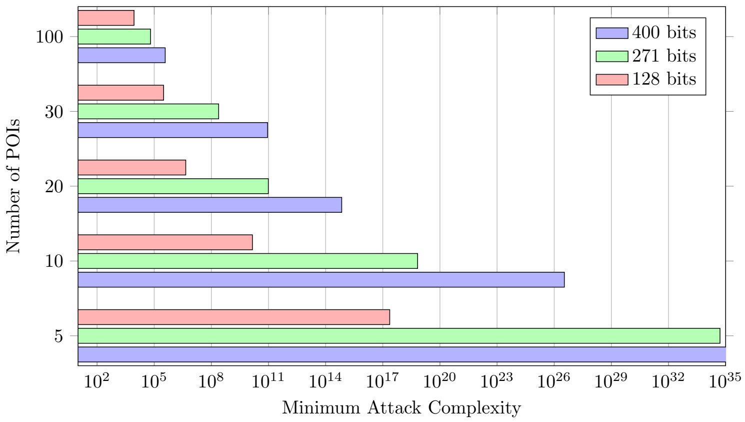


Figure 4.6: Minimum attack complexity as the number of measurements needed to average to recover (parts) of the Isap state from Fulmine. As a security margin we set $\gamma = 100$.

and all mentioned state sizes. However, for higher numbers of POIs the minimum attack complexities tend towards practically feasible. Namely, when using 100 POIs, 10 000 measurements can be enough to learn 128 bits of the state, and 500 000 measurements are the minimum to recover the full state.

However, using that many POIs often hampers template building or leads to overfitting effects reducing the classification rate. Besides, side-channel leakage is not distributed such as to perfectly use the channel. This becomes visible in the gap between channel capacity and MI in Figure 4.4. While this might allow an attacker to recover a few states more easily, in consideration of all possible states the attack complexity yet stays above the bounds in Figure 4.6. Namely, for non-ideal distributions of the leakage, an attacker will, in general, require even more measurements to learn the specified amount of information.

From a practical perspective, conducting such powerful attack would require an attacker to successfully build templates on the respective state. In many cases, this is however not possible, e.g., when the attacker does not have control over the state on a suitable device. Even further, the complexity to build, measure, and evaluate such large set of templates is clearly impractical. In this respect, the implementation of Isap on Fulmine can for the used measurement setup be considered secure against power analysis attacks also above the bounds in Figure 4.6.

4.5 Conclusion

In this chapter, we presented a novel approach to determine leakage upper bounds for side channels of cryptographic implementations under a single data input. Without any further leakage assumptions we showed that the channel capacity of transmission channels with multiple in- and outputs gives the natural upper bound for information leakage in multivariate side channels with Gaussian noise.

We then considered the case where attackers are capable of performing multiple measurements of the same execution in order to improve their SNR. We showed that the gain in the SNR of multivariate leakages resulting from signal averaging is exponential in the number of POIs. This observation gives a tool for attackers to learn about the feasibility of an attack and for implementors to assess the minimum attack complexity of state recovery in leakage-resilient schemes allowing for multiple decryptions like ISAP. We verified the soundness of our model and our bounds using the ASIC FULMINE implementing ISAP and the Keccak[400] permutation. Finally, we gave lower bounds on the complexity for recovering the ISAP state using power analysis. The results provide evidence that recovery of the ISAP state on FULMINE is practically infeasible with power analysis and the used measurement setup.

5

Side-Channel Attacks on Leakage-Resilient Encryption

Cryptographic schemes based on frequent re-keying such as leakage-resilient encryption inherently prevent Differential Power Analysis (DPA) on the secret key by limiting the amount of data being processed under one key. Yet, these schemes do not make any statements with respect to other confidential data. In particular, the actual goal of an encryption scheme is to ensure plaintext confidentiality. The fact that leakage-resilient encryption guarantees bounded leakage of the key material does not imply bounded leakage of the plaintexts. For this reason, together with Mario Werner and Stefan Mangard, we investigated in [UWM17c] the effect of re-keying and leakage-resilient encryption on the plaintext's side-channel security. The idea to this publication originated from discussions between Christoph Dobraunig, Stefan Mangard, Florian Mendel, Mario Werner, and me. In the publication, I was the main author contributing attacks, experiments with re-keyed stream ciphers, and most of the text, while Mario Werner contributed the text and the experiments in terms of re-keyed block ciphers. In this chapter, we use text and results from [UWM17c] and make the following detailed contributions.

Contribution. We show that frequent re-keying as it occurs within leakage-resilient encryption is vulnerable to plaintext recovery using side-channel attacks. More concretely, we show that encrypting a constant plaintext multiple times with different keys facilitates DPA to recover the constant plaintext. In this respect, leakage-resilient stream ciphers such as in [Pie09; PSV15; Sta+10] leak a constant plaintext through a plain, first-order DPA and, moreover, a second-order, template-based DPA can be utilized to learn a constant plaintext that is the input of a block cipher that is protected by re-keying such as in [Med+11; Med+10; TS15]. We verified both attacks on a Field Programmable Gate Array (FPGA) and a microcontroller to emphasize their practicality.

However, we stress that our attacks are not limited to side-channel countermeasures such as leakage-resilient encryption, but are also relevant for any other scenario where the same data is encrypted using different keys. In particular, we show that the presented attacks are applicable to several scenarios in practice. For example, the encryption of Random Access Memory (RAM) such as in Intel SGX [Gue16] is often initialized using a random key, resulting in the same data being encrypted using a different key on every startup. Another prominent example are messages that are sent to multiple users that each use a different key.

We emphasize that our plaintext recovery attacks are a realistic and serious threat. In particular, whenever long-term keys are encrypted, for example, when they are loaded into an encrypted memory, plaintext recovery implies the leakage of sensitive key material. As a consequence, cryptographic implementations with dedicated DPA countermeasures should be considered for all these settings that encrypt the same data multiple times using different keys.

Outline. This chapter is organized as follows. Section 5.1 presents side-channel plaintext-recovery attacks and Section 5.2 discusses their applications. Section 5.3 elaborates on the attacks' practical verification and Section 5.4 concludes this chapter.

5.1 Side-Channel Plaintext-Recovery Attack

Chapter 2 illustrated how re-keying based encryption schemes effectively prevent DPA attacks on the secret key. However, as it has not been considered so far, this section analyzes frequent re-keying in terms of plaintext confidentiality and shows that frequent re-keying is vulnerable to plaintext recovery using DPA. In particular, encrypting constant data with different keys allows to attack leakage-resilient stream ciphers using a first-order DPA and re-keyed block cipher encryption using a second-order template attack.

5.1.1 Stream Cipher Mode

For illustration of the first-order DPA to recover constant plaintexts when they are encrypted using a leakage-resilient stream cipher such as in Figure 2.2, we consider the encryption of a single, constant plaintext block P_i. The choice of a fresh nonce N upon every encryption results in different key streams and thus in the plaintext P_i being encrypted using different pads Y_i, Y_i', Y_i''. As a result, an attacker will, for the same plaintext P_i, observe different ciphertexts C_i, C_i', C_i'' and the respective power consumptions of the implementation. This observation facilitates DPA on the constant plaintext. Namely, the varying pad

Y_i allows to distinguish correct from wrong guesses of the plaintext P_i. In the DPA, the attacker can therefore model the power consumption of the stream cipher implementation as $HW(Y_i) = HW(C_i \oplus P_i)$ for all observed ciphertexts C_i, C_i', C_i'' and for all guesses of P_i, where HW denotes the Hamming Weight. Applying an appropriate statistical distinguisher, e.g., correlation, to the power model and the observed power consumption then reveals the correct plaintext P_i.

Note that this DPA targets the linear XOR operation. We also stress that this kind of DPA is not limited to leakage-resilient stream ciphers as in Figure 2.2, but applicable to stream ciphers in general. Namely, for cryptographic security, the pad of a stream cipher (and thus its initial value) must not repeat over different plaintexts. Therefore, producing different key streams, i.e., re-keying, is mandatory for any stream cipher implementation. The discussed attack scenario will thus appear every time a plaintext is encrypted with a stream cipher more than once.

5.1.2 Block Cipher Mode

The leakage-resilient block cipher encryption mode in Figure 2.3 is not subject to the previously discussed first-order DPA attack. This is due to the fact that a full block cipher, i.e., the bottom ciphers in Figure 2.3, separates the known ciphertext from the constant plaintext, and therefore, without knowledge of the random key, we cannot verify any plaintext hypothesis. However, a second-order template attack targeting the first round of the data-encryption block cipher can be mounted on the construction instead.

The idea of the attack against the block cipher used for data encryption is similar to the idea behind Unknown Plaintext Template Attacks (UPTAs) [HTM09]. As explained in Chapter 2, UPTAs recover the secret key of a block cipher without having access to both the plain- and the ciphertext by using templates to extract the required information from the side-channel leakage of multiple block cipher invocations and combining it in a DPA-like manner. Re-keying effectively prevents key recovery via these UPTAs since the key for the block cipher is constantly changed. However, an UPTA-like plaintext-recovery attack can still be mounted on the block cipher given that the plaintext is constant. In this case, the role of the key and the plaintext are simply swapped.

In more detail, an UPTA-like plaintext recovery attack on the initial round of a block cipher consists of two phases. In *phase one*, two sets of templates are trained using labeled training traces. The first template set provides combined information about both the key and the plaintext. It is therefore typically trained on the intermediate value after the first nonlinear function. The second template set provides information about the secret key. None of the two template sets is expected to determine the respective values unambiguously. However, the combination of the information provided by both template sets should be significant.

Attacking, for example, the first round of the AES block cipher with such an UPTA-like plaintext recovery attack results in the setup shown in Figure 5.1. The first template set has to be trained on the S-box output $\mathcal{S}(P_0 \oplus \overline{K}_0)$ and

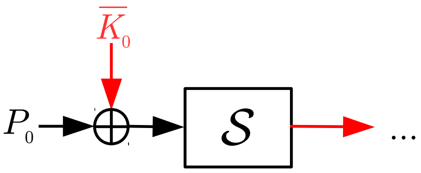


Figure 5.1: UPTA-like plaintext recovery attack on one plaintext byte in the first AES round. Two template sets, one on the whitening key $\overline{K}_0$, and one on the S-box output $\mathcal{S}(P_0 \oplus \overline{K}_0)$, have to be trained.

provides information about the plaintext and the whitening key. The second template set is trained on the whitening key $\overline{K}_0$ itself and is supposed to learn information about the actual key value.

In *phase two*, the templates are matched with all target traces. For every trace, matching the templates from the first set provides attackers with a matrix of probabilities for all possible key and plaintext hypotheses. However, these matrices can not be combined across different target traces. Therefore, the templates from the second set are used to weight the key hypotheses within the aforementioned matrices. As a result, the attacker is able to reduce the matrices with plaintext/key probabilities to vectors of plaintext probabilities. These vectors can then be easily combined across all target traces.

5.2 Implications and Applicability

The principle of the presented attacks on leakage-resilient encryption seems quite natural when taking into consideration that more information about certain data is leaked the more often it is processed in different ways. Namely, the vulnerability exploited in these schemes arises from mixing a varying component, the key, with a fixed component, the plaintext, in the process of re-keying. While such attacks that exploit multiple encryptions of fixed data using different keys seem implausible at first glance, there are indeed several practical use cases where such settings occur. Quite noteworthy, these settings are not limited to re-keying as a DPA countermeasure.

5.2.1 Communication

One example in practice are communication protocols. In SSL, for example, the communicating parties in each session agree on a new session key. As a result, sending the same data via multiple SSL sessions leads to the encryption of this data using different session keys and therefore enables side-channel plaintext-recovery attacks. One practical situation where this happens is an (embedded) web sever that receives multiple download requests for a certain file that is thus leaked using the presented attacks. Yet, for the cipher modes nowadays employed in such communication protocols it seems easier to use DPA to directly recover the key. However, using re-keying or leakage-resilient schemes as a countermeasure still allows to recover constant plaintexts.

Note that key wrapping does not solve the problem of plaintext recovery attacks in multi-party communication settings. Even though key wrapping guarantees that the plaintext itself is encrypted only once, the used data encryption key still needs to be encrypted for all communicating parties with their respective long-term keys. Therefore, side-channel plaintext-recovery attacks on the key wrapping procedure, where the data encryption key is the plaintext input, are still possible.

5.2.2 Memory Encryption

Re-keying nowadays also occurs in the setting of memory encryption. For example, in implementations of transparent RAM encryption, like recently introduced by Intel with SGX [Gue16], it is common to choose a random RAM encryption key on every system reboot. Therefore, every time (confidential) data is loaded into the memory after startup, it is re-encrypted using a different key. This facilitates the attacks presented in this chapter to learn the plaintext data. This can be particularly critical if there are long-term keys being loaded into the RAM.

Similar to the communication example, the application of re-keying and leakage-resilient encryption does not close this vulnerability. Namely, memory is typically encrypted using a block-wise granularity. Therefore, updating small amounts of data, e.g., a single byte, will cause a read-modify-write operation on the respective block to take place. In order for re-keying to work properly, the whole block must then be re-encrypted using a freshly chosen key, triggering the re-encryption of the unchanged data in the block. As a result, the presented attacks will not just work across several system boots, but also within a single session of the system.

We stress that a similar effect also occurs for RAM encryption using the counter mode such as in [Gue16]. Hereby, the nonce input to the block cipher is composed of the block address and the respective block's counter. Therefore, whenever some data block is copied or written back to the memory, either the address or the respective block's counter changes, leading to a different pad, and thus re-encryption of the same data, again allowing for side-channel plaintext-recovery attacks.

5.2.3 Remarks and Countermeasures

Re-keying successfully prevents key recovery through DPA attacks without the need for DPA countermeasures in cryptographic implementations. Even more, leakage-resilient encryption nicely allows to give proofs of security in the presence of side-channel adversaries. However, the presented attacks also make clear that the re-keying approach may facilitate new attack scenarios that have been left unconsidered so far. This work gives an example of such a scenario that designers and implementers need to be aware of.

The best countermeasure to the presented attack would be to avoid the re-encryption of constant data at all. However, multiple encryption of the same data using different keys frequently appears in practice. Therefore, contradicting the original intention of re-keying, adding countermeasures to the cryptographic implementation is one possible solution for these use cases. While these countermeasures cannot prevent the attack scenario completely, mechanisms like masking [PR13] can at least increase the attack order to render the attack complexity [Cha+99] for plaintext recovery too high.

5.3 Practical Evaluation

In this section, we practically verify the attacks described in Section 5.1 and present our results on both, an FPGA and a microcontroller.

5.3.1 Stream Cipher Mode

We implemented the stream cipher depicted in Figure 2.2 on the Sakura-G [Sat14] side-channel evaluation board featuring a Xilinx Spartan 6 LX75 FPGA. The implementation uses a single AES-128 core as the encryption primitive E that computes one AES round per cycle and that is shared between the key update procedure and the pad computation. Therefore, the implementation natively processes plaintext blocks P_i of 128-bit size. Once the pad is computed and the input data block is ready, the pad is applied fully in parallel to the input. The implementation communicates with the host PC via the USB interface that emulates a virtual COM device. To ease the attack setup, the implementation also provides a dedicated signal to trigger the power measurements.

The implementation was operated at 24 MHz. For the power measurements, we sampled the signal at measurement point J3 using a LeCroy WP725Zi oscilloscope at 250 MS. The measurement point J3 gives an amplified signal of the voltage drop over a shunt resistor on the VCC line. We then performed the attack as described in Section 5.1.1 by using plaintext byte hypotheses and correlation as the statistical distinguisher. The attack could successfully identify all plaintext bytes in less than 10 000 traces. For example, Figure 5.2 shows the results for a single plaintext byte that could be recovered using 3 000 traces already. However, to improve the results both measurement setup and trace processing are possible starting points.

5.3.2 Block Cipher Mode

The practical evaluation of the UPTA-like plaintext-recovery attack on the block cipher mode was performed using a ChipWhisperer-Lite [New16] side-channel evaluation board, sampling at 29.5 MHz. As the target board, an Atmel XMEGA128D4-U microcontroller, clocked at 7.4 MHz, was used. As in the original UPTA paper, the Hamming Weight (HW) leakage model is well suited

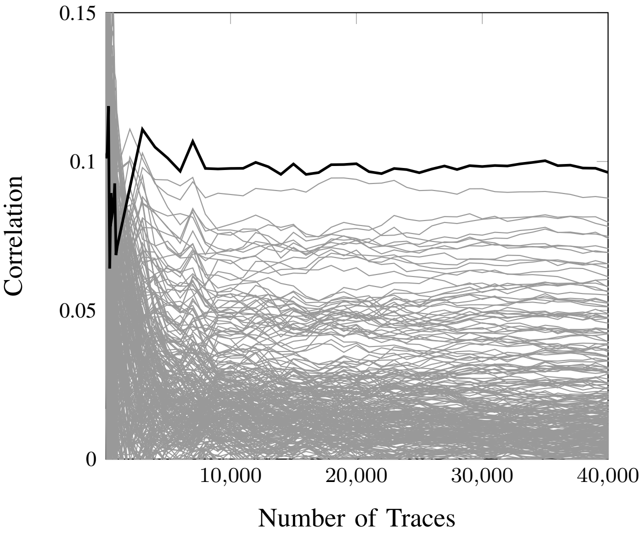


Figure 5.2: Single plaintext byte result of a 1st-order DPA on the leakage-resilient stream cipher in Figure 2.2. The correct plaintext byte is highlighted in bold.

for this processor. On the software side, a byte-oriented C implementation of AES-128 from the AVR-Crypto-Lib [AVR16] was chosen.

The actual attack was performed following a divide-and-conquer approach, where every plaintext byte is attacked in isolation. However, only a single template trace set as well as a single target trace set was used to attack all bytes.

In the first phase of the attack, byte-wise template sets to classify the HWs of both the key and the S-box output were trained using 30 000 power traces recorded during random encryptions with known plaintext and key. For each template set, the POIs were chosen by selecting the samples of a trace with the highest variance between the means of all HW classes of the respective template set. The variance for the key and the S-box template set is visualized for a single state byte in Figure 5.3. The POIs that contribute the most information to the desired templates can be clearly seen in the 3 000-sample long traces. In total, 50 POIs were selected for each template.

In the second phase, every byte's key and S-box template was matched with every target trace. This results in probabilities for the modeled HWs at the template positions. For each trace, the probabilities of a single byte's S-box template were then used to compute the probabilities for all potential plaintext and key values for the respective byte. Afterwards, the key dependency was removed by weighting the probabilities based on the result of the key template matching. As a result, probabilities for the different plaintext values remain which can be combined for all traces.

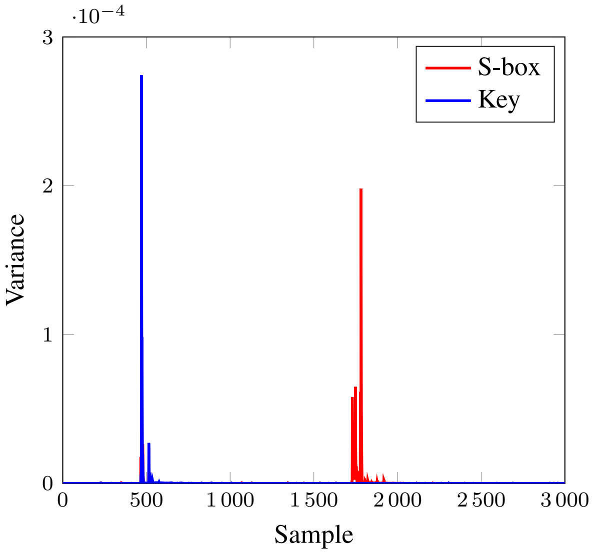


Figure 5.3: Point of Interest (POI) detection for the S-box and the key template. The main key leakage is located at sample 470. The S-box output leaks the most at sample 1782.

An exemplary development of these plaintext value probabilities is visualized for one byte of the plaintext in Figure 5.4. In this figure, roughly 2 000 target traces are sufficient to uniquely determine the correct plaintext value. Across all plaintext bytes, most of the plaintext values could be determined with less than 5 000 traces.

Note that the presented attack is only supposed to prove that UPTA-like plaintext-recovery attacks are indeed possible and practical. The required number of traces should hence not be taken as reference for the expected attack complexity. Optimizing the attack would easily be possible using a more sophisticated measurement setup or by exploiting the leakage from additional samples within the traces.

5.4 Conclusion

In this chapter, we investigated the side-channel security of frequent re-keying and leakage-resilient encryption. While such schemes have several advantages such as inherently preventing DPA on secret key material and giving provable leakage bounds without the need for a protected cryptographic implementation, we showed that schemes based on frequent re-keying do not sufficiently protect confidential plaintexts from DPA. In particular, whenever confidential data, e.g., a long-term key, is (re-)encrypted multiple times using different keys, the cryptographic device generates additional leakage on this data that an attacker can exploit. As a result, constant plaintexts encrypted using leakage-resilient

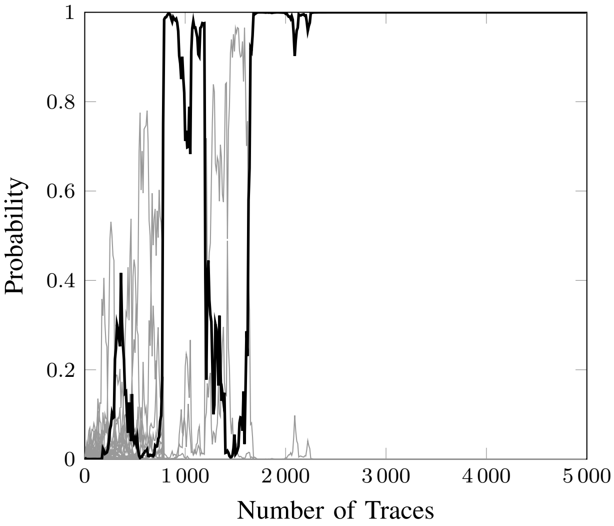


Figure 5.4: Plaintext probabilities of an UPTA-like attack on one plaintext byte. The correct plaintext value is highlighted in bold black.

stream ciphers are recovered using a standard, first-order DPA, and a template-based, second-order DPA can reveal plaintexts that are encrypted multiple times with a block cipher using different keys.

The consideration of plaintext confidentiality in the presence of side-channel adversaries thus reveals a weakness of current re-keying based schemes that designers and implementers need to be aware of. This issue is emphasized by several sensible applications where care has to be taken as these inherently perform re-encryption of constant plaintexts, e.g., multi-party communication and RAM encryption. As a consequence, cryptographic implementations with DPA countermeasures, such as masking [PR13], should be considered to avoid the leakage of plaintexts in these uses cases inherently encrypting data using multiple keys.

Part II

Side-Channel Security for Memory Encryption

Apart from side-channel attacks as discussed in the previous chapters, physical access to Internet-of-Things (IoT) devices allows attackers to read confidential data from or tamper with data in external memory as well. While there are memory encryption and authentication techniques to prevent these kind of attacks, side-channel attacks, on the other hand, have not been considered in this context before. In this part, we extend the focus of memory encryption and authentication to side-channel attacks and make the following main contributions:

- We show that all current memory encryption and authentication schemes are vulnerable to Differential Power Analysis (DPA) and Differential Fault Analysis (DFA) attacks and practically show the feasibility of DPA on state-of-the-art systems by revealing the key from ext4 disk encryption on a Zynq-7010 System on Chip (SoC).
- We present the first memory encryption and authentication scheme that is secure against side-channel attacks and suitable for all kinds of memory. We implement and evaluate our scheme to show that it is as efficient as state-of-the-art memory authentication techniques without side-channel protection.

6

Side-Channel Attacks on Memory Encryption

Memory and disk encryption is a common measure to protect sensitive information in memory from adversaries with physical access and is being implemented in an increasing number of real-world applications, such as `dm-crypt` [DMC15], iOS [App15], Mac OS X [App12], Android [Goo15], Windows [Fer06], and ext4 [MM; Lin15]. However, physical access also comes with the risk of physical attacks such as side-channel and active fault attacks. Together with Stefan Mangard, we therefore analyzed the security of current memory encryption schemes and their implementation within `dm-crypt`, Android 5.0, Mac OS X, and ext4 in terms of these physical attacks. The respective publication in [UM16] was awarded the best paper at COSADE 2016. The idea to this publication originated from Stefan Mangard and me, whereas I was the main author contributing attacks, experiments, and text. In this chapter, we use text and results from this publication [UM16] and make the following scientific contributions.

Contribution. As our first contribution, a detailed analysis shows that Differential Power Analysis (DPA) and Differential Fault Analysis (DFA) breaks all contemporary memory and disk encryption schemes used in practice. Most prominently, it presents tricks to be applied to DPA and DFA in order to obtain the keys from the tweakable ciphers Xor-Encrypt-Xor (XEX) and XEX-based Tweaked codebook mode with ciphertext Stealing (XTS). Supporting the analysis results, our second contribution exploits the Electromagnetic Emanation (EM) side channel of a Zynq-7010 System on Chip (SoC) in a practical attack on the recently introduced ext4 disk encryption mechanism that completely discloses the confidential disk content. We thus conclude that securing memory against physical adversaries requires protected implementations, e.g., [Mor+11b; Bil+14; ISW03], to be used with contemporary memory encryption schemes, which however increases the cost of memory encryption by at least a factor of four, or new schemes that are designed to be resilient to physical attacks.

Outline. This chapter is organized as follows. Section 6.1 introduces memory encryption and gives an overview on common state-of-the-art implementations. The memory encryption schemes are analyzed with respect to both DPA and DFA in Section 6.2. The practical feasibility of such attacks is evaluated in Section 6.3, and Section 6.4 concludes the chapter.

6.1 Memory Encryption

Memory encryption deals with the encryption of data contained in memory such as Random Access Memory (RAM), memory cards and hard disks. However, in practice different variants and notations are being used for memory encryption. This section therefore defines memory encryption and gives an overview on common memory encryption schemes and implementations.

6.1.1 Definition

The encryption of memory is usually performed using dedicated memory encryption schemes as these schemes have to fulfill several requirements: (1) ensure random access to all memory blocks, (2) provide sufficiently fast bulk encryption, and (3) the only information an adversary can derive from the encrypted memory is whether a memory block has changed or not.

Definition 1. *A memory encryption scheme is an encryption scheme* $ENC\colon \mathcal{K} \times \mathcal{A} \times \{0,1\}^{s_b} \to \{0,1\}^{s_b}$*, which*

1. *uses a key* K *from key space* $\mathcal{K}$*, and*
2. *splits the memory of size* s_{memory} *into* $n_b = \lceil \frac{s_{memory}}{s_b} \rceil$ s_b*-bit memory blocks,*
3. *identifies each of the memory blocks by their address in address space* $\mathcal{A}$*, and*
4. *provides address-dependent en-/decryption for each of these memory blocks.*

Definition 1 considers the encryption of a flat memory space and requires the encryption process to incorporate address information. The address information allows memory encryption schemes to fulfill requirement (3) as for this reason each memory block is encrypted differently. Otherwise, it would be easily recognizable if certain data is contained in different memory locations and valid (but encrypted) data could simply be copied to different addresses (*splicing attack* [Elb+09]). The requirements (1) and (2) are typically satisfied by splitting the memory space into blocks using two different granularities: the memory is divided into larger sectors (or pages) and each sector (or page) is divided into encryption blocks.

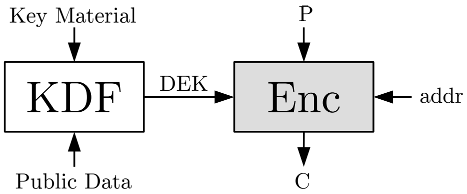


Figure 6.1: Generic model of memory encryption.

The encryption mode then ensures fast bulk encryption within each sector and random access on sector level.

6.1.2 Memory Encryption in Practice

In practice, memory encryption is often named disk encryption referring to the type of memory used. There are two variants of disk encryption: (1) *block device* or *full disk encryption*, and (2) *file-level disk encryption*. While *full disk encryption* performs encryption directly on the raw memory space of a whole disk, block device, or partition, i.e., beneath a file system, *file-level disk encryption* performs encryption on file level on top of or within a file system. Both variants use the same sort of memory encryption schemes, but apply them to different portions of the memory. Throughout this chapter, the term memory encryption thus denotes any of these variants.

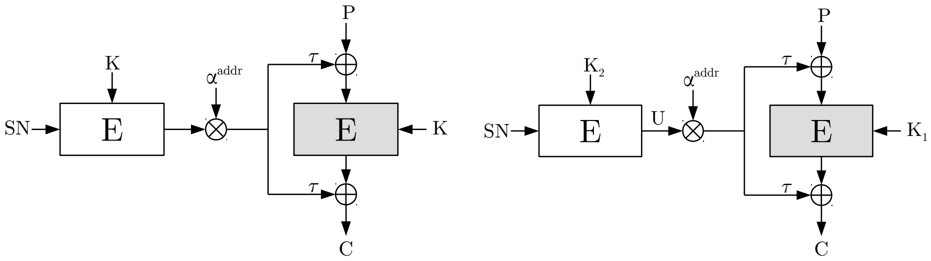
Another aspect of practical implementations of memory encryption is that they usually employ a Key Derivation Function (KDF) to derive the Data Encryption Key (DEK) to be used within the memory encryption scheme from, e.g., a user password and public nonces. The combination of such a KDF and a memory encryption scheme leads to the generic model of memory encryption in Figure 6.1. The following will use this model to first describe typical schemes for both the KDF and the encryption part, and will then show how these are used in several practical implementations.

Key Derivation Functions

To derive a key from a user password or a PIN, password hashing functions such as PBKDF2 [Kal00] or scrypt [Per09] are typically used. This password-derived key is then mostly used as a Key Encryption Key (KEK) to decrypt the actual master key MK of the memory using an ordinary block cipher. Depending on the concrete setup, such master key MK is directly used as the DEK for the memory encryption scheme or is used to further derive or decrypt keys, e.g., DEKs for the encryption of single files in file-level disk encryption.

Encryption Schemes

Common implementations exclusively deal with the encryption of external storage, e.g., hard disks. These implementations, e.g., in `dm-crypt`, mainly utilize the



(a) XEX mode. **(b)** XTS mode.

Figure 6.2: Tweakable ciphers for disk encryption.

modes XEX [Rog04], XTS [IEE08b], and Cipher Block Chaining (CBC) with Encrypted Salt-Sector IV (ESSIV) [Fru05]. The tweakable block ciphers XEX and XTS are shown in Figure 6.2. Both encryption modes apply a tweak τ to the block cipher E that results from a binary-field multiplication of the encrypted sector number with the memory block address. While XEX uses only one key, XTS uses two different keys for the two instances of the block cipher. The CBC mode with ESSIV is depicted in Figure 6.3. ESSIV ensures a secret Initial Vector (IV) and thus prevents watermarking attacks [Saa04]. It computes the IV by encrypting the sector number with the hashed key (i.e., salt).

Differently, research on the design and construction of secure systems further considered the encryption of the main memory. Primarily variants of the counter mode encryption were proposed such as in Figure 6.4 [Suh+03b; Rog+07]. The pad is the encryption of a block-specific seed that comprises an IV, the memory block address, and a timestamp (or counter). It is mostly favored due to the little latency it introduces on the path to the memory.

6.1.3 State-of-the-Art Implementations

The following presents common implementations within `dm-crypt`, Android, Mac OS X, and ext4, and shows that the memory encryption schemes presented before have high prevalence throughout all of these implementations.

dm-crypt. `dm-crypt` [DMC15] is a disk encryption utility that provides transparent encryption of arbitrary block devices within Linux ≥ 2.6, i.e., block device encryption. `dm-crypt` can be configured to use one of several available encryption modes, i.a., CBC-ESSIV and XTS (default), using different block ciphers, e.g., AES-128 [DR02]. The utility requires the user to supply the block device DEK when mounting the block device. For more convenient usage, however, Linux Unified Key Setup (LUKS) [Fru11] can be used. LUKS adds a meta-data header to the block device that stores the encrypted DEK. The respective KEK is derived from a user password using PBKDF2.

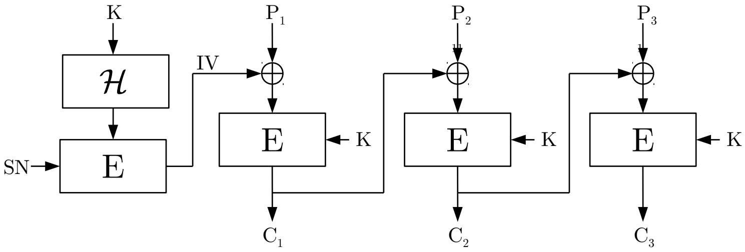


Figure 6.3: Disk encryption via CBC and ESSIV.

Mac OS X. Mac OS X from version 10.7 (Lion) onwards provides block device encryption using the tool `FileVault 2` [App12; CGM12]. Mac OS X encrypts block devices using XTS and AES-128 with separate DEKs that are chosen randomly upon setup of each encrypted block device. For key storage, Mac OS X uses a three-tier hierarchy of DEKs, KEKs and Derived Key Encryption Keys (DKEKs). The DEK is encrypted using a randomly chosen KEK that is encrypted using at least one DKEK. DKEKs can, e.g., be derived from a password or be the public key of a corporate certificate. Both the DEK and the KEK are stored encrypted in a meta-data block on the block device.

Android. Android is equipped with full disk encryption for devices such as flash memory. In Android 5.0, encryption of block devices is based on `dm-crypt` that is configured to use AES-128 and CBC-ESSIV [Goo15]. Its DEK is sized 128 bits by default and stored encrypted on the block device. The respective KEK is derived from a user password and a hardware-bound key using scrypt and a signing procedure within a Trusted Execution Environment (TEE).

Ext4. Since Linux 4.1, the ext4 file system offers file-level disk encryption [MM; Lin15]. It allows to set up encryption for a specific folder that is assigned a master key derived from a user passphrase and a salt using PBKDF2. While ext4 encrypts file content and names, meta data and file system structure is available in plaintext. Each file uses an individual DEK that is derived from the master key MK by encrypting MK with AES-128 in Electronic Code Book (ECB) mode and using a file nonce N_f as the key, i.e., $DEK_f = E(N_f; MK)$. The respective nonce N_f is stored in the file's meta-data section. The file DEK is used to encrypt the file contents using XTS and AES-128.

6.2 Physical Attacks on Memory Encryption

Physical access as the motivation for memory encryption and the prevalence of the memory encryption schemes from Section 6.1 necessitate their analysis with respect to physical attacks such as side-channel and fault attacks. The following analysis of memory encryption schemes w.r.t. physical attacks shows that both DPA [KJJ99] and DFA [BS97] attacks are easily capable of breaking

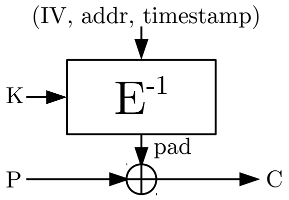


Figure 6.4: Counter mode memory encryption.

all the schemes presented, i.e., they reveal the DEK that allows to decrypt all memory content. Most remarkably, it demonstrates how to obtain the AES-128 keys in the tweakable block ciphers XEX and XTS with practical complexity.

6.2.1 Differential Power Analysis

As introduced in Chapter 2, DPA and its variants, e.g., Correlation Power Analysis (CPA) [BCO04], are attack techniques that allow the recovery of an encryption key based on side-channel leakage collected during multiple en-/decryptions using this key. In the following, we detail how DPA techniques can be applied to the memory encryption schemes described in Section 6.1.

XEX Mode

The tweak τ makes sure that the block cipher behaves differently for each memory address. In spite of this, DPA-style attacks are applicable with little modifications. Therefore, the adversary focuses on one particular memory block, i.e., fixed sector and fixed memory address. For this memory block, the adversary observes ciphertexts and power traces of several encryption processes. The captured power traces are then used twice to attack different rounds of the block cipher shaded gray in Figure 6.2a, as the following illustrates for AES-128:

1. From an attacker's point of view, the last round key rk_{10} is blinded with the tweak τ. However, for a fixed sector and memory address, the tweak τ is constant. A DPA that targets the input of the last round's S-box will thus reveal the last round key XOR-ed with the tweak, i.e., $rk_{10} \oplus \tau$.
2. Knowledge of $rk_{10} \oplus \tau$ is sufficient to target the input of the second-last round's S-box in a second DPA. It reveals the second-last round key rk_9, which can be used to compute the key K.

Two consecutive DPAs on the same set of traces allow to gain knowledge of the key K. The DPAs disclose the information contained in all memory blocks across all sectors, even though only one particular block in one specific sector is actually attacked.

Note that besides standard DPA, also Unknown Plaintext Template Attacks (UPTAs) [HTM09] are applicable to directly obtain rk_{10}. However, such attacks require a preceding profiling step to create suitable templates. On the other hand, if the adversary additionally has knowledge of the accessed sector, e.g., from the observation of memory addresses on the bus, the attack generally becomes easier. In this case, the encryption of the sector number within the tweak computation can be attacked to immediately learn K from power traces of memory accesses to different sectors. However, depending on the practical circumstances, either of those attacks is more suitable, e.g., the adversary may want to avoid raising suspicion by not probing the memory bus.

XTS Mode

Contrary to XEX, a successful DPA on XTS in Figure 6.2b requires the knowledge of the accessed sector number. This knowledge allows to first obtain K_2 from the tweak computation by monitoring accesses to different sectors. Once K_2 is known, the tweak τ used for encrypting any memory block can be computed, which enables a straight-forward attack on the key K_1 by monitoring the power consumption during arbitrary memory accesses. Alternatively, the attack technique from XEX can be used to learn K_1 and the sector tweaks τ without knowing the concrete sector number. Further, note that another approach to perform a DPA on XTS is to target the modular multiplication during tweak computation as presented in [LFD17].

Counter Mode

Known-plaintext scenarios allow for DPA attacks that recover the key K in counter mode encryption. They facilitate the computation of the encryption pads from the known plain- and ciphertexts and thus DPA on the last round of the cipher. Typically, plaintexts would be assumed to be unknown since memory encryption is applied. However, known-plaintext scenarios will certainly occur in memory encryption. One such case would be publicly known (or observable) data that is sent to a device, e.g., via external interfaces, and that is consecutively encrypted and stored in main memory, e.g., within an input buffer.

If there are insufficiently many known plaintexts, a known input seed also allows for a DPA—one that does not even require any ciphertext. Often, the counters and addresses within the seed will be publicly accessible (or observable). If the IV is public as well, the seed will be fully known and a DPA in the first round of the cipher be possible. The IV will mostly be stored publicly on the disk for disk encryption, but might be chosen randomly at startup and remain inaccessible for encryption of the main memory. Still, the approach in [Jaf07], where a DPA is performed on the counter mode of AES without knowledge of the counter value, might be applicable.

CBC Mode with ESSIV

Independently of the IV derivation, DPA attacks on the CBC mode are trivially possible through the observation of ciphertexts and power traces of the respective encryption processes. The recovered key K then allows to compute each sector's IV (ESSIV) and hence to obtain any plaintext.

6.2.2 Differential Fault Analysis

Differential Fault Analysis (DFA) [BS97] describes techniques that use algebraic properties of ciphers to find out about the key from one correct and one or several faulty cipher invocations with the same input. Various techniques to inject faults into a device exist, e.g., power and clock glitches, laser shots, and electromagnetic pulses. However, the following investigation does not consider how the faults are injected, but elaborates on how faults are exploited in order to obtain the key. It details DFA attack scenarios on the schemes from Section 6.1, and most noteworthy, how to break the tweakable block ciphers XEX and XTS with practical complexity 2^{35} if AES-128 is used.

XEX Mode

The attack procedure of DFA to learn the key K is tightly linked with the employed cipher. Exemplarily, we show how to use DFA to extract the key from AES-128 in XEX mode. The DFA targets the block cipher that is shaded gray in Figure 6.2a and consists of two basic steps:

1. An arbitrary byte fault in round 8 is used to extract the XOR of round key 10 and the tweak ($rk_{10} \oplus \tau$).
2. A byte fault in round 7 and a modified representation of the AES round function lead to round key 9 and thus the key K.

Learning $rk_{10} \oplus \tau$. From an attacker's point of view, the last round key rk_{10} is blinded with the tweak τ. This requires the tweak τ to be constant for DFA, i.e., the attack operates on a fixed sector and a fixed memory block. By forcing re-encryption of the same plaintext in the desired block, the adversary gets the chance to inject an arbitrary byte fault during round 8 of the encryption process of the tweakable cipher. Application of a suitable DFA technique, e.g., [PQ03; SMC09], to the pair of right and faulty ciphertext results in the value $rk_{10} \oplus \tau$.

Learning round key 9. The DFA to learn rk_9 benefits from an alternative representation of the AES round function. As shown in Figure 6.5, it is obtained from swapping MixColumns and AddRoundKey. The linearity of MixColumns allows this transformation if the round key is modified accordingly, i.e.,

$$\begin{aligned}\mathrm{MixColumns}(G) \oplus rk_9 &= \mathrm{MixColumns}(G \oplus \mathrm{MixColumns}^{-1}(rk_9)) \\ &= \mathrm{MixColumns}(G \oplus rk_{9,mc}).\end{aligned}$$

In the following, the alternative representation of the round function is used for round 9. The attack starts by injecting a random byte fault during round 7. As the MixColumns operation propagates the fault to the other state bytes, all

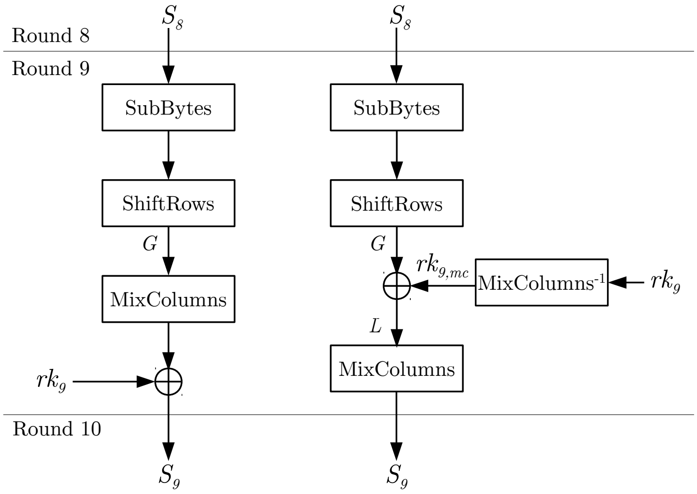


Figure 6.5: AES round function (left) and its alternative representation (right).

bytes are affected by the end of round 8. The observed pair of right and faulty ciphertext C, C' and the value $rk_{10} \oplus \tau$ are used to compute backward to obtain the respective values L, L' in round 9.

Interpreting L, L' as a pair of right and faulty ciphertext, the remaining cipher looks like a round-reduced version of the AES with one inner round missing. The last round consists of AddRoundKey, ShiftRows, and SubBytes and uses the round key $rk_{9,mc}$. The benefit of this approach is that now any DFA technique that targets the last round key of the AES, e.g., [PQ03; SMC09], is suitable to obtain $rk_{9,mc}$ from the pair L, L' and the fault differences at the end of round 8. Round key 9 is then easily computed as $rk_9 = \text{MixColumns}(rk_{9,mc})$.

If the technique in [SMC09] is used to learn $rk_{9,mc}$, the attack has the complexity 2^{34} and thus is clearly possible on nowadays' computers. According to [SMC09], the required faults can be injected by temporal overclocking only. Note that similar approaches work if ciphers different to AES are utilized.

XTS Mode

Although XTS using AES-128 relies on two 128-bit keys, DFA breaks this mode with total complexity 2^{35}. First, the DFA technique that was just applied to XEX trivially recovers the key K_1 with complexity 2^{34}. Second, the following small trick uses faults in the tweak computation to also learn K_2 with complexity 2^{34}. It determines the faulty tweak τ' from the observed faulty ciphertext C' and the correct tweak τ.

The procedure to recover K_2 requires the values of K_1, P, and $rk_{1,10} \oplus \tau$ to be known, where $rk_{1,10}$ denotes round key 10 derived from K_1. These preconditions usually apply if the previous DFA on XEX was utilized to learn K_1. As a result, the tweak τ and the intermediate value U (cf. Figure 6.2b) can be computed:

$U = \alpha^{-addr} \cdot \tau$. A random fault that is injected in one byte of the state in round 9 of the AES affects four bytes of U. Although the respective faulty U' is not directly observable, it can be brute-forced with complexity 2^{32}. This is done by trying all values for the faulty bytes of U', computing the respective tweaks τ', encrypting the original plaintext P using τ' and K_1, and matching the result against the faulty ciphertext C'. Once U' is known, four bytes of $rk_{2,10}$ (round key 10 derived from K_2) are revealed using the technique in [PQ03]. Hereby, the possible key space for $rk_{2,10}$ is reduced by the possible differences that can be observed at the output of MixColumns in round 9 that result from a single byte fault during round 9. Similarly, three more faults in different bytes of the state of round 9 recover the remaining 12 bytes of $rk_{2,10}$ and thus K_2.

Counter Mode

DFA on a block cipher operated in counter mode (cf. Figure 6.4) requires access to the output of the cipher, i.e., the pad. Since encryption pads must not repeat, consecutive encryptions of plaintexts will not use the same pad and encryption seed. As a result, DFA is limited to the decryption process. If the same ciphertext is loaded from the same memory address several times and the adversary can inject faults during the pad computations and observe the respective plaintexts, the correct and faulty pads can be computed and the master key K be learned via a suitable DFA technique. The required plaintexts may be observed from communication of the device via external interfaces.

CBC Mode with ESSIV.

Independently of the initial vector derivation, DFA is trivially possible by restricting analysis to one specific memory block within the CBC chain of one particular sector. Therefore, re-encryption of the same plaintext has to be triggered for the desired memory block, e.g., through placing the same message in an input buffer by repeatedly sending the same message to the device. Faults injected during re-encryption are directly observable in the resulting ciphertext. This facilitates the application of a suitable DFA technique in order to learn the master key K. Note that for this to work, all memory blocks in the sector prior to the target block must not change during re-encryption.

6.3 EM Attack on Ext4 Encryption

As our analysis points out, contemporary memory encryption schemes are clearly vulnerable to physical attacks. However, it remains to show that such attacks are indeed feasible on contemporary systems. This section therefore demonstrates a practical attack on the disk encryption scheme incorporated into the ext4 file system. The EM attack conducted on a Zynq Z-7010 SoC reveals the used master key and thus all content by exploiting the leakage of the first round of an AES execution.

6.3.1 Analysis of Ext4 Disk Encryption

Disk encryption within the ext4 file system works on file level and allows to encrypt arbitrary directories using a specified master key MK. For each file in such directory, the master key MK is used to derive an individual data encryption key DEK_f to encrypt the respective file's content and name. Key derivation is done by encrypting MK with AES-128 in ECB mode using a public file nonce N_f as the key. It starts whenever DEK_f is needed and not already present in main memory. The size of both MK and DEK_f is 512 bits and chosen such as to be able to encrypt files with AES-256 in XTS mode in future versions. However, currently only AES-128 in XTS mode is supported and thus the last 256 bits of DEK_f and MK are not used. The file nonce N_f is stored in an extended attribute of the file's inode.

Clearly, given the master key MK and a public file nonce N_f, the respective file key DEK_f can be derived. However, the key derivation chosen in ext4 also allows to compute the master key MK given any DEK_f and the respective nonce N_f. Therefore, an attacker who wants to learn MK using power analysis can choose between two equivalent targets, namely (1) data encryption of file content, and (2) the derivation of the file key DEK_f. In terms of target (1), the strategy from Section 6.2 can be straight-forwardly applied, but one may need files that are sufficiently large to be able to learn K_2 within XTS. With respect to target (2), one needs to monitor accesses to many different files as such trigger key derivations. To practically verify the feasibility of attacks on disk encryption, we opted for target (2).

6.3.2 General Attack Flow

The attack we performed assumes an encrypted folder on an SD card using the ext4 file system. It further assumes the attacker is able to trigger the creation of new files within the encrypted folder via external interfaces, e.g., by uploading data via a running web server or writing log files.

To perform the attack, the attacker first dumps the (encrypted) content of the SD card. They may not be able to read the actual content from such file system dump, but can learn about the directory structure as meta data is not encrypted. Second, the attacker triggers the creation of sufficiently many files on the SD card, observes the EM side channel, and stores the respective EM traces. Third, the attacker again dumps the content of the SD card. By comparing its content with the initial dump from before the measurements, the attacker can learn which files have been created. The meta data of the newly created files allows to both learn the used nonces N_f and their creation date, which in turn allows to map the newly created files on the SD card to the EM traces. In the next step, the attacker creates the power model for the key derivation, i.e., $DEK_f = E(N_f; MK)$. Finally, the power model is matched with the EM traces to reveal the master key.

To investigate the encrypted directory in the file system, debugging and forensic tools are highly suitable. We used the tool `debugfs` to find new files in

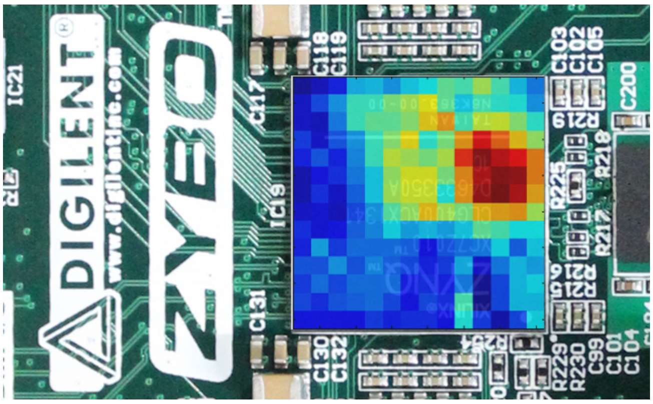


Figure 6.6: Distribution of t-test results on the chip surface.

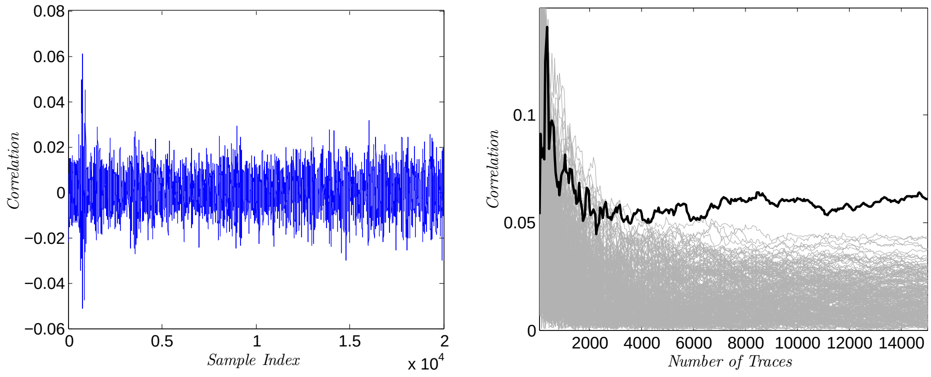
the file system and to learn their creation date and the respective nonces. Note that the access times are also available within the file system, which allows for the described attack also when monitoring arbitrary file accesses.

6.3.3 Experimental Setup and Results

The feasibility of the attack on ext4 encryption in Section 6.3.2 was verified using the Digilent ZYBO board. The board hosts a Xilinx Zynq Z-7010 SoC, 512 MB of DDR3 RAM, and several IO interfaces, i.a., an SD card slot. The Zynq Z-7010 SoC combines an Artix-7 Field Programmable Gate Array (FPGA) and a state-of-the-art hard macro comprising a 650-MHz dual-core ARM Cortex-A9 processor, IO modules, and memory controllers. The measurement devices required to capture the EM traces involved a LeCroy WavePro 725Zi oscilloscope, a Langer RF B 3-2 magnetic field probe, and a Langer PA 303 pre-amplifier.

The general leakage behavior of the Zynq Z-7010 was examined by running the AES T-table implementation included in the Linux 4.3 kernel in a bare-metal application. Therefore, the EM probe was placed in different locations using a stepper table to evaluate a fixed vs. random t-test. This revealed the spots of high leakage as shown in Figure 6.6 and allowed for successful DPA on the bare-metal AES.

The setup for the complete disk encryption scenario was established by configuring the Zynq SoC to use a 350-MHz memory clock and a 625-MHz Central Processing Unit (CPU) clock and deploying Linux 4.3 to the ZYBO board. An ext4 file system was created on an SD card and one directory encrypted such that it is only readable by the system running on the ZYBO board. The attack procedure from Section 6.3.2 was executed by repeatedly creating new files via the UART interface. The oscilloscope was triggered to capture an EM trace at 5 GS by setting a GPIO pin just before creating a new file. The SD card content was then analyzed on a PC using `debugfs`, the EM traces aligned, and a



(a) Time domain (correct key). **(b)** Key hypotheses.

Figure 6.7: Single-byte correlation results for ext4 key derivation.

DPA performed on the S-box output of the first AES round using the Hamming Weight (HW) power model.

The results of the DPA on a single byte of the master key are given in Figure 6.7. Using 15 000 EM traces, Figure 6.7a clearly presents the correlation of the power model of the correct key guess in the time domain. Moreover, in Figure 6.7b the correct key byte (black) is clearly distinguished from the remaining key hypotheses with 5 000 measurements.

In this feasibility study, the Linux kernel was reconfigured to omit symmetric multiprocessing, dynamic frequency scaling, and caches. Moreover, AES executions were highlighted in the captured EM traces through another hardware-triggered signal to help finding AES executions. This however does not affect the applicability of the attack. For example, [Lon+15] showed the practicality of attacking a free-running OpenSSL implementation of AES with active caches and frequency scaling on the TI Sitara platform that uses an ARM Cortex-A8. However, further improvement of both setup and trace processing would definitely be interesting future work.

6.4 Conclusion

Summarizing, this chapter unveiled that contemporary mechanisms that aim to ensure the confidentiality of memory content in the presence of adversaries with physical access are clearly vulnerable to physical attacks. In particular, it showed that all common implementations of memory and disk encryption schemes can easily be broken using DPA and DFA. The attacks are powerful enough to even break the tweakable cipher XTS that is most commonly used. Further, the feasibility of such attacks on state-of-the-art computing systems was verified by exploiting the EM side channel on the Zynq Z-7010 SoC. The attack revealed the master key of the disk encryption scheme incorporated into the ext4 file system and thus all encrypted content.

Our results suggest that if memory encryption is supposed to use current schemes in the future, cipher implementations with appropriate countermeasures must be used. However, the secure cipher implementations proposed so far were mainly designed for the use in embedded devices and might thus not yield the desired throughput for memory encryption. For example, the 1st-order threshold implementations in [Bil+14; Mor+11b] require 246 and 266 clock cycles for one AES execution, respectively. Additionally, these implementations add an area overhead of a factor of four that must hence also be expected for secure memory encryption based on such protected implementations. It thus remains future work to implement memory encryption that fulfills both the requirement for sufficient throughput and security against side-channel adversaries. Alternatively, a viable approach for the future is to develop new memory encryption schemes that resist the presented attacks by design.

7

Side-Channel Secure Memory Encryption and Authentication

Chapter 6 showed that many of nowadays' memory encryption schemes are vulnerable to Differential Power Analysis (DPA) when attackers can observe a device performing memory encryption during operation. This threat is even emphasized by the practical attacks in [UM16; Lon+15; SRH16; Bal+15] that document the feasibility of DPA on state-of-the-art System on Chips (SoCs). While techniques like masking [Cha+99; GP99] can protect the respective cryptographic primitives against DPA, the overhead of DPA-protected implementations is high. To overcome this limitation, together with Mario Werner and Stefan Mangard, we researched on new schemes for memory encryption that can prevent DPA by design and published the resulting DPA-secure memory encryption and authentication scheme in [UWM17b]. A follow-up work including its practical implementation and a more detailed comparison with the state of the art is currently in submission [UWM17a]. In these works, I was the main author contributing the idea, most of the text, and parts of the implementation, Stefan Mangard added to the idea for higher-order security, and Mario Werner contributed parts of the text and parts of the implementation. In this chapter, we use text and results from [UWM17b] and [UWM17a] and make the following contributions.

Contribution. We solve the problem of protecting data in memory against physical attackers in possession of a running device and, in particular, we prevent DPA attacks on memory encryption and authentication without additional overhead over conventional schemes. We approach the topic by reviewing the security of fresh re-keying [Koc03; Med+10] for memory encryption as a DPA countermeasure. While re-keying completely thwarts DPA on the cryptographic key, it shows that the read-modify-write access patterns inevitably occuring memory allow for side-channel plaintext-recovery attacks as presented in Chapter 5. In particular, re-keying block ciphers in read-modify-write operations causes constant plaintext parts to be encrypted using different keys, which can be exploited in profiled, higher-order DPA attacks to learn the constant plaintexts. As a result, re-keying merely provides first-order DPA security for the memory content itself.

Second, we build on our analysis and present MEAS—the first Memory Encryption and Authentication Scheme secure against DPA attacks. The scheme is suitable for all kinds of memory including Random Access Memory (RAM) and Non-Volatile Memory (NVM). By making use of synergies between fresh re-keying and authentication trees [Mer80; HJ05; Elb+07], MEAS simultaneously offers security against first-order DPA and random access to all memory blocks. In more detail, MEAS uses separate keys for each memory block that are stored in a tree structure and changed on every write access in order to strictly limit the use of each key to the encryption of two different plaintexts at most. For higher-order DPA security, MEAS performs data masking by splitting the plaintext values into shares and storing the encrypted shares in memory. This allows to flexibly extend DPA protection to higher orders in trade for additional memory. For all DPA protection levels, MEAS does not require DPA-protected implementations of the cryptographic primitives, making MEAS suitable for Common Off-The-Shelf (COTS) systems equipped with unprotected cryptographic accelerators. However, MEAS is also an ideal choice for constructing a DPA-secure system from scratch as engineers do not have to cope with complex DPA protection mechanisms within the cipher implementation.

Third, we give two lightweight MEAS instances suitable for RAM that encrypt and authenticate the tree nodes with strictly bounded data complexity per key. MEAS-v1 uses the PRINCE cipher and derives a fresh key for each encryption block using the sponge ASCON. MEAS-v2 provides faster tree traversal by using the same key for the encryption of several, but a sufficiently small number of, e.g., 4 or 8, blocks using the tweakable cipher QARMA.

Finally, we implement both MEAS instances on the Xilinx Zynq-7020 SoC Field Programmable Gate Array (FPGA) to practically evaluate the performance of RAM encryption and authentication with MEAS. We show that MEAS provides protection against the very powerful DPA attacks, and still features the same performance and memory overhead as state-of-the-art memory authentication schemes, which completely lack side-channel protection. In particular, we show that a 4-ary, first-order DPA secure instance of MEAS-v2 is a highly suitable choice for encrypting and authenticating RAM in practice.

Outline. This chapter is organized as follows. In Section 7.1, we first state our threat model and requirements, and we then discuss the state of the art on memory encryption and authentication. We analyze the re-keying countermeasure in terms of memory encryption in Section 7.2 and use the results to present our first-order DPA secure MEAS in Section 7.3. Section 7.4 then presents data masking to achieve higher-order DPA security in MEAS. We give two lightweight instances of MEAS suitable for RAM in Section 7.5 and detail their implementation in Section 7.6. An evaluation of MEAS is done in Section 7.7 and we finally conclude in Section 7.8.

7.1 Memory Encryption and Authentication

Memory encryption and authentication is more and more commonly adopted in consumer products, e.g., in Intel SGX [Gue16], AMD [KPW16], and ARM systems [HT13]. However, as Chapter 6 showed, memory encryption and authentication schemes currently lack the consideration of side-channel attacks. In this section, we hence define two threat models for memory encryption and authentication: the *non-leaking chip model* restates the state of the art [Suh+03a; Gue16; Elb+09; Suh+03b; Owu+13; Rog+07], and the extended *leaking chip model* further takes side-channel leakage into account. Moreover, we summarize present techniques for memory encryption and authentication and its requirements.

7.1.1 Threat Model and Requirements

The *non-leaking chip model* in previous works assumes a single, secure microchip performing all relevant computations, e.g., a Central Processing Unit (CPU). An attacker cannot perform any kind of active or passive attacks against this chip. All other device components outside this chip, e.g., buses, RAM modules and Hard Disk Drives (HDDs), are under full control of the adversary. Therefore, a physical attacker can, e.g., probe and tamper with buses, exchange peripherals, or turn the whole device on and off. For off-chip memory, this means that an attacker with physical access is capable of freely reading and modifying the memory content.

While reading can give an attacker access to confidential data stored inside the memory, modification breaks memory authenticity in several ways [Elb+09]: In *spoofing attacks*, an attacker simply replaces an existing memory block with arbitrary data, in *splicing attacks*, the data at address A is replaced with the data at address B, and in *replay attacks*, the data at a given address is replaced with an older version of the data at the same address.

Our *leaking chip model* extends the *non-leaking chip model* by considering passive side-channel attacks. It assumes that the microchip performing all relevant computations leaks information on the processed data via side channels, e.g., power and Electromagnetic Emanation (EM). Physical attackers can observe this leakage and perform side-channel attacks.

Hence, cryptographic schemes protecting the confidentiality and authenticity of off-chip memory in the *leaking chip model* have to fulfill three main requirements.

1. The only information an adversary can learn from memory is whether a memory block (i.e., ciphertext) has changed or not.

2. Prevention of spoofing, splicing, and replay attacks.

3. Protection against side-channel attacks.

In addition, fast random access to all memory blocks, high throughput (fast bulk encryption), and low memory overhead are desired.

7.1.2 Memory Encryption

Memory encryption schemes usually split the memory address space into blocks of predefined size, e.g., sector size, page size, or cache line size. Each of these blocks is then encrypted independently using a suitable encryption scheme. The partitioning of the address space into memory blocks aims to provide fast random access on block level and fast bulk encryption within the instantiated encryption scheme. Hereby, the chosen block size strongly affects possible trade-offs w.r.t. meta-data overhead, access granularity, and speed.

As shown in Chapter 6, several memory encryption schemes have been proposed in the *non-leaking chip model*, e.g., the tweakable encryption modes Xor-Encrypt-Xor (XEX) [Rog04] and XEX-based Tweaked codebook mode with ciphertext Stealing (XTS) [IEE08b], Cipher Block Chaining (CBC) with Encrypted Salt-Sector IV (ESSIV) [Fru05], and counter mode encryption [Suh+03b; Rog+07].

7.1.3 Memory Authentication

Like for memory encryption, memory authentication schemes split the memory address space into blocks and aim for separate authentication of each of these blocks. Several memory authentication schemes have been proposed in the *non-leaking chip model*.

For example, a keyed Message Authentication Code (MAC) using the block address information can protect against spoofing and splicing attacks. However, it still allows for replay attacks. In order to protect against replay attacks, authenticity information must be stored in a trusted environment, e.g., in secure on-chip memory, that an attacker cannot modify. Authentication trees minimize this demand for secure on-chip storage, namely, only the tree's root is stored in secure memory, while the remaining tree nodes can be stored in public memory. Such trees therefore protect against spoofing, splicing, and replay attacks. Authentication trees over n_b memory blocks with arity a have logarithmic height $\underline{l} = \log_a(n_b)$.

In the following we describe three prominent examples of authentication trees, namely, Merkle trees [Mer80], Parallelizable Authentication Treess (PATs) [HJ05], and Tamper Evident Counter (TEC) [Elb+07] trees. Note however that there are also hybrid variants like Bonsai Merkle trees [Rog+07], which use elements from both Merkle trees and PATs. The description assumes binary trees, the operator $||$ denotes concatenation.

Merkle Trees

Merkle trees [Mer80] use a hash function $\mathcal{H}$ to hash each of the n_b memory blocks P_i:

$$h_{\underline{l},i} = \mathcal{H}(P_i) \qquad\qquad 0 \leq i \leq n_b - 1.$$

These hashes $h_{\underline{l},i}$ are recursively hashed together in a tree structure and the root hash $h_{0,0}$ is put on the secure chip:

$$\begin{aligned} h_{j,i} = \mathcal{H}(h_{j+1,2i} || h_{j+1,2i+1}) \qquad\qquad & 0 \leq i \leq \frac{n_b}{2^{\underline{l}-j}} - 1, \\ & 0 \leq j \leq \underline{l} - 1. \end{aligned}$$

Parallelizable Authentication Trees

PATs [HJ05] use a nonce-based MAC and a key K to authenticate each of the n_b data blocks P_i using a tag $T_{\underline{l},i}$:

$$T_{\underline{l},i} = MAC(K; N_{\underline{l},i}; P_i) \qquad\qquad 0 \leq i \leq n_b - 1.$$

The nonces $N_{\underline{l},i}$ are recursively authenticated in a tree structure using again nonce-based MACs. While the key K and the root nonce $N_{0,0}$ must be stored on the secure chip, all other nonces and the tags are stored publicly in off-chip memory:

$$\begin{aligned} T_{j,i} = MAC(K; N_{j,i}; N_{j+1,2i} || N_{j+1,2i+1}) \qquad\qquad & 0 \leq i \leq \frac{n_b}{2^{\underline{l}-j}} - 1, \\ & 0 \leq j \leq \underline{l} - 1. \end{aligned}$$

Tamper Evident Counter Trees

While Merkle trees and PATs provide memory authenticity, TEC trees [Elb+07] additionally provide memory confidentiality. Therefore, TEC trees use Added Rendundancy Explicit Authenticity (AREA) [Fru05] codes. Hereby, each plain memory block P_i is padded with a nonce $N_{\underline{l},i}$ and then encrypted with key K using a common block cipher:

$$C_{\underline{l},i} = E(K; P_i || N_{\underline{l},i}) \qquad\qquad 0 \leq i \leq n_b - 1.$$

For verification, a ciphertext $C_{\underline{l},i}$ is decrypted to $P'_i || N'_{\underline{l},i}$ and $N'_{\underline{l},i}$ compared with the original nonce $N_{\underline{l},i}$. Hereby, the authenticity is ensured by the diffusion of the block cipher as it makes it hard for the adversary to modify the encrypted nonce $N_{\underline{l},i}$. The nonce $N_{\underline{l},i}$ is formed from the memory block address and a counter $ctr_{\underline{l},i}$ [Elb+07]. The nonce counters are recursively authenticated using AREA codes in a tree structure. The key K and the root counter $ctr_{0,0}$ are stored on the secure chip:

$$\begin{aligned} C_{j,i} = E(K; ctr_{j+1,2i} || ctr_{j+1,2i+1} || N_{j,i}) \qquad\qquad & 0 \leq i \leq \frac{n_b}{2^{\underline{l}-j}} - 1, \\ & 0 \leq j \leq \underline{l} - 1. \end{aligned}$$

7.2 Re-Keying for Memory Encryption

Present memory encryption and authentication schemes are designed to protect off-chip memory against adversaries with physical access assuming a microchip that is secure against all active and passive attacks. However, in Internet-of-Things (IoT) scenarios, the assumption that the microchip is secure against all passive attacks is often too strong since, in practice, a microchip running an algorithm leaks information on the processed data via various side channels. This side-channel leakage allows attackers to perform passive side-channel attacks on memory encryption, such as presented in Chapter 6. In this respect, Chapter 2 pointed out that DPA is particularly powerful and can be prevented by frequent re-keying without adding costly DPA countermeasures to the implementation itself. Simultaneously, there are more and more practical systems being deployed with unprotected cryptographic accelerators by vendors not being aware of side-channel attacks. As a result, re-keying based schemes are an interesting option for protecting memory encryption and authentication against DPA.

In this section, we hence investigate the security of re-keying in the context of memory encryption and authentication. It shows that contrary to other use cases, the re-keying operation itself can be realized without DPA countermeasures when protecting memory. However, the application of re-keying to memory encryption does not prevent the side-channel plaintext-recovery attacks presented in Chapter 5.

7.2.1 The Re-Keying Operation

Up until now, the principle of re-keying was applied only to communicating parties aiming for confidential transmission. Hereby, constructions following Figure 2.2 and Figure 2.3 prevent DPA, but require the initialization with a fresh session key $\overline{K}_0$ and thus secure key synchronization between the communicating parties. For this reason, both constructions use a secure re-keying function $g : (K, N) \mapsto \overline{K}_0$ to derive a fresh session key $\overline{K}_0$ from a pre-shared master secret K and a public, random nonce N [Med+10; GGM86; Sta+10]. However, this approach shifts the DPA problem to the re-keying function g, which thus needs DPA protection through mechanisms like masking.

The encryption and authentication of data stored in memory gives different conditions for the instantiation of re-keying based schemes. In particular, encrypting data in memory means that en- and decryption is performed by the same party, i.e., a single device encrypts data, writes it to the memory, and later reads and decrypts the data. Therefore, key synchronization becomes unnecessary and the cryptographic scheme can be re-keyed using random numbers without the need for any cryptographic primitive or function being implemented with DPA countermeasures, i.e., without the need for a secure re-keying function g.

7.2.2 Re-Keying and Plaintext Confidentiality

As Chapter 5 showed, frequent re-keying effectively prevents DPA on the key, but in certain applications yet results in a loss of plaintext confidentiality. Unfortunately, this is the case for the application of re-keying to memory encryption as well. The main observation that leads to this conclusion are read-modify-write operations that inevitably occur in any encrypted memory. These take place whenever the write granularity is smaller than the encryption granularity. In combination with re-keying, these read-modify-write operations cause constant plaintext parts to be encrypted several times using different keys. It is this mixing of constant plaintext parts with varying keys that allows to perform the side-channel plaintext-recovery attacks from Chapter 5 to learn the confidential, constant plaintexts. For re-keyed stream ciphers such as in Figure 2.2, this means to perform a plain, first-order DPA. For re-keyed block encryption such as in Figure 2.3, this means to perform a profiled, second-order DPA.

Both variants of side-channel plaintext-recovery attacks do not target the actual keys, but the confidential memory content. While these attacks cannot be prevented in the memory scenario, note that the effort and complexity of profiled, second-order DPA attacks is very high in practice. Hence, re-keyed block encryption provides a suitable basis to construct a memory encryption scheme with first-order DPA security. We further pursue this approach in Section 7.3. To obtain higher-order security, we extend our design in Section 7.4 and propose masking of the stored plaintext values. This effectively increases the number of values to be recovered via templates in side-channel plaintext-recovery attacks on block encryption without the need for masking being implemented in the cipher.

7.3 DPA-Secure Memory Encryption and Authentication

Section 7.2 indicates that frequent re-keying of a block cipher based mode is a suitable approach to construct a memory encryption and authentication scheme with first-order DPA security from unprotected cryptographic primitives. However, one major requirement in Section 7.1 is to provide fast random access in memory, but random access is not a feature of present re-keying based encryption schemes.

A common way to provide fast random access to large memory is to split the memory into blocks that can be directly accessed. However, encrypting each of these memory blocks by the means of fresh re-keying would render the number of keys to be kept available in secure on-chip storage too high. This problem is quite similar to memory authentication with replay protection, which also requires block-wise authenticity information to be stored in a trusted manner. To tackle this issue, state-of-the-art authenticity techniques (cf. Section 7.1) employ tree constructions to gain scalability and to minimize the required amount of expensive on-chip storage.

In this section, we therefore use the synergies between frequent re-keying and memory authentication to present Meas—a Memory Encryption and Authentication Scheme with first-order DPA security built upon unprotected cryptographic primitives and suitable for all kinds of large memory, e.g., RAM and NVM. Similar to existing memory authentication techniques, Meas uses a tree structure to minimize the amount of secure on-chip storage. However, instead of hashes or nonces, keys are encapsulated within the tree. In more detail, the leaf nodes of the tree, which store the actual data, are encrypted and authenticated using an authenticated encryption scheme that is provided with fresh keys on every write access. Similarly, the inner nodes of the tree, which store the encryption keys for their respective child nodes, are encrypted with an encryption scheme that uses a fresh key on every write. Meas is shown secure in the *leaking chip model*, and in particular, its DPA security is substantiated by limiting the number of different processed inputs per key to $q = 2$ such as in [Bel+14; Pie09; Sta+10; TS15].

In the following, we first present the construction of Meas, followed by a security analysis considering authenticity and side-channel attacks.

7.3.1 Construction

The construction of Meas is designed to be secure according to the *leaking chip model*. Therefore, Meas requires a Simple Power Analysis (SPA)-secure block encryption scheme ENC and an SPA-secure authenticated encryption scheme AE. Both ENC and AE have to fulfill the common security properties for (authenticated) encryption schemes and must be based on block encryption such as in [TS15]. Section 7.5 will detail concrete instances for both ENC and AE. Besides, Meas requires a secure random number generator for generating keys.

An example of the tree construction proposed for Meas is depicted in Figure 7.1. For the sake of simplicity, this example as well as the following description assumes the use of a binary tree, i.e., arity $a = 2$. However, instantiating the tree with higher arity is easily possible.

The structure of Meas is as follows. The data in memory is split into n_b plaintext blocks P_i. Each of these P_i is encrypted and authenticated to a ciphertext-tag pair (C_i, T_i) using the authenticated encryption scheme AE with data encryption key DEK_i:

$$(C_i, T_i) = \mathrm{AE}(DEK_i; P_i) \qquad\qquad 0 \leq i \leq n_b - 1.$$

The encryption scheme ENC then encrypts the data encryption keys DEK_i to the ciphertexts $C_{\underline{l}-1,i}$ using key encryption keys $KEK_{\underline{l}-1,i}$. The operator $||$ denotes concatenation.

$$C_{\underline{l}-1,i} = \mathrm{ENC}(KEK_{\underline{l}-1,i}; DEK_{2i}||DEK_{2i+1}) \qquad 0 \leq i \leq \frac{n_b}{2} - 1.$$

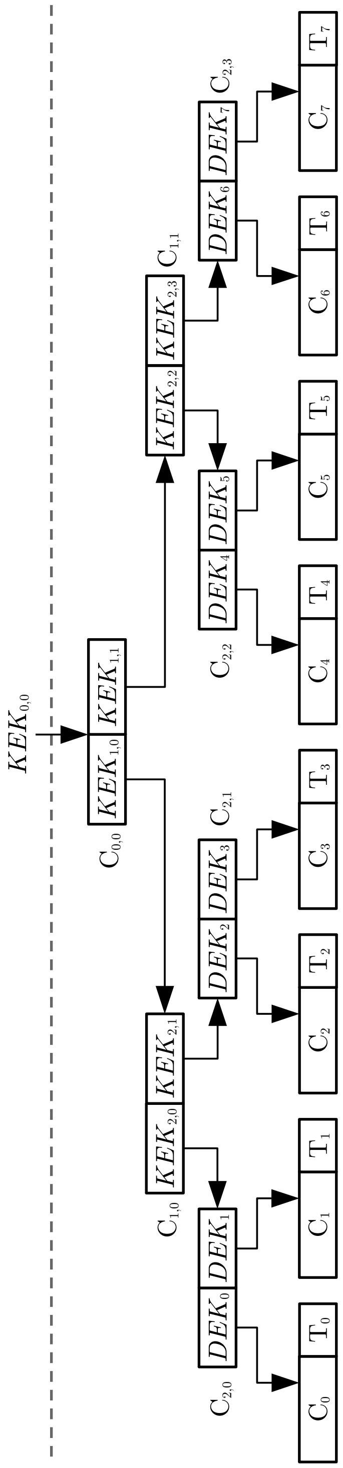


Figure 7.1: MEAS' tree construction for $n_b = 8$ data blocks and with an arity of $a = 2$.

Recursively applying ENC in a similar way to the key encryption keys finally leads to the desired tree.

$$C_{j,i} = \mathrm{ENC}(KEK_{j,i}; KEK_{j+1,2i}||KEK_{j+1,2i+1}) \qquad 0 \leq i \leq \frac{n_b}{2^{\underline{l}-j}} - 1$$
$$0 \leq j \leq \underline{l} - 2$$

While all ciphertexts and tags are stored in public, untrusted memory, the root key $KEK_{0,0}$ is stored on the leaking chip.

Read Operation

When reading data (C_i, T_i) from memory, all the keys on the path from the root key $KEK_{0,0}$ down to the respective data encryption key DEK_i are decrypted one after another. The data encryption key DEK_i is then used to decrypt and authenticate the respective memory block (C_i, T_i).

For example in Figure 7.1, to obtain the plaintext block P_2 stored in (C_2, T_2), the root key $KEK_{0,0}$ is used to decrypt $KEK_{1,0}$. Then, $KEK_{1,0}$ is used to decrypt $KEK_{2,1}$, which permits to decrypt DEK_2. Finally, DEK_2 is used with (C_2, T_2) to authenticate and decrypt the respective plaintext P_2.

Note that the decryption of the encapsulated keys can only be performed sequentially. However, this is not considered a problem since computation is typically much faster than storage (e.g., RAM or HDD). On the other hand, caching of the intermediate nodes (key encryption keys) is supported by MEAS in order to achieve good performance, e.g., small average access latency.

Write Operation

Writing data to the memory is where the actual re-keying is performed. Namely, the process of updating P_i with P'_i involves the replacement of all keys along the path from the root key $KEK_{0,0}$ down to the respective data encryption key DEK_i with randomly generated ones. On the other hand, the keys for the adjacent subtrees are only re-encrypted under the new node keys. This re-keying can be performed in a single pass from the root to the leaf node of the tree.

For example in Figure 7.1, when block P_5, which is stored in (C_5, T_5), gets replaced, also the keys $KEK_{0,0}$, $KEK_{1,1}$, $KEK_{2,2}$ and DEK_5 have to be changed. Therefore, the node $C_{0,0}$ is decrypted to extract $KEK_{1,0}$ and $KEK_{1,1}$. The new node $C'_{0,0}$ can then be determined by encrypting $KEK_{1,0}$ and a new $KEK'_{1,1}$ with the new key encryption key $KEK'_{0,0}$. The nodes $C_{1,1}$ and $C_{2,2}$ are updated in the same way. The new data block (C'_5, T'_5) is then the result of authenticated encryption of P'_5 under the new data encryption key DEK'_5.

Note that it is not necessary to check authenticity when a full block is written to the memory. Only read-modify-write operations on a data block require an authenticity check. This authenticity check is automatically performed when the data is read prior to modification and thus does not inhere any additional costs. Also note that read-modify-write operations require only one single tree traversal, because the data encryption key required for the read operation automatically becomes available in the last steps of the write (and re-keying) procedure.

7.3.2 Authenticity

The design of MEAS protects data authenticity with respect to spoofing, splicing, and replay attacks using both the authentic root key and the AE scheme. In particular, spoofing and splicing attacks on the leaf nodes are directly detected by the AE scheme since different keys are used for each block. Moreover, the AE scheme indirectly also protects the inner tree nodes for properly chosen schemes AE and ENC. In such case, any tampering with the ciphertext of an intermediate node will lead to a random but wrong key to be decrypted. This tampering will thus propagate down to the leaf node to give an erroneous, random data encryption key and finally an authentication error.

Replay protection for all nodes is the result of the authentic root key, which gets updated on every write to any leaf node, i.e., choosing a new, random root key on every write access ensures that the secure root reflects the current state of the tree in public memory. Vice versa, the authenticity tags in the leaf nodes output by the AE scheme reflect the authenticity of the path from the root to the respective data block. Therefore, if the authenticity check of a leaf node fails, any node on the path from the root to the leaf may be corrupted.

Handling corruption

As soon as a corrupted leaf node has been detected, the authenticity of the tree must be restored before any further actions are taken. Otherwise, an adversary may be able to perform DPA attacks on encryption keys by introducing authenticity failures on purpose.

Restoring authenticity of the tree is simple and requires no additional support. It is sufficient to replace all corrupted data (leaf) nodes with random values since regular writes restore authenticity from the root to the respective leaf node. Restoring authenticity in this manner also causes re-keying on all nodes on the path from the root to the leaf to take place. This re-keying procedure effectively thwarts any DPA that otherwise could be performed by malicious modification of stored ciphertexts.

For example in Figure 7.1, if the authenticity check of the node (C_4, T_4) fails, any of the nodes $C_{0,0}, C_{1,1}, C_{2,2}$ and (C_4, T_4) can be erroneous. Therefore, the plaintext P_4 is replaced with a random plaintext P'_4 in order to restore the authenticity. Hereby, new keys $KEK'_{0,0}, KEK'_{1,1}, KEK'_{2,2}$ and DEK'_4 are chosen and the stored values $C'_{0,0}, C'_{1,1}, C'_{2,2}$ and (C'_4, T'_4) are updated accordingly. This procedure restores the authenticity of the path from $KEK_{0,0}$ to DEK_4, but leaves any adjacent subtree intact. Moreover, the choice of fresh keys $KEK'_{0,0}, KEK'_{1,1}, KEK'_{2,2}$ and DEK'_4 prevents first-order DPA through adversaries repeatedly modifying $C'_{0,0}, C'_{1,1}, C'_{2,2}$ or (C'_4, T'_4).

Recovering from corruption

Depending on the actual application, there are different approaches to deal with the corruption. A straightforward approach, which is suitable for RAM encryption, is to simply reset the tree and start from scratch. The memory encryption engine of SGX [Gue16], for example, follows this approach and requires a system restart to recover. However, applying this idea to block-level disk encryption is impractical since a reset of the tree is equivalent to destroying the data of the whole block device.

Another, more graceful approach is to recover from the corruption when possible. In the case of RAM encryption, it is, for example, possible that the operating system kills (and restarts) only those processes which actually accessed a corrupt data block. In the setting of disk encryption, it can be enough to report which files or directories were destroyed to enable appropriate error handling.

Given a single authentication failure, it is not possible to determine which node is corrupt. However, since corruptions in higher tree nodes lead to authenticity failures in more data blocks, it is possible to identify the subtree which is affected by the data corruption using multiple adjacent reads. This can even be done quite efficiently in a binary search like approach (i.e., $\mathcal{O}(\log m)$ reads), assuming that only a single node has been corrupted.

For example in Figure 7.1, when the authenticity check of data block 2, i.e., (C_2, T_2), fails, then data block 3 is checked next. If block 3 is authentic, then only block 2 (child of DEK_2) is corrupt. Otherwise, either block 0 or block 1 is checked next. If this next block is authentic, then only blocks 2 and 3 (children of $KEK_{2,1}$) have been corrupted. In case of another error, a final check in the right subtree (children of $KEK_{1,1}$) is needed to determine if only the left subtree (children of $KEK_{1,0}$) or the whole tree is corrupt. Note however that locating the corruption requires each authenticity failure to be followed by a re-keying step as described in Section 7.3.2 in order to resist DPA. For example, if data block 2 is read and detected to be corrupt, the path from the root key to data block 2 must be re-keyed. If during the location phase data block 3 is detected to be unauthentic as well, also the path from the root key to data block 3 must be re-keyed. The same procedure applies to all other checks in the location phase.

7.3.3 Side-Channel Discussion

We discuss the side-channel security using three types of attackers with increasing capabilities. The first type solely uses passive attacks and tries to exploit the side-channel leakage during operation. The second type additionally induces authenticity errors by tampering with the memory and strives for exploiting error handling behavior. The third type further tries to gain an advantage by restarting, i.e., power cycling, the whole system at arbitrary points in time.

Passive Attacks. The protection of MEAS against DPA lies within the re-keying approach. Therefore, every randomly generated key is used for the encryption and decryption of exactly one tree node with one specific plaintext. As soon as the plaintext of a node changes in any way, also a new key for the encryption of the respective node is generated.

For a certain key, a physical attacker who only passively observes MEAS can thus at most acquire side-channel traces of one encryption and arbitrarily many decryptions of one single plaintext. Even though the trace number is possibly high, the best an attacker can do is to combine all the traces to a single rather noise free trace of this one key-plaintext pair. To perform a DPA, on the other hand, traces for multiple different plaintexts are required. In the presence of a passive physical attacker, MEAS is therefore secure against first-order DPA attacks given that both ENC and AE are SPA secure.

Passive Attacks and Memory Tampering. An active physical attacker who tampers with the memory content can gain additional information by corrupting the ciphertext of certain nodes. Namely, such tampering gives side-channel information from the decryption of different data for one single key. However, even with such tampering it is only possible to acquire one additional side-channel trace for a specific key. This is due to the fact that every tampering is detected as soon as the leaf node is authenticated. Handling the authentication error involves restoring authenticity and thus re-keying which makes the gathering of further traces impossible. As a result, the number of acquirable traces (i.e., under the same key, but with different ciphertexts) is clearly bounded by two. Given the assumptions in related work on leakage-resilient cryptography [Pie09; Sta+10; TS15], bounding the input data complexity per key by two makes MEAS secure against first-order DPA for malicious memory corruption.

Passive Attacks, Memory Tampering and Restarts. The side-channel security of MEAS relies on the assumption that tree operations are performed atomically. This means that, e.g., once a read operation is started, all steps involved in MEAS, i.a., the MAC verification and the re-keying on authenticity failure, must be performed and completed. This assumption holds true for a running device since physical fault attacks on the leaking chip are outside the threat model. However, restarting the device during operation can break this assumption. In this case, attackers can use a combination of power cycling and memory tampering to collect arbitrarily many side-channel traces and perform a first-order DPA against a non-volatile key. However, this attack can be prevented when the concrete use case is known.

For the encryption and authentication of RAM, there is simply no reason to maintain persistent keys between system restarts. Similar to SGX, the device generates a new random key on startup which effectively thwarts the attack. For NVM, however, a persistent root key is unavoidable. Yet, there are easy and secure solutions for NVM too. For example, one could store one additional bit on the leaking chip to record whether a presumably atomic operation is currently active. This allows to detect aborted operations in MEAS on startup and thus to take further actions, e.g., counting and storing the number of aborted operations on the leaking chip and appropriate error handling when a certain threshold is reached. Such countermeasures can also be integrated with the transaction/journaling functionality of a file system.

Summarizing, Meas itself does not contain any mechanism to deal with malicious power cycling. However, for both RAM and NVM simple and cheap solutions are available.

7.4 Higher-Order DPA Security

The tree construction presented in the previous section provides memory confidentiality and authenticity in the presence of a first-order side-channel adversary. However, profiled, second-order attacks as outlined in Chapter 5 still reveal the content of the tree nodes protected by the means of re-keying. Since the loss of confidentiality of a node close to the root would also reveal large chunks of the protected memory, i.e., all child nodes, protection against higher-order DPA is desirable.

In this section, we propose masking of the plaintext values to extend the protection of Meas to higher-order DPA. The extension works with cryptographic primitives implemented without DPA countermeasures and allows to dynamically adjust the protection order depending on the actual threat.

7.4.1 Concept

The basic idea to provide higher order DPA security is to add a masking scheme (cf. Section 2.2.1) to Meas. However, unlike the masking of specific cryptographic implementations, the proposed data masking scheme operates with unprotected primitives. Therefore, the plaintext data in each tree node of Meas is first masked, and then the masked plaintext and the masks are encrypted separately and both stored in memory. On decryption, both the masked plaintexts and the masks are decrypted and the masks applied to obtain the original plaintext value.

The masking scheme requires new masks to be chosen whenever the key of a tree node is changed. This is the case on every write access to a specific node. As a result, the data being encrypted is randomized. This prevents that constant data is encrypted under different keys. Moreover, it requires adversaries trying to learn a constant plaintext using profiled side-channel plaintext-recovery attacks such as described in Section 5.1.2 to additionally extract information on every single mask from the side-channel. Therefore, the order of the attack increases accordingly.

7.4.2 Masking Details

The following masking approach can be applied accordingly to both the intermediate nodes, which use an encryption scheme ENC, and the leaf nodes, which use an authenticated encryption scheme AE. However, for simplicity we only consider the encryption of an arbitrary tree node using an encryption scheme ENC.

When encrypting a tree node in Meas, the node's plaintext P is split into $u+1$ blocks $P_0, ..., P_u$ according to the size of the underlying encryption primitive, i.e., 128 bits in case of AES. In order to protect this node against d-th order DPA, $d-1$ random and secret masks $M_0, ..., M_{d-2}$ have to be generated. These masks are then applied to each plaintext block P_i to give random values R_i:

$$R_i = P_i \oplus M_0 \oplus ... \oplus M_{d-2} \qquad\qquad 0 \leq i \leq u.$$

In the actual encryption, both the masks $M_0, ..., M_{d-2}$ and the random values $R_0, ..., R_u$ are processed and the respective ciphertext C is stored in memory:

$$c = \mathrm{ENC}(DEK; M_0||...||M_{d-2}||R_0||...||R_u).$$

Whenever the node has to be read, the ciphertext is decrypted to give $M_0||...||M_{d-2}||R_0||...||R_u$. To obtain the plaintext blocks P_i, the masking is reverted by again XOR-ing all masks $M_0, ..., M_{d-2}$ to each block R_i.

7.4.3 Side-Channel Discussion

The re-keying of the (authenticated) encryption scheme guarantees that adversaries are not capable of building suitable DPA power models from the observation of ciphertexts and thus prevents DPA against the key completely.

To prevent the loss of plaintext confidentiality from the profiled, second-order attacks outlined in Chapter 5, the proposed masking scheme randomizes the plaintext input using $d-1$ random, secret masks. As a result, the scheme requires adversaries to combine side-channel information from $(d+1)$ different values to recover the plaintext, i.e., to perform a $(d+1)$-th order DPA. In particular, such DPA requires to learn side-channel information on the varying key, an intermediate value in the cipher, and the $d-1$ masks. On the other hand, the masking scheme requires to additionally encrypt $d-1$ masks in each tree node. However, for a properly chosen encryption scheme ENC, these encryption operations cannot be exploited in a DPA, because both the masks and the keys are random and always changed simultaneously on every write access to the respective tree node.

Note, however, that in order for the masking to protect MEAS also in the presence of hardware glitches, the sum of plaintext and the masks must be stored in a register prior to the encryption operation. This is automatically the case if the masking is implemented in software. Hereby, the result is stored in a register and may then, e.g., be further processed in a cryptographic hardware accelerator.

Besides, we also emphasize that profiled DPA attacks such as in Chapter 5—which are counteracted by the proposed masking scheme—are quite hard to conduct on state-of-the-art systems. For example, while the profiled side-channel plaintext-recovery attack in Chapter 5 was performed against a software implementation on an 8-bit microcontroller, a profiled DPA will take significantly more effort on hardware implementations embedded in a complex system-on-chip. Moreover, the attack complexity also rises rapidly with the attack order. As a result, small protection orders will already be sufficient for MEAS in practice. However, a detailed analysis of the side-channel leakage of a device implementing MEAS is indispensable for a proper choice of the protection order.

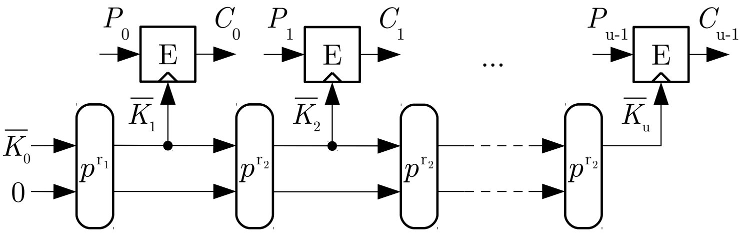


Figure 7.2: Schematic overview of ENC in Meas-v1.

7.4.4 Implementation Aspects

The definite choice of the implemented protection order allows for various trade-offs influenced by several parameters: the cost for storing the masks, the concrete leakage behavior of the device, and the risk. Hereby, the leakage behavior and the cost for storing the masks are closely coupled.

A DPA is more likely to be successful on a device the more side-channel leakage the device gives. Therefore, a higher protection order is needed the more the device leaks, which leads to higher storage costs for masks. Alternatively, the leakage of the device might be reduced by hiding countermeasures [MOP07] in the implementation, such as shuffling. However, such countermeasures can only be built into newly designed devices. Nevertheless, besides the actual strength of a potential attacker, the actual leakage behavior of the device forms the basis for the choice of the protection order and thus memory cost.

Besides, the choice of the protection order is also strongly influenced by the concrete risk of an attack. In more detail, a trade-off between the protection order and the risk is possible. Namely, the higher the risk of an attack to a specific block, the better should be the protection of the respective block, i.e., the higher should be the protection order. Concretely in Meas, the tree nodes stored in levels closer to the root are a more interesting target for an attacker since revealing the keys stored in these nodes would allow to decrypt large parts of the memory. Therefore, tree nodes closer to the root are at higher risk and thus need a higher protection order. However, the number of nodes in one tree level decreases the closer the respective level is to the root. As a result, increasing the protection order for tree nodes at higher risk has only little memory overhead in Meas and thus is an inexpensive improvement of security against higher-order DPA.

7.5 Instantiation

The design of Meas requires an SPA-secure block encryption scheme ENC and an SPA-secure authenticated encryption scheme AE. Using existing proposals of leakage-resilient block encryption [TS15] and a leakage-resilient MAC [PSV15], both ENC and AE can be easily obtained from unprotected cryptographic imple-

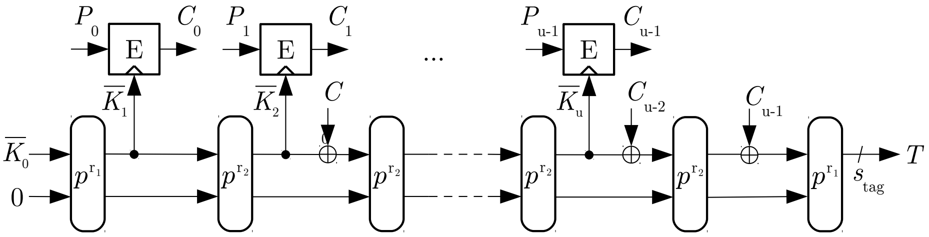


Figure 7.3: Schematic overview of AE in Meas-v1.

mentations of standard primitives like AES and SHA-2 and the generic composition encrypt-then-MAC [BN08]. However, for the encryption and authentication of RAM, more lightweight constructions for ENC and AE are desirable.

In this section, we present two lightweight Meas instances for the purpose of RAM encryption and authentication. The first, Meas-v1, uses the lightweight block cipher PRINCE for encryption, and the sponge Ascon for key stream generation and authentication. As a result, Meas-v1 uses a fresh key for the encryption of each plaintext block to prevent DPA on the key. The second, Meas-v2, improves on Meas-v1 in terms of efficiency in trade for a slightly increased number of, e.g., 4 or 8, different inputs processed under the same key. In particular, it omits key derivation in intermediate tree nodes to instead directly access the encrypted keys required for the next tree level using the tweakable block cipher QARMA. The security of Meas-v2 thus relies on the infeasibility of DPA for 4- or 8-limited data complexity per key.

7.5.1 Meas-v1

Our instance Meas-v1 is intended for RAM encryption and authentication and constructs ENC and AE by combining two different primitives: a lightweight block cipher E for encryption, and an r-round permutation p^r for sponge-based key derivation and authentication. While ENC uses the sponge merely for key stream generation, the sponge duplex construction [Ber+11a] is used in AE to also absorb the computed ciphertext and to compute the tag. Algorithm 1 gives the description of the respective algorithms. Their schematic is illustrated in Figure 7.2 and Figure 7.3, respectively. Since Meas applies (authenticated) encryption to message blocks of fixed, well-defined length, we describe ENC and AE without a padding rule and assume the messages to be a multiple of the s_{BC}-bit block size of E. Note that for optimization, AE absorbs the ciphertexts C_i with some delay. This allows to compute the permutation p^{r_2} and the encryption E in parallel.

We use PRINCE [Bor+12] as the block cipher E and the Ascon permutation [Dob+16] with $r_1 = 8$ and $r_2 = 6$ rounds for the sponge. PRINCE uses a key of $s_{key} = 128$ bits to process blocks of $s_{BC} = 64$ bits and the Ascon state S is sized $s_p = 320$ bits. These parameters allow to implement both ENC and AE

Algorithm 1: Specification of MEAS-v1.

Encryption: ENC($\overline{K}_0; P$)

Input: key $\overline{K}_0 \in \{0,1\}^{s_{key}}$, plaintext $P \in \{0,1\}^*$
Output: ciphertext $C \in \{0,1\}^*$

```
$P_0, \ldots, P_{u-1} \leftarrow s_{BC}$-bit blocks of $P$
$S \leftarrow \overline{K}_0 || 0^{s_p - s_{key}}$
$S \leftarrow p^{r_1}(S)$
for $i = 0, \ldots, u-1$ do
    $\overline{K} \leftarrow S[0 \ldots s_{key} - 1]$
    $C_i \leftarrow E(\overline{K}; P_i)$
    $S \leftarrow p^{r_2}(S)$
end for
return $C_0 || \ldots || C_{u-1}$
```

Authenticated Encryption: AE($\overline{K}_0; P$)

Input: key $\overline{K}_0 \in \{0,1\}^{s_{key}}$, plaintext $P \in \{0,1\}^*$
Output: ciphertext $C \in \{0,1\}^*$, tag $T \in \{0,1\}^{s_{tag}}$

```
$P_0, \ldots, P_{u-1} \leftarrow s_{BC}$-bit blocks of $P$
$S \leftarrow \overline{K}_0 || 0^{s_p - s_{key}}$
$S \leftarrow p^{r_1}(S)$
for $i = 0, \ldots, u-1$ do
    $\overline{K} \leftarrow S[0 \ldots s_{key} - 1]$
    $C_i \leftarrow E(\overline{K}; P_i)$
    if $i \neq 0$ then
        $S \leftarrow (S[0 \ldots s_{BC} - 1] \oplus C_{i-1}) || S[s_{BC} \ldots s_p - 1]$
    end if
    $S \leftarrow p^{r_2}(S)$
end for
$S \leftarrow p^{r_1}((S[0 \ldots s_{BC} - 1] \oplus C_{u-1}) || S[s_{BC} \ldots s_p - 1])$
$C \leftarrow C_0 || \ldots || C_{u-1}$
$T \leftarrow S[0 \ldots s_{tag} - 1]$
return $(C, T)$
```

Algorithm 2: Specification of ENC in MEAS-v2.

```
Encryption: ENC($\overline{K}_0; P$)
Input: key $\overline{K}_0 \in \{0,1\}^{s_{key}}$, plaintext $P \in \{0,1\}^*$
Output: ciphertext $C \in \{0,1\}^*$
  $P_0, \dots, P_{u-1} \leftarrow s_{BC}$-bit blocks of $P$
  for $i = 0, \dots, u-1$ do
    $C_i \leftarrow E(\overline{K}_0; addr(P_i); P_i)$
  end for
  return $C_0 || \dots || C_{u-1}$
```

with adequate throughput and low latency in hardware. The size of the tag s_{tag} can be chosen according to the desired security level, e.g., $s_{tag} = 64$ or 128 bits.

Both ENC and AE guarantee the use of each key for the encryption of a single plaintext P_i and for any tree node size. Note however that an implementation must only decrypt those plaintext parts within intermediate tree nodes (i.e., keys) that are actually needed for accessing the requested data block in the leaf. Namely, these keys become authenticated when accessing the leaf node which allows to detect malicious modifications of these keys in memory. On the other hand, decrypting keys that are not used any further allows attackers to modify the respective keys' ciphertext and thus to induce a DPA setting without being detected [Dob+17].

7.5.2 Meas-v2

One performance bottleneck of MEAS-v1 is the sequential key derivation within a tree node. On the other hand, direct access to a certain key within an intermediate tree node can significantly increase performance. By relaxing the constraints for DPA security, direct access to certain keys within a specific tree node becomes feasible.

For this purpose, we construct ENC using a tweakable block cipher. This allows to efficiently en-/decrypt parts of a tree node similar to Electronic Code Book (ECB), but provides better security in terms of ciphertext distinguishability. Given a tweakable block cipher $E(K; \tau; P)$ that encrypts a s_{BC}-bit plaintext P with the key K and tweak τ, a tree node comprising u plaintext blocks $P_0, ..., P_{u-1}$ is thus encrypted by simply computing $E(K; addr(P_i); P_i)$ for $i = 0, ..., u-1$, where the tweak τ is set to be the address of the respective P_i in memory. This is summarized in Algorithm 2.

On the other hand, we keep the design of AE in MEAS-v2 the same as in MEAS-v1. However, to avoid the implementation of another cipher for the use in AE, we recommend using the same tweakable cipher $E(K; \tau; P)$ in AE as well and setting the tweak τ in AE to either the block address or a constant. As the tweakable

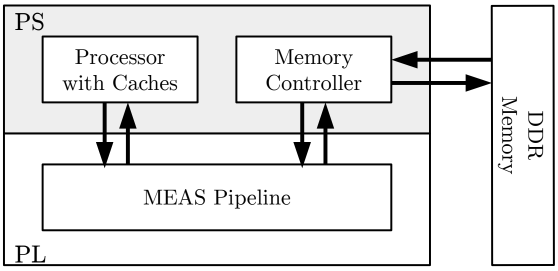


Figure 7.4: Zynq platform with MEAS pipeline.

block cipher $E(K; \tau; P)$, we use the lightweight design QARMA-64 [Ava17] with the parameter $r = 6$.

In terms of DPA, the mentioned approach increases the number of different inputs processed using a single key according to the number of plaintext blocks u in a tree node. However, for many practical implementations DPA will remain infeasible also for, e.g., 4 or 8, different encryptions using the same key. This assumption facilitates the efficient and secure implementation of MEAS-v2 for, e.g., binary and 4-ary trees.

7.6 Implementation

The two lightweight instances MEAS-v1 and MEAS-v2 are designed for RAM encryption and authentication. In order to show their practical applicability to this use case, an implementation allowing the evaluation of performance and implementation cost is desirable. In this section, we thus present an implementation of both MEAS instances on the Xilinx Zynq platform.

7.6.1 Platform

For the implementation, we chose a ZedBoard featuring the Xilinx Zynq XC7Z020 SoC and 512 MB DDR3 RAM. This SoC consists of two parts: (1) a Processing System (PS) comprising a dual-core ARM Cortex-A9 processor as well as several peripherals, and (2) a Xilinx Artix-7 Programmable Logic (PL). The PS is connected to the PL via 32-bit Advanced Extensible Interfaces (AXI). The PL has access to the RAM via 64-bit AXI.

For memory encryption and authentication, we designed an encryption pipeline capable of MEAS that is placed in the PL. As shown in Figure 7.4, the software running on the ARM core is configured such that the processor accesses the main memory via the PL, where all accesses are transparently encrypted and authenticated using MEAS.

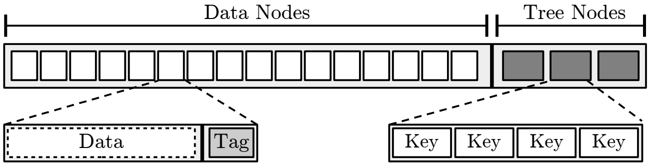


Figure 7.5: Memory layout for 4-ary MEAS.

7.6.2 Memory Layout

The implementation of MEAS requires to place all the tree nodes as well as their meta data somewhere in the RAM. For this purpose, and as shown in Figure 7.5, the physical memory is partitioned into two parts. In the first part, all data (leaf) nodes of MEAS are placed. These also contain their respective authenticity tags. The consecutive, second part contains all intermediate tree nodes storing the keys.

7.6.3 Address Translation

In order to provide the functionality of MEAS transparently to the CPU, a translation of the CPU's memory requests to the encrypted physical memory is required. Without consideration of tree node fetches, Figure 7.6 illustrates this CPU address translation. The CPU memory request is split according to the block size of the data (leaf) nodes. The MEAS implementation then issues independent requests to each of these data leaf nodes. Hereby, the size of the authentication tags is taken into account, which causes both an address shift and additional tags to be fetched.

However, the tree construction requires to also load several keys to decrypt a certain data (leaf) node. These key load operations are handled the same way as the requests to the data leaf nodes themselves. In particular, the MEAS implementation issues, translates, and processes the respective key load requests to intermediate tree nodes transparently without further CPU interaction and follows an address translation similar to data leaf nodes.

7.6.4 MEAS Pipeline

The pipeline architecture of our MEAS implementation is visualized in Figure 7.7. Its design results from the typical data flow in encrypted memory accesses. In particular, all requests run through a series of modules performing different actions. Hereby, the single modules interact by using a simple handshake mechanism. The width w of the respective data stream can be set to either 64 or 128 bits.

Our implementation communicates with CPU and memory via five different AXI4 interfaces: (1) the CPU address port, (2) the CPU read port, (3) the CPU write port, (4) the memory read port, and (5) the memory write port. Read requests use the modules shaded in light grey. Write requests are implemented

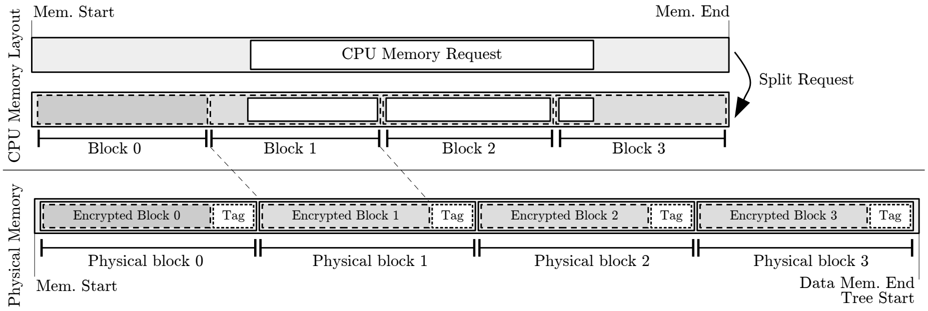


Figure 7.6: Data node requests for 4-ary Meas.

as read-modify-write operations and additionally use the modules depicted in dark grey. Dashed lines mark modules needed to process or optimize key related requests to intermediate tree nodes.

Data Flow

The implementation in Figure 7.7 processes a typical memory request as follows. First, the CPU issues a request on the `CPU Address Port`. The `Request Modifier` then splits and aligns the request according to the block size of the data (leaf) nodes. It further issues the respective key load requests within intermediate tree nodes. The `Memory Reader` fetches the required (encrypted) data from the main memory via the `Memory Read Port`. The `Key Injector` then inserts the key to be used for decryption into the data stream fetched from the memory. This key might either be a root key stored in the `Secure Root`, or be the result of a previous key load request that is obtained by the `Key Processing` module. The `Decryption` module performs the actual decryption procedure according to our instances in Section 7.5.

For key load requests, the requested key is extracted from the decrypted data using the `Key Processing` module. For read requests, the decrypted data is filtered according to the original CPU request by the `Data Filter` and returned to the CPU via the `CPU Read Port` by the `Read Responder`. To correctly handle CPU read requests with wrapping burst functionality, the `Wrap Burst Cache` performs a re-ordering of the decrypted data if necessary. For write requests, the `Data Modifier` modifies the decrypted data according to the data received from the CPU via the `CPU Write Port`. This is where the actual read-modify-write procedure takes place. The modified data is encrypted again using the `Encryption` module and written to the main memory via the `Memory Write Port` by the `Memory Writer`.

To improve the performance of the Meas pipeline, the `Secure Root` can implement an arbitrary number of roots to support multiple parallel trees in memory. Multiple roots effectively reduce both the tree height and the memory overhead in case more secure memory is available on the secure chip. To further

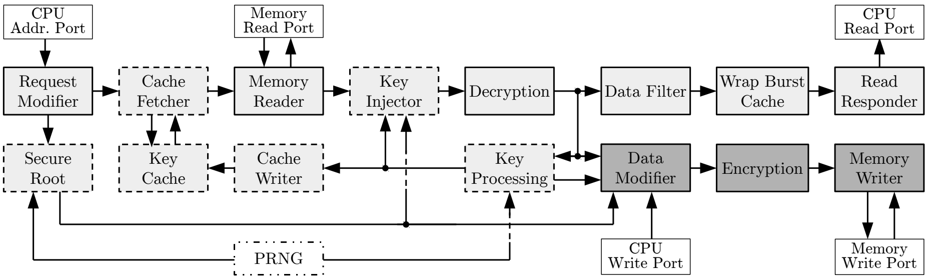


Figure 7.7: MEAS encryption and authentication pipeline.

improve the performance of read requests, the MEAS pipeline incorporates a `Key Cache` for faster key retrieval within the tree. For this purpose, the `Cache Fetcher` queries the cache for the key requested in a key load request. On a hit, the key load request is dropped and the key forwarded. Otherwise, the key load request is forwarded without modification. The `Key Cache` is filled using the `Cache Writer`, which receives the keys to be stored in the cache from the `Key Processing` module.

Re-Keying

Write requests in MEAS require the re-keying of all nodes on the path from the root to the respective data leaf node. This re-keying operation takes place in the `Secure Root` for the root keys themselves, and in the `Key Processing` module for non-root keys stored within the tree. In particular, besides filtering out the decryption keys from the decrypted data in key load requests, the `Key Processing` updates the respective keys during write requests. The new key is generated by the Pseudo-Random Number Generator (PRNG). This `PRNG` uses a Keccak[400] instance that is initialized with a random secret and that securely squeezes a secret, pseudo random stream. The freshly generated keys are provided to the `Data Modifier` to update the keys in the respective write requests and for encryption.

7.7 Evaluation

MEAS is a novel approach to provide authentic and confidential memory with DPA protection. While there already exist several concepts for memory encryption and authentication (cf. Section 7.1), all of them lack the consideration of side-channel attacks.

In this section, we compare MEAS with these state-of-the-art techniques regarding security properties, parallelizability, randomness, and memory overhead. Our methodology to assess the memory overheads is independent of any concrete implementation, precisely states the asymptotic memory requirements of all schemes, and is realistic for any real-world instance. In addition, we evaluate the practical performance of our MEAS implementation from Section 7.6 compared to TEC trees when encrypting RAM. It shows that MEAS efficiently provides first-order DPA-secure memory encryption and authentication at roughly the same memory overhead and performance as existing authentication techniques, which, on the other hand, completely lack the consideration of DPA at all. In particular, the 4-ary instance of MEAS-v2 is a highly suitable choice for DPA-secure encryption and authentication of RAM.

7.7.1 Security Properties

Comparing the contestants in Table 7.1 regarding security properties shows that only MEAS and TEC trees provide both confidentiality and authenticity in the form of spoofing, splicing and replay protection. DPA security, on the other hand, is only featured by MEAS and Merkle trees. However, Merkle trees do not provide confidentiality and their DPA security can be considered a side effect. Namely, the hash functions used in Merkle trees simply do not use any secret material, i.e., keys or plaintexts, which is the common target in DPA attacks.

7.7.2 Parallelizability

A more performance oriented feature, on which previous tree constructions typically improved on, is the ability to compute the cryptographic operations involved in read and write operations in parallel. Having this property is nice in theory, but is in practice not the deciding factor to gain performance. To make use of a scheme's parallelism, multiple parallel implementations of the cryptographic primitives as well as multi-port memory, to read and write various nodes in parallel, are required. Since these resources are typically not available, a common, alternative approach to improve performance is the excessive use of caches.

Table 7.1: Comparison of MEAS with other constructions for scalable authentic and/or confidential memory which offer block wise random access.

	Auth.	Conf.	DPA Security	Parallel[a]		Memory Overhead
				Read	Write	
MEAS	✓	✓	✓			$\frac{a}{a-1} \cdot \frac{s_{key}}{s_b} + \frac{s_{tag}}{s_b}$
PAT	✓			✓	✓	$\frac{a}{a-1} \cdot \frac{s_{tag}+s_{nonce}}{s_b}$
TEC Tree	✓	✓		✓	✓	$\frac{a}{a-1} \cdot \frac{s_{tag}+s_{nonce}}{s_b}$
Merkle Tree	✓		✓	✓		$\frac{a}{a-1} \cdot \frac{s_{hash}}{s_b}$

---

[a] Requires multiple cryptographic implementations and multi-port memory in practice.

In Meas, due to the key encapsulation approach used to achieve its DPA security, parallelizing the computations within the encryption scheme is not possible. However, this is not necessarily a problem preventing the adoption of Meas in practice since on-chip computation is very fast compared to off-chip memory accesses. Additionally, like for all authentication trees, caches for intermediate nodes are a very effective and important measure to reduce the average latency. In summary, the performance of any authentication tree (and Meas) is mainly determined by the tree height, which depends on both the tree arity and the number of blocks in the authenticated memory, and the cache size. As a result, given a concrete implementation of the cryptographic primitive, the actual runtime performance of all authentication trees is expected to be quite similar, which is also emphasized by the implementation results following in Section 7.7.6.

7.7.3 Memory Overhead

Table 7.1 further contains the memory overhead formulas that have been derived for each scheme. These formulas take into account the tree arity a, and the sizes for data blocks s_b, nonces s_{nonce}, hashes s_{hash}, tags s_{tag}, and keys s_{key}. The overhead formulas neglect the influence of the actual number of data blocks n_b given that it vanishes with rising node counts. The overheads therefore have to be considered as an upper bound which gets tight with $n_b \to \infty$. This approach gives exact and comparable results that are independent of the actual implementation and that are realistic for any memory with more than 128 data blocks.

The different parameters involved may make the overhead comparison seem difficult at first glance. However, it gets quite simple when actual instantiations are considered. Instantiating the trees for a fixed security level with $s_{nonce} = s_{tag} = s_{key}$ and $s_{hash} = 2 \cdot s_{tag}$, for example, shows that Merkle trees, PATs, and TEC trees have identical overhead. The overhead of Meas, on the other hand, is even lower, especially with small arity. This is due to the fact that in Meas only leaf nodes are directly authenticated. On the other hand, PATs and TEC trees directly protect the authenticity of every tree node.

The memory overhead of Meas, PATs, Merkle trees, and TEC trees is also visualized in Figure 7.8 for different block sizes. For practical instantiations, the block size will be chosen according to the system architecture, namely, page size, sector size, or cache line size. Both the sectors of modern disks as well as memory pages in state-of-the-art systems are sized 4 096 bytes (=32 768 bits). Such large block size is out of scope of Figure 7.8 as it has negligible memory overhead in any case. Besides, the memory overhead for a block size of 4 096 bits (sector size in older hard disks) is also very low, e.g., 7.3% for 4-ary Meas. However, the memory overhead of Meas for block sizes fitting nowadays cache architectures is also practical given the security features it provides. While today's typical cache line size is 512 bits, modern CPUs often come with features such as Adjacent Cache Line Prefetch [Int16], which effectively double the cache line fetches from memory to 1024 bits. In a 4-ary Meas, for example, such block size results in decent 29.2% memory overhead.

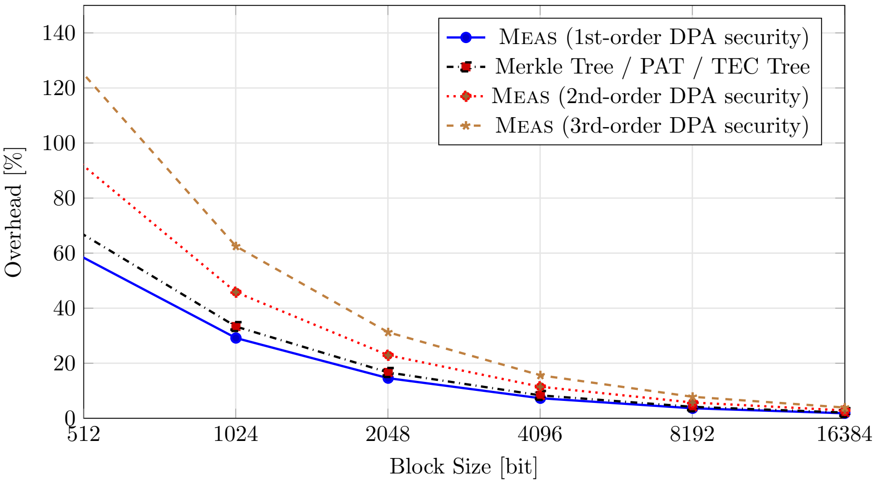


Figure 7.8: Memory overhead comparison for 4-ary trees depending on protection order and block size with a security level of 128 bits ($a = 4$, $s_{nonce} = s_{tag} = s_{key} = 128$, $s_{hash} = 256$).

Note that these relatively small overheads—quite similar to existing authentication techniques—in combination with additional and exclusive DPA protection are the main advantage of MEAS. Using existing memory encryption and authentication schemes with DPA-protected implementations, on the other hand, would result in overheads of at least a factor of four [Bil+14; Mor+11b; PSV15; BGS15] and thus be far more expensive, eventually rendering memory encryption and authentication in many applications impractical.

7.7.4 Memory Overhead with Masking

The memory overhead of MEAS with higher-order DPA protection additionally depends on the protection order d and the size of the masks s_{mask}. This size s_{mask} typically equals the block size of the cryptographic primitive s_{state}. For a tree with n_b data leaves and n_{inter} intermediate tree nodes, the memory overhead O_{inter} attributed to the intermediate tree nodes is

$$\begin{aligned} O_{inter} &= \frac{1}{n_b \cdot s_b} \cdot n_{inter} \cdot (a \cdot s_{key} + (d-1) \cdot s_{mask}) \\ &= \frac{1}{n_b \cdot s_b} \cdot \frac{1-n_b}{1-a} \cdot (a \cdot s_{key} + (d-1) \cdot s_{mask}), \end{aligned} \tag{7.1}$$

since $n_{inter} = (1-a^{\underline{l}})/(1-a)$ and $n_b = a^{\underline{l}}$ when the tree of height $\underline{l}$ is full. The memory overhead O_{leaf} attributed to the data leaf nodes is as follows:

$$O_{leaf} = \frac{s_{tag} + (d-1) \cdot s_{mask}}{s_b} \tag{7.2}$$

From Equation 7.1 and Equation 7.2, we can derive the limit of the memory overhead as the number of memory blocks approaches infinity:

$$\lim_{n_b \to \infty} O_{inter} + O_{leaf} = \frac{a}{a-1} \cdot \frac{s_{key} + (d-1) \cdot s_{mask}}{s_b} + \frac{s_{tag}}{s_b}.$$

In addition to the memory overhead without masking, Figure 7.8 shows the memory overhead with masking for a 4-ary tree and 128-bit security, i.e., the keys, the tags, and the masks are sized 128 bits. It shows that masking adds multiplicatively to the memory overhead for all block sizes. However, for larger block sizes, the memory overhead of MEAS becomes negligible regardless of the protection order. Note that the protection order stated for MEAS in Figure 7.8 applies to all nodes in MEAS. If however, and as explained in Section 7.4.4, different protection orders are used for nodes at different risk, the depicted plots mark the border cases for the actual memory overhead. For example, if low-level tree nodes do not use masking (i.e., having first-order DPA security) and first-order masking is applied to all other nodes (i.e., having second-order DPA security), the actual memory overhead is lower- and upper-bounded by the plot with first- and second-order protection, respectively.

An evaluation of the memory overhead of MEAS over different protection orders and arity is depicted for 1024-bit blocks and 128-bit security in Figure 7.9. Hereby, it turns out that the memory overhead is strongly influenced by the tree's arity leading to two main observations. First, a higher arity clearly lowers the memory overhead, but for an arity higher than eight, the reduction resulting from another increase of the arity becomes quite small. Second, the memory overhead rises linearly with the protection order, but the increase is stronger the lower the tree's arity is. This is due to the masks for randomization of the plaintext being chosen and stored for each tree node. As a result, higher arity leads to more plaintext blocks sharing such masks in one tree node and thus lower memory overhead due to the masking.

7.7.5 Randomness

MEAS consumes a considerable amount of randomness. In particular, fresh random keys and masks must be chosen for all nodes on the path from the root to the leaf whenever a write operation is performed. For MEAS with protection order d, this sums up to $(s_{key} + (d-1) \cdot s_{mask}) \cdot (\underline{l}+1)$ random bits needed on each write operation, where $\underline{l}$ is the tree height. Implementations of Merkle trees, PATs and TEC trees without consideration of side channels however do not require any random value if all nonces are chosen as counters. Yet, cipher implementations that protect PATs and TEC trees against side-channel attacks rely on significant amounts of randomness too. Namely, implementations with protection order d split their state into $(d+1)$ shares. This demands for at least $d \cdot s_{state}$ random bits per cipher invocation that get necessary for all accessed nodes on both reads and writes. Contrary to that, MEAS does not require randomness during read accesses.

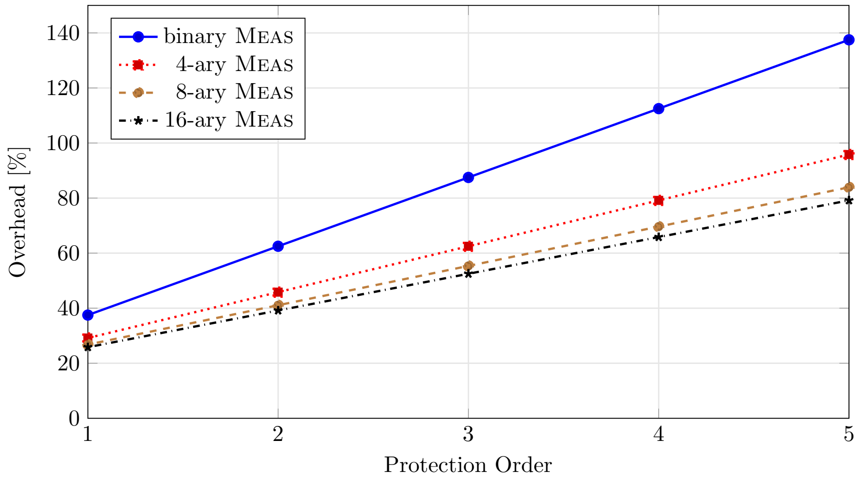


Figure 7.9: Memory overhead of Meas depending on arity and protection order (1024-bit blocks, 128-bit security).

7.7.6 Implementation Results

We extensively evaluated the performance of our Meas implementation from Section 7.6. In particular, we ran both Meas-v1 and Meas-v2 on the Digilent ZedBoard using different tree arities. As a state-of-the-art reference, we further implemented and ran a variant of TEC trees with different arities based on the same architecture as given in Figure 7.7. These TEC trees use Ascon [Dob+16] for authenticated encryption.

The evaluations for TEC trees were done with 64-bit counters (nonces) and 64-bit tags, which is a common instance for TEC trees in RAM [Gue16]. For our evaluations of Meas, we used a side-channel protection order $d = 1$ and 128-bit keys. Besides, we operated Meas with 128-bit tags as 64-bit tags only gave negligibly better results. Another relevant evaluation parameter is the data block size s_b. A suitable choice for s_b typically is the processor's cache line size. While the cache of the ARM Cortex-A9 processor on the ZedBoard's Xilinx XC7Z020 SoC features 256-bit cache lines, we configured the cache to always fetch 512-bits from memory by enabling the double line fill feature [ARM]. For this reason, both Meas and the TEC tree use a data block size of $s_b = 512$ bits. To speed up our designs, we made use of 1024 root keys (or root nonces for TEC trees) and a cache with 1024 slots to store keys (or nonces, respectively).

All our implementations use the 32-bit GP0 AXI interface to the CPU and the 64-bit HP0 AXI interface to the memory. As a result, a natural choice for the width w of the internal data stream that connects the various modules in Figure 7.7 is 64 bits. For the TEC tree implementation, we hence set $w = 64$ bits. On the other hand, Meas operates heavily on 128-bit keys, which could make a 128-bit internal data stream more efficient. For this reason, we evaluated the

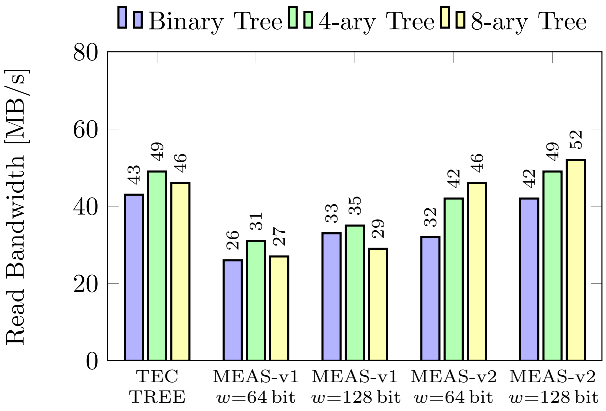


Figure 7.10: Read performance determined with tinymembench (`NEON read prefetched (64 bytes step)`).

performance impact of the internal data stream width by running both Meas-v1 and Meas-v2 with both $w = 64$ and $w = 128$ bit internal stream width.

In our evaluations, we booted Linux (Xilinx Linux kernel 4.4, tag 2016.2) [Xil16] in encrypted and authenticated memory, and measured the memory performance using a set of benchmarks. In particular, we executed tinymembench 0.3 [Sia13] and LMBENCH 3.0-a9 [SM07] for determining the memory bandwidth and latency, respectively. We performed these benchmarks for 256 MB of encrypted and authenticated memory provided to the ARM CPU and an FPGA clock frequency of 50 MHz. Note that at 50 Mhz, the 32-bit GP0 interface bounds the achievable memory bandwidth with 200 MB/s.

Memory Bandwidth

Figure 7.10 and Figure 7.11 show the read and the write memory bandwidth for all our designs and different tree arities. As mentioned before, we compare both Meas-v1 and Meas-v2 with 64- and 128-bit internal data stream width to our TEC tree implementation. As expected, it shows that Meas-v2 performs clearly better than Meas-v1 in terms of read bandwidth, yielding up to 52 MB/s. Meas-v2 only fetches and decrypts the actually required keys from within intermediate tree nodes and thus allows for faster read access. On the other hand, the write performance is generally lower and only a little better for Meas-v2 than for Meas-v1, because the re-keying step requires to read, modify, and re-encrypt full intermediate nodes in Meas-v2 as well. However, the slightly better write performance of up to 11 MB/s is caused by ENC lacking initialization and key derivation in Meas-v2. In terms of the internal data stream width, it shows that despite the 64-bit memory interface, the 128-bit internal data stream gives better results for both Meas-v1 and Meas-v2. This is mainly due to the instant

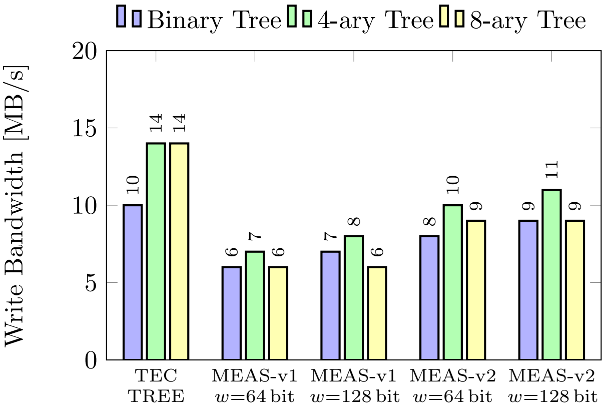


Figure 7.11: Write performance determined with tinymembench (`NEON fill`).

availability of the 128-bit keys from caches in the read case, and the faster processing of decrypted keys in the write case.

In terms of tree arity, 4-ary trees give the best write bandwidth for both MEAS and the TEC tree. As a closer investigation shows, an arity of four results in the least amount of data being processed when accessing a data block. Regarding read bandwidth, 4-ary trees give the best performance for TEC trees and MEAS-v1. However, for MEAS-v2 higher arity leads to higher read performance, as MEAS-v2 reduces the amount of data to be read from memory during read accesses by providing direct access to the keys within intermediate tree nodes.

Latency

Figure 7.12 shows the latency of all our MEAS designs and the TEC tree for different arities. As the main bottleneck of both memory bandwidth and latency is the processing of all the tree nodes, our latency results behave quite similar to the measured read bandwidth in Figure 7.10. In particular, MEAS-v2 offers clearly better latency than MEAS-v1 across all arities, namely down to 1315 ns (roughly 65 FPGA clock cycles), while the TEC tree behaves quite similar to MEAS-v2. For the TEC tree and MEAS-v1, an arity of four yields the lowest latency. However, for MEAS-v2 read accesses become faster the higher the arity is. As before, an internal data stream sized 128 bits yields lower latency than 64-bit streams.

Resource Utilization

Figure 7.13 shows the utilization of flip flops, Look-Up Tables (LUTs), and 36 KB Block RAMs (BRAMs) on the XC7Z020 SoC FPGA for MEAS and the TEC tree. In total, this XC7Z020 provides 106 400 flip flops, 53 200 LUTs, and 140 instances of 36 KB BRAMs. As the tree arity hardly influences hardware utilization, we focus on the results for 8-ary trees. These results show that all

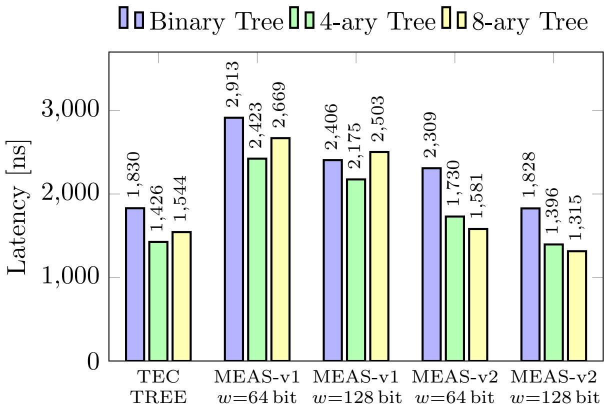


Figure 7.12: Memory latency determined with LMBENCH (`lat_mem_rd 8M`).

designs are dominated by logic, with an utilization of up to 35 % of LUTs, while the demand for flip flops and BRAMs stays below 10 %. Compared to the TEC tree, Meas consumes 60-80% more logic, because it implements a (tweakable) block cipher and a PRNG in addition to the Ascon permutation.

Comparison

Our evaluations indicate quite similar performance of TEC trees and Meas-v2. However, while Meas has higher implementation cost, the implemented TEC tree does not offer DPA protection. Moreover, note that our instance of Meas uses 128-bit keys and tags, but the TEC tree implementation operates with smaller 64-bit nonces and tags.

Summarizing our evaluation results and especially taking into account write performance and side-channel constraints, we conclude that Meas-v2 with arity four is a DPA-secure, highly practical, and hence suitable choice to encrypt RAM. However, as 4-ary Meas-v2 encrypts four 128-bit keys per intermediate tree node with a 64-bit cipher, 4-ary Meas-v2 relies on the assumption of DPA being infeasible given eight different encryptions per key. If DPA on such 8-limiting construction is considered feasible, binary Meas-v2 and 4-ary Meas-v1 are viable alternatives with solid performance results and only four and a single encryption per key, respectively.

7.8 Conclusion

Authentic and encrypted memory is a requirement for storing and processing data in hostile environments where attackers have physical access. The consideration of the imminent threat of side-channel attacks against the involved cryptographic primitives is thus the natural next step.

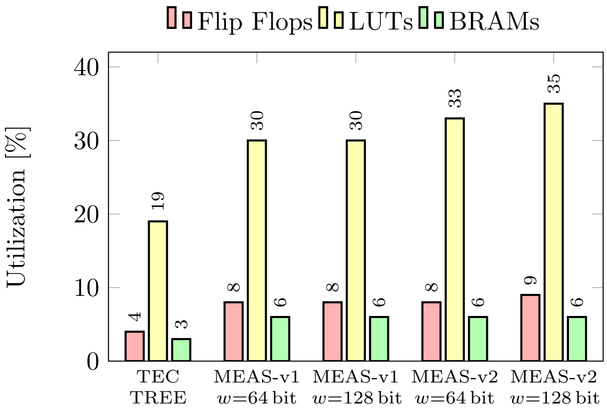


Figure 7.13: FPGA utilization on XC7Z020 for 8-ary trees.

In this chapter, we therefore presented MEAS, the first Memory Encryption and Authentication Scheme which is secure against DPA attacks. The scheme does not require any DPA-protected primitive, allowing its use in COTS systems. Moreover, MEAS provides fast random access on the configured block level and can be adopted for all kinds of use cases including RAM and disk encryption.

The scheme combines the concept of fresh re-keying with authentication trees by storing the involved keys in an encrypted tree structure. While this prevents first-order DPA, masking of the plaintext values flexibly extends the protection of MEAS to higher-order DPA if required. Compared to existing schemes, MEAS exclusively offers DPA protection by design at roughly the same memory overhead and performance. This is a clear benefit over state-of-the-art memory authentication and encryption techniques, which would face impractical implementation and runtime overheads for DPA-protected implementations if adapted accordingly.

Part III

Bilinear Pairings for Embedded Devices

While the previous parts focused on the security of Internet-of-Things (IoT) platforms when attackers have direct physical access, the increasing number of devices in IoT networks also is a severe risk to privacy. In this respect, modern cryptography offers several novel protocols that are useful for IoT scenarios, such as privacy-preserving group signatures [CH91; BBS04], attribute-based credentials [CL02], and identity-based encryption [Sha84; BB04]. These protocols are most commonly built from bilinear pairings, but computing bilinear pairings is complex and many IoT devices have tight resource constraints. As a result, the wide deployment of such protocols in the IoT is difficult. In addition, the tight binding of an identity to the encryption operation in identity-based encryption hampers revocation and the anonymity aspects of privacy-preserving protocols make misuse detection difficult. As a result, leaking sensitive key material via side-channel attacks is particularly critical in such schemes. In this part, we focus on these issues by the following main contributions.

- We make pairing-based cryptographic protocols available to constrained IoT devices. Based on the ARM Cortex-M0+, we present three hard-/software co-designs for pairing-based cryptography. These side-channel protected implementations are both lightweight and efficient enough to provide pairing-based protocols also in embedded applications with user interaction.

- We analyze the side-channel security of pairing-based protocols for unprotected implementations by presenting a Correlation Power Analysis (CPA) attack that extracts the secret key from identity-based encryption and promote several randomization techniques as suitable countermeasures.

8

Efficient Pairings and ECC for Embedded Devices

Pairing-based cryptography offers a wide range of protocols interesting for future Internet-of-Things (IoT) applications. However, one obstacle for the widespread deployment of pairing-based protocols are their high hardware and software requirements. As a result, there have been optimized pairing implementations for desktop computers [Ara+11; Beu+10], for smart phones [Gre+12; SR13], and as dedicated hardware modules [FVV09; Kam+09]. Yet, cost-sensitive IoT applications often do not have the budget for such powerful hardware components. Lightweight implementations on Reduced Instruction Set Computer (RISC) processors [Szc+08; GOL12; DSD07], on the other hand, offer impractical runtimes that range between between 1.9 and 17.9 seconds for a single pairing. Moreover, it is unclear to which degree timing-analysis, power-analysis, or fault-analysis attacks have been considered in all those implementations.

For this reason, together with Erich Wenger, we have implemented constant-runtime, side-channel protected optimal-Ate pairings using Barreto-Naehrig (BN) curves [BN05] on an ARM Cortex-M0+ [ARM14a; Atm13] and explored the design space for lightweight hardware extensions to make pairing-based protocols suit resource-constrained applications. We started this research during my master thesis [Unt13], continued our work during my PhD studies, and published our final results in [UW14a]. In this publication, I was the main author contributing most of the text and the implementation, whereas Erich Wenger added parts of the implementation and parts of the text. In this chapter, we use text and results from [UW14a] and make the following contributions.

Contribution. We present three reusable pairing platforms that offer runtimes of down to 162 ms and require 10.1 kGE of dedicated hardware at most – significantly less than similarly fast hardware implementations by related work. Our rigorous hardware/software co-design approach equipped one platform with a multiply-accumulate instruction-set extension and another platform with a drop-in accelerator[1] [Wen13]. By building a flexible, specially crafted drop-in module with several novel design ideas, we were able to improve the runtime of pairing and group operations up to ten times. This concept platform consisting of Central Processing Unit (CPU), Random Access Memory (RAM), Read-Only Memory (ROM), and drop-in module consumes merely 49 kGE of hardware in total with 10.1 kGE of those being spent for the drop-in accelerator. To show the practicality of this platform, we evaluated the performance of several high-level pairing protocols [BB04; BB08; Hwa+11] – each operating in significantly less than one second. To demonstrate its versatility, we further evaluated Elliptic Curve Cryptography (ECC) for `secp160r1`, `secp256r1` [Cer00; Nat09], and Curve25519 [Ber06], requiring 11.9-36.8 ms for a side-channel protected point multiplication. These results make the drop-in based platform highly suitable for embedded computing, smart cards, wireless sensor nodes, Near-Field Communication (NFC), and the IoT.

Outline. This chapter is structured as follows: Section 8.1 gives an overview on pairings and Section 8.2 covers the implementation aspects of the high-level pairing arithmetic. In Section 8.3, the architectural options to build suitable pairing platforms are presented. The respective platforms are evaluated in Section 8.4 and compared with related work in Section 8.5. The (re-)usability of our drop-in platform is content of Section 8.6. A conclusion is finally done in Section 8.7.

8.1 Background on Pairings

The wide range of cryptographic protocols in pairing-based cryptography is based on three cyclic order-n_G groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and a bilinear pairing operation. A bilinear pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ accepts an element of the two additive groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, maps these to the multiplicative group $\mathbb{G}_T$, and hereby fulfills several properties:

1. Bilinearity: $e(aP, bQ) = e(P, Q)^{ab} \;\; \forall P \in \mathbb{G}_1,\ Q \in \mathbb{G}_2,\ a, b \in \mathbb{Z}\,.$
2. Non-degeneracy: $\forall P \in \mathbb{G}_1 \setminus \{\mathcal{O}\}\ \exists\, Q \in \mathbb{G}_2 : e(P, Q) \neq 1\,.$
3. Computability: $e(P, Q)$ can be computed efficiently.

The groups $\mathbb{G}_1$, $\mathbb{G}_2$ are typically groups over elliptic curves and $\mathbb{G}_T$ is the subgroup of a large extension field. However, only certain elliptic curves allow the definition

---

[1] Wenger [Wen13] applied the concept to binary-field based elliptic-curve cryptography while we apply the concept to prime-field based elliptic-curve cryptography.

of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ with an admissible bilinear pairing, e.g., [BN05; MNT01]. In this chapter, we focus on the pairing-friendly elliptic curves by Barreto and Naehrig [BN05] of the form $\boldsymbol{E} : y^2 = x^3 + b$ with $b \neq 0$ (*BN curves*). Ate pairings $a(Q, P)$ based on these curves can be described as follows:

$$a\colon\ \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T\ :\ \boldsymbol{E}(\mathbb{F}_{p^{12}}) \times \boldsymbol{E}(\mathbb{F}_p) \to \mathbb{F}^*_{p^{12}}\,. \tag{8.1}$$

Note that for $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ to have the same prime order n_G, $\mathbb{G}_2$ and $\mathbb{G}_T$ need to be subgroups of $\boldsymbol{E}(\mathbb{F}_{p^{12}})$ and $\mathbb{F}^*_{p^{12}}$, respectively. The BN curves use a parameter u such that a desired security level is achieved. This allows the computation of the prime p and the prime group order n_G in dependence of u:

$$\begin{aligned} p(u) &= 36u^4 + 36u^3 + 24u^2 + 6u + 1 \\ n_G(u) &= 36u^4 + 36u^3 + 18u^2 + 6u + 1\,. \end{aligned}$$

As another benefit, BN curves possess an efficiently computable group homomorphism that exploits the curve's sextic twist $\boldsymbol{E}'$. Utilization of this homomorphism allows to compress elements in $\mathbb{G}_2$, which leads to a more efficient definition of the Ate pairing, namely

$$a\colon\ \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T\ :\ \boldsymbol{E}'(\mathbb{F}_{p^2}) \times \boldsymbol{E}(\mathbb{F}_p) \to \mathbb{F}^*_{p^{12}}\,. \tag{8.2}$$

The pairing a itself consists of the evaluation of a rational function $f_{\Lambda,Q}$ and a final exponentiation that maps all cosets to the same unique representative:

$$a = f_{\Lambda,Q}(P)^{(p^{12}-1)/n_G}.$$

Owing to the Frobenius homomorphism, the final exponentiation by $(p^{12} - 1)/n_G$ can be split into an easy part $(p^6 - 1)(p^2 + 1)$ and a hard part $(p^4 - p^2 + 1)/n_G$. The function $f_{\Lambda,Q}$ can in general not be evaluated directly. However, Miller [Mil04] described an important property of rational functions, namely

$$f_{i+j,P} = f_{i,P} f_{j,P} \frac{\ell_{[i]P,[j]P}}{\nu_{[i+j]P}}\,.$$

The property allows the computation of $f_{\Lambda,Q}$ in polynomial time by merely evaluating vertical (ν) and straight (ℓ) lines in elliptic curve points using a double-and-add approach. Values of Λ with low Hamming weight result in a particularly fast computation of $f_{\Lambda,Q}$, the pairing becomes optimal. In this work, we used the efficient optimal-Ate pairing by Vercauteren [Ver10].

8.2 High-Level Arithmetic

The computation of bilinear pairings over BN curves requires several layers of arithmetic. As illustrated in Figure 8.1, all arithmetic is based on a multi-precision integer arithmetic layer. On top of that, prime-field arithmetic and a tower of extension fields are built upon. The elliptic curve groups used as $\mathbb{G}_1$ and $\mathbb{G}_2$ utilize the prime field and its quadratic extension field, respectively. The largest extension field $\mathbb{F}_{p^{12}}$ is used by $\mathbb{G}_T$. The pairing computation itself is based on the groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, and their underlying field arithmetic.

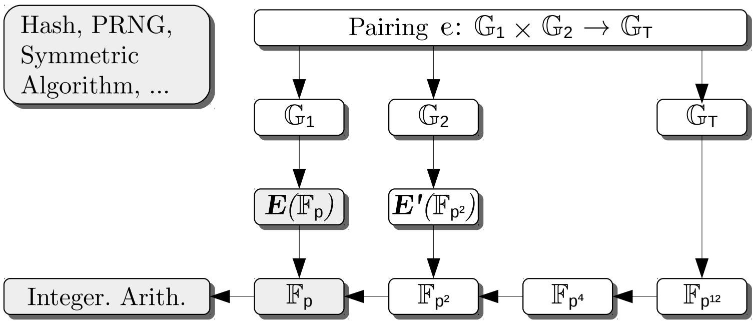


Figure 8.1: Arithmetic required for pairings over Barreto-Naehrig curves.

8.2.1 Implementation Details

The following gives an overview on the implementation details of our software implementation of bilinear pairings according to Figure 8.1.

Methodology

Our state-of-the-art implementations are based on the techniques used by Beuchat et al. [Beu+10] and Devegili et al. [Dev+06]. The pairing implementation uses the fast formulas by Costello et al. [CLN10], the inversion trick by Aranha et al. [Ara+11], a lazy reduction technique in $\mathbb{F}_{p^2}$ [Beu+10; SR13], and a slightly modified variant of the final exponentiation by Fuentes-Castañeda et al. [FKR11] that requires less memory. We will detail this optimization in Section 8.2.2. Similarly, the prime-field inversion using Fermat's little theorem is optimized according to Section 8.2.3. Apart from that, we use dedicated squaring formulas [GS10] for operations in $\mathbb{G}_T$ and during the hard part of the final exponentiation as these take place in the cyclotomic subgroup of $\mathbb{F}^*_{p^{12}}$. The point multiplications in both elliptic curve groups use Montgomery ladders that are based on fast formulas [HJS11] in homogeneous projective co-Z coordinates.

Parameters

As this work aims to offer a certain degree of flexibility, we decided to support two elliptic curves, namely, BN158 [GOL12] ($u = \mathtt{40\,00800023}_h$) and BN254 [Nog+08] ($u = -\mathtt{40800000\,00000001}_h$) of the form $y^2 = x^3 + 2$, which were crafted for the 80- and 128-bit security level[1], respectively. These lead to particularly fast execution times as the respective constants Λ of $f_{\Lambda,Q}$ have low Hamming weights.

---

[1] Note that recent results on solving the Discrete Logarithm Problem (DLP) in $\mathbb{G}_T$ [MSS16; KB16; BD17] suggest a reduced security level of roughly 100 bits for BN254. Unfortunately, such analysis is currently missing for BN158. However, while the embedding degree of BN curves and a prime of 158 bits so far implied DLP security of more than 80 bits in $\mathbb{G}_T$, we suspect that the new DLP solving techniques result in less than 80 bits of security for BN158 and hence refrain from using BN158 any further.

The extension field $\mathbb{F}_{p^2}$ is represented as $\mathbb{F}_p[i]/(i^2 - \beta)$ with $\beta = -1$. The extension field $\mathbb{F}_{p^{12}}$ is built as $\mathbb{F}_{p^2}[z]/(z^6 - \zeta)$, with $\zeta = (1 + i)$ for BN254 and $\zeta = \frac{1}{1+i}$ for BN158.

Implementation Attacks

An important aspect in the implementation of pairings and group arithmetic for embedded applications is the consideration of side-channel attacks. While scalar factors or exponents are typically the secret operands for operations in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, an elliptic curve point may have to be protected in the case of pairing operations.

As a countermeasure to timing attacks, all implemented algorithms have constant, data-independent runtime. Therefore, e.g., some fast but vulnerable point multiplication algorithms are not used. Both the point multiplications in $\mathbb{G}_1$, $\mathbb{G}_2$ and the exponentiations in $\mathbb{G}_T$ hence use Montgomery ladders. The implementation's countermeasures against first-order Differential Power Analysis (DPA) attacks comprise Randomized Projective Coordinates (RPC) [Cor99] in both the pairing computation and the point multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$. To detect fault attacks on data, point multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$ include several point verifications. DPA and fault attacks on exponentiations in $\mathbb{G}_T$ as well as fault attacks on pairings were also taken into consideration, but can better be handled on the protocol layer using randomization.

8.2.2 Optimized Final Exponentiation

The hard part of the final exponentiation by Fuentes-Castañeda et al. [FKR11] yields fast execution by reducing the number of multiplications and exponentiations in $\mathbb{F}_{p^{12}}$. As a drawback, it requires four large temporary variables in $\mathbb{F}_{p^{12}}$. In order to attain a low-memory implementation, we decreased the number of temporary variables by adapting their formulas without noticeably degrading performance. Therefore, we initially set $t_0 = f^p$ and compute the chain

$$f^u \to f^{2u} \to f^{4u} \to f^{6u} \to f^{6u^2} \to f^{12u^2} \to f^{12u^3} .$$

Following, a and b are set to $a = f^{6u} \cdot f^{6u^2} \cdot f^{12u^3}$ and $b = a \cdot (f^{2u} \cdot f)^{-1}$. The computation of the result, namely

$$\begin{aligned} f &= f^{6u^2} \cdot f \cdot f^p , \\ f &= [f \cdot a][b]^p[a]^{p^2}[b]^{p^3} , \end{aligned}$$

requires one more multiplication and one more Frobenius action than originally. However, the respective implementation in Algorithm 3 requires three temporary variables instead of four when the exponentiation and the multiplication on Line 16 are done simultaneously using a dedicated function. Since variables in $\mathbb{F}_{p^{12}}$ are large and RAM is more expensive than ROM, this approach aids to keep chip area low.

Algorithm 3 Memory-optimized hard part of the final exponentiation for pairings over BN curves.

```
Input: $f \in \mathbb{F}_{p^{12}}$
Output: $f^{\phi_{12}(p)/n_G} \in \mathbb{F}_{p^{12}}$
$t_0 \leftarrow f^p$
$b \leftarrow f^u$ if
if $u < 0$ then $b \leftarrow \overline{b}$   ▷ Conjugate
$b \leftarrow b^2$
$a \leftarrow b^2$
$a \leftarrow a \cdot b$
$b \leftarrow b \cdot f$
$b \leftarrow \overline{b}$
$f \leftarrow f \cdot t_0$
$t_0 \leftarrow a^u$ if
if $u < 0$ then $t_0 \leftarrow \overline{t_0}$
$f \leftarrow f \cdot t_0$
$a \leftarrow a \cdot t_0$
$t_0 \leftarrow t_0^2$ if
if $u < 0$ then $t_0 \leftarrow \overline{t_0}$
$a \leftarrow a \cdot t_0^u$   ▷ Interleaved
$b \leftarrow b \cdot a$
$t_0 \leftarrow b^p$
$t_0 \leftarrow t_0 \cdot a$
$t_0 \leftarrow t_0^p$
$t_0 \leftarrow t_0 \cdot b$
$t_0 \leftarrow t_0^p$
$t_0 \leftarrow t_0 \cdot f$
$f \leftarrow t_0 \cdot a$
return $f$
```

8.2.3 Optimized Prime-Field Inversion

The parameterized prime $p(u)$ facilitates an optimized exponentiation-based prime-field inversion for positive u that have low Hamming weight. In such cases, the inverse $a^{-1} \in \mathbb{F}_p$ can be expressed as

$$\begin{aligned} a^{-1} \bmod p = a^{p-2} \bmod p &= a^{36u^4+36u^3+24u^2+6u-1} \bmod p \\ &= a^{6u(4u+6u^2(1+u))} \cdot a^{6u-1} \bmod p\,. \end{aligned}$$

Precomputation of the constant $6u-1$ and the chain of computations

$$a^{6u-1} \rightarrow a^{6u} \rightarrow a^{12u^2} \rightarrow a^{24u^2} \rightarrow a^{36u^2} \rightarrow a^{36u^3} \rightarrow a^{36u^4}$$

enables the computation of the inverse as

$$a^{-1} \bmod p = a^{6u-1} \cdot a^{24u^2} \cdot a^{36u^3} \cdot a^{36u^4} \bmod p\,.$$

Consequently, prime field inversion is done using three fast exponentiations by u, one exponentiation by $6u-1$, five multiplications, and two squarings. Since the exponents are fixed and publicly known, Montgomery ladders are not required and runtime remarkably benefits from the low Hamming weight of u.

8.3 Hardware Architectures

To meet the high requirements of pairing-based cryptography in embedded devices, our goal was to equip a stand-alone microprocessor, designated for embedded applications, with a dedicated hardware unit such that (i) pairing computations

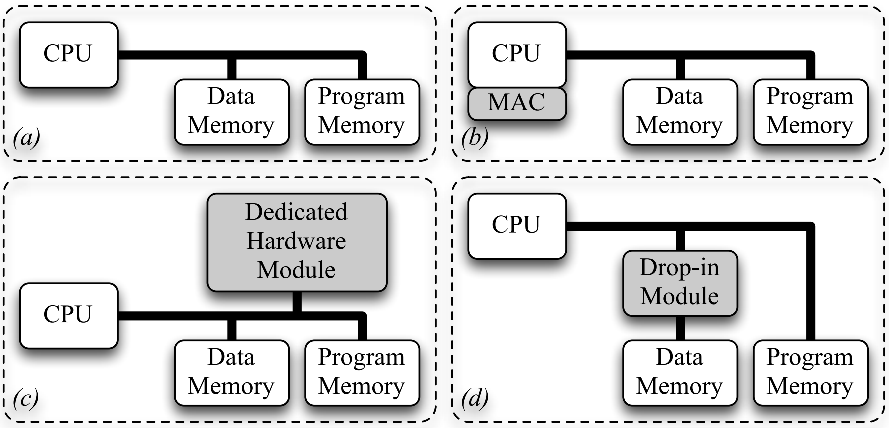


Figure 8.2: Architectural options for fast and flexible pairing designs.

are usable within interactive (e.g., authentication) protocols, (ii) a pre-existing microprocessor platform is modified only minimally, (iii) the overall hardware requirements, i.e., the costs, are kept small and considerably below 100 kGE needed in related work [FVV09; Kam+09], and (iv) embedded applications such as wireless sensor nodes and NFC should be practically feasible.

Figure 8.2 summarizes potential architectures that can be used to attain such goals. The straightforward solution *(a)*, a sole off-the-shelf microprocessor, requires minimal hardware-development time, however potentially delivers insufficient performance. The runtimes desirable for interactive protocols can only be achieved by either adding powerful, dedicated instructions *(b)*, or by adding dedicated co-processors. Contrary to a dedicated hardware module *(c)*, a drop-in module *(d)* is memoryless and requires neither a Direct Memory Access (DMA) controller nor a multi-master bus. Wenger [Wen13] showed the advantages of the drop-in concept in comparison to a dedicated hardware module for binary-field ECC. However, the applicability of this technique for prime-field based pairings is still an open question.

Following up the potential architectures, we consecutively evaluate the practicability of a plain microprocessor design *(a)*, a multiply-accumulate instruction-set extension *(b)*, and a dedicated drop-in module *(d)*.

8.3.1 The Used Microprocessor

The accomplishment of the initially set goals highly depends on the used microprocessor. As the runtime figures by Szczechowiak et al. [Szc+09] and Gouvêa et al. [GOL12] discourage the use of an 8-bit or 16-bit microprocessor, a 32-bit microprocessor is preferred as a basis. Moreover, the bottleneck between computation unit and RAM is less of an issue if 32-bit interfaces are used. We hence decided to utilize a self-built processor functionally equivalent to the ARM

Cortex-M0+ [ARM14a], because the Cortex-M0+ was especially designed for embedded applications and currently is one of the smallest 32-bit processors in production. The Cortex-M0+ has 16 32-bit general-purpose registers of which 8 are efficiently usable. It comes with a mixed 16/32-bit Thumb/Thumb-2 instruction set and optionally either a 32-cycle or single-cycle 32-bit multiplier. In its minimum configuration, ARM specifies its Cortex-M0+ to require only 12 kGE in a 90 nm process technology.

8.3.2 The Software Framework

The biggest advantage of an off-the-shelf microprocessor are the vast (open-source) toolchains. Thus a high-level framework capable of pairing-based cryptography using BN curves was created in C. It provides extension field arithmetic, elliptic curve operations, and bilinear pairings. The framework focuses on both good performance and low memory consumption. To achieve the latter, several optimizations were incorporated into the framework. First, virtually all of the memory is allocated on the stack. As stack variables are discarded at the end of each function, stack allocation facilitates the reduction of required memory by separating code into different functions. Second, allocated memory is re-utilized where possible. Third, memory-optimized algorithms are used, e.g., for the final exponentiation as in Section 8.2.2. Last, compiler optimizations are used to decrease the program size. Therefore, the compiler options `-ffunction-sections`, `-fdata-sections` and the linker options `-gc-sections`, `--specs=nano.specs` are passed to the bare-metal ARM GNU toolchain (version 4.7.4).

The high-level pairing framework is common to all three evaluated platforms. The main difference between these platforms is the implemented finite-field arithmetic. While *(a)* and *(b)* control the whole finite field arithmetic in software, *(d)* relies on finite-state machines to perform additions, subtractions and multiplications in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. Nevertheless, all implementation options ensure constant runtime and consider side-channel attacks.

8.3.3 Assembly-Optimized Software Implementation *(a)*

The plain microprocessor platform *(a)* is based on a Cortex-M0+ with a single-cycle multiplier. Its hand-crafted assembly routines for optimized prime-field arithmetic always perform a reduction step to ensure constant runtime. This is accomplished by storing the reduction result either to the target or a dummy memory location via masking of the operand addresses. The crucial prime-field multiplication utilizes an unrolled Separate Product Scanning (SPS) method of the Montgomery multiplication [Mon85] that is derived from [KAJ96]. The SPS variant is chosen because of the particular $\mathbb{F}_{p^2}$-multiplication technique [Beu+10; SR13] we use, which performs the required three multiplications and two reductions separately. Product scanning can further be efficiently implemented on the processor if three registers are used as an accumulator, as presented in [WUW13]. The reduction step for the curve BN254 is further optimized as several multiply-accumulate operations can be skipped due to the sparse prime [GOL12].

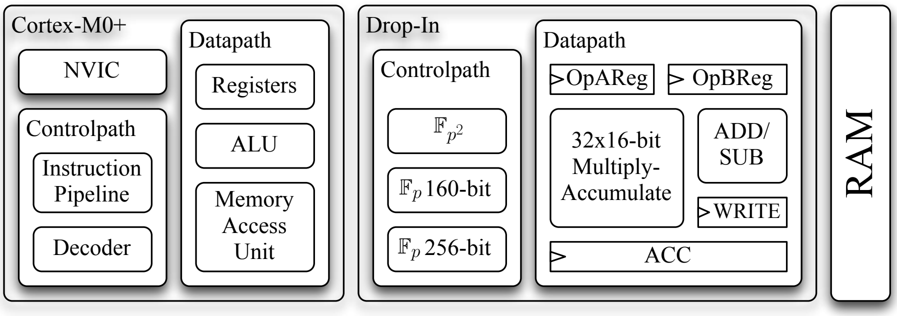


Figure 8.3: High-level representation of architecture *(d)* (without program memory). Note that the sizes of the blocks are not proportional to their respective hardware footprints.

8.3.4 Multiply-Accumulate Hardware Extensions *(b)*

The performance of the prime-field multiplication significantly suffers from the $32 \times 32 \rightarrow 32$ bit multiplier of the Cortex-M0+, which results in 80% of a pairing's runtime being spent in $\mathbb{F}_p$ multiplications. To improve this, the processor core is equipped in *(b)* with a multiply-accumulate extension similar to [WUW13]. It adds the result of a full $32 \times 32 \rightarrow 64$ bit multiplication to three accumulation registers in a single cycle. In order to avoid a modification of the compiler toolchain, the `TST` instruction, which is not required for prime-field multiplication, is reinterpreted as a multiply-accumulate instruction if a certain bit in the control register is set. The control register is manipulated accordingly at the beginning and the end of a prime-field multiplication. Besides accelerated multiply-accumulate operations, the prime-field multiplication requires less registers for temporary variables, which we exploit by caching some of the operand words in the product scanning routine.

8.3.5 The Drop-in Module *(d)*

As a consequence of the high-level runtime and area goals, it is highly important to maximize the utilization of the invested chip hardware. To achieve this, a lightweight hardware drop-in accelerator is placed between processor and data memory. The respective design, which is shown in Figure 8.3, uses a Cortex-M0+, but any other processor is equally suitable.

The drop-in module provides unrolled state machines and an appropriate arithmetic unit for 160-bit and 256-bit $\mathbb{F}_p$ multiplication, $\mathbb{F}_p$ addition and $\mathbb{F}_p$ subtraction. It further encompasses state machines to control $\mathbb{F}_{p^2}$ addition, $\mathbb{F}_{p^2}$ subtraction, $\mathbb{F}_{p^2}$ multiplication and $\mathbb{F}_{p^2}$ squaring. Several memory-mapped registers are used to control the drop-in module. A lightweight arbiter is built in which always gives preference to the CPU when the CPU wants to access the data memory. In such case, the drop-in module is prepared to stall its operation.

Table 8.1: Propagation of data within the pipelined drop-in module

Bus	`OpBReg`	`OpAReg`	**Mult.**	**Accum.**
`LD OpB+0`				
`LD OpA+0`	`WR`			
`LD OpB+0`	`SH`	`WR`		
`LD OpA+1`	`WRSH`		`MUL1`	
`LD OpB+1`	`SH`	`WR`	`MUL2`	`SHIFT`
`LD OpA+0`	`WRSH`		`MUL1`	
`LD OpB+2`	`SH`	`WR`	`MUL2`	
`ST RES+0`	`WRSH`		`MUL1`	
`LD OpB+1`	`SH`		`MUL2`	`SHIFT`
`LD OpA+1`	`WRSH`		`MUL1`	
`LD OpB+0`	`SH`	`WR`	`MUL2`	

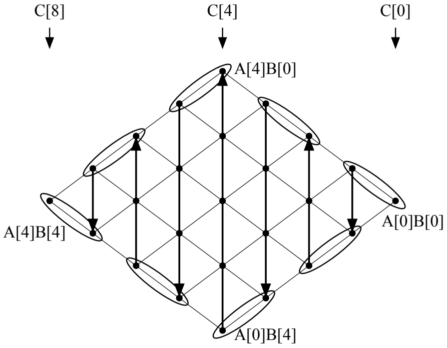


Figure 8.4: 5 × 5-word zig-zag product scanning multi-precision multiplication method.

The core element of our drop-in module is a multiply-accumulate unit that is used to perform a Finely Integrated Product Scanning (FIPS) [KAJ96] Montgomery multiplication. For $n_w = \lceil \frac{\mathrm{ld}(p)}{w} \rceil$ w-bit words, this algorithm approximately performs $2n_w^2 + n_w$ w-bit integer multiplications that require roughly $4n_w^2$ load operations. However, instead of using a dual-port memory, we perfectly utilize bus and multiplier by using a two-cycle multiply-accumulate unit that is based on a $w \times w/2$-bit multiplier. This saves 3 kGE for $w = 32$ in an 130 nm process compared to a traditional $w \times w$-bit multiplier.

A finite-field operation is started by writing three memory pointer registers (`OpA`, `OpB`, and `RES`) and a control register. As those registers are mapped at consecutive addresses, the store-multiple instruction (`STM`) of the Cortex-M0+ can be used to efficiently start an operation. A started finite-field multiplication is performed using the following hardware components: a $w \times w/2 = 32 \times 16$-bit multiplier, a $\lceil \mathrm{ld}(2n_w) \rceil + 2w = 68$-bit `ACC`umulator, a $w = 32$-bit register for operand A (`OpAReg`), a $3w/2 = 48$-bit register for operand B (`OpBReg`), and a $w = 32$-bit `WRITE` register. In `OpBReg`, the top 32 bits are always written by the bus and the lowest 16 bits are used as an operand of the multiplier. Therefore, a sequence of shift/rotate operations is necessary to actually multiply the loaded operands.

Table 8.1 visualizes the dataflow within the drop-in module. For a single multiply-accumulate operation five clock cycles are necessary. As the drop-in module heavily relies on pipelining, practically only two cycles are needed. The following steps are performed: (i) `OpB+i` is applied to the bus. (ii) `OpB+i` is `WR`itten to `OpBReg` and `OpA+j` is applied to the bus. (iii) `OpAReg` is `WR`itten and `OpBReg` is `SH`ifted by 16 bits. (iv) The first multiplication cycle (`MUL1`) multiplies the lower 16 bits of `OpB+i` with `OpA+j` and `OpBReg` is shifted again. (v) During the second multiplication cycle (`MUL2`) the accumulator is optionally `SHIFT`ed. When shifted, the lowest 32-bit of the accumulator are stored in the `WRITE` register. This data is later written to the address `RES+i+j`, when the bus is not utilized.

As the fully utilized bus needs some free cycles to write the result, we use a zig-zag product scanning technique (cf. Figure 8.4) [WW11]. In this technique, consecutive columns are traversed in different order, which allows caching of a single operand from one column to the next. This frees the bus for $2n_w$ cycles, which are exactly the $2n_w$ cycles required to store the computed results.

Although the implemented FIPS multiplication is quite complex, the software running on the CPU is completely independent of the methodology used to perform finite-field arithmetic within the drop-in module. However, there are two implementation guidelines the software has to deal with. First, constant variables have to be temporarily copied to the data memory when being used. Second, there are two techniques to wait for the drop-in module to finish. A function delegating an operation to the drop-in module can either start an operation and wait for it to finish, or wait for a previously started operation to finish and only then start a new operation. The latter case is more performant because the CPU and the drop-in module potentially work in parallel, i.e., the control flow operations involved in the invocation of the routines that call the drop-in module are done while the drop-in module is computing. However, temporary variables on the stack are freed once a function finishes, which requires adding additional wait statements within the extension-field arithmetic to prevent the drop-in from accessing reallocated memory locations. Nevertheless, the utilization of the drop-in is increased from 77.6% to 85.1% when the function first waits for previous operations to finish. Similarly, the utilization of the RAM is raised from 75.7% to 80.1% (cf. 34.6% in *(b)*, 17.0% in *(a)*).

8.4 Implementation Results

To verify the achievement of the area and performance goals initially set, the three microprocessor-based platforms *(a)*, *(b)* and *(d)* were evaluated with respect to hard- and software. Regarding the overall hardware platforms, runtime, area, power, and energy consumption are distinctive. Regarding the software part, the

Table 8.2: Performance of various operations on architectures *(a)*, *(b)*, and *(d)*.

Design		$\mathbb{F}_p$		$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	$\mathbb{G}_1 \times \mathbb{G}_2$	**RAM**	**ROM**
	Add	**Mul**	**Inv**	**Mul**	**Mul**	**Exp**	**Pairing**		
	[Cycles]	[Cycles]	[kCycles]	[kCycles]	[kCycles]	[kCycles]	[kCycles]	[Byte]	[Byte]
BN158									
Cortex-M0+	112	1 800	331	4 828	11 775	22 871	17 389	1 856	13 980
MAC	112	361	72	1 129	4 042	10 736	7 828	1 796	11 232
Drop-in	56	161	29	493	1 577	4 322	3 182	1 876	10 364
BN254									
Cortex-M0+	166	3 782	1 122	16 071	38 277	72 459	47 643	2 828	18 116
MAC	166	934	285	4 323	11 449	27 460	17 960	2 836	12 572
Drop-in	75	335	97	1 566	4 858	12 076	7 763	2 880	10 764

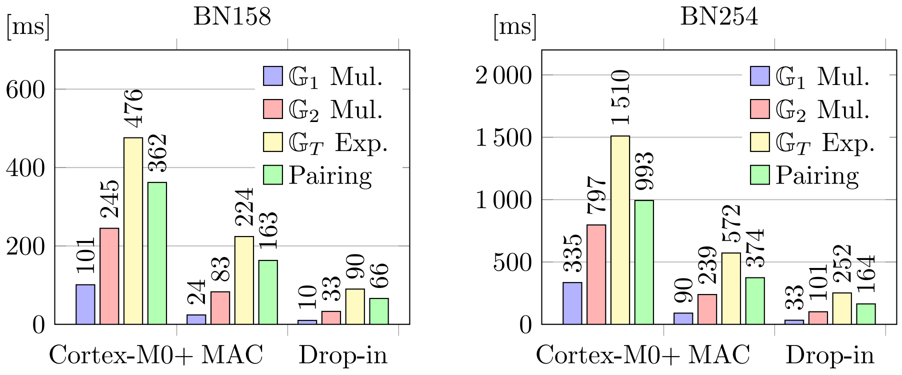


Figure 8.5: Group operations at 48 MHz.

evaluation focuses on the runtimes of the underlying finite-field arithmetic and the most expensive operations used within protocols: the point multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$, the exponentiation in $\mathbb{G}_T$, and the pairing operation.

The results in Table 8.2 show that the multiply-accumulate extension speeds up the prime-field multiplications by factors of 4.0-5.0[1], but leaves the prime-field additions unaffected. The same speed-ups are observed for prime-field inversions and point multiplications in $\mathbb{G}_1$. However, the impact of the multiply-accumulate extension on the performance of both pairings and operations in $\mathbb{G}_2$, $\mathbb{G}_T$ is lower and lies between a factor of 2.1 and 3.3. Considering the performance of the drop-in module, an even greater speed-up is observed compared to the plain software implementation. In this case, prime-field multiplications, inversions and point multiplications in $\mathbb{G}_1$ are up to 11.3 times faster, which eventually results in an up to 6.1 times faster computation of pairings. On average, operations using BN158 are 3.0 times faster than operations using BN254.

Throughout all implementations, the demand for data memory is kept relatively low, with a maximum of 1 876 bytes and 2 880 bytes for BN158 and BN254, respectively. Similarly, the program sizes are kept small, e.g., 18 KB for BN254. Given a typical clock frequency of 48 Mhz, the performance results of the point multiplications in $\mathbb{G}_1$, $\mathbb{G}_2$, the exponentiation in $\mathbb{G}_T$, and the pairing operation are illustrated in Figure 8.5. The respective runtimes support our choice of a 32-bit architecture: for BN254, the drop-in based platform does pairing computations in highly practical 164 ms. The pure *embedded* software implementation performs the same computation in 993 ms.

While Table 8.2 focuses on the software part, the most important hardware characteristics are visualized in Table 8.3. The runtime is given for a single pairing computation. Both area and power measurements were determined for an 130 nm low-leakage UMC technology. The area results in a 90 nm UMC

---

[1] The implementation for BN158 with multiply-accumulate extension utilizes the FIPS method and discards lazy reduction in $\mathbb{F}_{p^2}$ [Beu+10; SR13] as it yields better performance.

Table 8.3: Implementation characteristics for 130 nm and 90 nm process technologies.

Platform	Area RAM [kGE]	 ROM [kGE]	 CPU [kGE]	 Dedicated [kGE]	Total [kGE]	Power [mW]	Runtime [ms]	Energy [mJ]
BN158								
Cortex-M0+	11.4	15.6	18.4	-	45.4	5.92	362	2.14
MAC	11.1	13.8	27.1	-	52.0	7.38	163	1.20
Drop-in	11.4	13.8	17.0[a]	10.8	52.9	10.25	66	**0.68**
Drop-in 90nm	10.5	12.0	12.6[a]	**10.1**	**45.2**	-	66	-
BN254								
Cortex-M0+	16.0	19.3	18.4	-	53.7	5.80	993	5.76
MAC	16.0	15.6	27.1	-	58.8	7.33	374	2.74
Drop-in	16.2	13.8	17.0[a]	10.8	57.7	9.96	162	**1.61**
Drop-in 90nm	14.3	12.0	12.6[a]	**10.1**	**49.0**	-	162	-

---

[a] Bit-serial multiplier.

technology are explicitly marked. The designs were synthesized and their power and runtime evaluated for a clock frequency of 48 MHz. Both data and program memory were realized using RAM and ROM macros of appropriate sizes. The program memory encompasses all routines required to implement pairing-based protocols, i.e., pairings, operations in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. These platforms are hence ready-to-use for future applications based on pairings over BN curves.

According to Table 8.3, BN254 pairing computations with reasonable performance are available at the cost of 57.7 kGE in an 130 nm process technology. Switching to the more advanced 90 nm process technology shrinks the design to 49.0 kGE, constituting one of the smallest available hardware designs for pairings with practical relevance. In terms of power consumption, the plain microprocessor design is, as expected, the most economical. The multiply-accumulate extension and the drop-in module increase power consumption by 25% and 70%, respectively. Due to their increased performance, these platforms are more energy-efficient though. Their respective demand for energy is 2.1 and 3.5 times lower.

Table 8.4: Related software implementations of Ate pairings over BN curves.

	Platform	RAM [Byte]	ROM [Byte]	Runtime [kCycles]	Frequ. [MHz]	Runtime [ms]
Ours	Cortex-M0+	**2 828**	**18 116**	**47 643**	48	**993**
Gouvêa [GOL12]	MSP430	6 500	36 000	79 440	8	9 930
Devegili [DSD07]	Philips HiPerSmart™	<16 000	-	90 462	36	2 513
Gouvêa [GOL12]	MSP430X/MPY32	6 500	34 400	47 736	25	1 909

Table 8.5: Related hardware platforms (130 nm).

	Area		**Time**
	Ded.	**Total**	
	[kGE]	[kGE]	[kCycles]
Ours (Drop-in)	**11**[a]	**58**	7 763
Fan [FVV09]	183	183	**593**
Kammler [Kam+09]	71[b]	164	5 340
Kammler [Kam+09]	67[b]	145	6 490
Kammler [Kam+09]	53[b]	130	10 816

---

[a] Drop-in module.

[b] Core excl. 26 kGE of original RISC

8.5 Comparison with Related Work

As a consequence of our hardware/software co-design approach, comparison with related work focuses on two aspects. On the one hand, the pure software implementation on the Cortex-M0+ is brought into relation to other software implementations on low-resource hardware. On the other hand, the resulting hardware design is compared with other dedicated pairing hardware implementations.

The comparison of our software implementation with related implementations of Ate pairings over BN curves with primes of roughly 256-bit size is summarized in Table 8.4. Gouvêa et al. [GOL12] provide highly optimized software implementations for the 16-bit microcontroller MSP430 and a variant of its successor MSP430X, which is equipped with a 32-bit multiplier (MPY32). The implementation by Devegili et al. [DSD07] is evaluated on a 32-bit Philips HiPerSmart™ smart card, which has a SmartMIPS architecture and clearly is a direct competitor of Cortex-M0+-based smart cards. However, it is unclear to which extent side-channel resistance is considered by either of them.

As both the MSP430 and the Cortex-M0+ use a 16-bit instruction-set, it is important to highlight the exceptionally low program and data memory footprint of our implementations. It is however hard to compare the quality of an implementation when different frameworks and different microprocessors are involved.

Other pairing implementations for 32-bit ARM processors are limited to the Cortex-A series, such as in [Gre+12]. However, their pairing's runtime of 9.9 ms on a 1.2 GHz Cortex-A9 is as well hardly comparable with our pairing's runtime on the Cortex-M0+ since the multi-core Cortex-A processors provide massively higher clock frequencies along with a more powerful instruction set.

Regarding related hardware platforms, Table 8.5 covers hardware implementations of pairings with primes sized roughly 256 bits. Fan et al. [FVV09] proposed a dedicated pairing cryptoprocessor with parallelized, full-precision $\mathbb{F}_p$ arithmetic. Its centerpiece is a hardware implementation of a hybrid modular multiplication algorithm that performs both polynomial and coefficient reduction. Their area

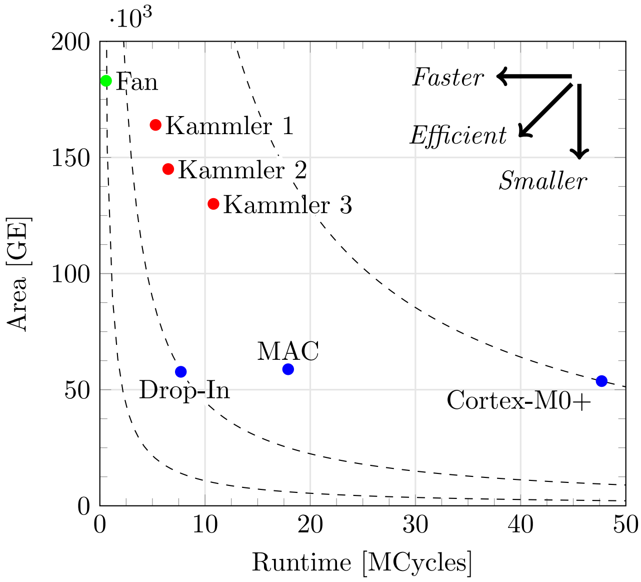


Figure 8.6: Characteristics of related hardware.

figures, however, exclude the required RAM. Kammler et al. [Kam+09] extended a 5-stage 32-bit RISC core with instructions for $\mathbb{F}_p$ arithmetic. Their Application-Specific Instruction-set Processor (ASIP) uses a Montgomery multiplier structure that can be synthesized in different configurations and sizes. Unfortunately, their area figures do not contain the program memory.

In comparison to [FVV09] and [Kam+09], our drop-in-based platform is 2.2-3.1 times smaller with regard to total area consumption. In both [Kam+09] and our case the CPU and the data memory can be reused for other applications. In terms of dedicated hardware, our drop-in-based platform is 16.6 times smaller than the work of Fan et al. In exchange, their design is faster and provides the best area-runtime product according to Figure 8.6. However, it depends on the application how much hardware area is actually acceptable to be spent on a dedicated pairing accelerator.

8.6 Re-usability of our Drop-in Architecture

To emphasize the practicability of our low-area platforms for deploying cryptography to embedded environments, several protocols that are relevant in such context have been assessed in terms of the performance to expect.

Table 8.6: Performance of pairing-based protocols on the drop-in platform.

	$\mathbb{G}_1$ Mul	$\mathbb{G}_2$ Mul	$\mathbb{G}_T$ Exp	$\mathbb{G}_1 \times \mathbb{G}_2$ Pairing	**BN158** [ms]	**BN254** [ms]
Leakage Resilient KEM [KP10]						
Encaps.	0	1	1	0	123	353
Decaps.	2	0	0	2	153	389
Identity-Based Encryption KEM [BB04; IEE08a]						
Encaps.	3	0	1	0	121	349
Decaps.	0	0	0	1.5[a]	99	243
Short Signatures [BB08]						
Sign	1	0	0	0	10	33
Verify	0	2	0	1	132	364
Short Group Signatures [Hwa+11]						
Sign	9	2	0	1.5[a]	258	739
Verify	9	2	0	3	357	981
Link	0	0	0	3	199	485

---

[a] Ratios and products of pairings are counted as 1.5 pairing computations.

8.6.1 Using the Drop-in Module for Pairing-based Protocols

The short signature scheme by Boneh and Boyen [BB08] is interesting for constrained signature devices as it aids to reduce communication. As a representative of group signatures, which help to provide anonymous authentication, the scheme by Hwang et al. [Hwa+11] was chosen. To be able to establish a random session key without the necessity of verifying public keys, the identity-based encryption scheme by Boneh and Boyen [BB04] in its Key Encapsulation Mechanism (KEM) variant was evaluated as it combines good performance with small parameters. Additionally, the leakage-resilient bilinear ElGamal KEM by Kiltz and Pietrzak [KP10] is taken into consideration because it is proven to have bounded side-channel leakage.

The number of computationally expensive operations and the expected overall runtime of each of the aforementioned protocols are presented in Table 8.6. The runtimes are given for the drop-in module based platform. As the figures suggest, all of the protocols may be performed on the device with user interaction as response times lie noticeably below one second.

8.6.2 Using the Drop-in Module for ECC

In order to emphasize the re-usability of our drop-in module based design, we also evaluated the performance of the standardized curves [Cer00; Nat09] `secp160r1` and `secp256r1` and the performance of Curve25519 by Bernstein [Ber06], which many people consider as a replacement for the standardized National Institute of Standards and Technology (NIST) curves. Again, we follow the point multiplication methodology from [WUW13], which relies on Montgomery ladders, randomized projective coordinates and multiple point validation checks. All implementations have similar hardware footprints and require 4.1 kGE (500 bytes) for RAM, 6.2 kGE (3 200 bytes) for ROM, 10.1 kGE for the drop-in module, 12.6 kGE for the Cortex-M0+, and 33 kGE in total (in a 90 nm UMC technology). Point multiplications for `secp160r1`, `secp256r1`, and Curve25519 need 570 kcycles, 1 765 kcycles, and 1110 kcycles, respectively. Note that we do not take advantage of the special form of the underlying primes. However, with runtimes of 11.9-36.8 ms (at 48 MHz) the drop-in concept is clearly an enabler of elliptic-curve based interactive protocols.

8.7 Conclusion

According to our evaluations of three microprocessor-based hardware designs, the utilization of a compact 32-bit microprocessor results in notably small pairing implementations. Requiring merely 45.2-49.0 kGE of chip area, we provided one of the smallest available hardware designs capable of bilinear pairings. The most prominent platform was obtained from the construction of a dedicated drop-in hardware module for prime-field arithmetic. Its low area requirements and highly practical runtime facilitate pairing-based cryptography in interactive embedded applications.

9

Side-Channel Attacks on Bilinear Pairings

Identity-based encryption constitutes a promising alternative to traditional cryptography that works without symmetric keys or public key infrastructures. Due to the latest developments in efficient pairing implementations such as in Chapter 8, pairing-based identity-based encryption is expected to play an important role in providing secure Internet-of-Things (IoT) applications in the future as well. However, IoT devices are inherently exposed to side-channel attacks. For this reason, together with Erich Wenger, we started to investigate the side-channel security of bilinear pairings during my master thesis [Unt13]. We further worked on this topic during my PhD studies and published our final results in [UW14b]. In this publication, I was the main author contributing the attack and most of the text. In this chapter, we use text and results from [UW14b] and make the following contributions.

Contribution. We present a practical Correlation Power Analysis (CPA) attack that leaks a user's private key in the identity-based encryption scheme by Boneh and Boyen [BB04]. In particular, our attack reveals the secret input point of the practical optimal-Ate pairing defined over Barreto-Naehrig (BN) curves from a software implementation on a 32-bit ARM Cortex-M0(+) processor. For this purpose, this work exploits the leakage of a finite field multiplication [Hut+09] within the pairing computation. Besides the CPA attack, we provide future work with evidence on how power analysis attacks perform relatively to each other on a Field Programmable Gate Array (FPGA), an Application Specific Integrated Circuit (ASIC), and using power simulations. Moreover, we emphasize that the projective point randomization technique [Cor99] is a countermeasure that is applied to Ate pairings on BN curves almost without effort.

Outline. This chapter is structured as follows. In Section 9.1, we investigate related work and further highlight how our work complements it. Besides the background of identity-based encryption, a high-level view of the attack setting is given in Section 9.2. A general description of the attack is part of Section 9.3 and Section 9.4 discusses the practical results of the attack. Following possible countermeasures in Section 9.5, a conclusion is drawn in Section 9.6.

9.1 Related Work

The first to investigate side-channel attacks in the context of pairing computations were Page and Vercauteren [PV04]. They focused on pairings over ternary fields, pointed out the possibility of timing and Simple Power Analysis (SPA) attacks of improperly implemented finite field multiplications, and proposed a Differential Power Analysis (DPA) attack that sequentially extracts one bit after another using the technique by Messerges [Mes02]. Similarly, Kim et al. [Kim+06] showed each a timing, a CPA and a DPA attack that potentially extract a secret value involved in the computation of the Eta pairing over hyperelliptic curves using binary fields. This work in contrast focuses on a CPA attack on optimal-Ate pairings using large prime fields, whose arithmetic differs enormously to that in binary or ternary fields used in, e.g., [PV04].

Whelan and Scott [WS06] investigated the side-channel vulnerability of the Tate, the Ate, and the Eta pairing more generally. They concluded that the computation of a bilinear pairing $e(P, Q)$ of the two elliptic curve points P and Q is inherently more secure if its first parameter P is the secret as it seemed impossible to build the hypothesis for a DPA attack. However, for the Tate pairing not using elliptic curve twists, Blömer et al. [BGL13] concluded theoretically that schemes using bilinear pairings with its first argument P being secret are not less vulnerable to side-channel attacks than otherwise. We complement their work by presenting results of a practical attack on the secret first argument of an Ate pairing computation over BN curves that uses elliptic curve twists.

An attack similar to the one presented in this chapter was done by Ghosh and Chowdhury [GC11]. In their attack, the secret parameter Q of the Tate pairing $e(P, Q)$ over BN curves was revealed. In more detail, a finite field addition during the evaluation of the line function in the Miller loop was targeted. The operation involves the secret input Q as well as the x-coordinate of an intermediate elliptic curve point that derives from the public input point P. Starting from the Least Significant Bit (LSB), they recover the secret x-coordinate successively by performing a difference-of-means test for each bit. They gather the necessary power measurements from their own FPGA-based pairing cryptoprocessor. Contrary to attacking a finite field addition within the pairing computation, we exploit the leakage of a finite field multiplication. Thereby we utilize the technique of

Hutter et al. [Hut+09], who efficiently attack a multi-precision integer multiplication within Elliptic Curve Digital Signature Algorithm (ECDSA)-enabled Radio-Frequency Identification (RFID) devices.

Private keys in identity-based encryption were shown to be vulnerable to side-channel attacks in [EFD09]. In a DPA attack, they demonstrated the feasibility of extracting the secret input of a prime-field based pairing computation from a hardware circuit which has an 8-bit datapath and which merely performs the operations leaking the secret information. In contrast, we extract the private key from a full and practical implementation of identity-based encryption on a 32-bit architecture, which is significantly harder to be performed successfully due to the exponentially larger number of possible values for each word of the secret. Besides, our results are based on three different measurement setups, while [EFD09] use power simulations only.

Several countermeasures to inhibit attacks on pairing computations were shown in the past. Page and Vercauteren [PV04] proposed two variants of point blinding mechanisms to counteract DPA attacks. In addition to that, Whelan and Scott [WS06] proposed multiplying the Miller variable in each iteration with a different random value. Unluckily, all of the mentioned countermeasures offer rather bad performance. Point blinding requires the computation of a second pairing at least, while the multiplication of the Miller variable involves an additional finite field multiplication in each iteration of the Miller loop. However, Kim et al. [Kim+06] adopted the fast and effective randomization countermeasure by Coron [Cor99] to the Eta pairing. They provided modified formulas to deal with the randomized projective coordinates of one of the two input points. In this chapter, we intend to raise awareness of the randomization countermeasure in the context of optimal-Ate pairings over BN curves. In this case, it is not even necessary to modify the formulas to deal with the randomized coordinates.

9.2 Identity-based Encryption

In 1984, Shamir [Sha84] proposed the concept of identity-based encryption for secure communication in company networks and mailing systems without the necessity of public key infrastructures. The concept uses identity strings instead of public keys for encryption, e.g., someone's e-mail address in a mailing system, which inherently allows sending encrypted e-mails. In order to achieve that, a trusted third party is responsible for providing public parameters and for generating the users' private keys.

One fast identity-based encryption scheme that is already in practical use is the BB_1 scheme that was presented by Boneh and Boyen [BB04]. Besides, they proposed a very practical BB_1-based Key Encapsulation Mechanism (KEM) for the future Institute of Electrical and Electronics Engineers (IEEE) standard on identity-based encryption. The KEM variant of the scheme specifies the four algorithms *Setup*, *Derive*, *Encapsulate* and *Decapsulate*. The *Setup* algorithm is run at the trusted third party and creates a master secret and the public parameters. Also the *Derive* algorithm is run at the trusted third party in

Algorithm 4 Ate pairing over BN curves.

```
Input: $P \in \boldsymbol{E}(\mathbb{F}_p), Q \in \boldsymbol{E}'(\mathbb{F}_{p^2})$
Output: $a(Q, P)$
$T \leftarrow Q, f \leftarrow 1$
for $i = \lfloor ld(s) \rfloor - 2$ downto 0 do
    $f \leftarrow f^2 \cdot \ell_{T,T}(P)$    ▷ subject to our attack
    $T \leftarrow [2]T$
    if $s_i = 1$ then
        $f \leftarrow f^2 \cdot \ell_{T,Q}(P)$
        $T \leftarrow T + Q$
    end if
end for
$f \leftarrow f^{(p^{12}-1)/n_G}$ return $f$
```

order to generate each user's private key. The two algorithms *Encapsulate* and *Decapsulate* are run by the respective users, who may use embedded devices. The *Encapsulate* algorithm provides both a session key and a ciphertext that is decryptable by the intended recipient only. The recipient recovers the session key from the received ciphertext by invoking the *Decapsulate* algorithm with their private key as a parameter.

The scheme uses three cyclic order-n_G groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ that allow the definition of a bilinear pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The produced ciphertext $C = (C_0, C_1)$ consists of two elements in $\mathbb{G}_1$ and the respective private keys $D_{id} = (D_{0,id}, D_{1,id})$ are comprised of two elements in $\mathbb{G}_2$. The scheme's *Decapsulate* algorithm recovers the session key $\overline{K}$ from a ciphertext C with the aid of the user's private key D_{id}, the properties of the bilinear pairing e, and a hash function $\mathcal{H}$:

$$\overline{K} = \mathcal{H}(e(C_0, D_{0,id})/e(C_1, D_{1,id})).$$

In this algorithm, the session key is obtained from bilinear pairing computations involving both a public and a secret operand. The secret operand to the bilinear pairing—the user's private key in this particular case—is the target of adversaries.

Note that we focus our analysis on the BB_1 scheme, but the subsequent attack is applicable to all schemes that involve pairing computations with a secret and a public operand.

9.2.1 Vulnerability

For the identity-based encryption scheme, this work uses optimal-Ate pairings [Ver10] based on the pairing-friendly elliptic curves by Barreto and Naehrig [BN05]. However, our attack applies to Ate pairings in general, which are computed according to Algorithm 4. Since our attack aims to recover the pairing's secret input, a more detailed investigation of the algorithm is necessary.

Algorithm 5 Initial sequence of Ate pairing computations.

```
Input: $P \in \boldsymbol{E}(\mathbb{F}_p), Q \in \boldsymbol{E}'(\mathbb{F}_{p^2})$
1: $(\mathrm{X_T}, \mathrm{Y_T}, \mathrm{Z_T}) \leftarrow (\mathrm{x_Q}, \mathrm{y_Q}, 1)$
2: $\mathrm{L}_{1,0} \leftarrow \mathrm{X}_\mathrm{T}^2$
3: $\mathrm{L}_{1,0} \leftarrow 3 \cdot \mathrm{L}_{1,0}$
4: $\mathrm{L}_{1,0} \leftarrow \mathrm{L}_{1,0} \cdot \mathrm{x_P}$
5: ...
```

The algorithm to compute the Ate pairing $a(Q, P)$ basically consists of the Miller loop in Lines 1-9 and the final exponentiation step in Line 10. The evaluation of the tangent line $\ell_{T,T}(P)$ in Line 3 and the point doubling in Line 4 of Algorithm 4 can be interleaved using the fast formulas by Costello et al. [CLN10]. The respective sequence of operations at the beginning of the first iteration of the Miller loop is shown in Algorithm 5.

This sequence of operations is vulnerable to a side-channel attack and may be exploited to extract either of the pairing's two parameters P and Q. In the *Decapsulate* routine of the aforementioned BB_1 identity-based encryption scheme, the pairings $a(D_{0,id}, C_0)$ and $a(D_{1,id}, -C_1)$ are computed. In both cases, the input parameter Q of the pairing $a(Q, P)$ is the secret to be extracted.

In the following, we assume the input point Q of $a(Q, P)$ to be secret and P to be public. In Line 4 of Algorithm 5, the x-coordinate of the publicly known input $x_P \in \mathbb{F}_p$ is multiplied with the unknown intermediate value $L_{1,0} \in \mathbb{F}_{p^2}$. This finite field multiplication consists of two separate prime field multiplications of x_P with the two $\mathbb{F}_p$ elements of $L_{1,0}$. A prime field multiplication is often partitioned into a multiplication and a reduction step. The multiplication steps within those two prime field multiplications allow the extraction of the two $\mathbb{F}_p$ elements of the unknown intermediate $L_{1,0}$ using a CPA attack. The original secret input Q is then easily computed from $L_{1,0}$ using Tonelli-Shanks square root computation in $\mathbb{F}_{p^2}$ and the elliptic curve equation. Accordingly, the two pairing computations $a(D_{0,id}, C_0)$ and $a(D_{1,id}, -C_1)$ in the identity-based encryption scheme allow the recovery of the two parts of the user's private key $D_{0,id}$ and $D_{1,id}$.

To counteract the attack, an idea may be to design protocols such that P is secret and Q is public. However, in this setup the same prime field multiplication $L_{1,0} \cdot x_P$ can be attacked to reveal the secret P since we are able to compute $L_{1,0}$ for any public input.

Other implementation formulas than the ones by Costello et al. [CLN10] may also be vulnerable to such type of attack. In particular, the same type of attack can be performed on the revised formulas for point doubling and tangent line evaluation by Aranha et al. [Ara+11]. With a slightly modified hypothesis, the same attack is feasible on the formulas using Jacobian coordinates by Hankerson et al. [HMS08], Beuchat et al. [Beu+10], and Aranha et al. [Ara+11]. Moreover, other protocols and schemes using pairing computations are exposed as well if these involve one both constant and secret parameter.

9.3 General Attack

As indicated before, the attack to extract the secret parameter used in the optimal-Ate pairing $a_{opt}(Q, P)$ is performed on a prime-field multiplication. A prime-field multiplication on an embedded processor usually consists of a multi-precision integer multiplication of the two input operands A and B that is succeeded by a modular reduction. In order to attack the multi-precision integer multiplication, we followed the ideas presented by Hutter et al. [Hut+09].

The public operand A and the both constant and secret operand B of the multi-precision integer multiplication consist of n_w words of w bits, where w denotes the architecture's word size. In this respect, a multi-precision integer multiplication can be written as the sums of word multiplication products, i.e.,

$$\underline{C} = A \cdot B = \sum_{i=0}^{n_w-1} \sum_{j=0}^{n_w-1} A[i]B[j]2^{(i+j)w},$$

where $A[i]$ denotes the i-th word of A. Its implementation may use, for example, operand scanning or product scanning [KAJ96]. We focus on product scanning, but the attack can easily be adapted for other implementation variants as well. As secret multi-precision integers span a large space of possible values, the attack is split into two basic steps:

Step 1: The **word multiplications** $A[i] \cdot B[j]$ are attacked to reduce the number of candidates for each of the n_w words of B. The k most probable candidates for each word are chosen to be used in the second step.

Step 2: The **accumulated intermediate sums of products** resulting from the respective word multiplications are attacked using solely the remaining k candidates for each word of B.

We now describe these two steps for attacking a multi-precision integer multiplication within prime-field multiplications in more detail. Note however that this attack is not limited to prime-field multiplications that separate the multiplication and the reduction step, but may also be applied to, e.g., Finely Integrated Product Scanning (FIPS) implementations, by taking the public modulus into consideration.

Step 1: Attacking Word Products

Generally, each word $B[j] \, \forall \, j = 0, ..., n_w - 1$ can be any value between 0 and $2^w - 1$. In this respect, *Step 1* tries to extract the most probable k candidates of the 2^w possible values for each word of B. This is done by attacking the products of each word of the secret B with the i-th word of the public input $A[i]$. All of the n_w words of the public input A are equally suitable for this. Depending on the details known about the implementation, a Hamming Weight (HW) or a Hamming Distance (HD) model may be used to construct a matrix that reflects the hypothetical power consumption of the respective multiplications. For the HW model and n_t executions of the algorithm, the hypothesis matrix is as follows:

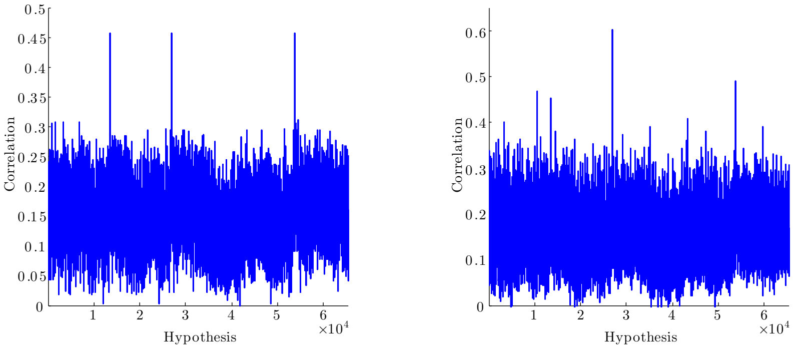
$$\begin{array}{cc} & \xrightarrow{\text{Hypothesis}} \\ \text{Execution}\big\downarrow & \begin{pmatrix} \mathrm{HW}(A_0[i]\cdot 0) & \dots & \mathrm{HW}(A_0[i]\cdot(2^w-1)) \\ \vdots & \ddots & \vdots \\ \mathrm{HW}(A_{n_t-1}[i]\cdot 0) & \dots & \mathrm{HW}(A_{n_t-1}[i]\cdot(2^w-1)) \end{pmatrix} \end{array}$$

Hereby, $A_l[i]$ denotes the i-th word of the input A used in the l-th execution of the algorithm. Correlation of this hypothesis matrix with the power traces measured during the respective n_t executions of the algorithm results in a correlation matrix that shows how each hypothesis correlates with every sample in the power traces. The correlation matrix thus allows to identify the regions in the power traces where each of the multiplications $A[i]\cdot B[j]\,\forall\, j=0,...,n_w-1$ take place. Figure 9.2d, for example, shows eight regions of high correlation that correspond to the respective multiplications $A[i]\cdot B[j]$. Evaluating each of these regions over all hypotheses allows the extraction of the most likely candidates for each multiplication and hence for each word $B[j]$ of the secret.

As pointed out by Hutter et al. [Hut+09], shifted variants of the correct hypothesis also lead to high correlation since multiplication is a linear operation. In the best case, each word can be identified uniquely, but in the worst case w equally likely hypotheses remain. An evaluation of all possible values for the secret input of a word multiplication is depicted in Figure 9.1. In the HW model, three equally likely candidates remain. Their respective values are bit-shifted versions of the correct secret. In this case, *Step 2* is necessary to uniquely determine the attacked word from the remaining k candidates. In the HD model, the correct value of the secret word becomes clearly visible, but other hypotheses also yield high correlations. In this instance, *Step 2* helps to gain certainty about the correctness of the most likely candidate found.

Step 2: Attacking Sums of Products

Based on the k most probable candidates that were determined for each word $B[j]$ in *Step 1*, *Step 2* aims to uniquely determine the full secret value B. In this iterative process, one word after another is revealed by consecutively attacking the single words of the final result $\underline{C}$. Initially, the first two words of the secret value B are determined. For this purpose all combinations of the candidates found for the first two words of B and all different inputs of A are used to create a suitable hypothesis matrix that models the second word of the result, $\underline{C}[1]$,



(a) Hamming weight model. **(b)** Hamming distance model.

Figure 9.1: Correlation of multiplication result for 16-bit hypotheses.

which is computed as follows:

$$\underline{C}[1] = A[0]B[1] + A[1]B[0] + (A[0]B[0] \gg w).$$

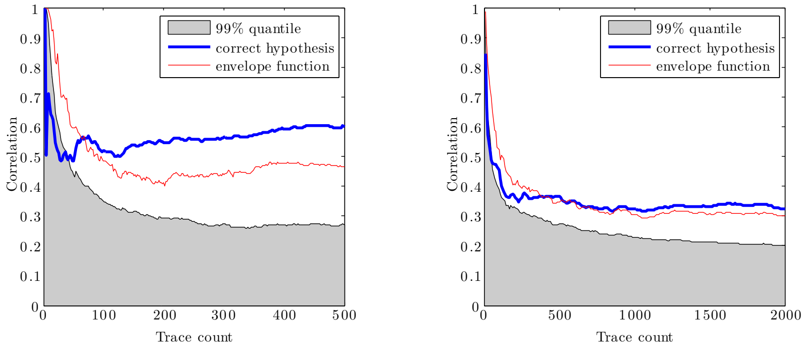
The power hypotheses for $\underline{C}[1]$ are then correlated with the recorded power traces. The resulting correlation matrix uniquely determines the first two words of the secret B. These revealed parts of the secret, namely $B[0]$ and $B[1]$, are then used together with the candidates for $B[2]$ to create a new hypothesis for the third word of the result, $\underline{C}[2]$. In general, the hypothesis for $\underline{C}[l]$ to uniquely determine $B[l]$ is built as

$$\underline{C}[l] = \left(\sum_{i \geq 0, j \geq 0}^{i+j=l} A[i]B[j] + \left(\sum_{m=0}^{l-1} \left(\sum_{i \geq 0, j \geq 0}^{i+j=m} A[i]B[j] \right) \gg (l-m)w \right) \right) \mod 2^w.$$

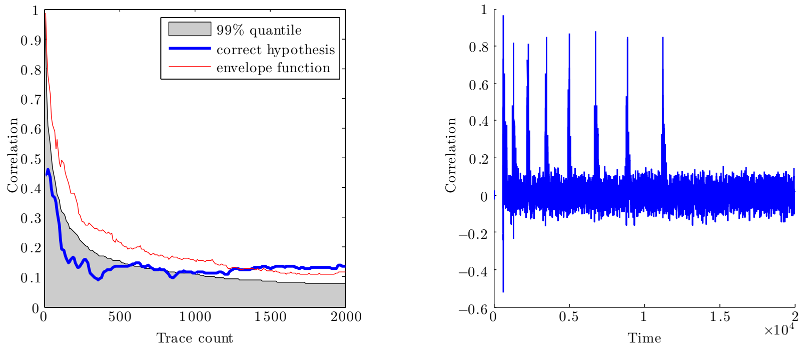
In this manner, the candidates found in the first step are used to successively determine the complete secret value B.

9.4 Practical Setup and Results

The attack presented in the previous section was conducted in practice. An embedded software implementation of the BB_1-KEM identity-based encryption scheme that is based on our pairing framework from Chapter 8 and suitable for both the ARM Cortex-M0 [ARM14b] and the Cortex-M0+ [ARM14a] was chosen as a target. However, note that for illustration of the attack this implementation had all randomization countermeasures disabled. Both the ARM Cortex-M0 and the ARM Cortex-M0+ work on 32-bit operands, but merely support a



(a) Toggle counts. **(b)** FPGA.



(c) ASIC. **(d)** Correlation over time.

Figure 9.2: Correlation of word multiplication results.

$32 \times 32 \rightarrow 32$ bit multiplication that discards half of the product. Therefore, each of the n_w^2 word multiplications in the multiplication step of the Separate Product Scanning (SPS) multiplication method is split into four $16 \times 16 \rightarrow 32$ bit multiplications that are aligned and accumulated appropriately. A suitable multiplication routine that simultaneously does the accumulation necessary for product scanning was presented by Wenger et al. [WUW13] and is shown in Algorithm 6.

The attack described in Section 9.3 is rather hard to perform on a 32-bit platform as each of the words of the secret operand can attain any value between 0 and $2^{32} - 1$. This leads to extremely large hypothesis matrices and requires high computational effort. Therefore, the attack was modified to better suit the targeted platform. Since each 32-bit multiplication is split into four 16-bit multiplications, *Step 1* of the practical attack targets the 16-bit half-words of the secret operand. The respective hypothesis matrix is built from the multiplication results of the least significant half-word of the public input with all possible values for the attacked half-word (2^{16} possibilities). This matrix targets the

Algorithm 6 Multiply-Accumulate routine for Cortex-M0 and Cortex-M0+ processors.

Input: `r1, r2` are 32-bit operands
Input: `r8, r9` are pointers to the operands
Output: {`r5, r4, r3`} is the accumulator

```
mov  r1, r8
ldr  r1, [r1, #offset1]
mov  r2, r9
ldr  r2, [r2, #offset2]

uxth r6, r1
uxth r7, r2
lsr  r1, r1, #16
lsr  r2, r2, #16

mov  r0, r6
mul  r0, r0, r7  ▷ low × low
mul  r6, r6, r2  ▷ low × high
mul  r2, r2, r1  ▷ high × high
mul  r1, r1, r7  ▷ high × low
mov  r7, #0
add  r5, r5, r0  ▷ low × low
adc  r4, r4, r2  ▷ high × high
adc  r3, r3, r7

lsl  r0, r6, #16
lsr  r2, r6, #16
add  r5, r5, r0  ▷ low × high
adc  r4, r4, r2
adc  r3, r3, r7

lsl  r0, r1, #16
lsr  r2, r1, #16
add  r5, r5, r0  ▷ high × low
adc  r4, r4, r2
adc  r3, r3, r7
```

multiplications in Line 10 and 11 of Algorithm 6. The first of these multiplications reveals the lower half, and the latter the upper half of each word of the secret operand. As one of the operands is overwritten by the multiplication result, a HD model is used to reflect the hypothetical power consumption of the changing registers.

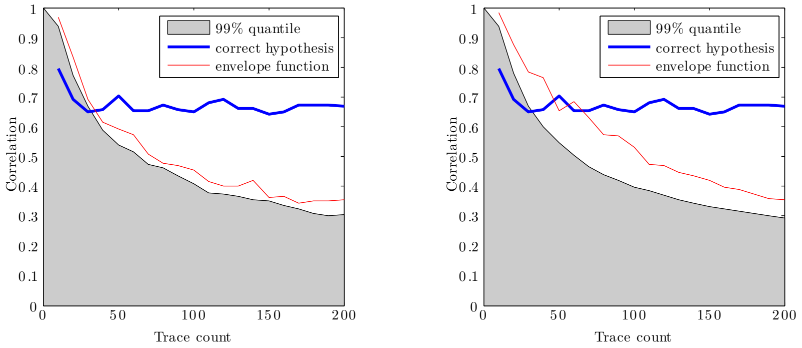
Step 2 of the attack was adapted accordingly. The candidates for the 16-bit half-words of the unknown operand are used to compute the hypothetical outcome for each word of the final result. The respective words are contained by the accumulator registers at various times. A simple HW model was preferred to describe the actual power consumption as the changes of the accumulator registers are rather complex to model.

Three different setups were used to collect the power traces necessary to practically perform the attack. In the first setup, a self-built processor functionally equivalent to the ARM Cortex-M0+ and its respective software implementation were deployed to the Xilinx Virtex-II Pro xc2vp30 FPGA [Xil14] on a Sasebo G board [RIS14]. In the second setup, the same hardware platform was synthesized for a UMC 130 nm process and power simulations were run to obtain the count of bit toggles in each clock cycle. In the third setup, the same software implementation was deployed to an ARM Cortex-M0 Microcontroller Unit (MCU) by NXP (LPC1114FN28 [NXP14]). For all three setups, the same set of input data was used, which allows comparison of the quality of side-channel leakage. Mixing results of the Cortex-M0 and the Cortex-M0+ seems acceptable as the two processors differ only slightly. In particular, the Cortex-M0+ comes with two pipeline stages while the Cortex-M0 is in possession of three, which mainly affects branching and only marginally influences the attack.

For the power measurements on the FPGA and the ARM Cortex-M0, a MATLAB Side-Channel Analysis toolbox was utilized to communicate with the cryptographic device using its serial interface. It was further used to retrieve the power traces from the oscilloscope. The FPGA and the Cortex-M0 were operated at a clock frequency of 25 MHz and 10 MHz, respectively. To attain good measurements, both clock frequencies were chosen such that the sampling rate of the oscilloscope is an integer multiple of the device clock frequency. A trigger signal was used to align the power traces, which were measured on an 1 Ω resistor on the line from the device to VCC using a differential probe.

The effort to successfully perform the presented attack was evaluated for the three different setups. For *Step 1*, which targets the multiplication of half-words, Figure 9.2a-9.2c show the number of traces required to distinguish the correct hypothesis and its shifted variants from the others. Apart from the correct hypothesis' correlation, these figures show the envelope function and the 99% quantile of all hypotheses, i.e., the range of correlations of all hypotheses but the highest 1%. The envelope function represents the highest correlation of any hypothesis but the correct one in each of the experiments with different trace counts.

When using the noiseless toggle counts instead of power measurements, the attack is already possible with data from less than 100 different traces. The rather



(a) k=5. **(b)** k=10.

Figure 9.3: Correlation of accumulation register.

old Virtex-II FPGA has quite high leakage, which results in successful attacks with merely 800 traces. When attacking the ARM Cortex-M0 by NXP that is built with modern process technologies, the attack succeeds with approximately 1 500 traces. Contrary to the other two experiments, the correct hypothesis' correlation is much lower. Therefore, it takes significantly more traces for the correct hypothesis to elevate from the hypotheses in the 99% quantile.

The results for *Step 2* are similar. Figure 9.3 shows the correlation of the second result word $A[0]B[1] + A[1]B[0] + (A[0]B[0] \gg w)$ depending on the number of traces when using toggle counts. The experiment was done with different numbers of candidates k learned for each half-word of the secret B in *Step 1*. These were determined as the top k correlating hypotheses when using 100 power traces. Since half-word candidates are found in *Step 1*, there remain k^4 candidates to build the hypothesis matrix for the second partial sum. Using the $k = 5$ most likely candidates for each half-word resulted in a sooner success than when using the $k = 10$ most likely candidates. For higher numbers of candidates, tested with $k = 15$ and $k = 20$, no difference could be observed compared to $k = 10$. From the results in Figure 9.2a and Figure 9.3b, we conclude that the complete attack succeeds with the same number of traces as required in *Step 1*.

9.5 Countermeasures

The presented CPA attack successfully extracts the secret input point of a bilinear pairing. To mitigate such kind of attacks, several general countermeasures have been presented before, e.g., point blinding [PV04] and randomization of the Miller variable [WS06]. Point blinding techniques leave the pairing algorithm untouched and solve the problem on a higher level, i.e., instead of computing $e(P, Q)$ directly, one could either compute $e(P, Q) = e(aP, bP)^{1/ab}$ with a and b being random values or $e(P, Q) = e(P, Q + R)/e(P, R)$ with R being a random point. However, in the first case two additional point multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$

and an exponentiation in $\mathbb{G}_T$ are required, and in the second case the computation of a second pairing is necessary. Since either of those two approaches degrades performance massively, both can hardly be applied to embedded scenarios. Less expensive and hence better suited for embedded devices is the randomization of the Miller variable as in [WS06]. This countermeasure requires that in each iteration of the Miller loop all intermediate variables contributing to f (cf. Algorithm 4) are multiplied with a random value. Due to the final exponentiation, this does not affect the final result of the pairing algorithm. Still, this kind of countermeasure is not very efficient.

Therefore, we recommend counteracting side-channel attacks on pairings in embedded devices by following the idea of Randomized Projective Coordinates (RPC) [Cor99] and randomizing the intermediate point T in the computation of $a_{opt}(Q, P)$ in Algorithm 4. In particular, instead of initializing T trivially with $(X_T, Y_T, Z_T) = (x_Q, y_Q, 1)$, one chooses a random value ρ and assigns $(X_T, Y_T, Z_T) = (\rho x_Q, \rho y_Q, \rho)$ to the homogeneous projective point T. Independently of which of the two input points Q and P is secret, this approach prevents attackers from building a suitable hypothesis in the presented attack. Apart from this single initialization step, the countermeasure does not incur any overhead. Moreover, the randomization can easily be adapted to other sets of implementation formulas and different variants of projective coordinates.

9.6 Conclusion

This chapter featured a CPA attack on bilinear pairings that poses a significant threat to pairing-based protocols. In this respect, we pointed out how the pairing computation can leak a user's private key in the popular identity-based encryption scheme BB_1 by Boneh and Boyen [BB04]. We thereby illustrated that many implementation formulas of the widely used Ate pairings $a(Q, P)$ over BN curves are vulnerable to power analysis attacks, independently of which of the two input parameters is secret. We practically verified the feasibility of our attack using three different setups and finally emphasized that Coron's projective point randomization techniques are equally important for pairing implementations as they are for elliptic curve cryptography. Therefore, we recommend using RPC in all future pairing implementations threatened by side-channel attacks.

10

Conclusions

Internet-of-Things (IoT) devices are inherently exposed to physical attackers and for this reason suffer from a wide variety of powerful attacks, ranging from physical probing of the Printed Circuit Board (PCB), over manipulation of the memory, to side-channel attacks. In this thesis, we have put significant effort in securing IoT devices against side-channel attacks.

In particular, the first part of this thesis was devoted to cryptography offering side-channel security when the leakage of its implementation is assumed to be bounded. Chapter 3 used the cryptographic properties of sponges to provide a tool for modeling bounded side-channel leakage in permutation-based designs. This thesis hereby gave a new approach to include side-channel security into the design phase of cryptographic schemes and to assess the practical security of cryptographic implementations in the presence of limited amounts of side-channel leakage. However, note that implementations must yet be designed to keep the side-channel leakage small in order to obtain decent security levels. As well, it is a particular challenge to practically determine the amount of side-channel leakage an implementation gives about a value.

In this regard, Chapter 4 illustrated that evaluating the capacity of a side channel is a suitable approach to determine the leakage about a value from a single processing in bits. The presented technique works independent of the concrete leakage function and Gaussian noise is the sole assumption. In addition, this leakage quantification suits many leakage-resilient schemes as these use a secret only once. However, the amount of leakage also depends on factors like the measurement equipment and, as shown in this thesis, rises quickly for multivariate side channels when noise is removed through signal averaging. The attacker's capabilities must hence be clearly specified to get reliable leakage estimations.

On the other hand, even when the leakage about a key is bounded by both the cryptographic scheme and its implementation, there can also be side effects depending on the concrete use case. In particular, Chapter 5 showed that frequent re-keying used in leakage-resilient modes protects the key, but allows for Differential Power Analysis (DPA) attacks that unveil plaintext parts that stay the same when the key is changed. This is, e.g., relevant for memory encryption, which inherently produces read-modify-write operations. Consequently, this insight is a strong reminder to take care of possible side effects when applying side-channel countermeasures in practical applications.

In the second part, this thesis focused on memory encryption and authentication. State-of-the-art techniques are designed to protect the memory from attackers with physical access, but so far completely neglect the threat of side-channel analysis. While the disregard of side-channel attacks is not a security concern for applications like the encryption of USB flash drives, where the attacker usually owns a shut off device, side-channel attacks are a severe threat in case of attackers with access to running IoT devices operating on encrypted memory. In this respect, we showed in Chapter 6 that DPA breaks all memory encryption schemes used in practice. While this could be expected from a theoretical point of view, our practical attack on ext4 disk encryption clearly demonstrated their applicability to state-of-the-art System on Chips (SoCs), making them a highly relevant threat to a broad range of IoT devices. Apart from that, this thesis also revealed the vulnerability of memory encryption to active Differential Fault Analysis (DFA) techniques.

As a consequence of our attacks in Chapter 6, Chapter 7 presented MEAS—the first memory encryption and authentication scheme that is secure against DPA and that protects memory in IoT devices from attackers with runtime physical access. The scheme combines ideas from fresh re-keying and authentication trees to yield both DPA protection and a minimal root of trust. The efficiency of our scheme illustrates how to achieve practical side-channel protection when the use case is known and side-channel attacks are considered already in the design phase. In this respect, our prototype implementation of MEAS offers DPA protection for memory encryption and authentication without additional costs over conventional schemes. This allows to deploy MEAS to Random Access Memory (RAM) without significant efforts, and to Non-Volatile Memory (NVM) by integrating a non-volatile, secure storage, such as in a Trusted Platform Module (TPM), to maintain the root of trust. Yet, we think that the cost of memory encryption and authentication must be taken into account when designing future IoT architectures such as by mechanisms to reduce memory pressure.

In the third and last part of this thesis, we shifted our focus to privacy concerns in the IoT, which can be overcome through techniques from pairing-based cryptography. We showed in Chapter 8 that pairing-based protocols, such as privacy-preserving group signatures, are ready for their widespread deployment in the IoT by using a hard- and software co-design approach to give a both efficient and lightweight implementation of bilinear pairings. However, recent advances in solving the Discrete Logarithm Problem (DLP) [MSS16; KB16; BD17] suggest reduced security for pairings and hence demand for transferring our design methodology to larger elliptic curves offering better security. As a result, the practicality of bilinear pairings in embedded devices requires further investigation. Yet, note that the elliptic curve BN254 we used in this thesis is still assumed to offer 100-bit security, which is sufficient for many applications. We are hence confident that our advances towards efficient pairing-based cryptography will help, e.g., privacy-enhancing technologies, to find their way into our everyday lives.

Finally, by recovering the secret key from an unprotected implementation of identity-based encryption in a practical DPA attack, Chapter 9 showed that side-channel attacks need to be concerned for pairing-based cryptography as well. While our recommendation is to prevent side-channel attacks directly within the pairing algorithm by randomizing projective elliptic curve point coordinates, we emphasized that there is a range of point and exponent blinding techniques available to prevent side-channel analysis when using unprotected implementations as well. Unfortunately, these techniques come at the cost of massive overheads that are unsuitable for lightweight applications. Thus, protecting implementations using randomized projective coordinates currently is the best approach to prevent side-channel analysis. However, we also think that adapting pairing-based protocols to perform re-keying or to give provable leakage resilience, such as in [KP10], is a promising approach for further exploration.

Outlook. The results in this thesis indicate that further research on side-channel attacks and countermeasures will be necessary in the future. In particular, our approach in Chapter 4 to quantify leakage under a single data input leaves several open questions. For example, it is yet unclear if there is a reasonable leakage bound when combining different side channels, e.g., Electromagnetic Emanation (EM), power, and timing. Moreover, additional research is required on how to apply the procedure of leakage quantification to generic processor architectures and to implementations with large state sizes that make profiling impossible. Further, methods and choices in the profiling step need to be evaluated regarding their impact on the estimated leakage since profiling defines what is considered to be the side-channel signal. In addition, our re-keying function IsapRk2, our instance Meas-v2, and many other (leakage-resilient) constructions allow each key to be used with a small number of different inputs, e.g., $q = 2$. Hence, there needs to be research on finding suitable leakage bounds in these settings as well.

However, there clearly is no useful leakage bound for a secret that is used together with many different inputs, such as for a constant plaintext in the case of re-keying. As a result, there needs to be research on whether protected implementations are the sole option to cope with this threat, or if other techniques may help as well. For instance, it would be interesting to investigate the effect of oblivious RAM techniques on the side-channel security of Meas as it may hide the repeated en-/decryption of the same plaintext by permanently changing the memory layout. Apart from that, side-channel analysis in this thesis strongly focused on DPA techniques, but recent research showed other techniques, such as cache attacks, to be similarly dangerous. As IoT devices become more and more evolved, these kind of techniques and side channels should in future analyses, designs, and implementations also be explicitly taken into consideration.

In terms of memory encryption and authentication, there is also more research to be done. For example, the approach to side-channel resistant memory encryption and authentication used in MEAS raises the question of whether an efficient, side-channel resistant memory encryption scheme can also be designed without involving the overhead of authentication trees. Another example is the development of techniques to securely improve the performance and hide the latency of encrypted memory. In addition, other aspects, such as address-bus leakage and fault attacks, need to be concerned when securing memory as well. A future challenge hence is the integration of different countermeasures to form a single scheme to secure memory. At least in terms of fault attacks, there is evidence that re-keying does not only counteract DPA, but can also prevent DFA [Med+10].

Summarizing, the analyses, designs, tools, and implementations provided in this thesis are an important step towards inherent side-channel security and contribute to secure platforms and the practicality of (pairing-based) privacy-preserving schemes in the IoT. From these advances, we conclude that the vision of a secure IoT is not wishful thinking, but the ongoing progress in research will lead to better security architectures and make secure IoT devices both realistic and practical.

Bibliography

[And+15] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. “Security of Keyed Sponge Constructions Using a Modular Proof Approach.” In: *Fast Software Encryption – FSE 2015*. Ed. by Gregor Leander. Vol. 9054. LNCS. Springer, 2015, pp. 364–384. ISBN: 978-3-662-48115-8.

[App12] Apple Inc. *Apple Technical White Paper: Best Practices for Deploying FileVault 2*. `http://training.apple.com/pdf/WP_FileVault2.pdf`. 2012.

[App15] Apple Inc. *iOS Security*. `https://www.apple.com/business/docs/iOS_Security_Guide.pdf`. 2015.

[Ara+11] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio López. “Faster Explicit Formulas for Computing Pairings over Ordinary Curves.” In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, 2011, pp. 48–68. ISBN: 978-3-642-20464-7.

[ARM] ARM Ltd. *Core Link™ Level 2 Cache Controller L2C-310 Technical Reference Manual*. ID080112.

[ARM14a] ARM Ltd. *Cortex-M0+ Processor*. June 2014. URL: `http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php`.

[ARM14b] ARM Ltd. *Cortex-M0 Processor*. June 2014. URL: `http://www.arm.com/products/processors/cortex-m/cortex-m0.php`.

[Atm13] Atmel Corporation. *Atmel SAM D20 ARM-based Microcontroller Datasheet*. Dec. 2013. URL: `http://www.atmel.com/Images/Atmel-42129-SAM-D20_Summary.pdf`.

[Ava17] Roberto Avanzi. “The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes.” In: *IACR Trans. Symmetric Cryptol.* 2017 (2017), pp. 4–44.

[AVR16] AVR-Crypto-Lib. *AVR-Crypto-Lib*. 2016. URL: `https://trac.cryptolib.org/avr-crypto-lib`.

[Bal+15] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. "DPA, Bitslicing and Masking at 1 GHz." In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. LNCS. Springer, 2015, pp. 599–619. ISBN: 978-3-662-48323-7.

[BB04] Dan Boneh and Xavier Boyen. "Secure Identity Based Encryption Without Random Oracles." In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew K. Franklin. Vol. 3152. LNCS. Springer, 2004, pp. 443–459. ISBN: 3-540-22668-0.

[BB08] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups." In: *J. Cryptology* 21 (2008), pp. 149–177.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short Group Signatures." In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew K. Franklin. Vol. 3152. LNCS. Springer, 2004, pp. 41–55. ISBN: 3-540-22668-0.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation Power Analysis with a Leakage Model." In: *Cryptographic Hardware and Embedded Systems – CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. LNCS. Springer, 2004, pp. 16–29. ISBN: 3-540-22666-4.

[BD17] Razvan Barbulescu and Sylvain Duquesne. "Updating key size estimations for pairings." In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 334. URL: `http://eprint.iacr.org/2017/334`.

[Bel+14] Sonia Belaïd, Fabrizio De Santis, Johann Heyszl, Stefan Mangard, Marcel Medwed, Jörn-Marc Schmidt, François-Xavier Standaert, and Stefan Tillich. "Towards fresh re-keying with leakage-resilient PRFs: cipher design principles and analysis." In: *J. Cryptographic Engineering* 4 (2014), pp. 157–171.

[Bel+15] Sonia Belaïd, Jean-Sébastien Coron, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, and Emmanuel Prouff. "Improved Side-Channel Analysis of Finite-Field Multiplication." In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. LNCS. Springer, 2015, pp. 395–415. ISBN: 978-3-662-48323-7.

[Ber+09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak sponge function family main document." In: *Submission to NIST (Round 2)* 3 (2009), p. 30.

[Ber+11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications." In: *Selected Areas in Cryptography – SAC 2011*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. LNCS. Springer, 2011, pp. 320–337. ISBN: 978-3-642-28495-3.

[Ber+11b] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *Cryptographic sponge functions (Version 0.1)*. http://sponge.noekeon.org/. 2011.

[Ber+11c] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *The Keccak SHA-3 submission*. http://keccak.noekeon.org/. 2011.

[Ber+12] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. "Permutation-based encryption, authentication and authenticated encryption." In: *Workshop Records of DIAC 2012*. 2012, pp. 159–170.

[Ber+14a] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, and Ronny Van Keer. *CAESAR submission: Ketje*. http://ketje.noekeon.org/. 2014.

[Ber+14b] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, and Ronny Van Keer. *CAESAR submission: Keyak*. http://keyak.noekeon.org/. 2014.

[Ber06] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records." In: *Public Key Cryptography – PKC 2006*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Vol. 3958. LNCS. Springer, 2006, pp. 207–228. ISBN: 3-540-33851-9.

[Beu+10] Jean-Luc Beuchat, Jorge Enrique González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. "High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves." In: *Pairing-Based Cryptography – Pairing 2010*. Ed. by Marc Joye, Atsuko Miyaji, and Akira Otsuka. Vol. 6487. LNCS. Springer, 2010, pp. 21–39. ISBN: 978-3-642-17454-4.

[BFG14] Sonia Belaïd, Pierre-Alain Fouque, and Benoît Gérard. "Side-Channel Analysis of Multiplications in GF(2128) - Application to AES-GCM." In: *Advances in Cryptology – ASIACRYPT 2014*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. LNCS. Springer, 2014, pp. 306–325. ISBN: 978-3-662-45607-1.

[BGL13] Johannes Blömer, Peter Günther, and Gennadij Liske. "Improved Side Channel Attacks on Pairing Based Cryptography." In: *Constructive Side-Channel Analysis and Secure Design – COSADE 2013*. Ed. by Emmanuel Prouff. Vol. 7864. LNCS. Springer, 2013, pp. 154–168. ISBN: 978-3-642-40025-4.

[BGS15] Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. "Masking and leakage-resilient primitives: One, the other(s) or both?" In: *Cryptography and Communications* 7 (2015), pp. 163–184.

[Bil+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. “A More Efficient AES Threshold Implementation.” In: *Progress in Cryptology – AFRICACRYPT 2014*. Ed. by David Pointcheval and Damien Vergnaud. Vol. 8469. LNCS. Springer, 2014, pp. 267–284. ISBN: 978-3-319-06733-9.

[BN05] Paulo S. L. M. Barreto and Michael Naehrig. “Pairing-Friendly Elliptic Curves of Prime Order.” In: *Selected Areas in Cryptography – SAC 2005*. Ed. by Bart Preneel and Stafford E. Tavares. Vol. 3897. LNCS. Springer, 2005, pp. 319–331. ISBN: 3-540-33108-5.

[BN08] Mihir Bellare and Chanathip Namprempre. “Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm.” In: *J. Cryptology* 21 (2008), pp. 469–491.

[Bog+14] Andrey Bogdanov, Christoph Dobraunig, Maria Eichlseder, Martin M. Lauridsen, Florian Mendel, Martin Schläffer, and Elmar Tischhauser. “Key Recovery Attacks on Recent Authenticated Ciphers.” In: *Progress in Cryptology – LATINCRYPT 2014*. Ed. by Diego F. Aranha and Alfred Menezes. Vol. 8895. LNCS. Springer, 2014, pp. 274–287. ISBN: 978-3-319-16294-2.

[Bor+12] Julia Borghoff et al. “PRINCE - A Low-latency Block Cipher for Pervasive Computing Applications (Full version).” In: *IACR Cryptology ePrint Archive* 2012 (2012), p. 529. URL: `http://eprint.iacr.org/2012/529`.

[BS97] Eli Biham and Adi Shamir. “Differential Fault Analysis of Secret Key Cryptosystems.” In: *Advances in Cryptology – CRYPTO 1997*. Ed. by Burton S. Kaliski Jr. Vol. 1294. LNCS. Springer, 1997, pp. 513–525. ISBN: 3-540-63384-7.

[Cer00] Certicom Research. *Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0*. Sept. 2000. URL: `http://www.secg.org/`.

[CGM12] Omar Choudary, Felix Gröbert, and Joachim Metz. “Infiltrate the Vault: Security Analysis and Decryption of Lion Full Disk Encryption.” In: *IACR Cryptology ePrint Archive* 2012 (2012), p. 374. URL: `http://eprint.iacr.org/2012/374`.

[CH91] David Chaum and Eugène van Heyst. “Group Signatures.” In: *Advances in Cryptology – EUROCRYPT 91*. Ed. by Donald W. Davies. Vol. 547. LNCS. Springer, 1991, pp. 257–265. ISBN: 3-540-54620-0.

[Cha+99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. “Towards Sound Approaches to Counteract Power-Analysis Attacks.” In: *Advances in Cryptology – CRYPTO 1999*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, 1999, pp. 398–412. ISBN: 3-540-66347-9.

[CL02] Jan Camenisch and Anna Lysyanskaya. “A Signature Scheme with Efficient Protocols.” In: *Security and Cryptography for Networks – SCN 2002*. Ed. by Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano. Vol. 2576. LNCS. Springer, 2002, pp. 268–289. ISBN: 3-540-00420-3.

[CLN10] Craig Costello, Tanja Lange, and Michael Naehrig. “Faster Pairing Computations on Curves with High-Degree Twists.” In: *Public Key Cryptography – PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, 2010, pp. 224–242. ISBN: 978-3-642-13012-0.

[Cor99] Jean-Sébastien Coron. “Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems.” In: *Cryptographic Hardware and Embedded Systems – CHES 1999*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. LNCS. Springer, 1999, pp. 292–302. ISBN: 3-540-66646-X.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. “Template Attacks.” In: *Cryptographic Hardware and Embedded Systems – CHES 2002*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. LNCS. Springer, 2002, pp. 13–28. ISBN: 3-540-00409-2.

[CT12] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[Dev+06] Augusto Jun Devegili, Colm O’hEigeartaigh, Michael Scott, and Ricardo Dahab. “Multiplication and Squaring on Pairing-Friendly Fields.” In: *IACR Cryptology ePrint Archive* 2006 (2006), p. 471. URL: http://eprint.iacr.org/2006/471.

[DFS15] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. “Making Masking Security Proofs Concrete - Or How to Evaluate the Security of Any Leaking Device.” In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, 2015, pp. 401–429. ISBN: 978-3-662-46799-2.

[DMC15] DM-Crypt. *Dm-crypt: Linux Kernel Device-Mapper Crypto Target*. http://www.saout.de/misc/dm-crypt/. 2015.

[Dob+14] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, and Florian Mendel. “On the Security of Fresh Re-keying to Counteract Side-Channel and Fault Attacks.” In: *Smart Card Research and Advanced Applications – CARDIS 2014*. Ed. by Marc Joye and Amir Moradi. Vol. 8968. LNCS. Springer, 2014, pp. 233–244. ISBN: 978-3-319-16762-6.

[Dob+16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. “Ascon v1.2.” In: (2016).

[Dob+17] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. "ISAP - Towards Side-Channel Secure Authenticated Encryption." In: *IACR Trans. Symmetric Cryptol.* 2017 (2017), pp. 80–105.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[DSD07] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. "Implementing Cryptographic Pairings over Barreto-Naehrig Curves." In: *Pairing-Based Cryptography – Pairing 2007*. Ed. by Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto. Vol. 4575. LNCS. Springer, 2007, pp. 197–207. ISBN: 978-3-540-73488-8.

[Dzi+16] Stefan Dziembowski, Sebastian Faust, Gottfried Herold, Anthony Journault, Daniel Masny, and François-Xavier Standaert. "Towards Sound Fresh Re-keying with Hard (Physical) Learning Problems." In: *Advances in Cryptology – CRYPTO 2016*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, 2016, pp. 272–301. ISBN: 978-3-662-53007-8.

[EFD09] N. El Mrabet, M.-L. Flottes, and G. Di Natale. "A practical Differential Power Analysis attack against the Miller algorithm." In: *Research in Microelectronics and Electronics, 2009. PRIME 2009. Ph.D.* July 2009, pp. 308–311.

[Eis+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. "On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme." In: *Advances in Cryptology – CRYPTO 2008*. Ed. by David A. Wagner. Vol. 5157. LNCS. Springer, 2008, pp. 203–220. ISBN: 978-3-540-85173-8.

[Elb+07] Reouven Elbaz, David Champagne, Ruby B. Lee, Lionel Torres, Gilles Sassatelli, and Pierre Guillemin. "TEC-Tree: A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks." In: *Cryptographic Hardware and Embedded Systems – CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. LNCS. Springer, 2007, pp. 289–302. ISBN: 978-3-540-74734-5.

[Elb+09] Reouven Elbaz, David Champagne, Catherine H. Gebotys, Ruby B. Lee, Nachiketh R. Potlapally, and Lionel Torres. "Hardware Mechanisms for Memory Authentication: A Survey of Existing Techniques and Engines." In: *Trans. Computational Science*. LNCS 4 (2009). Ed. by Marina L. Gavrilova, Chih Jeng Kenneth Tan, and Edward D. Moreno, pp. 1–22.

[Fer06] Niels Ferguson. *AES-CBC + Elephant Diffuser A Disk Encryption Algorithm for Windows Vista*. Aug. 2006.

[FKR11] Laura Fuentes-Castañeda, Edward Knapp, and Francisco Rodríguez-Henríquez. “Faster Hashing to $\mathbb{G}_2$.” In: *Selected Areas in Cryptography – SAC 2011*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. LNCS. Springer, 2011, pp. 412–430. ISBN: 978-3-642-28495-3.

[FPS12] Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. “Practical Leakage-Resilient Symmetric Cryptography.” In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. LNCS. Springer, 2012, pp. 213–232. ISBN: 978-3-642-33026-1.

[Fru05] Clemens Fruhwirth. *New Methods in Hard Disk Encryption*. Tech. rep. 2005.

[Fru11] Clemens Fruhwirth. *LUKS On-Disk Format Specification*. `https://gitlab.com/cryptsetup/cryptsetup/wikis/LUKS-standard/on-disk-format.pdf`. 2011.

[FVV09] Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede. “Faster -Arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves.” In: *Cryptographic Hardware and Embedded Systems – CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. LNCS. Springer, 2009, pp. 240–253. ISBN: 978-3-642-04137-2.

[Gau+17] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini. “Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices.” In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* PP.99 (2017), pp. 1–14.

[GC11] Santosh Ghosh and Dipanwita Roy Chowdhury. “Security of Prime Field Pairing Cryptoprocessor against Differential Power Attack.” In: *Security Aspects in Information Technology – InfoSecHiComNet 2011*. Ed. by Marc Joye, Debdeep Mukhopadhyay, and Michael Tunstall. Vol. 7011. LNCS. Springer, 2011, pp. 16–29. ISBN: 978-3-642-24585-5.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. “How to construct random functions.” In: *J. ACM* 33 (1986), pp. 792–807.

[Gie+10] Benedikt Gierlichs, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. “Revisiting Higher-Order DPA Attacks:” in: *Topics in Cryptology – CT-RSA 2010*. Ed. by Josef Pieprzyk. Vol. 5985. LNCS. Springer, 2010, pp. 221–234. ISBN: 978-3-642-11924-8.

[GJ16] Qian Guo and Thomas Johansson. “A New Birthday-Type Algorithm for Attacking the Fresh Re-Keying Countermeasure.” In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 225. URL: `http://eprint.iacr.org/2016/225`.

[Gol+03] Andrea Goldsmith, Syed Ali Jafar, Nihar Jindal, and Sriram Vishwanath. “Capacity limits of MIMO channels.” In: *IEEE Journal on Selected Areas in Communications* 21 (2003), pp. 684–702.

[Gol05] Andrea Goldsmith. *Wireless communications.* Cambridge university press, 2005.

[GOL12] Conrado Porto Lopes Gouvêa, Leonardo B. Oliveira, and Julio López. "Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller." In: *J. Cryptographic Engineering* 2 (2012), pp. 19–29.

[Goo15] Google Inc. *Android Full Disk Encryption.* `https://source.android.com/security/encryption/`. 2015.

[GP99] Louis Goubin and Jacques Patarin. "DES and Differential Power Analysis (The "Duplication" Method)." In: *Cryptographic Hardware and Embedded Systems – CHES 1999.* Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. LNCS. Springer, 1999, pp. 158–172. ISBN: 3-540-66646-X.

[Gre+12] Gurleen Grewal, Reza Azarderakhsh, Patrick Longa, Shi Hu, and David Jao. "Efficient Implementation of Bilinear Pairings on ARM Processors." In: *Selected Areas in Cryptography – SAC 2012.* Ed. by Lars R. Knudsen and Huapeng Wu. Vol. 7707. LNCS. Springer, 2012, pp. 149–165. ISBN: 978-3-642-35998-9.

[Gro+16] Hannes Groß, Manuel Jelinek, Stefan Mangard, Thomas Unterluggauer, and Mario Werner. "Concealing Secrets in Embedded Processors Designs." In: *Smart Card Research and Advanced Applications – CARDIS 2016.* Ed. by Kerstin Lemke-Rust and Michael Tunstall. Vol. 10146. LNCS. Springer, 2016, pp. 89–104. ISBN: 978-3-319-54668-1.

[GS10] Robert Granger and Michael Scott. "Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions." In: *Public Key Cryptography – PKC 2010.* Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, 2010, pp. 209–223. ISBN: 978-3-642-13012-0.

[Gue16] Shay Gueron. "A Memory Encryption Engine Suitable for General Purpose Processors." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 204. URL: `http://eprint.iacr.org/2016/204`.

[Hal+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. "Lest we remember: cold-boot attacks on encryption keys." In: *Commun. ACM* 52 (2009), pp. 91–98.

[HJ05] William Eric Hall and Charanjit S. Jutla. "Parallelizable Authentication Trees." In: *Selected Areas in Cryptography – SAC 2005.* Ed. by Bart Preneel and Stafford E. Tavares. Vol. 3897. LNCS. Springer, 2005, pp. 95–109. ISBN: 3-540-33108-5.

[HJS11] Michael Hutter, Marc Joye, and Yannick Sierra. "Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation." In: *Progress in Cryptology – AFRICACRYPT 2011*. Ed. by Abderrahmane Nitaj and David Pointcheval. Vol. 6737. LNCS. Springer, 2011, pp. 170–187. ISBN: 978-3-642-21968-9.

[HMS08] Darrel Hankerson, Alfred Menezes, and Michael Scott. "Software Implementation of Pairings." In: *IOS Press Cryptology and Information Security Series on Identity-Based Cryptography*. M. Joye and G. Neven, 2008. Chap. 12, pp. 188–206.

[HT13] Michael Henson and Stephen Taylor. "Beyond Full Disk Encryption: Protection on Security-Enhanced Commodity Processors." In: *Applied Cryptography and Network Security – ACNS 2013*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. LNCS. Springer, 2013, pp. 307–321. ISBN: 978-3-642-38979-5.

[HTM09] Neil Hanley, Michael Tunstall, and William P. Marnane. "Unknown Plaintext Template Attacks." In: *Information Security Applications – WISA 2009*. Ed. by Heung Youl Youm and Moti Yung. Vol. 5932. LNCS. Springer, 2009, pp. 148–162. ISBN: 978-3-642-10837-2.

[Hut+09] Michael Hutter, Marcel Medwed, Daniel M. Hein, and Johannes Wolkerstorfer. "Attacking ECDSA-Enabled RFID Devices." In: *Applied Cryptography and Network Security – ACNS 2009*. Ed. by Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud. Vol. 5536. LNCS. 2009, pp. 519–534. ISBN: 978-3-642-01956-2.

[Hwa+11] Jung Yeon Hwang, Sokjoon Lee, Byung-Ho Chung, Hyun Sook Cho, and DaeHun Nyang. "Short Group Signatures with Controllable Linkability." In: *LIGHTSEC 2011*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 44–52. ISBN: 978-0-7695-4340-6.

[IEE08a] IEEE. *P1363.3TM/D1 Draft Standard for Identity-based Public-key Cryptography Using Pairings*. 2008.

[IEE08b] IEEE. *Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. IEEE Std 1619-2007*. Apr. 2008.

[Int16] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer Manuals*. 325462-058. 2016.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. "Private Circuits: Securing Hardware against Probing Attacks." In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, 2003, pp. 463–481. ISBN: 3-540-40674-3.

[Jaf07] Joshua Jaffe. “A First-Order DPA Attack Against AES in Counter Mode with Unknown Initial Counter.” In: *Cryptographic Hardware and Embedded Systems – CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. LNCS. Springer, 2007, pp. 1–13. ISBN: 978-3-540-74734-5.

[Jou04] Antoine Joux. “A One Round Protocol for Tripartite Diffie-Hellman.” In: *J. Cryptology* 17 (2004), pp. 263–276.

[KAJ96] Çetin Kaya Koç, Tolga Acar, and Burton S. Kaliski Jr. “Analyzing and comparing Montgomery multiplication algorithms.” In: *IEEE Micro* 16 (1996), pp. 26–33.

[Kal00] Burt Kaliski. “PKCS# 5: Password-based Cryptography Specification Version 2.0.” In: (2000).

[Kam+09] David Kammler, Diandian Zhang, Peter Schwabe, Hanno Scharwächter, Markus Langenberg, Dominik Auras, Gerd Ascheid, and Rudolf Mathar. “Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves.” In: *Cryptographic Hardware and Embedded Systems – CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. LNCS. Springer, 2009, pp. 254–271. ISBN: 978-3-642-04137-2.

[KB16] Taechan Kim and Razvan Barbulescu. “Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case.” In: *Advances in Cryptology – CRYPTO 2016*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, 2016, pp. 543–571. ISBN: 978-3-662-53017-7.

[Kim+06] Tae Hyun Kim, Tsuyoshi Takagi, Dong-Guk Han, Ho Won Kim, and Jongin Lim. “Side Channel Attacks and Countermeasures on Pairing Based Cryptosystems over Binary Fields.” In: *Cryptology and Network Security – CANS 2006*. Ed. by David Pointcheval, Yi Mu, and Kefei Chen. Vol. 4301. LNCS. Springer, 2006, pp. 168–181. ISBN: 3-540-49462-6.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis.” In: *Advances in Cryptology – CRYPTO 1999*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, 1999, pp. 388–397. ISBN: 3-540-66347-9.

[Koc03] P.C. Kocher. *Leak-resistant cryptographic indexed key update*. US Patent 6,539,092. Mar. 2003. URL: `https://www.google.com/patents/US6539092`.

[KP10] Eike Kiltz and Krzysztof Pietrzak. “Leakage Resilient ElGamal Encryption.” In: *Advances in Cryptology – ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, 2010, pp. 595–612. ISBN: 978-3-642-17372-1.

[KPW16] David Kaplan, Jeremy Powell, and Tom Woller. *AMD Memory Encryption*. http://developer.amd.com/resources/articles-whitepapers/. 2016.

[Lam+03] Damjan Lampret et al. “Openrisc 1000 architecture manual.” In: *Description of assembler mnemonics and other for OR1200* (2003).

[LFD17] Chao Luo, Yunsi Fei, and A. Adam Ding. “Side-channel power analysis of XTS-AES.” In: *Design, Automation & Test in Europe — DATE 2017*. Ed. by David Atienza and Giorgio Di Natale. IEEE, 2017, pp. 1330–1335. ISBN: 978-3-9815370-8-6.

[Lin15] Linux Kernel Organization Inc. *Linux Kernel 4.3 Source Tree*. https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/log/?id=refs/tags/v4.3. 2015.

[Lon+15] Jake Longo, Elke De Mulder, Dan Page, and Michael Tunstall. “SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip.” In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. LNCS. Springer, 2015, pp. 620–640. ISBN: 978-3-662-48323-7.

[Med+10] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. “Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices.” In: *Progress in Cryptology – AFRICACRYPT 2010*. Ed. by Daniel J. Bernstein and Tanja Lange. Vol. 6055. LNCS. Springer, 2010, pp. 279–296. ISBN: 978-3-642-12677-2.

[Med+11] Marcel Medwed, Christophe Petit, Francesco Regazzoni, Mathieu Renauld, and François-Xavier Standaert. “Fresh Re-keying II: Securing Multiple Parties against Side-Channel and Fault Attacks.” In: *Smart Card Research and Advanced Applications – CARDIS 2011*. Ed. by Emmanuel Prouff. Vol. 7079. LNCS. Springer, 2011, pp. 115–132. ISBN: 978-3-642-27256-1.

[Med+16] Marcel Medwed, François-Xavier Standaert, Ventzislav Nikov, and Martin Feldhofer. “Unknown-Input Attacks in the Parallel Setting: Improving the Security of the CHES 2012 Leakage-Resilient PRF.” In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. 2016, pp. 602–623. ISBN: 978-3-662-53886-9.

[Men+12] Florian Mendel, Bart Mennink, Vincent Rijmen, and Elmar Tischhauser. “A Simple Key-Recovery Attack on McOE-X.” In: *Cryptology and Network Security – CANS 2012*. Ed. by Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis. Vol. 7712. Springer, 2012, pp. 23–31. ISBN: 978-3-642-35403-8.

[Mer80] Ralph C. Merkle. “Protocols for Public Key Cryptosystems.” In: *IEEE Symposium on Security and Privacy – S&P 1980*. IEEE Computer Society, 1980, pp. 122–134. ISBN: 0-8186-0335-6.

[Mes00] Thomas S. Messerges. “Using Second-Order Power Analysis to Attack DPA Resistant Software.” In: *Cryptographic Hardware and Embedded Systems – CHES 2000*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1965. LNCS. Springer, 2000, pp. 238–251. ISBN: 3-540-41455-X.

[Mes02] Thomas S. Messerges. “Power Analysis Attacks and Countermeasures for Cryptographic Algorithms.” PhD thesis. University of Illinois, 2002.

[Mil04] Victor S. Miller. “The Weil Pairing, and Its Efficient Calculation.” In: *J. Cryptology* 17 (2004), pp. 235–261.

[Miz+13] Hiroaki Mizuno, Keisuke Iwai, Hidema Tanaka, and Takakazu Kurokawa. “Information Theoretical Analysis of Side-Channel Attack.” In: *Information Systems Security – ICISS 2013*. Ed. by Aditya Bagchi and Indrakshi Ray. Vol. 8303. LNCS. Springer, 2013, pp. 255–269. ISBN: 978-3-642-45203-1.

[MM] Ted Ts’o Michael Halcrow Uday Savagaonkar and Ildar Muslukhov. *Ext4 Encryption Design Document*. `https://docs.google.com/document/d/1ft26lUQyuSpiu6VleP70_npaWdRfXFoNnB8JYnykNTg`.

[MNT01] Atsuko Miyaji, Masaki Nakabayashi, and Shunzou Takano. *New Explicit Conditions of Elliptic Curve Traces for FR-Reduction*. 2001.

[Mon85] Peter L. Montgomery. “Modular Multiplication without Trial Division.” In: *Mathematics of Computation* 44 (1985), pp. 519–521.

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

[Mor+11a] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. “On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs.” In: *Conference on Computer and Communications Security – CCS 2011*. Ed. by Yan Chen, George Danezis, and Vitaly Shmatikov. ACM, 2011, pp. 111–124. ISBN: 978-1-4503-0948-6.

[Mor+11b] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. “Pushing the Limits: A Very Compact and a Threshold Implementation of AES.” In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, 2011, pp. 69–88. ISBN: 978-3-642-20464-7.

[MSJ12] Marcel Medwed, François-Xavier Standaert, and Antoine Joux. "Towards Super-Exponential Side-Channel Security with Efficient Leakage-Resilient PRFs." In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. LNCS. Springer, 2012, pp. 193–212. ISBN: 978-3-642-33026-1.

[MSS16] Alfred Menezes, Palash Sarkar, and Shashank Singh. "Challenges with Assessing the Impact of NFS Advances on the Security of Pairing-Based Cryptography." In: *Paradigms in Cryptology. Malicious and Exploratory Cryptology — Mycrypt 2016*. Ed. by Raphael C.-W. Phan and Moti Yung. Vol. 10311. LNCS. Springer, 2016, pp. 83–108. ISBN: 978-3-319-61272-0.

[Nat09] National Institute of Standards and Technology (NIST). *FIPS-186-3: Digital Signature Standard (DSS)*. 2009. URL: `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.

[New16] NewAE Technology Inc. *ChipWhisperer*. 2016. URL: `https://newae.com/tools/chipwhisperer/`.

[Nog+08] Yasuyuki Nogami, Masataka Akane, Yumi Sakemi, Hidehiro Katou, and Yoshitaka Morikawa. "Integer Variable chi-Based Ate Pairing." In: *Pairing-Based Cryptography – Pairing 2008*. Ed. by Steven D. Galbraith and Kenneth G. Paterson. Vol. 5209. LNCS. Springer, 2008, pp. 178–191. ISBN: 978-3-540-85503-3.

[NXP14] NXP Semiconductors. *LPC1114FN28 MCU Product Information*. June 2014. URL: `http://www.nxp.com/products/microcontrollers/cortex_m0_m0/lpc1100/LPC1114FN28.html`.

[OM07] Elisabeth Oswald and Stefan Mangard. "Template Attacks on Masking - Resistance Is Futile." In: *Topics in Cryptology – CT-RSA 2007*. Ed. by Masayuki Abe. Vol. 4377. LNCS. Springer, 2007, pp. 243–256. ISBN: 3-540-69327-0.

[Owu+13] Emmanuel Owusu, Jorge Guajardo, Jonathan M. McCune, James Newsome, Adrian Perrig, and Amit Vasudevan. "OASIS: on achieving a sanctuary for integrity and secrecy on untrusted platforms." In: *Conference on Computer and Communications Security – CCS 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM, 2013, pp. 13–24. ISBN: 978-1-4503-2477-9.

[Per09] Colin Percival. "Stronger Key Derivation via Sequential Memory-Hard Functions." In: *Self-published* (2009), pp. 1–16.

[Pie09] Krzysztof Pietrzak. "A Leakage-Resilient Mode of Operation." In: *Advances in Cryptology – EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. LNCS. Springer, 2009, pp. 462–482. ISBN: 978-3-642-01000-2.

[PM16] Peter Pessl and Stefan Mangard. “Enhancing Side-Channel Analysis of Binary-Field Multiplication with Bit Reliability.” In: *Topics in Cryptology – CT-RSA 2016*. Ed. by Kazue Sako. Vol. 9610. LNCS. Springer, 2016, pp. 255–270. ISBN: 978-3-319-29484-1.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. “A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD.” In: *Cryptographic Hardware and Embedded Systems – CHES 2003*. Ed. by Colin D. Walter, Çetin Kaya Koç, and Christof Paar. Vol. 2779. LNCS. Springer, 2003, pp. 77–88. ISBN: 3-540-40833-9.

[PR13] Emmanuel Prouff and Matthieu Rivain. “Masking against Side-Channel Attacks: A Formal Security Proof.” In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, 2013, pp. 142–159. ISBN: 978-3-642-38347-2.

[PSV15] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. “Leakage-Resilient Authentication and Encryption from Symmetric Cryptographic Primitives.” In: *Conference on Computer and Communications Security – CCS 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM, 2015, pp. 96–108. ISBN: 978-1-4503-3832-5.

[PV04] Dan Page and Frederik Vercauteren. “Fault and Side-Channel Attacks on Pairing Based Cryptography.” In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 283. URL: http://eprint.iacr.org/2004/283.

[RIS14] RISEC, AIST. *Side-Channel Attack Standard Evaluation Board*. June 2014. URL: http://www.risec.aist.go.jp/project/sasebo/.

[Riv08] Matthieu Rivain. “On the Exact Success Rate of Side Channel Analysis in the Gaussian Model.” In: *Selected Areas in Cryptography – SAC 2008*. Ed. by Roberto Maria Avanzi, Liam Keliher, and Francesco Sica. Vol. 5381. LNCS. Springer, 2008, pp. 165–183. ISBN: 978-3-642-04158-7.

[Rog+07] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. “Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly.” In: *International Symposium on Microarchitecture – MICRO 2007*. IEEE Computer Society, 2007, pp. 183–196. ISBN: 0-7695-3047-8.

[Rog04] Phillip Rogaway. “Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC.” In: *Advances in Cryptology – ASIACRYPT 2004*. Ed. by Pil Joong Lee. Vol. 3329. LNCS. Springer, 2004, pp. 16–31. ISBN: 3-540-23975-8.

[Ros+15] Davide Rossi et al. “PULP: A parallel ultra low power platform for next generation IoT applications.” In: *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE. 2015, pp. 1–39.

[Saa04] Markku-Juhani Olavi Saarinen. “Encrypted Watermarks and Linux Laptop Security.” In: *Information Security Applications – WISA 2004*. Ed. by Chae Hoon Lim and Moti Yung. Vol. 3325. LNCS. Springer, 2004, pp. 27–38. ISBN: 3-540-24015-2.

[Sat14] Satoh Lab./UEC. *Sakura G*. 2014. URL: `http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html`.

[Sch+14] Robert Schilling, Manuel Jelinek, Markus Ortoff, and Thomas Unterluggauer. “A low-area ASIC implementation of AEGIS128—A fast authenticated encryption algorithm.” In: *Austrian Workshop on Microelectronics — Austrochip*. Oct. 2014, pp. 1–5.

[Sch+18] Robert Schilling, Thomas Unterluggauer, Stefan Mangard, Frank Gürkaynak, Michael Muehlberghuber, and Luca Benini. “High Speed ASIC Implementations of Leakage-Resilient Cryptography.” In: *DATE 2018*. (in press). 2018.

[Sha84] Adi Shamir. “Identity-Based Cryptosystems and Signature Schemes.” In: *Advances in Cryptology – CRYPTO 1984*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, 1984, pp. 47–53. ISBN: 3-540-15658-5.

[Sia13] Siarhei Siamashka. *tinymembench*. (accessed 2017-03). 2013. URL: `https://github.com/ssvb/tinymembench`.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. “A Stochastic Model for Differential Side Channel Cryptanalysis.” In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. LNCS. Springer, 2005, pp. 30–46. ISBN: 3-540-28474-5.

[SM07] Carl Staelin and Larry McVoy. *LMbench - Tools for Performance Analysis*. (accessed 2017-03). 2007. URL: `http://lmbench.sourceforge.net`.

[SMC09] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. “A Diagonal Fault Attack on the Advanced Encryption Standard.” In: *IACR Cryptology ePrint Archive* 2009 (2009), p. 581. URL: `http://eprint.iacr.org/2009/581`.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. “A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks.” In: *Advances in Cryptology – EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. LNCS. Springer, 2009, pp. 443–461. ISBN: 978-3-642-01000-2.

[SR13] Ana Helena Sánchez and Francisco Rodríguez-Henríquez. “NEON Implementation of an Attribute-Based Encryption Scheme.” In: *Applied Cryptography and Network Security – ACNS 2013*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. LNCS. Springer, 2013, pp. 322–338. ISBN: 978-3-642-38979-5.

[SRH16] Sami Saab, Pankaj Rohatgi, and Craig Hampel. “Side-Channel Protections for Cryptographic Instruction Set Extensions.” In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 700. URL: http://eprint.iacr.org/2016/700.

[SSU14] Daniel Slamanig, Raphael Spreitzer, and Thomas Unterluggauer. “Adding Controllable Linkability to Pairing-Based Group Signatures for Free.” In: *Information Security – ISC 2014*. Ed. by Sherman S. M. Chow, Jan Camenisch, Lucas Chi Kwong Hui, and Siu-Ming Yiu. Vol. 8783. LNCS. Springer, 2014, pp. 388–400. ISBN: 978-3-319-13256-3.

[SSU16] Daniel Slamanig, Raphael Spreitzer, and Thomas Unterluggauer. “Linking-Based Revocation for Group Signatures: A Pragmatic Approach for Efficient Revocation Checks.” In: *Paradigms in Cryptology. Malicious and Exploratory Cryptology — Mycrypt 2016*. Ed. by Raphael C.-W. Phan and Moti Yung. Vol. 10311. LNCS. Springer, 2016, pp. 364–388. ISBN: 978-3-319-61272-0.

[Sta+10] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. “Leakage Resilient Cryptography in Practice.” In: *Towards Hardware-Intrinsic Security - Foundations and Practice*. Information Security and Cryptography. Springer, 2010, pp. 99–134.

[Suh+03a] G. Edward Suh, Dwaine E. Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. “AEGIS: architecture for tamper-evident and tamper-resistant processing.” In: *International Conference on Supercomputing – ICS 2003*. Ed. by Utpal Banerjee, Kyle Gallivan, and Antonio González. ACM, 2003, pp. 160–171. ISBN: 1-58113-733-8.

[Suh+03b] G. Edward Suh, Dwaine E. Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. “Efficient Memory Integrity Verification and Encryption for Secure Processors.” In: *International Symposium on Microarchitecture – MICRO 2003*. ACM/IEEE Computer Society, 2003, pp. 339–350. ISBN: 0-7695-2043-X.

[Szc+08] Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. “NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks.” In: *European Conference on Wireless Sensor Networks – EWSN 2008*. Ed. by Roberto Verdone. Vol. 4913. LNCS. Springer, 2008, pp. 305–320. ISBN: 978-3-540-77689-5.

[Szc+09] Piotr Szczechowiak, Anton Kargl, Michael Scott, and Martin Collier. “On the application of pairing based cryptography to wireless sensor networks.” In: *Security and Privacy in Wireless and Mobile Networks – WISEC 2009*. Ed. by David A. Basin, Srdjan Capkun, and Wenke Lee. ACM, 2009, pp. 1–12. ISBN: 978-1-60558-460-7.

[Tel99] Emre Telatar. “Capacity of Multi-antenna Gaussian Channels.” In: *European Transactions on Telecommunications* 10 (1999), pp. 585–595.

[TS14] Mostafa M. I. Taha and Patrick Schaumont. “Side-channel countermeasure for SHA-3 at almost-zero area overhead.” In: *Hardware Oriented Security and Trust – HOST 2014*. IEEE Computer Society, 2014, pp. 93–96. ISBN: 978-1-4799-4114-8.

[TS15] Mostafa M. I. Taha and Patrick Schaumont. “Key Updating for Leakage Resiliency With Application to AES Modes of Operation.” In: *IEEE Trans. Information Forensics and Security* 10 (2015), pp. 519–528.

[UM16] Thomas Unterluggauer and Stefan Mangard. “Exploiting the Physical Disparity: Side-Channel Attacks on Memory Encryption.” In: *Constructive Side-Channel Analysis and Secure Design – COSADE 2016*. Ed. by François-Xavier Standaert and Elisabeth Oswald. Vol. 9689. LNCS. Springer, 2016, pp. 3–18. ISBN: 978-3-319-43282-3.

[Unt+17] Thomas Unterluggauer, Thomas Korak, Stefan Mangard, Robert Schilling, Luca Benini, Frank K. Gürkaynak, and Michael Muehlberghuber. “Leakage Bounds for Gaussian Side Channels.” In: *CARDIS 2017*. (in press). 2017.

[Unt13] Thomas Unterluggauer. “Hardware-Software-Codesign of Side-Channel Evaluated Identity-based Encryption.” MA thesis. Graz University of Technology, 2013.

[UW14a] Thomas Unterluggauer and Erich Wenger. “Efficient Pairings and ECC for Embedded Systems.” In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. LNCS. Springer, 2014, pp. 298–315. ISBN: 978-3-662-44708-6.

[UW14b] Thomas Unterluggauer and Erich Wenger. “Practical Attack on Bilinear Pairings to Disclose the Secrets of Embedded Devices.” In: *Availability, Reliability and Security – ARES 2014*. IEEE Computer Society, 2014, pp. 69–77.

[UWM17a] Thomas Unterluggauer, Mario Werner, and Stefan Mangard. “MEAS: Memory Encryption and Authentication Secure Against Side-Channel Attacks Using Unprotected Primitives.” In: *J. Cryptographic Engineering* (2017). (in submission).

[UWM17b] Thomas Unterluggauer, Mario Werner, and Stefan Mangard. "Securing Memory Encryption and Authentication Against Side-Channel Attacks Using Unprotected Primitives." In: *Asia Conference on Computer and Communications Security – AsiaCCS 2017*. Ed. by Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi. ACM, 2017, pp. 690–702. ISBN: 978-1-4503-4944-4.

[UWM17c] Thomas Unterluggauer, Mario Werner, and Stefan Mangard. "Side-channel plaintext-recovery attacks on leakage-resilient encryption." In: *Design, Automation & Test in Europe — DATE 2017*. Ed. by David Atienza and Giorgio Di Natale. IEEE, 2017, pp. 1318–1323. ISBN: 978-3-9815370-8-6.

[Ver10] Frederik Vercauteren. "Optimal pairings." In: *IEEE Trans. Information Theory* 56 (2010), pp. 455–461.

[Wen13] Erich Wenger. "Hardware Architectures for MSP430-Based Wireless Sensor Nodes Performing Elliptic Curve Cryptography." In: *Applied Cryptography and Network Security – ACNS 2013*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. LNCS. Springer, 2013, pp. 290–306. ISBN: 978-3-642-38979-5.

[Wer+17b] Mario Werner, Thomas Unterluggauer, Robert Schilling, David Schaffenrath, and Stefan Mangard. "Transparent memory encryption and authentication." In: *Field Programmable Logic and Applications – FPL 2017*. Ed. by Marco D. Santambrogio, Diana Göhringer, Dirk Stroobandt, Nele Mentens, and Jari Nurmi. IEEE, 2017, pp. 1–6. ISBN: 978-9-0903-0428-1.

[Wer+18] Mario Werner, Thomas Unterluggauer, David Schaffenrath, and Stefan Mangard. "Sponge-Based Control-Flow Protection for IoT Devices." In: *IEEE EuroS&P 2018*. (in press). 2018.

[WS06] Claire Whelan and Michael Scott. "Side Channel Analysis of Practical Pairing Implementations: Which Path Is More Secure?" In: *Progress in Cryptology – VIETCRYPT 2006*. Ed. by Phong Q. Nguyen. Vol. 4341. LNCS. Springer, 2006, pp. 99–114. ISBN: 3-540-68799-8.

[WUW13] Erich Wenger, Thomas Unterluggauer, and Mario Werner. "8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors." In: *Progress in Cryptology – INDOCRYPT 2013*. Ed. by Goutam Paul and Serge Vaudenay. Vol. 8250. LNCS. Springer, 2013, pp. 244–261. ISBN: 978-3-319-03514-7.

[WW11] Erich Wenger and Mario Werner. "Evaluating 16-Bit Processors for Elliptic Curve Cryptography." In: *Smart Card Research and Advanced Applications – CARDIS 2011*. Ed. by Emmanuel Prouff. Vol. 7079. LNCS. Springer, 2011, pp. 166–181. ISBN: 978-3-642-27256-1.

[Xil14] Xilinx, Inc. *Xilinx Virtex-II Pro Data Sheet*. June 2014. URL: http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf.

[Xil16] Xilinx, Inc. *Linux Kernel xilinx-v2016.2*. (accessed 2017-03). 2016. URL: https://github.com/Xilinx/linux-xlnx.git.

About the Author

Author information as of December 2017.

Personal Information

Name:	Thomas Unterluggauer
Date of birth:	April 5th, 1988
Place of birth:	Villach, Austria

Education

- **03/2014 – present:** Doctoral studies, Graz University of Technology, Austria.
- **11/2011 – 11/2013:** Master studies in Information and Computer Engineering (Telematik), Graz University of Technology, Austria.
- **10/2008 – 11/2011:** Bachelor studies in Information and Computer Engineering (Telematik), Graz University of Technology, Austria.

Professional and Academic Experience

- **11/2013 – present:** Research assistant, Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria.
- **10/2009 – 10/2013:** Software developer and tester, IVM Technical Consultants, Graz, Austria.
- **04/2008 – 06/2008:** Software developer, Gomogi, Klagenfurt, Austria.
- **Summer 2006 and 2007**: Internship as software developer for geoinformation, Carinthia University of Applied Sciences, Villach, Austria.

Author's Publications

Author's publications as of December 2017 mapped to the corresponding chapters.

Chapter 3

Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. "ISAP - Towards Side-Channel Secure Authenticated Encryption." In: *IACR Trans. Symmetric Cryptol.* 2017 (2017), pp. 80–105

Chapter 4

Thomas Unterluggauer, Thomas Korak, Stefan Mangard, Robert Schilling, Luca Benini, Frank K. Gürkaynak, and Michael Muehlberghuber. "Leakage Bounds for Gaussian Side Channels." In: *CARDIS 2017.* (in press). 2017

Chapter 5

Thomas Unterluggauer, Mario Werner, and Stefan Mangard. "Side-channel plaintext-recovery attacks on leakage-resilient encryption." In: *Design, Automation & Test in Europe — DATE 2017.* Ed. by David Atienza and Giorgio Di Natale. IEEE, 2017, pp. 1318–1323. ISBN: 978-3-9815370-8-6

Chapter 6

Thomas Unterluggauer and Stefan Mangard. "Exploiting the Physical Disparity: Side-Channel Attacks on Memory Encryption." In: *Constructive Side-Channel Analysis and Secure Design – COSADE 2016.* Ed. by François-Xavier Standaert and Elisabeth Oswald. Vol. 9689. LNCS. Springer, 2016, pp. 3–18. ISBN: 978-3-319-43282-3

Chapter 7

Thomas Unterluggauer, Mario Werner, and Stefan Mangard. "Securing Memory Encryption and Authentication Against Side-Channel Attacks Using Unprotected Primitives." In: *Asia Conference on Computer and Communications Security – AsiaCCS 2017.* Ed. by Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi. ACM, 2017, pp. 690–702. ISBN: 978-1-4503-4944-4

Thomas Unterluggauer, Mario Werner, and Stefan Mangard. "MEAS: Memory Encryption and Authentication Secure Against Side-Channel Attacks Using Unprotected Primitives." In: *J. Cryptographic Engineering* (2017). (in submission)

Chapter 8

Thomas Unterluggauer and Erich Wenger. "Efficient Pairings and ECC for Embedded Systems." In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. LNCS. Springer, 2014, pp. 298–315. ISBN: 978-3-662-44708-6

Chapter 9

Thomas Unterluggauer and Erich Wenger. "Practical Attack on Bilinear Pairings to Disclose the Secrets of Embedded Devices." In: *Availability, Reliability and Security – ARES 2014*. IEEE Computer Society, 2014, pp. 69–77

Further Contributions

Conference Publications

Mario Werner, Thomas Unterluggauer, David Schaffenrath, and Stefan Mangard. "Sponge-Based Control-Flow Protection for IoT Devices." In: *IEEE EuroS&P 2018*. (in press). 2018

Robert Schilling, Thomas Unterluggauer, Stefan Mangard, Frank Gürkaynak, Michael Muehlberghuber, and Luca Benini. "High Speed ASIC Implementations of Leakage-Resilient Cryptography." In: *DATE 2018*. (in press). 2018

Mario Werner, Thomas Unterluggauer, Robert Schilling, David Schaffenrath, and Stefan Mangard. "Transparent memory encryption and authentication." In: *Field Programmable Logic and Applications – FPL 2017*. Ed. by Marco D. Santambrogio, Diana Göhringer, Dirk Stroobandt, Nele Mentens, and Jari Nurmi. IEEE, 2017, pp. 1–6. ISBN: 978-9-0903-0428-1

Hannes Groß, Manuel Jelinek, Stefan Mangard, Thomas Unterluggauer, and Mario Werner. "Concealing Secrets in Embedded Processors Designs." In: *Smart Card Research and Advanced Applications – CARDIS 2016*. Ed. by Kerstin Lemke-Rust and Michael Tunstall. Vol. 10146. LNCS. Springer, 2016, pp. 89–104. ISBN: 978-3-319-54668-1

Daniel Slamanig, Raphael Spreitzer, and Thomas Unterluggauer. "Linking-Based Revocation for Group Signatures: A Pragmatic Approach for Efficient Revocation Checks." In: *Paradigms in Cryptology. Malicious and Exploratory Cryptology — Mycrypt 2016*. Ed. by Raphael C.-W. Phan and Moti Yung. Vol. 10311. LNCS. Springer, 2016, pp. 364–388. ISBN: 978-3-319-61272-0

Daniel Slamanig, Raphael Spreitzer, and Thomas Unterluggauer. “Adding Controllable Linkability to Pairing-Based Group Signatures for Free.” In: *Information Security – ISC 2014*. Ed. by Sherman S. M. Chow, Jan Camenisch, Lucas Chi Kwong Hui, and Siu-Ming Yiu. Vol. 8783. LNCS. Springer, 2014, pp. 388–400. ISBN: 978-3-319-13256-3

Robert Schilling, Manuel Jelinek, Markus Ortoff, and Thomas Unterluggauer. “A low-area ASIC implementation of AEGIS128—A fast authenticated encryption algorithm.” In: *Austrian Workshop on Microelectronics — Austrochip*. Oct. 2014, pp. 1–5

Erich Wenger, Thomas Unterluggauer, and Mario Werner. “8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors.” In: *Progress in Cryptology – INDOCRYPT 2013*. Ed. by Goutam Paul and Serge Vaudenay. Vol. 8250. LNCS. Springer, 2013, pp. 244–261. ISBN: 978-3-319-03514-7

Open-Source Repositories

Mario Werner, Thomas Unterluggauer, Hannes Groß, Thomas Kastner, Christian Maierhofer, David Schaffenrath, Robert Schilling, and Erich Wenger. *GitHub Repository: Transparent Memory Encryption and Authentication*. `https://github.com/IAIK/memsec`. 2017

Thomas Unterluggauer, Erich Wenger, Raphael Spreitzer, Mario Werner, and René Hölbling. *GitHub Repository: Pairings in C*. `https://github.com/IAIK/pairings_in_c`. 2015

Erich Wenger, Thomas Unterluggauer, and Mario Werner. *GitHub Repository: Flexible Elliptic Curve Cryptography library in C*. `https://github.com/IAIK/flecc_in_c`. 2014