Advances in the Side-Channel Analysis of Symmetric Cryptography

Mostafa Taha

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Patrick R. Schaumont, Chair
Inyoung Kim
Mohamed R. Rizk
Sandeep K. Shukla
Joseph G. Tront

April 29, 2014
Blacksburg, Virginia

Keywords: Side-Channel Analysis, Practical Leakage Resiliency, AES, SHA-3
Copyright 2014, Mostafa Taha

Advances in the Side-Channel Analysis of Symmetric Cryptography

Mostafa Taha

(ABSTRACT)

Side-Channel Analysis (SCA) is an implementation attack where an adversary exploits unintentional outputs of a cryptographic module to reveal secret information. Unintentional outputs, also called side-channel outputs, include power consumption, electromagnetic radiation, execution time, photonic emissions, acoustic waves and many more. The real threat of SCA lies in the ability to mount attacks over small parts of the key and to aggregate information over many different traces. The cryptographic community acknowledges that SCA can break any security module if the adequate protection is not implemented. In this dissertation, we propose several advances in side-channel attacks and countermeasures. We focus on symmetric cryptographic primitives, namely: block-ciphers and hashing functions.

In the first part, we focus on improving side-channel attacks. First, we propose a new method to profile highly parallel cryptographic modules. Profiling, in the context of SCA, characterizes the power consumption of a fully-controlled module to extract power signatures. Then, the power signatures are used to attack a similar module. Parallel designs show excessive algorithmic-noise in the power trace. Hence, we propose a novel attack that takes design parallelism into consideration, which results in a more powerful attack. Also, we propose the first comprehensive SCA of the new secure hashing function SHA-3. Although the main application of SHA-3 is hashing, there are other keyed applications including Message Authentication Codes (MACs), where protection against SCA is required. We study the SCA properties of all the operations involved in SHA-3. We also study the effect of changing the key-length on the difficulty of mounting attacks. Indeed, changing the key-length changes the attack methodology. Hence, we propose complete attacks against five different case studies, and propose a systematic algorithm to choose an attack methodology based on the key-length.

In the second part, we propose different techniques for protection against SCA. Indeed, the threat of SCA can be mitigated if the secret key changes before every execution. Although many contributions, in the domain of leakage resilient cryptography, tried to achieve this goal, the proposed solutions were inefficient and required very high implementation cost. Hence, we highlight a generic framework for efficient leakage resiliency through lightweight key-updating. Then, we propose two complete solutions for protecting AES modes of operation. One uses a dedicated circuit for key-updating, while the other uses the underlying AES block cipher itself. The first one requires small area (for the additional circuit) but achieves negligible performance overhead. The second one has no area overhead but requires small performance overhead. Also, we address the problem of executing all the applications of hashing functions, e.g. the unkeyed application of regular hashing and the keyed application of generating MACs, on the same core. We observe that, running unkeyed application on an SCA-protected core will involve a huge loss of performance (3x to 4x). Hence, we propose a novel SCA-protected core for hashing. Our core has no overhead in unkeyed applications, and negligible overhead in keyed ones.

Our research provides a better understanding of side-channel analysis and supports the cryptographic community with lightweight and efficient countermeasures.

Acknowledgments

Mostafa Taha is an assistant lecturer at Assiut University, Egypt. He is partially supported by Assiut University and by the VT-MENA program of Egypt. The research was supported by the National Science Foundation (NSF) Grant no. 1115839.

Contents

1 Introduction **1**

1.1 Outline 3

Part I Side-Channel Countermeasures **7**

2 Background on Side-Channel Attacks **8**

2.1 Side-Channel Outputs 10

2.2 Attack Methodologies 11

2.2.1 Simple Power Analysis (SPA) 11

2.2.2 Differential Power Analysis (DPA) 12

2.3 Typical Attack Setup 14

3 Profiling of Highly Parallel Implementations **16**

3.1 Related Work 17

3.2 Proposed Attack Model 19

3.2.1 Targeted Operation 20

3.2.2 The Extra Round 21
3.2.3 Assumptions 22
3.2.4 Profiling Phase 23
3.2.5 Attack Phase 24
3.2.6 Two Regions 25
3.3 Results of a Case Study: AES 26
3.3.1 The Target Implementation 26
3.3.2 The Extra Round 27
3.3.3 Performance Metrics 28
3.3.4 Results 28
3.4 Summary 30

4 Side-Channel Analysis of SHA-3 31

4.1 Previous Work 34
4.2 Background 34
4.3 Effect of Changing the Key-Length 38
4.4 Targeted Operation 41
4.5 Systematic Algorithm 45
4.5.1 Application to MAC-Keccak Examples 46
4.6 Case Studies 49
4.6.1 Key-length = 768 bits 51
4.6.2 Key-length = 896 bits 52

4.6.3 Key-length = 1024 bits 53
4.7 Practical Results 54
4.8 Summary 59

Part II Side-Channel Countermeasures **60**

5 Background on Side-Channel Countermeasures **61**
5.1 Practical Countermeasures 62
5.1.1 Hiding 62
5.1.2 Masking 63
5.2 Theoretical Countermeasures 64
5.3 Do We Need Perfect Protection? 65
5.4 Our Design Concept 65

6 Efficient Leakage Resiliency **67**
6.1 Background and Previous Work 67
6.1.1 Stateless Key-Updating 69
6.1.2 Stateful Key-Updating 70
6.2 Framework for Lightweight Key-Updating 72
6.2.1 Assumptions 73
6.2.2 Key Requirements 74
6.2.3 Discussions 76

7 Solutions for AES Modes of Operation **78**

7.1 AES Modes of Operation . . . 78

7.2 System Overview . . . 79

7.2.1 Interaction with the underlying mode of AES . . . 80

7.3 Previous Work . . . 81

7.4 Key-updating with Dedicated Circuit . . . 83

7.4.1 Key-Updating Functions . . . 84

7.4.2 Security Analysis . . . 85

7.4.3 Supported Number of Encryptions . . . 86

7.4.4 Implementation . . . 86

7.5 Key-updating with Round-Reduced AES . . . 87

7.5.1 Key-Updating Functions . . . 88

7.5.2 Security Analysis . . . 89

7.5.3 Implementation . . . 92

7.6 Comparison . . . 93

7.7 Trading SCA-security for Performance . . . 96

7.8 Summary . . . 97

8 Solution for Keyed Applications of SHA-3 **98**

8.1 Keyed Applications of SHA-3 . . . 100

8.2 One Module for all Applications . . . 101

8.2.1 Security Analysis . . . 102

8.3 Implementation 103

8.3.1 Performance and trading SCA-protection for performance 104

8.3.2 Comparison 105

8.4 Summary 107

9 Conclusion **108**

9.1 Conclusion about Leakage Resiliency 108

9.2 Overall Conclusion 109

Bibliography **113**

List of Figures

1.1 Side-channel analysis using instantaneous power consumption. 2

1.2 Outline of contributions. 3

2.1 Conditions for successful SCA. 9

2.2 Possible side-channel outputs. 10

2.3 The attack model of DPA. 13

2.4 A typical setup for SCA. 15

3.1 Comparison between different profiling techniques. 18

3.2 The extra round in hardware implementations. 21

3.3 Two different attack regions. 22

3.4 The last round of AES. 26

3.5 The proposed regression model and the Hamming Distance model. 28

3.6 Global Success Rate. 30

4.1 The difference between DPA of AES, HMAC, and MAC-Keccak. 32

4.2 Terminology used in Keccak. 35

4.3 Keccak hashing algorithm and the padding rules. 36

4.4 The two steps of θ operation, θ_1 and θ_2. 37

4.5 Lane indices before and after the π step. 38

4.6 Effect of key-length on the number of controlled bits and the number of unknown bits. 40

4.7 Effect of key-length on the attack difficulty. 41

4.8 Column xor of θ operation. 43

4.9 DPA applied to some examples of MAC-Keccak. 48

4.10 The DPA of Key-length = 768 bits. 51

4.11 Data-dependent bits of key-length=1024 bits, before and after π step. 53

4.12 A power trace of MAC-Keccak. 55

4.13 SCA of an xor operation. 56

4.14 SCA of generating the θ_{plane} operation. 57

4.15 Success rate of different case studies. 58

5.1 Pillars of SCA attacks. 62

6.1 Stateless and stateful key-updating, as shown for the example of data encryption. 69

6.2 The system overview. 72

6.3 Stateless key-updating using a tree structure. 73

6.4 Stateful key-updating using a chain of whitening functions. 73

7.1 High-level representation of the proposed scheme. 80

7.2 Replacing the first and last round keys by fresh running keys. 81

7.3 Architecture of the re-keying scheme. 84

7.4 Average number of correct solutions at different key-space sizes. 91

7.5 The implementation overhead of the different techniques used for the stateless key-updating. 95

7.6 The implementation overhead of the different techniques used for the stateful key-updating. 96

8.1 Difference between block-ciphers and hashing functions with respect to DPA attacks. 99

8.2 Authenticated Encryption mode using KECCAK. 101

8.3 The new countermeasure, where $|n|$ is the bit-length of the nonce and f_r is a round-reduced version of Keccak f. 101

8.4 Relative area compared to the unprotected core. 105

8.5 Relative throughput at 128 bits of nonce compared to the unprotected core. 106

List of Tables

3.1 Results for the complete attack against AES. 29

7.1 Previous key-updating schemes. 82

7.2 Comparison between the implementation overhead of the key-updating schemes. 94

Chapter 1

Introduction

As information technology moves into embedded form factors, security and privacy face a new challenge. Cryptographic protocols were originally designed to provide sound security in black box scenarios where an adversary, Eve, cannot access the internals of a device. Now, the threat model is changed, as Eve holds the computing device in her palms. She can monitor, record, or tamper the activities of the device while it is performing cryptographic computations. This opens a new door to attacks. Consider, for example, the cryptographic module in Figure 1.1. It executes a cryptographic algorithm that uses a secret key. Here, Eve can measure the instantaneous power consumption, which depends on the actual data being processed, and use statistical tools to recover the secret key. In this regard, the instantaneous power consumption is a side-channel leakage and the whole process is called side-channel analysis (SCA).

Side-channel analysis is a group of passive non-invasive implementation attacks, where Eve can recover secured information by monitoring the cryptographic module while computing. SCA does not alter the normal operation of the module, hence passive, nor does it require any damage to the module itself, hence non-invasive. SCA depends on breaking a cryptographic primitive by targeting the underlying module itself rather than the algorithmic structure.

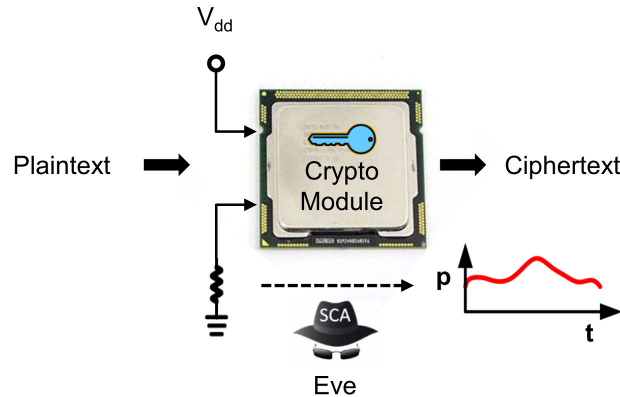

Figure 1.1: Side-channel analysis using instantaneous power consumption.

Classical cryptography used to put every effort in securing the algorithm assuming that the implementation is a black box, i.e. Eve can only know the inputs and the outputs. On the other hand, SCA respects the physical nature of the implementation in a gray box model, where Eve can collect information leakage from every execution.

Side-Channel Analysis is the most researched type of implementation attack. Implementation attacks, in general, also include active and invasive attacks. For example, fault analysis is a group of active attacks where Eve can alter the normal operation of the module. Eve can force a change in one of the internal registers, and monitor its effect on the final output [32]. Also, Eve can gradually change the input voltage or the clock frequency, and monitor the point where the module fails [61]. Invasive attacks include, for example, probing attacks. Here, Eve can decap the chip and insert micro-probes to capture the secret key right from the internal data paths [52].

SCA poses a practical threat to modern cryptographic systems. SCA has been demonstrated against many secure systems, and caused them to be recalled. It has been used to break the KeeLog remote keyless entry system [35], a Mifare DESFire RFID tag [82], the Atmel CryptoMemory nonvolatile memory [10], and the bitstream encryption of Xilinx Virtex-II FPGA [73], Virtex-4/5 FPGA [74] and Altera Stratix II [76]. These attacks were reported in conference papers, however, the actual number of broken systems may be higher.

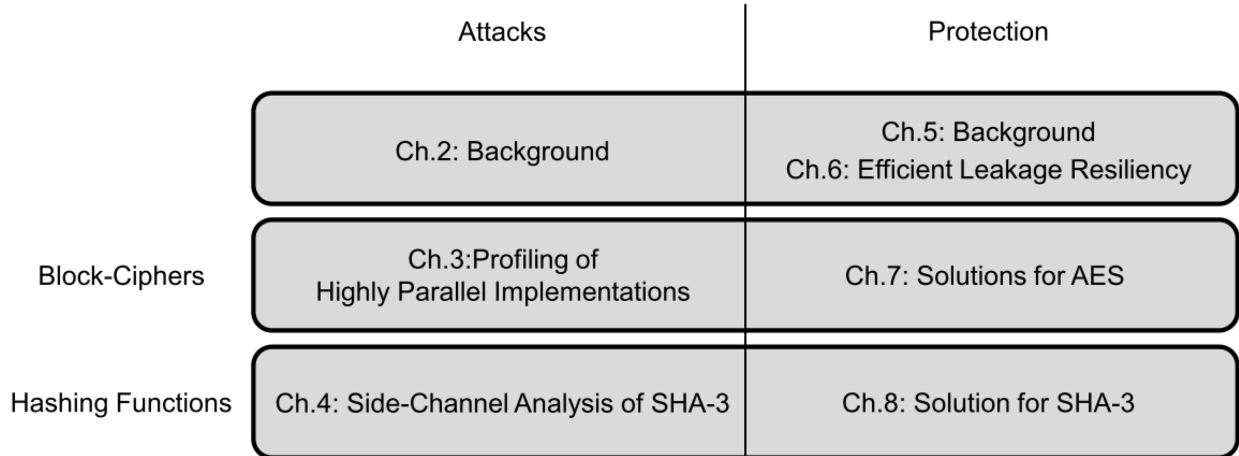

Figure 1.2: Outline of contributions.

Indeed, it is a reasonable assumption that, any cryptographic system without an adequate SCA protection can be broken.

1.1 Outline

We demonstrate several advances in the SCA of symmetric cryptography, as shown in Figure 1.2. The outline of this dissertation, the addressed challenges and contributions are briefly discussed as follows:

Part. 1.1 focuses on side-channel attacks. A brief background about different attack methodologies and a typical attack setup are covered in Chapter 2. Then, we address the following challenges:

1. *Profiling of Highly Parallel Implementations:* Profiling, in the context of SCA, characterizes the power consumption of a fully-controlled cryptographic module to extract accurate power signatures. The power signatures are used to attack similar modules. Parallel hardware designs, built with FPGA's and ASIC's, employ high design parallelism to achieve high computation speed. These parallel architectures represent

challenging targets for profiling. The captured power trace shows the system power consumption of all the parallel logic modules. While analyzing one logic block, activity in all the other logic blocks will add to the noise. This noise is called Algorithmic-Noise because it is generated by the algorithm itself. Our challenge is to improve SCA against highly parallel hardware modules.

Contribution: We propose a novel profiling technique that uses regression analysis to separate the power signature of each logic block. We use matrix representation to map every power trace of the profiling set to the activity of all the logic blocks. Then we use regression to recover accurate power signatures for every logic block. The proposed profiling technique achieves a higher attack success rate than the previous work. Results of this research are covered in Chapter 3.

2. *Side-Channel Analysis of SHA-3:* SHA-3 is a secure hashing function. It accepts a message of arbitrary length and generates a digest of a fixed length. Hashing is used as the core function in many cryptographic structures. SHA-3 was a competition by the National Institute of Standards and Technology (NIST) to choose a new hashing function. Recently, the competition ended with Keccak [15] as the winner. Keccak is the first hashing function to use a variable-length key in its keyed application: MAC-Keccak (Message Authentication Code using Keccak). Our challenge is to study the SCA properties of Keccak, and the effect of the variable-length key on the analysis.

   Contribution: We analyze the operations of Keccak for vulnerabilities to SCA. We also analyze the effect of changing the key-length on the overall effort required in SCA. Interestingly, we find that increasing the key-length does not always increase the required effort in the attack. Hence, we highlight some specific values of key-length where SCA becomes most difficult and name these values as the optimum key-length. Indeed, changing the key-length changes the attack methodology. Hence, we propose a systematic algorithm to choose an attack methodology based on the key-length, and we demonstrate complete attacks against five different case studies. Results of this research are covered in Chapter 4.

Part. 4.8 focuses on protection techniques against SCA. Chapter 5 highlights a brief background about SCA-countermeasures. Then, we address the following challenges:

3. *Efficient Leakage Resiliency:* In order to protect the implementation of any cryptographic primitive against SCA, a countermeasure must be employed. Unfortunately, all practical countermeasures (hiding and masking, as will be discussed) trade performance or area for implementation security with a higher than 2x implementation cost. One track of research, namely: leakage resilient cryptography, tries to change the secret key before every execution, hence prevents any differential attack. However, the proposed solutions were inefficient and required very high performance overhead. Our challenge is to design a practical and efficient leakage resilient encryption schemes.
   Contribution : We study the minimum requirements for heuristically secure leakage resilient schemes. Then, we highlight a generic framework for efficient leakage resiliency through lightweight key-updating. We propose two complete solutions for protecting AES modes of operation. The first one uses a dedicated circuit for key-updating, hence achieves negligible performance overhead at some area overhead. The other solution utilizes the cryptographic properties of the underlying AES itself, hence has negligible area overhead and small performance overhead. A system designer should choose between these two alternatives according to the scarce resource in the underlying module. The framework for key-updating will be covered in Chapter 6. The solutions for AES modes of operation will be covered in Chapter 7.

4. *Solution for Keyed Applications of SHA-3:* SHA-3 provides the interesting opportunity to have a single core that can perform hashing, MAC generation, authenticated encryption and more. However, using an SCA-protected core to perform unkeyed applications (e.g. regular hashing) involves a huge loss of performance (3x to 4x). Our challenge is to design a SHA-3 core that can be protected for the keyed applications at a marginal overhead, while being able to run normally for unkeyed applications.
   Contribution : We propose a novel SCA-protected core for SHA-3 that employs a new

message format and requires only two gates at 3.7 GE. For unkeyed applications, there is no loss of any kind. For keyed applications, there is a one-time performance loss that can be trivialized at long message lengths. Our contribution is essential to provide a single core for all the keyed and unkeyed applications of SHA-3. Results of this research will be covered in Chapter 8.

At the end, Chapter 9 concludes the dissertation.

Part I

Side-Channel Attacks

Chapter 2

Background on Side-Channel Attacks

SCA exploits any unintentional source of information leakage in the implementation. The real threat of SCA lies in the ability to mount attacks over small parts of the key, and to aggregate the information leakage through different runs to recover the full secret. SCA exploits weaknesses in the implementation of a module rather than exploiting weaknesses in the algorithm itself. Generally, a cryptographic module is vulnerable to SCA attacks if:

1. There is any internal variable that depends on the secret key.
2. And, this internal variable affects any observable quantity.

Figure 2.1 shows these conditions. Any internal variable that fulfils these conditions is called a sensitive variable.

Consider for example Algorithm 1, which shows a naive example of information leakage. The algorithm is a password checker. It assumes eight bytes of a secret key stored in variable `Key`, and eight bytes of user input stored in variable `Try`. The code sequentially checks every byte for correctness. On the first mismatch, the code returns `False` and `Break`.

Regarding functionality, this code works. However, the variable that holds the output of the `IF` statement is a sensitive variable. Indeed, it depends on the secret key and affects the

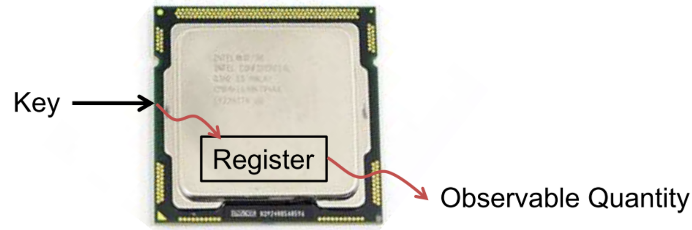

Figure 2.1: Conditions for successful SCA.

Algorithm 1 Password Checker

```
Require: Key[8]
Require: Try[8]
  for i = 0:7 do
    if Try[i] = Key[i] then
      Valid = true;
    else
      Valid = false;
      break;
    end if
  end for
    return Valid
```

execution time. A result of '1' (match) will initiate another loop while a result of '0' will terminate the loop.

With this in mind, Eve can enumerate all the guesses of the first byte, while measuring the execution time. The input that results in the longest execution time (due to starting a new loop), will match the correct secret key. Then, Eve can focus on the second byte, and so one.

Recovering information about small parts of the key is called 'divide-and-conquer' principle. This principle enables the real threat of SCA. Indeed, the brute force attack against such system should require 256^8 trials. Using SCA, particularly divide-and-conquer attack,

Figure 2.2: Possible side-channel outputs.

the brute force attack requires only $256 * 8$ trials. To put these numbers into perspective, if one try requires 1 μsec, the system should withstand $585K$ years against brute force attack. However, it will withstand only 2 msec against SCA attack. This naive example shows that, security of embedded systems cannot tolerate the threat of SCA, and a sound countermeasure must be employed.

Unfortunately, these implementation problems are not comprehended by cryptographers, because the theoretical security is not affected. Also, these problems are not fully comprehended by typical engineers, because the code is functioning properly. This elevated the need for cryptographic engineers, to focus on these types of problems.

2.1 Side-Channel Outputs

The previous example discussed leakage through execution time, however, Eve can exploit many other side-channel outputs, as shown in Figure 2.2. A side-channel output is any measurable quantity that is generated as a by-product of computation. Paul Kocher used the execution time to recover the private key of an RSA implementation [55]. He also exploited the instantaneous power consumption to reveal the secret key of DES [58]. Electromagnetic radiation was used by Agrawal *et al.* against DES and RSA [7]. Acoustic leakages were harvested from an RSA implementation on the CPU of commodity laptop computers [41]. Photonic emissions (light) were used against an AES implementation [89].

The execution time is a remarkable side-channel because it does not require close contact with the device. It can be demonstrated over small networks [27]. Electromagnetic attacks can also be used over relatively long distances [71]. Power attacks have the largest share of research in SCA [64], because the power trace is rich in information (compared to execution time). Also, power attacks need low cost equipment (compared to electromagnetic radiation) [80]. Still, protection against power attacks is a challenging target.

2.2 Attack Methodologies

There are two different methodologies to mount SCA attacks, simple analysis and differential analysis. Hereafter, the power consumption will be used as the intended side-channel output, however SCA methodologies are generic to any other output.

2.2.1 Simple Power Analysis (SPA)

Simple Power Analysis (SPA) exploits only one power trace to recover the secret key. One trace may represent the average of many power traces collected at the same inputs [58]. For example, SPA is possible when the key is used in conditional branching. Here, a sequence of instructions is performed only at a certain value of the key. If that sequence can be identified from the power trace, SPA can be used to directly read out the secret key. Also, Eve can use a cloned device to prepare a power signature for every possible value of a subkey (the considered small part of the key). Then, she can compare the measured trace to the power signatures searching for the best match. Protection against SPA can be achieved by preventing the use of secret values in conditional branching and by using random noise generators. The limitation of SPA is that, it does not combine information from traces at different input messages. Moreover, any noise in the measured power trace can turn SPA ineffective.

2.2.2 Differential Power Analysis (DPA)

Differential Power Analysis (DPA) was proposed to overcome the limitations of SPA [58]. In DPA, Eve tries to link the change in the input message from trace to trace, to the change in the corresponding power consumption. Here, DPA can effectively combine information from traces at different inputs. Hence, protection against DPA becomes much more difficult. Moreover, the effect of noise on DPA is minimal because, Eve can overcome any noise by increasing the number of traces.

DPA, at a high abstraction level, consists of two phases. Eve builds a power model that mimics the power consumption of the target module at a specific point in the algorithm (the sensitive variable). Then, she uses a distinguisher to compare the measured traces against the modeled ones searching for a key that results in the best match. Practically speaking, a complete attack requires five steps as shown in Figure 2.3:

1. Eve analyzes the algorithm searching for a sensitive variable that depends on both the input and a small segment of the key (subkey).
2. She uses the target module to collect a set of actual power traces at known input messages.
3. She calculates the sensitive variable for every possible key guess using the same input messages.
4. The power model is used to map the calculated variables into equivalent power consumption. The figure shows the Hamming Weight as the power model.
5. Finally, she uses a distinguisher to compare the modeled power trace to the actual power trace searching for a subkey that results in the best match.

Eve should repeat the last three steps for every subkey until the entire secret key is revealed. A huge body of research has been dedicated to the improvement of power models and distinguishers, as will be shown after defining the sensitive variable.

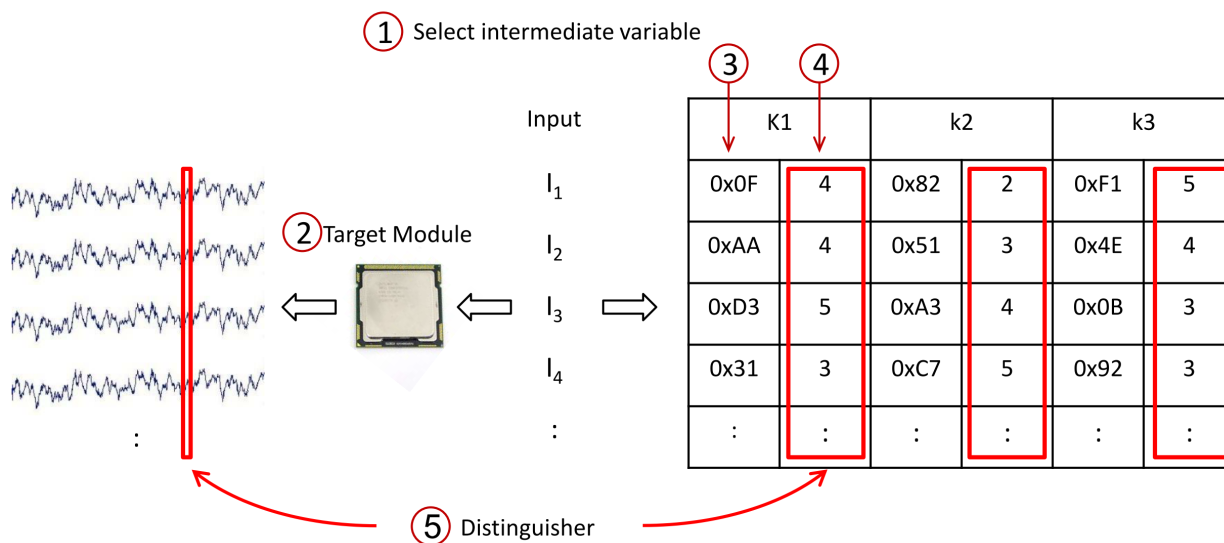

Figure 2.3: The attack model of DPA.

Sensitive Variable A sensitive variable is any variable within the cryptographic algorithm that depends on both the key and the input message with as low confusion and high diffusion as possible. The sensitive variable should also affect the leakage trace in some way. Variables that only depend on the key will generate the same power consumption at every trace, which can be easily missed in the measurement noise. Low confusion means that a small part of the key (subkey) affects the sensitive variable, i.e. a variable that can enable divide-and-conquer. Low confusion can typically be achieved by attacking the first/last round of the cryptographic algorithm. High diffusion means that any change in the subkey affects the entire sensitive variable, as typically achieved by non-linear operations in cryptographic algorithms. High diffusion helps DPA in finding the exact secret subkey as, if one bit is miss-guessed, the inspected power consumption will completely change.

Power Model A power model is a function that maps an internal variable into corresponding power consumption. The simplest power model is the *Hamming Weight* model, which is the number of ones in the binary representation of the variable. Indeed, the power required to change the state of a register from zero to a certain value, depends on the number of ones

in the binary representation of that value. If the initial state of the register is not zero, the *Hamming Distance* model can be used which is the number of bit flips between the initial and current state of a register. Eve can also use a fully-controlled cloned device to capture a power signature to every instance of a sensitive variable. Then, the captured signatures can be used as a power model. Using a cloned device to build power model is called *Profiling*. Collecting a signature for every instance of a sensitive variable (full profiling) is called the *Template* attack [26].

Another powerful attack is to directly compare the power consumption of two different traces, without using any power model. If a key guess lets two sensitive variables match, their power signatures should also match. This technique is called *Collision* attack [75]. Collision attack can also be mounted against two different points within the same trace [60].

Distinguisher A distinguisher is a mathematical formula to measure the similarity between measured traces and modeled ones. A distinguisher should be applied between the measured traces and the modeled traces at every possible guess of the secret key. Then, Eve should choose the key that results in the best match (highest / lowest output of the distinguisher). The simplest distinguisher is the *Distance of Means*, where Eve groups the traces that correspond to a sensitive value of '1' versus those that correspond to a sensitive value of '0'. If the two groups have a significant difference between their means, the key guess was correct. Otherwise, the key guess was wrong. The *Pearson Correlation Coefficient* can also be used to measure the linear relationship between the measured and modeled traces [24]. The traces can also be treated as random variables, where the *Mutual Information* can be used to find the amount of information sharing between them [42].

2.3 Typical Attack Setup

Figure. 2.4 shows the attack setup that we used to mount power attacks. The PC is used to

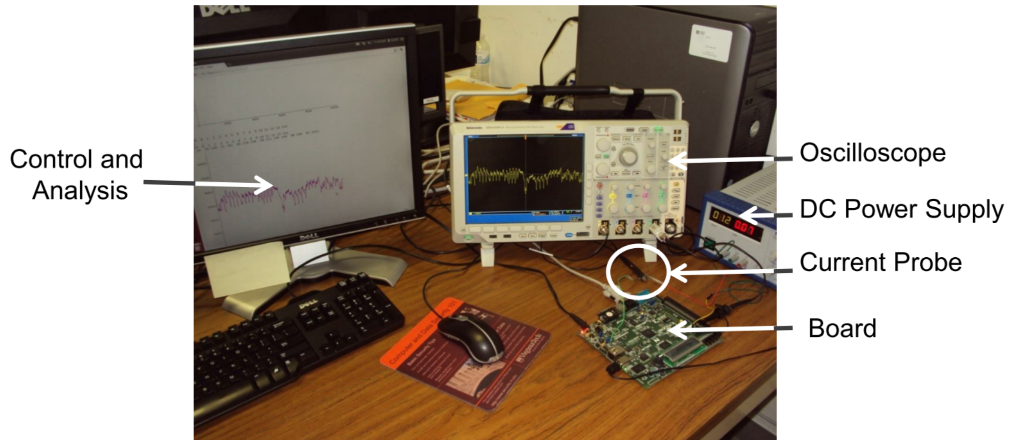

Figure 2.4: A typical setup for SCA.

program and control the target board. The target board executes a cryptographic algorithm. A DC power supply is used to feed the computing core of the board. A current probe measures the current flow to the computing core, which indicates the instantaneous power consumption. An oscilloscope captures the power trace and send the trace data back to the PC for analysis.

Recently, several initiatives tried to support the research in SCA. Many embedded boards were designed with an easy access to the power and synchronization signals [87, 54]. Other research focused on designing mathematical packages for analysis under different attack strategies [83]. Also, modular frameworks were proposed to control the target board, the measuring device, and the analysis code right from one scripting language [53, 81].

Chapter 3

Profiling of Highly Parallel Implementations

The major challenge faced by SCA in plain implementations (without countermeasures) is the noise. The dominant source of noise, especially in hardware designs, is the Algorithmic Noise. Hardware designs in ASIC's and FPGA's are inherently parallel. To achieve one round per clock, cryptographic modules are designed so that, all the input bits are processed simultaneously in small parallel logic blocks. While guessing one subkey in the attack, the power consumption of the other logic blocks will alter the measured trace by unknown values. Developing a profiled attack for this noisy environment is the objective of this chapter.

In this chapter, many of the DPA steps mentioned earlier will be revised. We start with the definition of 'Targeted Operation'. Then, we propose a novel profiling technique that takes the high design parallelism into consideration. The attack phase is also improved. We define two new targeted attack regions in the power trace, and we aggregate the attack results from each of them to get a more powerful attack phase. The results for AES running in an FPGA will be presented as a case study.

3.1 Related Work

The main contribution in this part is to introduce a new profiling technique. This section presents an overview of the common profiling techniques that have been used in the literature and how the proposed method is different and more suitable in the presence of high Algorithmic Noise.

Chari *et al.* presented the first profiling technique which is Template Attack [26]. Their main concern was to build a noise model for the power traces. They used the average and covariance matrix of a set of power traces measured at the same input as a template for that input. The profiling set should be used to build a template for every possible combination of plaintext and subkey. The template will be the basis of a multivariate normal distribution representing the noise model. In the attack phase, the authors used the maximum likelihood to find a template that best matches the attack traces which will lead directly to the correct subkey. Typically, the size of the profiling set should equal the size of all input combinations which can be reduced by building templates with respect to the sensitive variable directly [64]. The weak point of the Template Attack is the limited number of trace points that can be supported in the covariance matrix. The size of each covariance matrix grows quadratically with the number of trace points. Moreover, the complexity of calculating the matrix inversion, which is required in the attack phase, grows almost cubically with the number of trace points (depending on the algorithm used).

Schindler *et al.* proposed the Stochastic Attack [88] where they used regression analysis to give a different weight to every bit of the sensitive variable, as a way to improve the accuracy of the Hamming Weight (or Hamming Distance) model. The weights are estimated based on approximating the profiling traces to a function of the bits of their corresponding values. Schindler *et al.* assumed that the leakage of each bit is separable and that the system power trace is the sum of the leakages of these individual bits. However, the reality is more complicated, and the leakage depends on the relation between bits. This led to the extension of analysis dimensions to cover the effect of each two bits (37 dimensions) [51], three bits

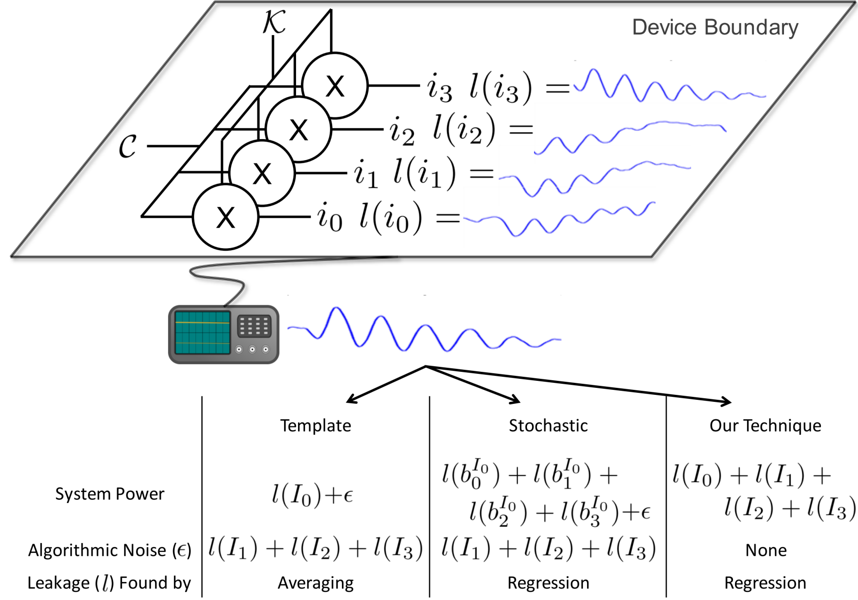

Figure 3.1: Comparison between different profiling techniques.

(93 dimensions) and so on, up to all the eight bits in 255 dimensions [50].

The difference between our proposed technique and the two previously mentioned profiling techniques is highlighted with an example in Figure 3.1. We assume that the targeted device uses four parallel logic blocks, the data width is four bits, and the Algorithmic Noise (ϵ) is the dominant source of noise. The source of Algorithmic Noise is the power consumption of other logic blocks running simultaneously with the targeted block. We also assume that:

- $\mathcal{C}$: The ciphertext.
- $\mathcal{K}$: The correct key used to collect traces.
- I_x: The sensitive variable at logic block number x.
- $b_y^{I_x}$: Bit number y of the sensitive variable I_x.
- $l(z)$: The leakage function of z.

The leakage function is the one-to-one mapping between the sensitive variable and the corresponding power consumption in the power model. The figure shows the sensitive variable as a function of both a subkey and a small segment of the ciphertext. The targeted device runs several logic blocks simultaneously, and each of them creates one sensitive variable and consumes a certain power. The system power trace is the aggregated sum of the power consumption of all the logic blocks. The Template Attack expects to see the leakage of one sensitive variable. It extracts an estimate of the power signature by averaging over a large number of traces to cancel out the effect of Algorithmic Noise. The averaging should be done with respect to every possible sensitive variable. The Stochastic Attack expects to see the aggregate leakage of individual bits (or combination of bits) of one sensitive variable. It extracts an estimate of the leakage of each bit by using the linear least squares regression. The number of required traces should be much larger than the number of unknown leakage functions to cancel out the effect of Algorithmic Noise.

Note that, all subkeys used in different logic blocks are known in the profiling set. This information can be used to build a more accurate power model. Our proposed technique understands that the system trace is the aggregate leakage of processing several sensitive variables. We estimate the leakage function of each sensitive variable (the value itself, not the individual bits) by regression. Hence, our technique is a mix between the previously two mentioned techniques. The design of our profiling technique acknowledges the presence of Algorithmic Noise, and uses it to get a more accurate power model. It is worth mentioning that, the Algorithmic Noise is removed in the profiling phase of our attack. However, this noise will still affect the attack phase.

3.2 Proposed Attack Model

In this section, we first present several preliminaries that are required by the proposed attack. The actual attack model is presented later on.

3.2.1 Targeted Operation

The term of 'Sensitive Variable' is not suitable for hardware modules, because the Hamming Distance model, for example, uses the xoring between two sensitive variables where the result of xoring is not an internal variable by itself. Hence, we propose to use the term 'Targeted Operation' which is more generic and intuitive.

The Targeted Operation is an operation running in the module where a sensitive variable represents its input or output. Eve should search for its power signature in the recorded trace. The Targeted Operation has an index and a dimension. The operation index is a number that uniquely identifies the leakage of a single instance of the operation, and that differentiates it from other instances with different leakages. The index must be data dependent and include the key as the sole unknown. If the same index is given to more than one instance of the operation, Eve acknowledges that the leakage of these instances is considered the same. For example, if we target the operation of charging a byte register, with the Hamming Weight as the index, we implicitly acknowledge that the instance of charging the register to 0x0F (in hexadecimal format) has the same leakage as the instance of charging it to 0xAA as they both have the same Hamming Weight of 4. The best index size is the one that allocates a different value to every leakage value. Template Attack is therefore better than the Hamming Weight as it assigns the byte value itself as the operation index, hence it can differentiate between the aforementioned instances of charging a byte register.

The operation dimension is the number of sensitive variables that are used to evaluate the index. For example, if we use the Hamming Distance as the index in the previous example, we need to evaluate the input to the register in two consecutive clock cycles in a 2-dimensional operation. The best dimension is the one that represents the actual source of power consumption. For example, if the power consumption depends on the switching activity, the 2-dimensional operation should be used.

Considering the same example of charging a byte register as above, if we use the Hamming Distance as the index, we acknowledge that the instance of updating the register value from

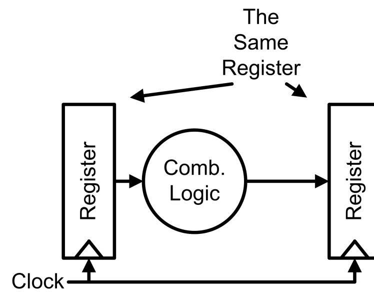

Figure 3.2: The extra round in hardware implementations.

0x0F to 0x1F has the same leakage as updating it from 0x07 to 0x0F as they both have the same Hamming Distance of 1. The index can be improved to be the result of xoring the two input values. In this case, the index size will be 256. This index can now differentiate between the aforementioned instances. The xoring has been used as the index in the stochastic attack of [50], and will also be used in this research.

3.2.2 The Extra Round

Figure 3.2 shows an implementation of one round of a round-based cryptographic algorithm utilizing one round per clock. The combinational logic shown includes at least an xor with the key, a non-linear diffusion operation, and a mixing operation for confusion. Once the input is ready in the register, the signal will pass through the combinational logic to generate the output and wait for the clock signal to store it in the same register. In the final round, the circuit behavior will still be the same. Once the final output is stored in the register, the combinational logic will execute and evaluate the output, which is never stored or used. This means that, the combinational logic runs on the final output one extra round; an operation that is not intended to happen. Here, we exploit this point and target the 2-dimensional switching activity of the register and the combinational logic between the final and extra rounds. The new attack region is marked at 'Region 1' in Figure 3.3 along with the time of the last two rounds in a symbolic representation of the power trace. This wide attack region increases the amount of information extracted from each trace for a more powerful attack.

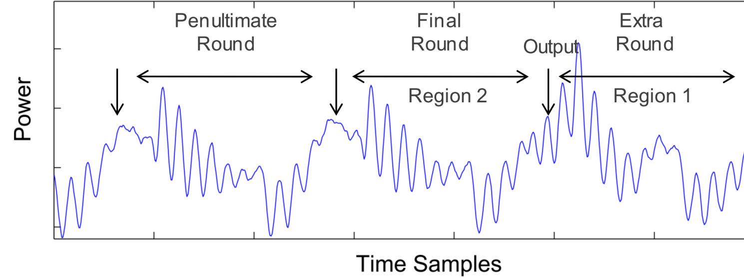

Figure 3.3: Two different attack regions.

3.2.3 Assumptions

In the proposed attack model, we define a *unit* as the register and the combinational logic circuit following it. The problem in extracting the power model is that the system power trace is the aggregated sum of all the units. Our objective is to separate them out into different power traces that represent different units.

We follow the same assumptions used in Section 3.1, and further assume that:

- R: The output of the penultimate round.
- N_U: The number of parallel units.
- R_u, $\mathcal{C}_u$: The inputs to unit number u.
- I_u: The index at unit number u, $I_u = R_u \oplus \mathcal{C}_u$.
- N_I: The size of the operation index.
- P: The system power trace.
- N_P: The number of profiling traces.

The Targeted Operation is the 2-dimensional operation of updating a unit input from R_u to $\mathcal{C}_u$. The operation index depends only on the unit inputs, hence we assume that the leakage of every unit is the same if they were fed with the same input regardless of the location of

the unit within the chip. This assumption is fairly reasonable as they were implemented using the same RTL, and they are different only for their place and route. This means that we need to have only one set of templates that are reusable over all the processing units.

In the following, we use a set of profiling traces P to extract an estimate of l for every instance of the unit, which is the unknown leakage model. Then we use the power model to analyze the set of attack traces at an unknown key.

3.2.4 Profiling Phase

In the profiling step, we assume that Eve collects a set of traces using known keys. The measured power trace will be the aggregated sum of the power consumption of N_U different units

$$P = \sum_{u=0}^{N_U-1} l(I_u) \tag{3.1}$$

Now consider that, every unit can implement N_I different instances of the operation. We assume that $g(i)$ is the number of units executing in an operation instance with $I = i$. Hence, the power consumption trace will be

$$P = \sum_{i=0}^{N_I-1} g(i)l(i) \tag{3.2}$$

Note that, the summation of $g(i)$ over i will always be equal to N_U. For N_P profiling traces, the power consumption can be expressed in a matrix format as

$$\mathbf{P} = \mathbf{G} \times \mathbf{L} \tag{3.3}$$

The size of matrix $\mathbf{P}$ is $(N_P \times C)$, where C is the number of samples covering the attack region. The size of $\mathbf{G}$ is $(N_P \times N_I)$. The size of $\mathbf{L}$ is $(N_I \times C)$. The best solution for $\mathbf{L}$ can be found using the least squares equation

$$\hat{\mathbf{L}} = (\mathbf{G}'\mathbf{G})^{-1}\mathbf{G}' \times \mathbf{P} \tag{3.4}$$

In case of a large index size and relatively small number of parallel units, the matrix $\mathbf{G}$ will be a sparse matrix with a maximum of N_U/N_I non-zero elements. The first row of $\hat{\mathbf{L}}$ is $\hat{l}(0)$, the second row is $\hat{l}(1)$ and so on. The matrix $\hat{\mathbf{L}}$ will be used as the power model in the next section.

3.2.5 Attack Phase

In the attack phase, we target a single key byte and choose a key guess. The key guess along with the input plaintexts can be used to calculate the operation index. Next, the matrix of modeled traces will be built by mapping every operation index to its corresponding model out of the $\hat{\mathbf{L}}$ matrix. Once the matrix of modeled power traces is built, a distinguisher will be used to compare modeled traces to measured ones. In this work, we need to compare a large range of trace points in the attack region hence; we use the sum of point-by-point correlation which reveals the complexity of calculating the covariance matrix associated with Template Attack. We assume that

- $\mathbf{P}$: The matrix of attack traces.
- $\bar{K}$: The subkey guess used to build the modeled traces.
- $\bar{\mathbf{P}}_{\bar{K}}$: The matrix of modeled traces.
- $D_{\bar{K}}$: The distinguisher result using $\bar{K}$ as the key guess.

We also assume that $\mathbf{P}_c$ and $\bar{\mathbf{P}}_{\bar{K},c}$ are column number c of $\mathbf{P}$ and $\bar{\mathbf{P}}_{\bar{K}}$ respectively. The distinguisher will be

$$D_{\bar{K}} = \sum_{c=0}^{C-1} \frac{cov(\mathbf{P}_c, \bar{\mathbf{P}}_{\bar{K},c})}{\sigma_{\mathbf{P}_c} \sigma_{\bar{\mathbf{P}}_{\bar{K},c}}} \tag{3.5}$$

The distinguisher should be calculated for all the possible key guesses. The correct key is the one that maximizes $D_{\bar{K}}$.

3.2.6 Two Regions

Every point of the power trace gives more information to the attack, only if we know how to use it. The idea of aggregating separate attacks against different points of the trace has been proposed in [11]. Here, we use a similar idea to aggregate the attack against the extra round with the attack against the final round. In this section, we describe how to build an attack against the final round ('Region 2' in Figure 3.3), and how to aggregate the attack results from the two regions.

Previously, we used the knowledge of the output of the penultimate round R in a 2-dimensional attack against the extra round. Here, we use the same value in a 1-dimensional attack against the final round. A 2-dimensional attack cannot be used here, because calculating the previous value of the register requires passing through another round up which mostly involves a confusion step. Passing through a confusion step requires the knowledge of all the subkeys involved, which violates the divide-and-conquer principle and broadens the search space.

The previously discussed analysis can still be used with only minor changes

- I_u: The index at unit number u, $I_u = R_u$.

The Targeted Operation is the 1-dimensional operation of storing R_u in a unit. The attack points involved in P should be those samples in time that correspond to the final round.

The combined distinguisher will be:

$$D_{\bar{K}} = D1_{\bar{K}} + D2_{\bar{K}} \tag{3.6}$$

where $D1_{\bar{K}}$, is the result of attacking the extra round and $D2_{\bar{K}}$ is the result of attacking the final round. The interesting point about attacking two different regions in the same trace is that the noisy wrong result in a one region will not show up (with high probability) in the other region as well however, the correct key will give high correlation values in both of them.

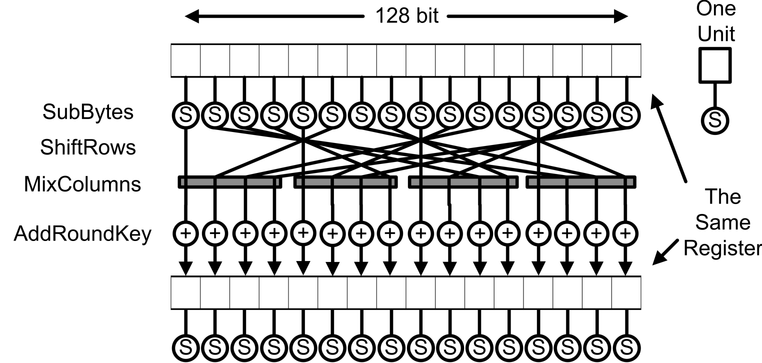

Figure 3.4: The last round of AES.

3.3 Results of a Case Study: AES

The DPA contest [3] offers a common set of power traces to make it possible to compare the performance of different SCA attack models. The traces are collected for a hardware 128-bit AES encryption algorithm running on SASEBO-GII board. There are two public sets of traces, a profiling set of 1 million traces, and an attack set of 20,000 traces for 32 different random keys (640,000 traces). Those keys are known, and used by the researcher to characterize his own attack model. In this section, we target this implementation (which is not our design) to demonstrate the existence of the extra round and the efficiency of the overall attack.

3.3.1 The Target Implementation

The AES design runs at one round per clock using 16 parallel S-box combinational circuits as shown in Figure 3.4. More information about the AES encryption algorithm can be found in the AES reference book [30]. The location of MixColumn function is shown in the figure with no effect on the data-path, as there is no MixColumn function in the final round.

In this case, the targeted unit is a one byte register and the S-box combinational logic following it where the number of parallel units N_U will be 16. The targeted operation in

the extra round will be the 2-dimensional operation of updating the unit inputs with the ciphertext. The operation index will be

$$I_u = \mathcal{C}_u \oplus \mathcal{S}^{-1}(\mathcal{SH}^{-1}(\mathcal{C}_u \oplus \mathcal{K}_u)) \tag{3.7}$$

where $\mathcal{S}^{-1}$ is the inverse of S-box function and $\mathcal{SH}^{-1}$ is the inverse of ShiftRows function. The targeted operation in the final round, will be the 1-dimensional operation of storing R_u in the unit. The operation index will be

$$I_u = \mathcal{S}^{-1}(\mathcal{SH}^{-1}(\mathcal{C}_u \oplus \mathcal{K}_u)) \tag{3.8}$$

The size of the operation index N_I in both cases will be 256. The attack regions are identified using the normal Correlation Power Analysis [24] with Hamming Distance Model.

3.3.2 The Extra Round

A simple test is conducted to check the existence of the extra round along with the effectiveness of the new modeling method. In this test, we calculate the correlation between the actual measured traces and the modeled traces in the attack region of the extra round. The modeled traces are built using the proposed regression model and the normal Hamming Distance model as a reference. The test is applied to key 1 of the public set, with all the 20000 traces. Figure 3.5 shows the result of this test. The peak correlation point in the Hamming Distance model is the point of updating the register value with the output ciphertext. The correlation goes down as the signal passes through the S-box circuit. However, the correlation of the proposed regression model achieves higher values in the region of S-box circuit. The correlation values in the figure are relatively high because we assume that we know the subkeys of all the units and aggregate the power models of them to cancel the Algorithmic Noise in this test. However, when we target only one unit in an actual attack, the 15 other units will be considered as noise. As previously mentioned, the Algorithmic Noise is removed in the proposed profiling phase however; it still affects the attack phase.

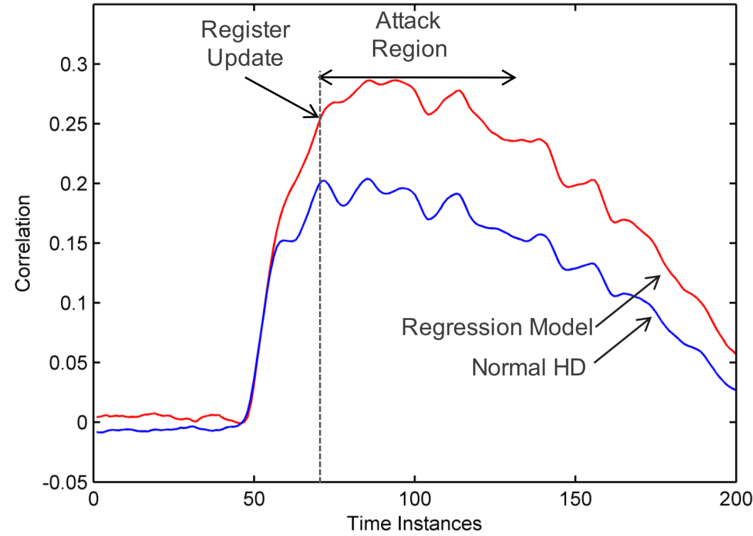

Figure 3.5: The proposed regression model and the Hamming Distance model.

3.3.3 Performance Metrics

The performance metrics used in this study are the same as those in the DPA Contest (V2):

- Global Success Rate (GSR): The number of traces so that all the subkey bytes are recovered concurrently.
- Partial Success Rate (PSR): The number of traces so that a selected subkey is recovered correctly. Every subkey byte will have its own PSR.
- Partial Guessing Entropy (PGE): The rank of the correct subkey within the result. Every subkey byte will have its own PGE.

3.3.4 Results

The proposed attack model has been used to attack the public set of the DPA Contest (V2). The GSR is shown in Figure 3.6. The figure shows that the proposed attack achieved 80% GSR at 4641 traces, where the 80% is calculated over the 32 experiments i.e. at least 26

Table 3.1: Results for the complete attack against AES.

Attack	GSR>80%	min PSR>80%	max PGE<10
Our Attack	2,755	2,226	1,420
Li[60]	2,256	2,155	3,181
Heuser[50]	3,589	2,748	1,356

experiments have been recovered completely by this number of traces.

This result reflects a side effect of using one power model to attack all the 16 logic blocks. The extracted power model is considered the average of all of them. However, there may be one (or more) logic blocks that have a shifted model due to differences in placing and routing. Hence, it will be difficult for our proposed attack to recover the correct subkey of this shifted logic block. As the GSR requires recovering all the subkeys concurrently, this shifted subkey will pull the GSR up.

To recover this side effect, we improve our attack with a 2/3 subkey byte search. In this improvement, we targete the penultimate round with a similar attack. The 2-dimensional operation at the penultimate round involves at least 10 subkey bytes because of the MixColumn step. In the 2/3 subkey byte search, we test all the combinations of the best 3 guess of the worst two bytes (a total of 10 tests). The choice of those worst bytes is done as part of the profiling phase. The GSR of the proposed attack including the 2/3 subkey byte search is also shown in Figure 3.6. This simple key search improve the 80% GSR from 4641 traces to 2755 traces.

To demonstrate the performance of the proposed complete attack, we compare its performance to the best two submissions of the DPA Contest in Table 3.1. The table shows the previously discussed 80% GSR. It also shows the minimum PSR $>$ 80%, where the 80% is calculated over the 32 experiments and the minimum reflects the worst case subkey out of the 16 subkey bytes. Finally, the table shows the maximum PGE $<$ 10, where the maximum reflects the worst case subkey out of the 16 subkey bytes.

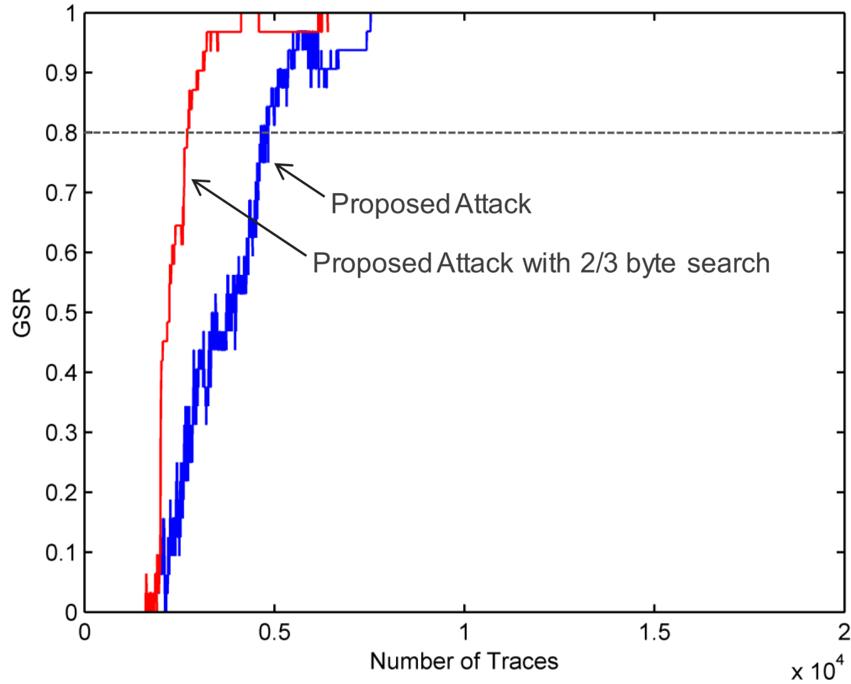

Figure 3.6: Global Success Rate.

3.4 Summary

In this chapter, we proposed a profiled attack model in presence of high Algorithmic Noise. We proposed a novel profiling technique that acknowledges the high noise in these designs. We also proposed two new insights in the attack phase. One insight is to exploit the effect of the fixed connections of combinational logic circuits. The other insight is to aggregate the results of attacking two different regions in the same trace.

Chapter 4

Side-Channel Analysis of SHA-3

A cryptographic hash function is a one-way function that converts an arbitrary-length message into a fixed-length digest. It is a fundamental step in the efficient implementation of electronic messages, where it can be used to ensure the integrity of messages. If the receiver generates the same digest as the sender, the message should be correct, can never be altered. When hashing the combination of a secret key and a message, a Message Authentication Code (MAC) is obtained. A MAC ensures integrity as well as authenticity of the message, as the verification of a MAC implies testing knowledge of the secret key.

Back in 2004, significant attacks on the standard hash functions of that time were getting published everywhere. These attacks almost completely broke MD5, SHA-0 and later on SHA-1. At that time, the United States National Institute of Standards and Technology (NIST) announced to phase out SHA-1 and replace it with the more stable, SHA-2. However being algorithmically similar to SHA-1, NIST had the fear that SHA-2 might itself get broken in the near future, and announced in Nov 2007 for a competition to select a new standard for cryptographic hashing, SHA-3. Out of 64 submissions, the competition ended on Oct 2012 with Keccak [15] as the winner. Keccak was particularly selected for being based on a new algorithmic construction, the *Sponge* construction, which is entirely different from previous hashing standards. This new construction opens new questions, challenges and opportunities

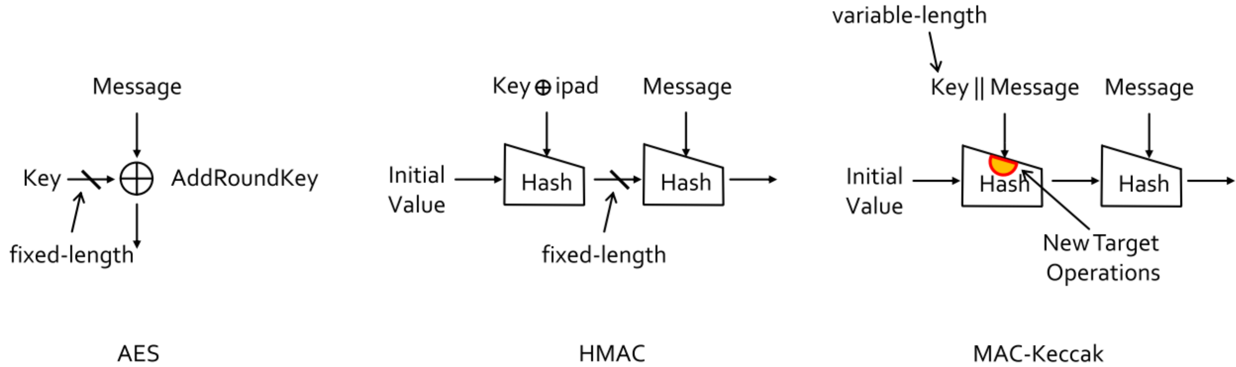

Figure 4.1: The difference between DPA of AES, HMAC, and MAC-Keccak.

in Side-Channel Analysis (SCA), which will be the focus of this chapter.

The idea of the *Sponge* construction is to have an internal state that is bigger than the input and output block sizes. This feature prevents the input block from directly affecting the entire state and protects the internal state from being fully exposed at the output. This construction allowed Keccak to securely create Message Authentication Codes (MACs) by hashing the direct concatenation between the key and the message in a cryptographic mode called MAC-Keccak [14]. Although previous MACs required that the key is hashed in a separate input block, MAC-Keccak allowed the secret key and the message to share one input block. Furthermore, the designers allowed the use of a variable-length key, which opens a new dimension in the analysis.

The difference between the DPA of MAC-Keccak and that of the typical cryptographic algorithms (e.g. AES and HMAC) is highlighted in Figure 4.1. The length of the secret key in typical cryptographic algorithms is fixed. Hence, the algorithm can be analyzed easily to select a set of target operations. The output variables of these operations are the sensitive variables. The AES encryption algorithm, for example, uses a fixed length key of 128, 192 or 256 bits and the target operation is the AddRoundKey [30]. Similarly, DPA on HMAC aims at recovering the internal state after hashing the key (xored with the ipad) [13]. In this case, the size of the required unknown is also fixed and equals to the size of the internal state,

where the target operation is the input to the next hashing cycle. On the contrary, the secret key in MAC-Keccak has a variable length, and it shares the message in one input block. As a result, changing the key-length (and consequently the message-length) within the input block, changes the set of target operations that need to be monitored for a successful DPA. Also, the target operations are located deep inside the algorithm itself instead of being only at the input. For instance, the key-length can be selected near the input block size to shrink the proportion of the input message. Here, a single target operation will not be sufficient due to the difference in size between the key and the known message. In such case, several consecutively dependent operations will be targeted to reach the point in the algorithm where every key-byte is affected by at least one message-byte. Hence, the DPA of MAC-Keccak requires new methodologies, which were not needed in the analysis of other cryptographic algorithms. These new features of MAC-Keccak along with the complexity of the Keccak algorithm itself formed the challenge addressed in this chapter.

In this chapter, we explore side-channel properties of MAC-Keccak, and the effect of changing the key-length. We show that increasing the key-length does not always increase the required effort in the attack. Hence, we highlight some specific values of key-length where SCA becomes most difficult and names these values as the optimum key-length. Also, we show that the complete recovery of the secret key requires mounting DPA against several consecutively dependent operations. Hence, we use a systematic approach to analyze MAC-Keccak under any key-length, to extract the required target operations, in the proper order of dependency. Finally, we present complete case studies of MAC-Keccak under several practically difficult key-lengths, and validate the analysis by successfully breaking the reference software code on a 32-bit Microblaze processor [2].

4.1 Previous Work

Previous work on DPA, including those dedicated to MAC-Keccak, haven't studied DPA under a variable key-length. McEvoy *et al.* resolved the dependency between target operations of HMAC with SHA-2, where the size of the secret key was fixed to the size of the secret intermediate hash value [68]. Zohner *et al.* presented an analysis step based on Correlation Power Analysis (CPA) to identify the key-length in use [97, 24] . They acknowledged the effect of changing the key-length on DPA, but they did not study the problem in detail. Relying on their method of identify the key-length, we assume that the key-length of MAC-Keccak is known upfront. Keccak developers proposed a side-channel countermeasure for MAC-Keccak based on secret sharing (masking) [29]. They presented results for attacking both protected and unprotected implementations of MAC-Keccak using simulated traces [16]. In their analysis, they studied the effect of changing the state-size assuming that the key-length is fixed and equals to the input block size (i.e. similar to previous MAC constructions). If the key-length changes, they will have to apply high-order DPA following the same attack methodology of this paper.

4.2 Background

Keccak uses a new *Sponge* construction chaining mode with a fixed permutation function called the Keccak-function. In the Sponge construction, the new input block affects only r (Rate) bits of the internal state [14]. The rest of the state, of size c (capacity) is not directly affected by the current input. The size of the internal state is $b = 25 * 2^l$ and $l = 0 : 6$ arranged in a 5x5x2^l array. The Rate and the capacity are used as a design parameter to trade security strength for throughput. Their sum is always equal to the state-size ($b = r+c$). To match the NIST output length requirements, the designers of Keccak proposed $b = 1600$ bits ($l = 6$), and $r = 1152$, 1088, 832 and 576 bits for output length of 224, 256, 384 and 512 bits respectively [18].

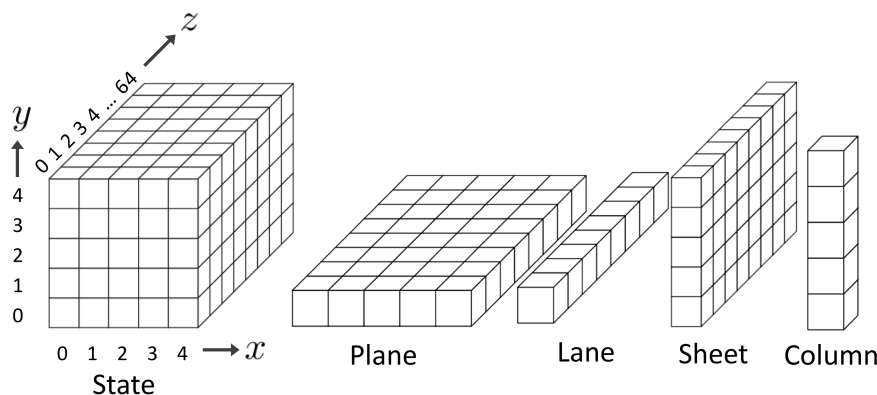

Figure 4.2: Terminology used in Keccak.

The internal state of Keccak is arranged in a 3-D array as shown in Figure 4.2. Each state-bit is addressed using three coordinates, and can be written as $S(X, Y, Z)$. The Keccak state also defines a *plane*, *lane*, *sheet* and *column*. These are defined as follows: a plane $\mathbb{P}(y)$ contains all state-bits $S(X, Y, Z)$ for which $Y{=}y$; a lane $\mathbb{L}(x, y)$ contains all state-bits for which $X{=}x$ and $Y{=}y$; a sheet $\mathbb{S}(x)$ contains all state-bits for which $X{=}x$; a column $\mathbb{C}(x, z)$ contains all state-bits for which $X{=}x$ and $Z{=}z$.

The hashing is done in three steps:

- *Padding and Initializing :* The input message is appended with a number of bits so that the total length is multiple of the rate. The padding starts with '1' and ends with '1' with all 0's in-between. Padding is mandatory, and therefore the minimum padding length is 2 and the maximum length is $(r + 1)$. Padding is shown in the first step of Figure 4.3. Initialization is done by setting the initial state to all zeros.

- *Absorbing :* Every r (rate) message bits are appended with c (capacity) bits of zeros to be of the same size as the state. Then, the bits get arranged in a 5x5x64 3-D array (called the state). The state is filled with r new data bits starting from $S(0, 0, 0)$ and filling in the Z direction, followed by the X direction, followed by the Y direction. The remaining part of the state (the capacity) is kept unchanged; it is filled with zeros in the first hashing operation. This filling sequence puts the new input bits in the lower

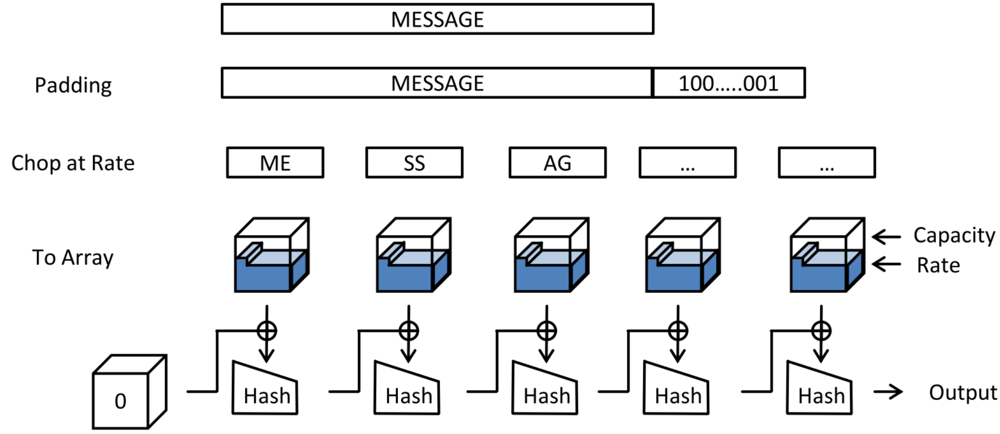

Figure 4.3: Keccak hashing algorithm and the padding rules.

planes (from $Y = 0$) leaving the zero bits at the upper planes, which is indicated by the shaded part in each state of Figure 4.3. The state is xored with the previous state. The result is used as the input to the Keccak function, which outputs a new state. The absorbing operation continues for every block of message bits as shown in the figure.

- *Squeezing :* The digest is the first bits of the output state.

The Keccak-function consists of 24 rounds of five sequential steps. The steps are briefly discussed here. Further details can be found in the Keccak reference [15]. The output of each round is:

$$Output = \iota \circ \chi \circ \pi \circ \rho \circ \theta(Input) \tag{4.1}$$

Throughout the following operations, operations on X and Y are done modulo 5, and operations on Z are done modulo 64.

- θ is responsible for diffusion. It is a binary xor operation with 11 inputs and a single output. Every bit of the output state is the result of xor between itself, and two neighbor columns:

$$S(X,Y,Z) = S(X,Y,Z) \oplus \left(\oplus_{i=0}^{4} S(X-1,i,Z)\right) \oplus \left(\oplus_{i=0}^{4} S(X+1,i,Z-1)\right) \tag{4.2}$$

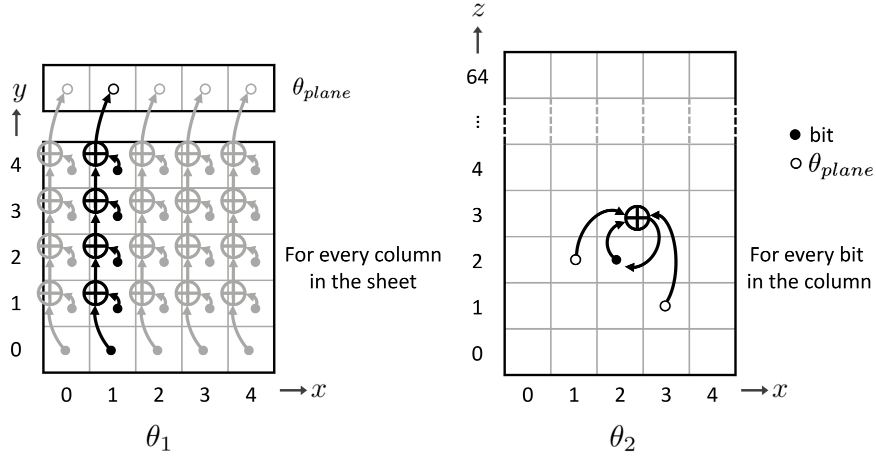

Figure 4.4: The two steps of θ operation, θ_1 and θ_2.

The θ operation is done over two successive steps. The first step θ_1 calculates the parity of each column. The result is θ_{plane}:

$$\theta_{plane}(X, Z) = \oplus_{i=0}^{4} S(X, i, Z) \tag{4.3}$$

In software implementations, θ_1 is implemented in incremental steps starting from $Y = 0$, as shown on the left of Figure 4.4. We name each xor operation by the first input, e.g. $\theta_1(X, Y, Z)$ is an xor operation with inputs: $S(X, Y, Z)$ and $S(X, Y+1, Z)$. The second step θ_2 computes the xor between every bit of the state and two parity bits of θ_{plane}, as shown on the right of Figure 4.4.

$$S(X, Y, Z) = S(X, Y, Z) \oplus \theta_{plane}(X-1, Z) \oplus \theta_{plane}(X+1, Z-1) \tag{4.4}$$

We name these operations by the name of the output bit, e.g. $\theta_2(X, Y, Z)$ is an xor operation with three inputs: the state-bit $S(X, Y, Z)$, the parity bit $\theta_{plane}(X-1, Z)$ and the parity bit $\theta_{plane}(X+1, Z-1)$.

- ρ is a binary rotation over each lane of the state.

- π is a binary permutation over lanes of the state. Every lane is replaced with another lane. In other words, π shuffles every row of lanes to a corresponding column. The lane indices before and after the π step is shown in Figure 4.5

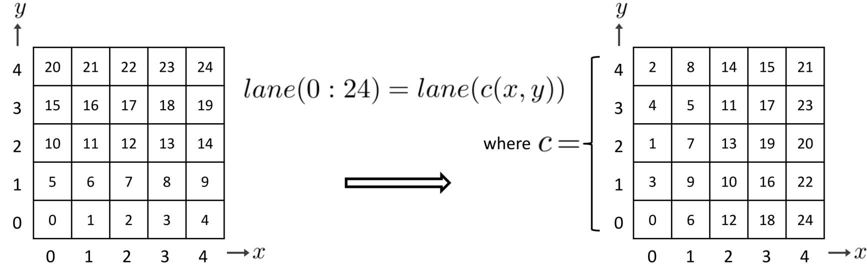

Figure 4.5: Lane indices before and after the π step.

- χ is responsible for the non-linearity. It flips a state-bit if its two adjacent bits along X are 0 and 1:

$$S(X, Y, Z) = S(X, Y, Z) \oplus \left(\overline{S(X+1, Y, Z)} \cdot S(X+2, Y, Z) \right) \tag{4.5}$$

  We name the χ operation by its output bit, e.g. $\chi(X, Y, Z)$ takes three state-bits: $S(X, Y, Z)$, $S(X+1, Y, Z)$ and $S(X+2, Y, Z)$.

- ι is a binary xor with a round constant.

It is clear that θ and χ are the only operations that can be targeted with DPA, as they involve mixing between key bytes and data bytes. However, ρ and π operations cannot be ignored; they are needed to track the location of data-dependent bytes in the χ operation.

As mentioned earlier, Keccak recommends a direct MAC construction, which is secured depending on the characteristics of the Sponge.

$$MAC(M, K) = H(K||M) \tag{4.6}$$

This construction features an arbitrary-length key, which raised the new challenge addressed by this research.

4.3 Effect of Changing the Key-Length

To study the effect of changing the key-length, we define the following quantities:

- *Number of Controlled Bits :* The number of message bits in the input block that are variable and known to Eve.
- *Number of Unknown Bits :* The number of unknown bits that are required to forge a MAC digest. Depending on the key-length, this can match the number of bits in the key or the number of bits in the hash state.
- *Attack Difficulty :* The ratio of the number of unknown bits over the number of controlled bits.

We assume that the key-length can be any number of bits greater than zero. Small lengths are an easy target for brute force, but there is no fixed minimum length. So, we keep it general to study the problem from SCA point of view. For this analysis, we assume that the total number of message bits is larger than the rate. In other words, whatever the key-length was, the message will fill up the rest of the input block with no effect of padding bits. This is a very reasonable assumption in typical applications of cryptographic hashing. We handle the key-length as a number of bits. Modifying the analysis to only complete bytes or words should be straight forward. There are two cases of the key-length (l_K):

- *key-length < rate :* In this case, the key will be part of the input block and the message will fill up the rest of the block. The number of unknown bits will be the key-length. The process of extracting these bits is called Key Recovery. The number of controlled bits will be the rate minus the key-length ($r - l_K$). The attack difficulty will be ($l_K/(r - l_K)$) ranging from ($1/(r - 1)$) to (($r - 1)/1$.
- *key-length ≥ rate :* In this case, the key will fill up the first input block and part of the second block. The message will fill up the rest of the second input block. The number of unknown bits will be size of the previous state (b). Note that the recovery of the rest of key bits in the second input block is not required where; the key bits in the second block will be xored with the previous state. The knowledge of the output of the xor operation is sufficient to mount a MAC Forgery. Hence, the number of unknown bits

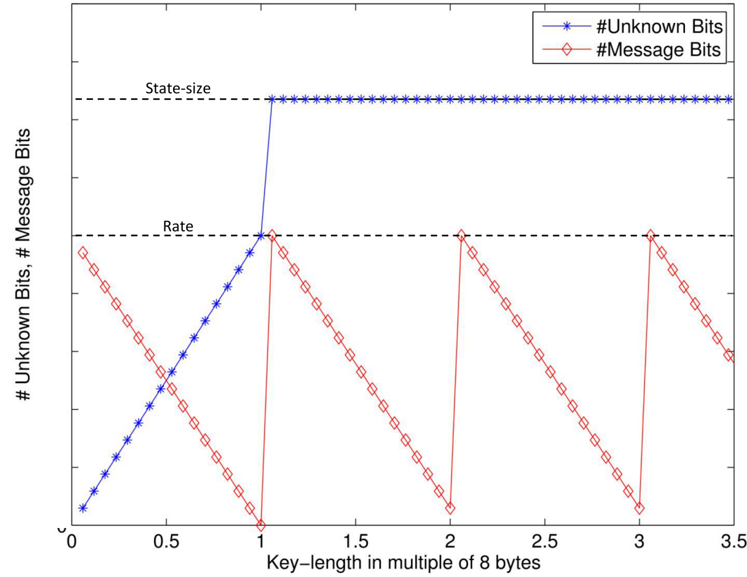

Figure 4.6: Effect of key-length on the number of controlled bits and the number of unknown bits.

will not increase beyond the state-size. The process of extracting these unknown bits is called MAC Forgery, as Eve is not required to recover the original key. She will only recover the required information to forge a MAC digest. The number of controlled bits will be the rate minus the key-length modulus the rate $(r - (l_K (\text{mod } r)))$. The attack difficulty will be $(b/(r - (l_K (\text{mod } r))))$ ranging from (b/r) to $(b/1)$.

Figure 4.6 shows the number of controlled bits and the number of unknown bits as a function of the key-length. The number of controlled bits in every block decreases by increasing the key-length where the sum of them will always be the rate. The number of unknown bits will be equal to the key-length while the key-length is less than the rate. Once the key fills up a complete input block $(l_K = r)$, the required unknown will be the previous state and the number of required unknown bits will be fixed at the state-size.

Figure 4.7 shows the attack difficulty as a function of the key-length. The figure also shows that the optimum key-length from SCA point of view is $((n * r) - 1)$, where $(n = 2 : \infty)$. Any increase of the key-length beyond $((2 * r) - 1)$ does not increase the security from SCA

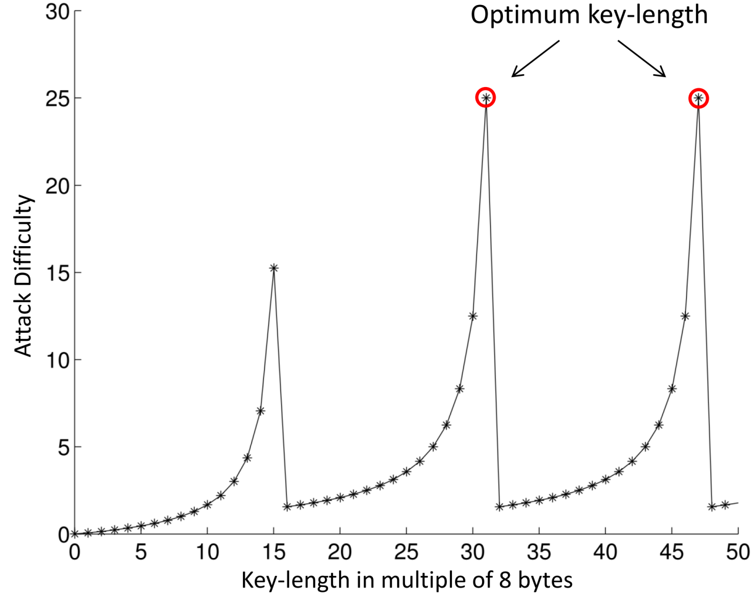

Figure 4.7: Effect of key-length on the attack difficulty.

point of view.

4.4 Targeted Operation

As discussed eariler, a Targeted Operation is an internal operation running within the module, where Eve searches for its signature in the recorded leakage. A Targeted Operation must depend on both the controlled bits and the key as Eve will search for the key that correctly links the change in the controlled input to the change in the power consumption. Hence, we assume that Eve targets the first hashing operation that includes the message as a part of the input block. That is the first hashing operation if $(l_K < r)$ and the $n+1$ hashing operation if $(l_K \geq n * r)$.

Here, we assume that the key prepends the message. The analysis can be easily ported to cases where the key is appended to the message. Following that assumption and that the state filling starts from bottom-up, the key bits will be in the lower planes and the controlled

bits will be in the middle planes. If the key-length is less than the rate, the upper planes will be filled with zeros. Otherwise, the upper planes will be filled with the previous state (key-state).

Due to the flexibility of the key-length in MAC-Keccak, no single targeted operation will be sufficient for all the cases. We study all the targeted operations that will be sufficient to mount an attack in any key-length. The operations are studied in the same order of the algorithm. The success of attacking a later operation depends on the success of attacking the previous operations, as the previous results will be used as assumptions in later attacks.

- *State xor:* The xor operation between the new input block and the previous state is the first targeted operation. This operation is useful only if the $(l_K \geq r)$, where the previous state is the required unknown. In this case, the State xor operation can recover the bits that are xored with the controlled bits $(r - (l_K (\bmod r)))$ out of the State-Size bits (b). The number of bits that are xored in parallel depends on the implementation ranging from all-at-once in hardware modules (FPGAs, and ASICs) to the processor width in software modules.

- *The θ operation :* Understanding the implementation of this step is critical in SCA. Typically, θ is done in two steps, first the module calculates the xor between the elements of each column (five elements) and generates what can be called θ_{plane}. Then, it calculates the xor between every element in the state, and two neighbor locations of the θ_{plane} as indicated earlier (Figure 4.4).

  The first step of θ : The analysis of the first operation (generation of the θ_{plane}) depends on the order of xoring the five elements as shown in Figure 4.8 where we assumed that the key-length is greater than the rate as a general case. The key-state bits (K_s) can be replaced by 0's otherwise. Generation of the θ_{plane} can be done all-at-once in hardware modules (FPGAs, and ASICs), or bottom-up or top-down in software modules, where the first method (bottom-up) is used in the reference software implementation [1].

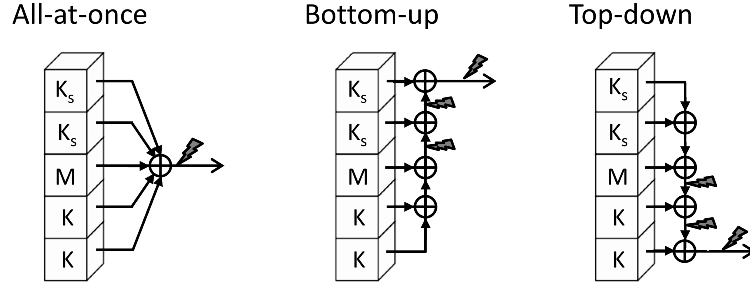

Figure 4.8: Column xor of θ operation.

- In all-at-once implementations, the output of the gate will take the effect of all the unknown bits at once. Hence, SCA cannot recover the value of individual unknown bits however; it can only get the result of xoring all of them. Targeting this operation can be a complete attack only if there is only one unknown bit in every column i.e. $l_K \leq p$; where p is the size of one plane (320 bits).
- In bottom-up implementations, all key bits will be xored with each other before being xored with a controlled bit. Then, the result will be xored with key-state (K_s) bits (if there are any) in sequential steps. Similarly, SCA can only reveal the result of xoring all the key bits. However in this implementation, SCA can recover all the key-state bits by monitoring the result of each xor operation (recover one key-state bit at every operation). Targeting this operation can be a complete attack only if there is only one key bit in every column ($l_K (\text{mod } r) \leq p$).
- In top-down implementations, the situation is exactly reversed from that in bottom-up implementations. SCA can only reveal the result of xoring all the key-state bits and it can recover the individual values of all the key bits. The role of bottom-up and top-down will be switched if MAC construction is built with the key appending the message.

Figure 4.8 shows the possible attack points in every implementation. In every case discussed above, if there is at least one controlled bit in every column ($r - l_K(\text{mod } r) \geq p$), the targeted operation will be enough to get the θ_{plane} which is a valuable

information for attacking the second step of θ.

The second step of θ: In this step, the algorithm calculates the xor between every element of the state and two neighbor elements of the θ_{plane}. If the θ_{plane} is correctly found in the previous step, SCA can recover all the unknown elements of the state.

To conclude, the two steps of θ can be used to build a complete attack only if there is at least one controlled bit in every column. For smaller number of controlled bits, Eve will have to follow the Keccak function in the later operations.

- *ρ and π operations :* These permutation operations cannot be used as an SCA target because they do not involve any mixing between known and unknown parts.

- *The χ operation :* This operation is a good SCA target. Every bit involved in this operation carries information from 11 controlled or unknown bits. Since it is a 3 input operation, the output will depend on 33 controlled or unknown bits.

- *The ι operation :* Similar to permutation operations, this operation does not reveal any SCA leakage because it does not involve any mixing between known and unknown parts.

Operation Width : A one last note in the analysis of the targeted operations is the width of the operation (the number of parallel bits included in the SCA). All Keccak functions are binary operations, which makes Eve free in choosing the suitable width i.e. she can build the CPA software to search for 1-unknown bit at a time, 2-unknown bits or one-unknown bye, etc... However, the choice of the operation width is a trade-off. If Eve chooses a one bit width, she will get a small search space of only $2^1 = 2$ but increased algorithmic noise where the power consumption of all the other parallel bits ((state-size - 1) in hardware modules and (processor-width - 1) in software modules) will contribute to noise. On the other hand, if Eve targets all the parallel bits at once (higher operation width), she will get zero algorithmic noise, but the search space will be huge 2 to the power of (number of parallel bits).

4.5 Systematic Algorithm

Here, we propose a systematic algorithm to identify the required target operations, in the proper order of dependency. We define a 'data-dependent variable' as any sensitive variable that depends on the input message, and 'unknown variable' as any sensitive variable that depends only on the key. The unknown variable should be constant from trace to trace. Also, we define '$\mathcal{D}$' as the set of all the known inputs and data-dependent variables that can be calculated using the information known to Eve.

Our algorithm depends on increasing the number of known sensitive variables (the size of $\mathcal{D}$) by mounting DPA against the unknown variables in a sequential way. The algorithm works as follows:

1. Add all the message bytes to the set $\mathcal{D}$.
2. Calculate all the data-dependent variables that depend on the available information.
3. Add these new calculated variables to the set $\mathcal{D}$.
4. Select the operations that process an element of $\mathcal{D}$ and a *constant* unknown.
5. Target the output of these operations with DPA to recover the unknown.
6. If the recovered unknown is enough to recover the required key, finish.
   Else,
   - Repeat from Step 2 using the information that became available from the just recovered unknown.

MAC-Keccak can be viewed as a Directed Acyclic Graph, where vertices (V) represent the internal operations and edges (E) represent inputs and sensitive variables. The vertices are numbered similar to the order of executing the operations within the algorithm. The pseudo-code of the systematic approach is shown as follows, where the output is the correct value of all the edges, including the required secret key.

We used the following data objects:

- `Flag[e]`: `C` for constant edges, `D` for unknown data-dependent edges, `SetD` for known data-dependent edges.
- `Init[e]`: The initial values of `Flag[e]`; `C` for key-bytes, `SetD` for message bytes, `D` for all the rest.
- `Required[v]`: The set of operations that need to be targeted to recover the full key.
- `e.Eval()`: A function to evaluate the value of edge `e`.
- `v.DPA()`: A function to apply DPA on the output of vertex `v`.

The pseudo-code is shown in Algorithm 2 and consists of three phases. The initialization phase flags the initial values of the edges. The exploration phase flags the constant edges with `C` and marks the set of target internal operations. The attack phase performs a greedy search for new target operations and mounts DPA attacks against them.

4.5.1 Application to MAC-Keccak Examples

We study some examples of MAC-Keccak as shown in Figure 4.9. We focus on one column of the state and study the propagation through θ_1, θ_2 and χ. The effect of ρ and π is neglected in the figure for clarity.

The first example assumes that there is one key-bit and four message-bits in every column (see Figure 4.9, left). The $\mathcal{D}$ set will be initialized with all the input message-bits. In this case, we will select and mount DPA against the first θ_1 operation, which involves one unknown variable (the secret key), and one element of $\mathcal{D}$ (the message-bits). The complete secret key should be recovered after this DPA iteration.

The second example assumes that there are two key-bits in every column (see Figure 4.9, middle). Similarly, the $\mathcal{D}$ will be initialized with all the input message-bits. However, the

Algorithm 2 Systematic Analysis for DPA of MAC-Keccak

```
Require: Graph G(V,E)
Require: Init[E]                                          ▷ Initialization Phase: step 1
  Flag[E] = Init[E];                                      ▷ Exploration Phase
  for each vertex v in V do
     if all Flag[v.input] == C then
        Flag[v.output] = C;
     end if
     if any Flag[v.input] == C & (any Flag[v.input] == D or SetD)  then
        Required[v] = 1;
     else
        Required[v] = 0;
     end if
  end for                                                 ▷ Attack Phase:
  while any Required == 1 do
     for each vertex v in V do
        if all Flag[v.input] == SetD then
           v.output.Eval();                               ▷ step 2
           Flag[v.output] = SetD;                         ▷ step 3
        end if
     end for
     for each vertex v in V do
        if any Flag[v.input] == C & any Flag[v.input] == SetD) then   ▷ step 4
           v.DPA();                                       ▷ step 5
           Flag[v.input] == SetD;
           Required[v] = 0;                               ▷ step 6
        end if
     end for
  end whilereturn Value of all E
```

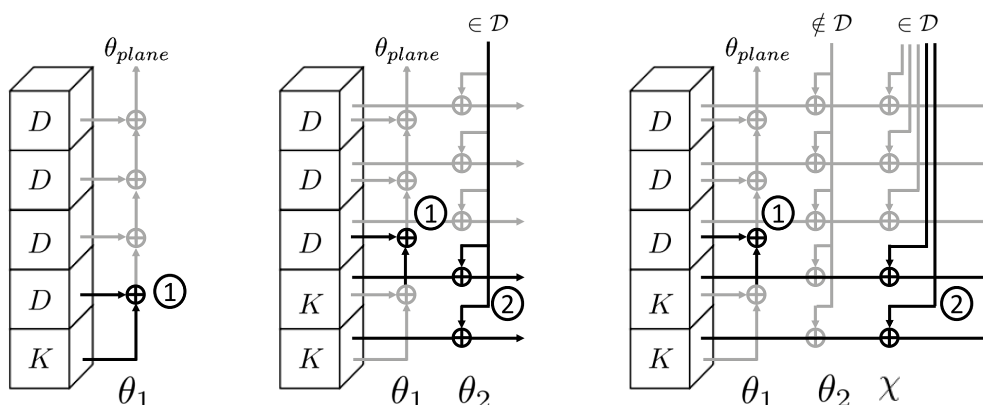

Figure 4.9: DPA applied to some examples of MAC-Keccak.

second θ_1 operation will be selected in this case, as the first operation has two unknown inputs. By mounting DPA against the selected operation, the involved unknown variable should be recovered, which is the output of xoring the two key-bits. Unfortunately, this recovered unknown is not enough to uncover the original secret key, nor it can be used to forge a MAC digest because each key-bit will have its individual effect in later operations. In this case, we will have to go for another DPA iteration. We will use the information available from the just recovered unknown to calculate the θ_{plane}, and add it to $\mathcal{D}$. In this iteration, we will select the first two operations of θ_2, for having unknown variables (secret key-bits) and elements of $\mathcal{D}$ (θ_{plane}-bits) at their inputs. By mounting DPA against the selected operations, the complete secret key should be recovered. Note that, we cannot skip the θ_1 operation and directly attack θ_2, as every θ_2 operation involves two columns of the state, which greatly increases the search space.

The third example is yet more complicated (see Figure 4.9, right). Similar to the previous example, we assume that there are two key-bits in every column. However, we assume that the θ_{plane}-bits required in the second DPA iteration are also unknown. Although this scenario is only possible if the key and message bits are interleaved in the input block, it shows that certain key-configurations may require targeting operations that are deep in the algorithm. In this case, the first DPA iteration will be similar to the previous example, and the θ_{plane}

should be partially recovered. The partially recovered θ_{plane} will be added to $\mathcal{D}$. We will trace the Keccak-function passing through ρ and π, where the location of bits within the state will be mixed. Finally, the χ operation will be selected, targeted and the unknown should be recovered. The recovered unknown can be used to trace-back the Keccak-function to calculate the original key.

The analysis of these MAC-Keccak examples shows several interesting findings:

- A cryptographic algorithm can be designed in such a way that builds a hierarchical dependency structure between different key-bytes. Attacking one key-byte depends on the successful recovery of another key-byte.
- The order of attacking the internal operations should respect the consecutive dependency between different key-bytes.
- The probability of achieving a successful attack is affected by the number of dependent DPA iterations.
- The order of attacking internal operations and the number of DPA iterations in the attack depends on the key-length and the location of the key-bytes (key configurations) within the input block.
- Key configurations can be selected to maximize the effort required to mount a successful attack, which will be the focus of our future work.

4.6 Case Studies

The selected target operations depend heavily on the location of key-bits within the state. Hence, it becomes important to visualize the relative locations of key-bits and message-bits within the state. We assume a state-size of $b = 1600$ bit, arranged in a $5 * 5 * 64$ array, and an input block size of $r = 1088$ bit. As the state fills bottom-up, the new input block will

first fill the lower three planes ($\mathbb{P}([0:2])$ in Figure 4.2), followed by the first two lanes of the fourth plane ($\mathbb{L}([0:1],3)$). We assume a key-length less than the Rate; i.e. the first input block will contain key-bits prepended on the message-bits. The key-bits will be in the lower planes of the state while the message-bits will be in the middle planes. Since this is the first hash block of the chain, the upper planes will contain zeros.

While there is no current standard for the key-length of MAC-Keccak, we present detailed analysis of three cases with different key-lengths; starting from 768 bits and adding 128 bits (or two Keccak lanes) in each case. The results of attacking these cases are highlighted in the following section. The results of two other cases, namely (Key-length = 832 and 960), are also presented without detailed analysis as they follow the same attack methodology.

The reasoning behind our choice of long key-lengths is that it gives the MAC-Keccak implementation higher resistance against DPA. Typically, the attack complexity increases linearly with the key-length where a separate attack is required for every key-byte. However, the complexity of attacking MAC-Keccak increases faster than linear with the key-length due to the consecutive dependency between different key-bytes. This behavior is validated by the results presented in the following section. Also, long key-lengths are not uncommon in the cryptographic community. They have been practically used in the applications of RSA [4].

The analysis of shorter key-lengths (128, 256 and up to 320 bits) is trivial and can be solved by only one DPA iteration (the easiest case in the previous section). The studied key-lengths are chosen to highlight the dependency of attack complexity on key-length, where every case requires a new DPA iteration. The proportional reduction of the amount of message-bits in each case will make the DPA increasingly harder. We considered cases where the key fills complete lanes, hence the Z index is always $Z = [0:63]$ however, the analysis can be applied to any other key-length.

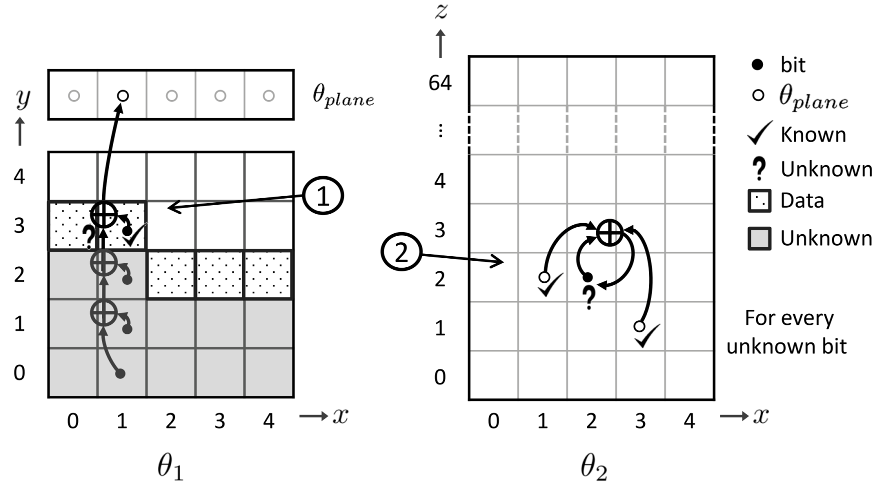

Figure 4.10: The DPA of Key-length = 768 bits.

4.6.1 Key-length = 768 bits

The key-length is chosen so that the input message-length = 320 bits, the size of one plane of the state. There will be a single message bit in every column. Figure 4.10 shows the position of the key-bits, message-bits and zeros using shaded, dotted and white squares respectively.

This case study is similar to the second example discussed in the previous section (Figure 4.9, middle). The attack involves the following steps:

- Add all the input message-bits to the set $\mathcal{D}$.
- Select and target the output of θ_1 xor operations between the key-bits and message-bits: $\theta_1([0:1], 2, Z)$ and $\theta_1([2:4], 1, Z)$.
- Recover the parity of the key-bits in every column.
- Calculate the θ_{plane}, and add it to $\mathcal{D}$.
- Select and target the output of θ_2 xor operations between every key-bit and the corresponding θ_{plane}-bits: $\theta_2([0:1], [0:2], Z)$ and $\theta_2([2:4], [0:1], Z)$.

- This should recover the required key-bits.

This attack required two DPA iterations.

4.6.2 Key-length = 896 bits

The key-length is chosen such that the input message-length = 192 bits. Here, both sheets $\mathbb{S}(2)$ and $\mathbb{S}(3)$ have no message-bits. This case study is more difficult than the previous one. The all-unknown sheets will lead to all unknown lanes in the θ_{plane}. These unknown lanes of θ_{plane} will require more effort in the attack.

Our attack will follow the following steps:

- Add all the input message-bits to the set $\mathcal{D}$.
- Select and target the output of θ_1 xor operations between key-bits and message-bits in sheets $\mathbb{S}([0,1,4])$: $\theta_1([0:1],2,Z)$ and $\theta_1(4,1,Z)$.
- Recover the parity of the key-bits in each column of those sheets.
- Calculate the partially recovered θ_{plane}, and add it to $\mathcal{D}$.
- Select and target the output of θ_2 xor operations of the message-bits of the neighboring sheets $\mathbb{S}([1,4])$: $\theta_2(1,3,Z)$ and $\theta_2(4,2,Z)$.
- Recover the two missing lanes of θ_{plane} ($\mathbb{L}([2,3],0)$).
- Add the recovered lanes to $\mathcal{D}$.
- Select and target the output of θ_2 xor operations between every key-bit and the corresponding θ_{plane}-bits: $\theta_2([0:3],[0:2],Z)$ and $\theta_2(4,[0:1],Z)$.
- This should recover the required key-bits.

The attack in this case required three DPA iterations.

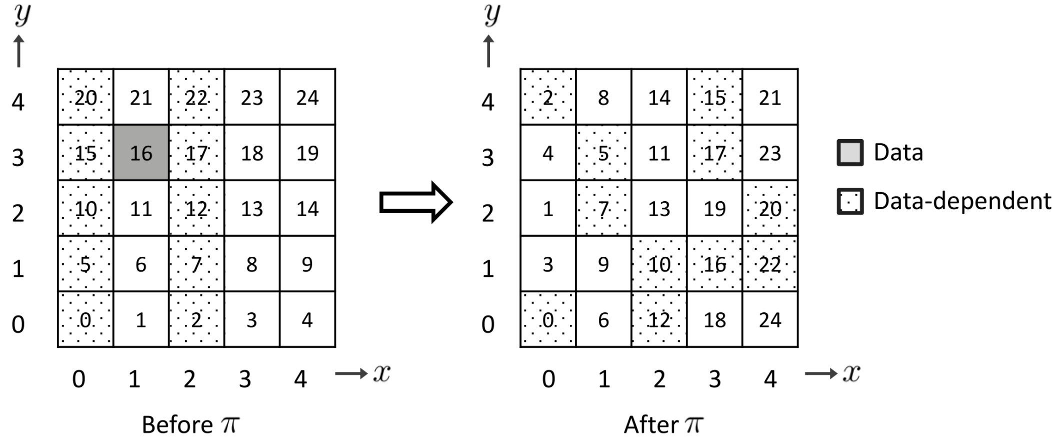

Figure 4.11: Data-dependent bits of key-length=1024 bits, before and after π step.

4.6.3 Key-length = 1024 bits

The length of the input message is 64 bits, only one lane ($\mathbb{L}(1,3)$) with 4 all-unknown sheets $\mathbb{S}([0,2,3,4])$. The attack in this case will logically be in these steps: recover the Keccak state after θ step, apply the mapping of ρ and π steps as shown in Figure 4.11, and recover the key using the χ step.

The exact attack will be as follows:

- Add the input message-bits to the set $\mathcal{D}$.
- Select and target the output of θ_1 xor operations between key-bits and message-bits in sheet $\mathbb{S}(1)$: $\theta_1(1,2,Z)$.
- Recover the parity of the key-bits in each column of that sheet.
- Calculate the partially recovered lane of θ_{plane}, and add it to $\mathcal{D}$.
- Select and target the output of θ_2 xor operations of the message-bits of that sheet $\mathbb{S}(1)$: $\theta_2(1,3,Z)$.
- Recover the data-dependent variables of lane $\mathbb{L}(1,3)$of the state after θ.

- Also in the same DPA iteration, select and target the output of θ_2 xor operations of all the key-bits of sheets $\mathbb{S}([0,2])$: $\theta_2([0,2],[0:4],Z)$.
- Recover the data-dependent variables of sheets $\mathbb{S}([0,2])$of the state after θ.
- Add the recovered data-dependent variables to $\mathcal{D}$.
- Select and target the output of χ operations between an unknown variable and two elements of $\mathcal{D}$: $\chi(0,0,Z)$, $\chi(1,1,Z)$, $\chi(3,1,Z)$, $\chi(4,2,Z)$, $\chi(1,3,Z)$, $\chi(3,4,Z)$.
- Recover the targeted unknown variables, and add them to $\mathcal{D}$.
- Select and target the output of χ operations between an unknown variable, one elements of $\mathcal{D}$, and one just recovered unknown: $\chi(1,0,Z)$, $\chi(4,0,Z)$, $\chi(0,2,Z)$, $\chi(3,2,Z)$, $\chi(0,3,Z)$, $\chi(2,3,Z)$, $\chi(2,4,Z)$, $\chi(4,4,Z)$
- This should recover all the rest of unknown variables.

The recovered state can be used to trace-back the Keccak-function to retrieve the original key, or it can be used directly to forge a MAC digest by inserting the recovered state directly in the χ step. The attack in this case required four DPA iterations.

4.7 Practical Results

The experimental evaluation of our analysis is conducted on the reference software code of Keccak [1] running on a 50MHz 32-bit Microblaze processor built on top of a Xilinx Spartan-3E FPGA. We use a Tektronix MIDO4104-3 oscilloscope with a CT-2 current probe to capture the instantaneous current of the FPGA core as an indication of the power consumption. To reduce the measurement noise, the processor is programmed to execute the same hashing operation 16 times, while the oscilloscope calculates the average of them. Figure 4.12 shows one recorded trace indicating all the steps of Keccak-function.

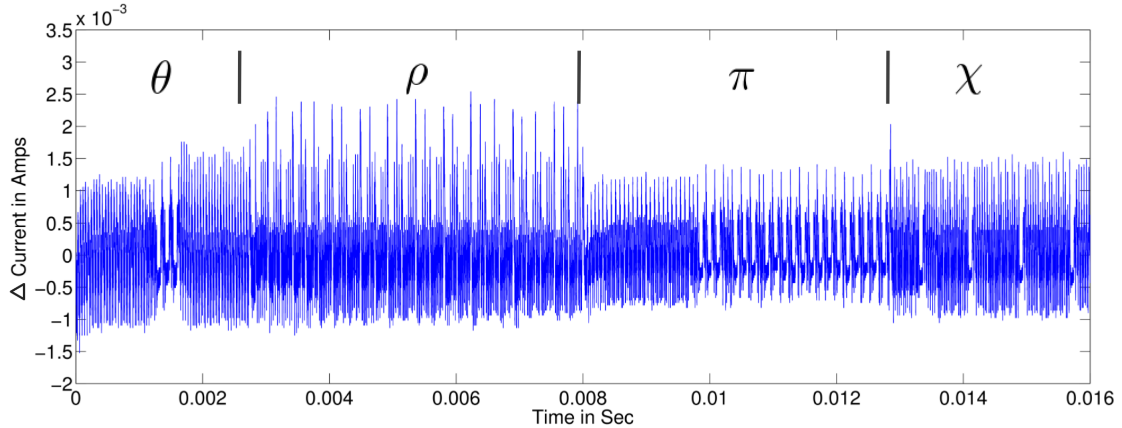

Figure 4.12: A power trace of MAC-Keccak.

The attack is conducted using the Correlation Power Analysis [24] with Hamming Weight power model. The power model is built using an 8-bit key guess at a time. This choice is motivated to reduce the algorithmic noise (because it is a 32-bit processor, where the processing of 24 bits will be considered noise) at a practical search space (256 different key guesses).

SCA of xor operation: Before conducting a complete attack, we first start with analyzing the side-channel leakage of one xor operation. We apply the CPA [24] with the Hamming Weight power model to the last xor operation of generating the θ_{plane} of MAC-Keccak. Figure 4.13 shows the result of this analysis. The red plot is for the correct key guess, the black plot is for the complement of the correct key and the green plots are for all other key guesses.

The figure clearly shows the effect of attacking simple xor binary operations. If the key is miss-guessed by one bit, the result power model will differ for only one bit. This is the reason for the high correlation in many of the green plots. In order to get over the high correlation of incorrect key guesses, Eve needs to collect increased number of power traces. The correct key guess shows up clearly after around 10,000 traces. Moreover, if the key is guessed to the complement of the correct key, the result will show a strong negative correlation. This

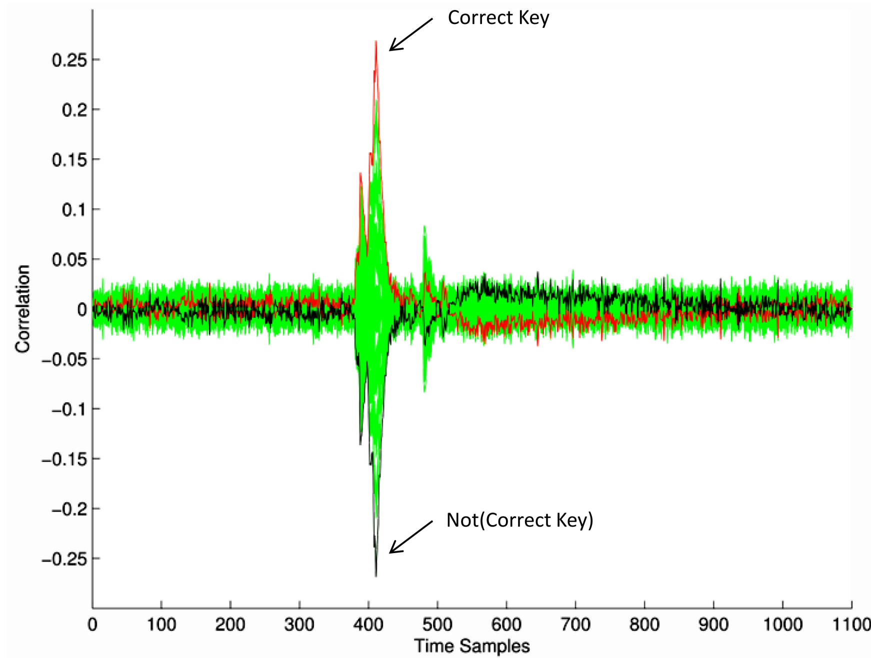

Figure 4.13: SCA of an xor operation.

means that, Eve needs to consider only the positive correlation.

SCA of generating the θ_{plane}: Another interesting observation found in the analysis of generating the θ_{plane} is that the peak positive correlation is always found in triple spikes, as shown in Figure 4.14. The reason of this can be explained with the help of the xor operation on the right hand side of the figure. We first assume that the key-length equals to the rate that the first three planes are filled with the message and the upper two planes are the previous key-state. The attack is meant to recover the first unknown (Ks1). The correct key-state bit at the output of the forth xor is the required unknown (Ks1) which shows up as the middle spike. Keeping in mind that, the power traces shows all the operations sequentially. The third xor operation shows up at the left spike with high correlation at a false key '0'. Similarly, the fifth xor operation shows up at the right spike with high correlation at a false key Ks1 $\oplus$ Ks2. This observation is critical for a correct attack. The three results represent actual operations running within the module that, the three correlation spikes should be of

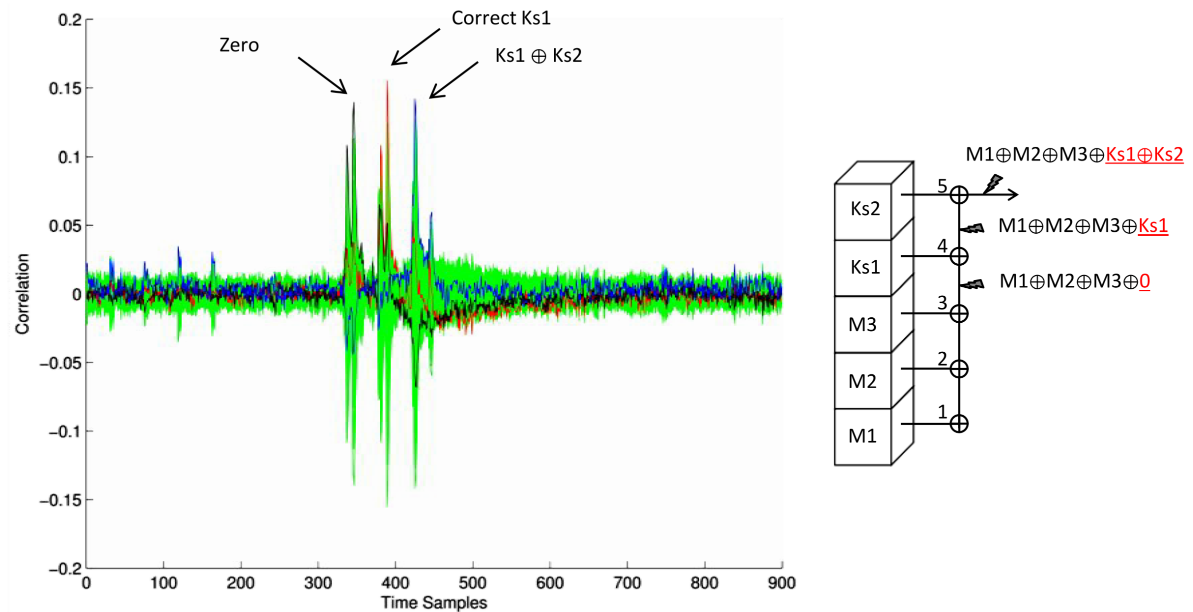

Figure 4.14: SCA of generating the θ_{plane} operation.

the same value. If the CPA is applied blindly without time profiling, the best guess will be uniformly distributed between '0', the correct key-state bit, and the xor of the rest of key-state bits. Based on this observation, we use a precise time profiling to target only the intended operation.

Results The results of attacking MAC-Keccak with key-lengths = (768, 896 and 1024), as discussed in the previous section, are shown in Figure 4.15. The figure also shows the results of attacking two other cases; key-lengths = (832 and 960), that were analyzed using the same approach. The figure shows the success rate of each case study as a function of the number of traces used in the analysis, where the success rate is the percentage of key-bytes that have been recovered successfully.

The consecutive dependency between different key-bytes in MAC-Keccak affects the success rate of the attack as follows. Assuming that a single DPA is successful with probability p. The key-bytes recovered in the first DPA iteration will have probability of success p. Key-bytes of the second DPA iteration will be successful only if all the involved key-bytes

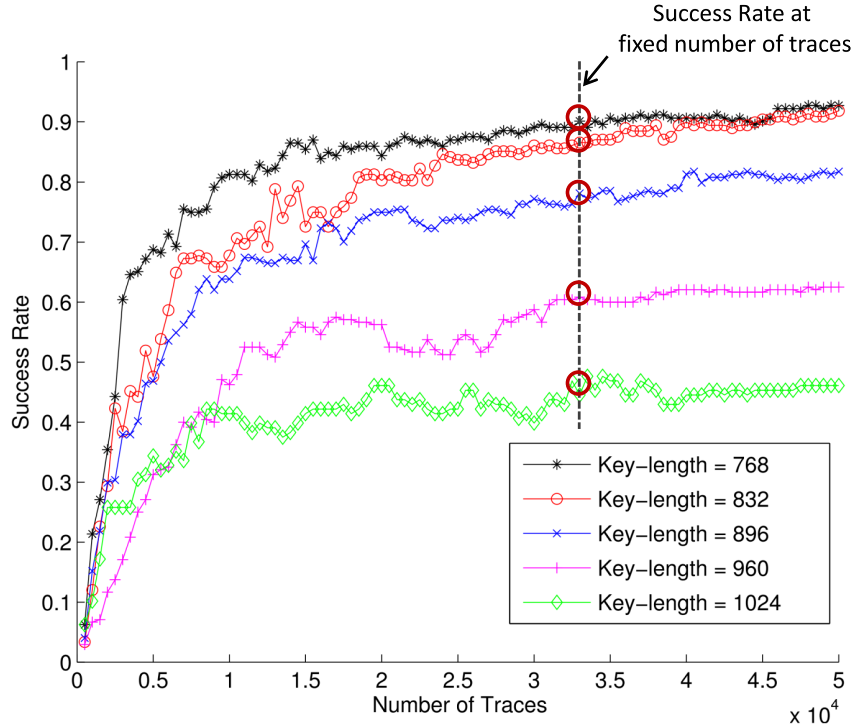

Figure 4.15: Success rate of different case studies.

of the previous iteration were recovered successfully and the second iterations itself was successful. Assuming that the number of previously recovered and involved key-bytes is n, the probability of success in the second DPA iteration will be $p^{(n+1)}$. Similarly, the probability of success of the third and fourth DPA iterations will drop very quickly depending on the number of previous, involved key-bytes. As a result, the complexity of a complete DPA attack on MAC-Keccak increases faster than linear by increasing the levels of consecutive dependency, which is reflected by the number of DPA iterations required. This behavior can be seen in the results by comparing the success rate of the studied cases at a fixed number of traces, as shown by the vertical line in the figure.

There is another remark in the figure, where the success rate of each case builds up very quickly in the first 10,000 traces, then in a slower way through the remaining traces. The reason of this behavior is that key-bytes are recovered with different probability of success within the same case. The key-bytes recovered in the first DPA iteration will build the

success rate quickly, while those recovered in later iterations will build the success rate in a slower and flatter way.

4.8 Summary

In this chapter, we presented a comprehensive analysis of the SCA properties of the Keccak hashing algorithm. We studied the effect of changing the key-length. We also demonstrated the challenge of selecting the proper target operation. We used a systematic approach to increase the number of known sensitive variables by attacking the unknown variables in a hierarchical way. We studied in full details several, practically-difficult, case studies of MAC-Keccak. The analysis was validated by a practical attack against the reference software code running on a 32-bit Microblaze processor.

Part II

Side-Channel Countermeasures

Chapter 5

Background on Side-Channel Countermeasures

The threat considered in the next chapters is that Eve recovers the secret key of the hardware implementation of a cryptographic module. Classical cryptography assumes that Eve can choose the input plaintext and the output ciphertext. SCA further assumes that Eve knows the underlying implementation and can capture the instantaneous power consumption.

SCA attacks are commonly based on three pillars, as shown in Fig. 5.1:

1. Sensitive variables affect leakage traces.
2. Eve can calculate hypothetical sensitive variables.
3. Eve is able to combine information from different traces.

The design of SCA-countermeasures is a vast research field. Contributions, in this regard, fall into two categories: practical countermeasures and theoretical ones.

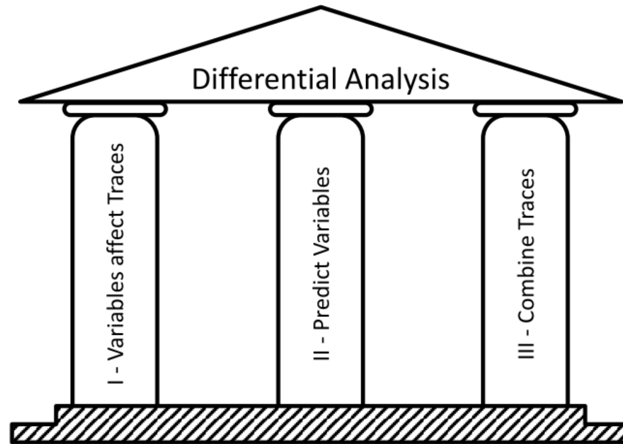

Figure 5.1: Pillars of SCA attacks.

5.1 Practical Countermeasures

Practical countermeasures focus on fixing the underlying implementation. They target breaking the first and second pillars of SCA through the use of hiding and masking respectively.

5.1.1 Hiding

Hiding depends on muting variations in power traces, hence breaking the link between sensitive variables and leakage traces (the first pillar). This is typically achieved by reducing the signal-to-noise ratio within the trace. Useful information is reduced using balanced circuits, while noise is increased using random noise generators.

Balanced circuits are redundant logic that process active data and its complementary at the same time, hence the system power consumption is kept almost constant. Balanced circuits can be realized in all design hierarchies, from low level logic [94] to custom instructions [63], and full high end processors [28].

Random noise generators can be used to add random fluctuations to the power signal [46]. It can also be used to add random delays to the clock signal, hence prevents the correct

alignment between traces [62].

Practically, achieving the perfect hiding is challenging due to engineering defects, where any difference between the complementary logic blocks (even the length of data paths) will lead to a useful leakage. However, once a perfect hiding is achieved, any cryptographic protocol can be implemented without useful SCA leakage. Remarkably, hiding is the only algorithm-independent countermeasure.

Unfortunately, hiding does not prevent an attack, but only increases the required number of traces (typically in orders of magnitude). Also, hiding has a very limited scope of protection against power (and sometimes electromagnetic) attacks, i.e. they cannot generally protect against fault or timing attacks. Finally, Cryptographic modules protected with hiding require more than double the area (e.g. [93]).

5.1.2 Masking

Masking depends on breaking Eve's ability to calculate hypothetical sensitive variables (the second pillar), by splitting the useful information into n shares based on random variable(s). Using n shares is called $(n-1)$ order masking. The random variables are generated on-the-fly and discarded afterwards. Each share is processed independently. Masking is designed so that, combining the shares at any point throughout the algorithm retrieves the original value. Hence, the final outputs (of each share) can be combined to retrieve the original output. A perfect masking scheme is achieved if every sensitive variable throughout the implementation is statistically independent of the secret key [22].

Although, Eve can no longer have a meaningful hypothesis against any share, she can still mount a successful attack by combining the information from targeting all the shares. Attacking n points in the trace at a time is called n order DPA. Generally $(n-1)$ order masking can be targeted by n order DPA.

Splitting information through linear operations is straightforward. However, splitting

information through non-linear operations is tricky and error prone, especially for high order masking. For this reason, an automatic tool was developed to measure the masking strength of a given implementation [36].

Masking is better than hiding in two domains. First, it can be implemented by software on commodity processors (at an overhead of 2x execution time). Also, it has a wider protection scope. It can prevent any leakage through the masked variables (power and electromagnetic), but any other source of leakage (e.g. timing) is not protected. However, masking is an algorithm-dependent countermeasure, i.e. new cryptographic algorithms need new masking schemes.

Unfortunately, masking schemes are limited by implementation effects (e.g. glitches [65]), or by design simplifications that make them vulnerable to new attacks (e.g. [72]). For example, the masked implementation of [79] has been broken in 22 traces [84]. Also, hardware modules supported with masking require more than double the area (e.g. [77]).

Practical countermeasures share a common advantage that, the algorithm output does not depend on the protection technique. Hence, if one side of the communicating parties is physically secured (e.g. server), it can use any unprotected implementation while obtaining the same output.

5.2 Theoretical Countermeasures

Theoretical countermeasures known as leakage resilient cryptography, focus on designing new primitives with inherent resilience against any information leakage. The argument is: if leakage-resiliency was considered as a required property in the design time, SCA can be prevented with provable solutions. The main target of this approach is to prevent Eve from combining information from different traces (the third pillar). This can be achieved by utilizing a key-updating mechanism (aka re-keying or key-rolling). The efforts of this group resulted in many new primitives including: pseudorandom generators [96], stream-

ciphers [34], block-ciphers [6] and many more.

The pros of leakage resilient cryptography are numerous. It completely prevents SCA attacks with provable solutions against any leakage (including fault and timing). Also, the protection extends to any differential attack, that requires multiple executions at the same key e.g. differential cryptanalysis. Unfortunately, the cons are also significant. First of all, it develops new primitives, leaving the current primitives and standards with problems of immediate need without solutions. Although leakage resilient primitives can be implemented using unprotected cores (i.e. no area overhead), the overall performance is at least halved (e.g. [90]).

More details about leakage resilient constructions will be discussed in the next chapter, where we introduce lightweight key-updating mechanisms.

5.3 Do We Need Perfect Protection?

It is commonly agreed that, if Eve acquire a cryptographic module and she has adequate resources, she will break the module one way or the other (e.g. using invasive attacks). The whole point of SCA-countermeasures is to exclude SCA from being the weakest point in the chain of security. Indeed, a practical market will not use high cost countermeasures in low end embedded devices, e.g. metro tickets.

For this reason, we design flexible solutions with the ability to trade some SCA security for a better performance.

5.4 Our Design Concept

Our focus is to design countermeasures for hardware cryptographic modules at a small implementation cost (area and performance). The solutions proposed in this dissertation are

based on two concepts. First, we use ideas from leakage resilient cryptography as tools to support the protection of current protocols and standards. Hence, our solutions approach the same goals of leakage resiliency, however we exclusively focus on supporting current standards with heuristically secure key-updating mechanism i.e. we do not propose new constructions.

The second concept is to achieve a design with fine granularity of SCA-protection versus performance. If the implementation cost of our provably secured application is too high for a particular market, a designer will have the ability to trade some SCA-security, with fine granularity, for a higher performance (lighter implementation).

Chapter 6

Efficient Leakage Resiliency

Leakage resiliency depends on thwarting the threat of SCA by changing the secret key after every execution. Indeed, many contributions in the domain of leakage resilient cryptography tried to achieve this goal. However, the proposed solutions had a very high performance overhead. In this work, we follow the same design guidelines of leakage resiliency. However, we focus on designing heuristically secure key-updating mechanisms at the smallest possible implementation cost (area and performance). For this purpose, we highlight a generic framework for key-updating that works with both block-ciphers and hashing functions. Then, we evaluate the minimum requirements for heuristically secure implementations.

6.1 Background and Previous Work

SCA security assume that Eve can acquire the input, the output and the instantaneous power consumption of every execution. In the domain of leakage resiliency, it is also assumed that Eve can run any polynomial-time function (called leakage function) on the power consumption to recover some bits of the secret key.

Leakage resiliency, being a protocol level protection, cannot protect the underlying imple-

mentation against Simple Power Attacks (SPA), where one execution of the leakage function can recover the full secret. Hence, the typical assumption is that the leakage function can recover a small part $\lambda < |K|$ of the secret key. This is a reasonable assumption in hardware modules, where the high parallelism and the measurement noise prevents any polynomial-time function from recovering the full secret. Differential Power Analysis (DPA) is represented by executing the leakage function over different executions (exactly $\lceil |K|/\lambda \rceil$), until the full secret key is revealed.

Leakage resiliency depends on changing the secret key after every execution. The updating function should possess a minimum set of requirements in order to prevent DPA attacks. For example, if the updating mechanism is linear or simple (e.g. a counter), Eve can build her hypothesis based on a key guess that follows the same updating mechanism, removing the effect of key-updating. This attack is called future-computation attack, because it is modeled as if the leakage function can recover some bits of a key that will show up in the future. Future-computation attack represents that main threat addressed by all leakage resilient cryptography.

The rest of this section reviews the two categories of key-updating and the notable contributions in each one. At the end of each subsection, we discuss how our solution improves over the current ones.

The two categories of key-updating are stateless and stateful. One mechanism or the other may be sufficient for a limited set of applications. However, the two mechanisms are both required for a complete and generic solution. For example, Fig. 6.1 shows how the two mechanisms complement each other for the application of data encryption. After exchanging a public nonce, a stateless key-updating is used to generate a pseudorandom secret state. Then, a stateful key-updating is used to generate fresh running keys ($k_1 : k_\infty$).

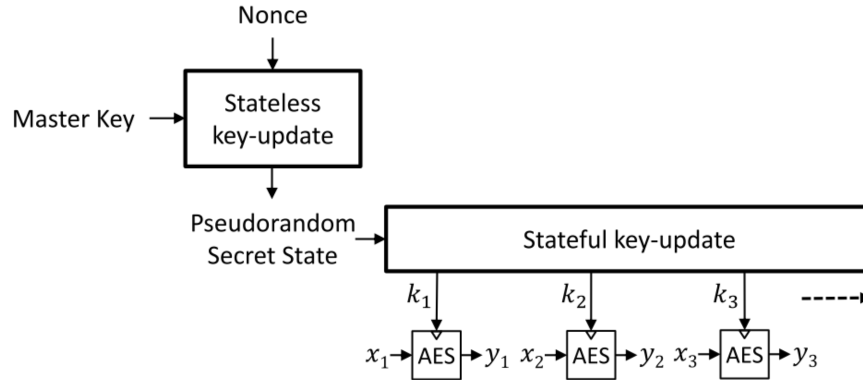

Figure 6.1: Stateless and stateful key-updating, as shown for the example of data encryption.

6.1.1 Stateless Key-Updating

Stateless key-updating assumes that the two communicating parties share only the secret key and a public variable (nonce) i.e. there is no shared secret state between them. This updating mechanism is required whenever there is no synchronization between the two communicating parties e.g. during initialization of a secret channel. Stateless key-updating provides a complete solution for applications with single cryptographic execution e.g. challenge response protocols.

There is no provably secure construction that supports stateless key-updating [90]. Intuitively speaking, the secret key cannot be updated to a new key unless a public variable is used (assuming no synchronization). Once a public variable interacts with a secret key, SCA will be possible. Some contributions tried to secure the stateless key-updating mechanism through hiding and masking [69, 39]. Although this approach limits the implementation overhead exclusively to the key-updating mechanism, allowing the use of unprotected cryptographic cores, the overall overhead is still significant (more than 100% [69]). Also, hiding and masking do not prevent an attack, but only make it harder (more traces to overcome hiding, and higher-order attacks to overcome masking).

On the other hand, leakage resiliency can be used to minimize the number of instances where a secret key is being used. This can be achieved using the tree structure (as proposed

by Goldreich, Goldwasser and Micali, known as GGM structure [44]), where the secret key is updated to a new secret through a series of sequential steps. Each step involves a sound source of randomization e.g. block cipher encryption with random plaintexts. The random plaintexts are used to prevent future-computation attacks and are selected based on one bit of the public variable (the nonce). The tree structure is proven secure against SCA attacks iff every step is SCA-secure [90]. Tree structures have the common drawback of high performance overhead. Typically, they require $O(|n|)$ executions of the underlying block cipher, where $|n|$ is the bit-length of the nonce. The performance overhead of the tree was reduced by involving more random variables per step (more bits of the nonce per step), while protecting each step using key-dependent algorithmic noise [70]. This solution is elegant as it demonstrates the first heuristically secure stateless key-updating. However, they required all the S-boxes of AES to leak identically, which is not guaranteed on most modules.

In this regard, we extend the work of [70] in two directions. First, we achieve sound protection for hardware implementations where the S-boxes leak differently, while reducing the performance overhead. Also, we propose solution for a different cryptographic primitive, the secure hashing function SHA-3.

6.1.2 Stateful Key-Updating

Stateful key-updating assumes that the two communicating parties share a common secret state (other than the key). They both can update the secret key into a new key without requiring any external variables. This scheme can provide a complete solution for synchronized applications e.g. key-fobs.

The first provably secure construction for stateful key-updating was the alternating structure [34, 85]. In this structure, two different keys are used in an alternating fashion. Hence, the computation of a future key depends not only on the current execution (where only one key is used) but also on another value that is not currently processed within the system. Unfortunately, this structure is inefficient, as it requires doubling the key size. Also, it assumes

that Eve cannot combine the leakage from the two computing parts, which is not a realistic assumption. Then, a direct structure was proposed replacing the alternating structure by using a fresh random variable at every key-update [37], which is not practical. Later, an efficient direct structure was proposed using only one random variable under the assumption that the leakage function is non-adaptive i.e. the leakage function is fixed and selected prior to or independent of the random variable. Actually, the leakage function is mostly determined by the underlying hardware, which makes such assumption a logical one [95].

Also, many contributions proposed heuristically secure stateful key-updating [39, 57, 95]. In these contributions, a full-features hashing function is used to update the secret key.

In this regard, we follow the work of [39] with two improvements. First, we enhance the security analysis by adapting a realistic assumption that only current and previous executions leak. Under this assumption, nonlinear key-updating functions with high diffusion can prevent future-computation attacks (as detailed later). Also, we manage to reduce the performance overhead. Our solution can be proven secure following the guidelines of [95]. The main difference is that we initiate the structure using a pseudorandom variable instead of using an external random input.

Also, we show that SCA-protection of the secure hashing function SHA-3 does not require any stateful key-updating, hence the performance overhead of our solution is minimal.

Although one key-updating mechanism or the other is enough for a specific application (as discussed), the two mechanisms are required for a complete and generic solution e.g. data encryption for the internet of things.

So far, future-computation attacks were prevented by computing the new secret key using some information that is not currently present in the system. Typically, this is achieved by using the alternating structure or by utilizing a fresh random at every step. In this paper, we propose a new method to achieve the same goal using a new requirement on the key-updating function.

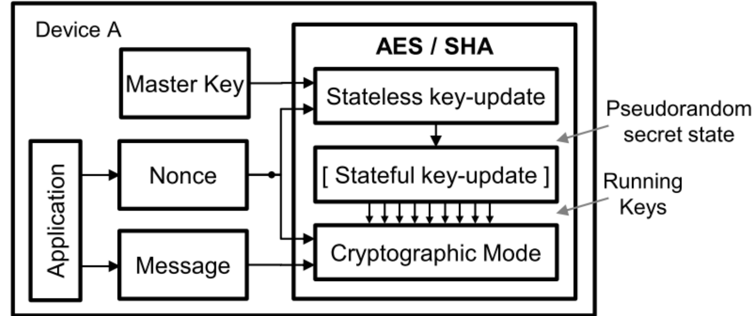

Figure 6.2: The system overview.

6.2 Framework for Lightweight Key-Updating

The proposed solution at the system level is shown in Fig. 6.2. We assume that an application on Device A needs to send secure data to an application on Device B (not shown). Both devices share a secret key, which we name master key. They can initiate the channel by exchanging a public nonce, and send the secure data using any cryptographic primitive (AES or SHA-3) running in a mode of operation. Although the black-box security of these modes is guaranteed by the cryptographic primitive, security is not guaranteed if Eve can monitor Device A. Here, we target protecting the master key against any SCA attack.

Device A starts with a stateless key-updating mechanism to compute a pseudorandom secret state out of the master key and the nonce. Then, the stateful key-updating is executed (only for block-cipher modes), to compute running keys. Finally, the actual cryptographic mode is called using the secret data and the same previously used nonce.

Our solution honors the tree structure for the stateless key-updating, as shown in Fig. 6.3. Each step involves processing a key and a single bit of the nonce through a lightweight whitening function (Wt: whitening in the tree). The tree starts from the master key, and ends with a pseudorandom secret state. If the secret state is (partially) observable by Eve (for the applications of SHA-3), an extra hashing step will be required.

For the stateful key-updating (only for block-cipher modes), we use a simple chain of

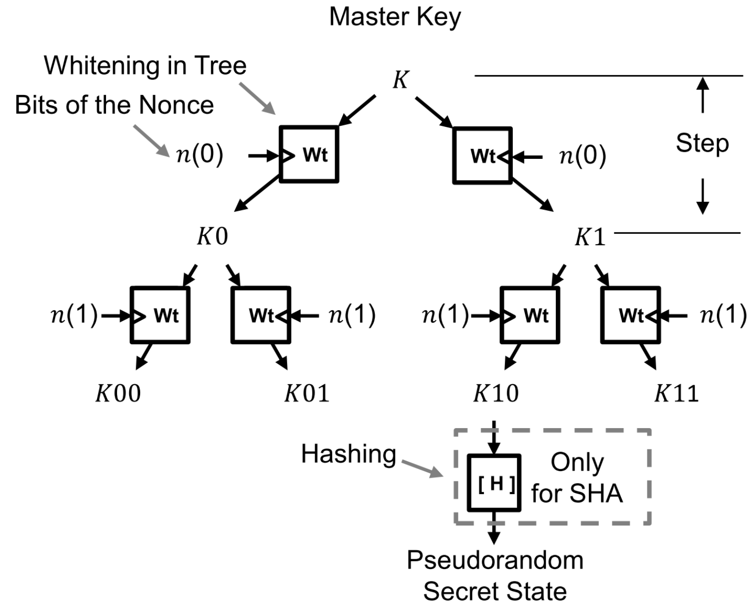

Figure 6.3: Stateless key-updating using a tree structure.

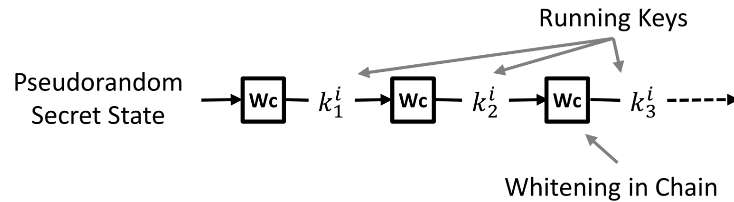

Figure 6.4: Stateful key-updating using a chain of whitening functions.

whitening functions (Wc: whitening in the chain) as shown in Fig. 6.4. Every execution of the whitening function generates a new running key.

The definition of Wt and Wc functions will be introduced in the next chapters, along with their security analysis and the implementation details.

6.2.1 Assumptions

During the design of the proposed solution, we follow these assumptions:

Parallel Hardware We assume that all the non-linear operations (S-boxes for AES, and χ for SHA-3) are processed in parallel. Hence, the system power consumption is the aggregation of all the leakages. This assumption is required to exploit the key-dependent algorithmic noise, which supports SCA security in the stateless key-updating.

Also, key-updating mechanisms are designed to be secure against simple and differential attacks. They can protect the underlying primitive (AES of SHA-3) against differential (power / electromagnetic / etc..) attacks. However, they cannot protect the underlying primitive against simple attack, where a single trace is used to recover the secret key. Our assumption of parallel hardware also prevents simple power analysis against the underlying primitive.

Only Current and Previous Iterations Leak This is a very logical assumption, as the power leakage is a physical quantity. The module as a physical entity does not have any clue about the next input message block. It is only Eve who can link future computations to the current leakage using the algorithm and the future inputs. Although a similar assumption was used in [90] (Only Current Iterations Leaks), we include leakage of previous iterations. Indeed, the use of Hamming Distance leakage function may reveal some information about the previously processed iteration. This assumption does not exclude future computation attack, but only breaks the direct (and mysterious) link between future computations and current leakage. Future computations and current leakage are still linked because they both depend on the current state and future inputs (if any). Under this assumption, power leakage will be used to recover key-bits of the current iteration, then the algorithm will be used to link the recovered key-bits to a future key.

6.2.2 Key Requirements

For the highlighted tree structure to be lightweight and secure against SCA, Wt function is required to be:

1. Non-Linear with High Diffusion: Each bit of a new key must depend on every bit of a previous key with a non-linear relationship. As discussed earlier, if the key-updating function is linear or simple (with low-diffusion), Eve can make a guess over a small part of the key and tracks its effect across some executions (divide-and-conquer principle). However, if the key-updating functions is non-linear with high diffusion, it is not useful for Eve to make a guess over a small part of the key, and it is not feasible to make a guess over the entire secret key (similar to brute force).

2. SPA-resistant: At the start of every session, the first execution of Wt will always process the master key. Hence, Wt should be protected against simple power analysis (SPA) attacks.

3. DPA-resistant against two differential traces: Wt will process the master key under two different fixed inputs. Hence, protection against DPA attacks over two differential traces is required.

4. At small area and performance overheads.

If these requirements are met, the tree structure will guarantee that:

- Each nonce will generate unique secret state. If every bit of a new key depends on the input nonce-bit, different value of the nonce (by definition) will result in different final outputs.

- SCA attack is prevented. If each step is protected against SPA attacks, the entire structure will be protected by induction.

Moreover, cryptographic properties of the underlying mode (block cipher or hashing function) will guarantee the one-wayness property. Hence, Eve cannot recover the master key using any observable output.

The Wc function should possess smaller set of requirements, namely: non-linearity with high diffusion at small area and performance overheads. SPA and DPA attacks against Wc

are not possible because the initial seed is pseudorandom and there is no further inputs. Hence, there is no basis for Eve to build her hypothesis.

6.2.3 Discussions

A Lightweight tree, not a GGM

The GGM structure (the original idea for the tree) is a method to realize secure pseudorandom functions (PRFs) from sequential steps of randomization e.g. block-cipher encryptions using plaintexts of random values. Hence, the final output of GGM is required to be pseudorandom. Most leakage resilient stateless key-updating used the GGM to achieve protection against both back-box attacks and side-channel attacks, where the final output is observable by Eve and used as a key-stream [90, 70]. However, we use a lightweight realization of the tree to achieve protection against only side-channel attacks, where the final output is still protected with a sound block-cipher or hashing function. The black-box security of our solution is maintained by the underlying primitive.

This domain change allowed two modifications:

1. In our solution, the decision bit ($n(i)$) selects between two fixed inputs (all 0's or all 1's) instead of selecting between two random variables. In this way, we lost the source of randomization. But, we kept the high-diffusion and the non-linearity, which are the main ingredients of SCA protection. Here, protection against SCA attacks is actually improved by allowing only two differential traces (at $n(i) = 0$ and $n(i) = 1$).

2. The whitening function is not required to exhibit strong black-box security but rather to only prevent future computation attacks. Following assumption No. 2, we achieve this by requiring that any bit of a new key to depend on every bit of a previous key through a non-linear operation.

For these modification, we called our structure a tree rather than a GGM.

Protocol Level Protection

Our solution represents protocol level protections against SCA, where the final output depends on the key-updating mechanism. Hence, the two communicating parties have to follow the same key-updating mechanism, even if one of them is physically secured (e.g. server). This is not the case for hiding or masking, where the final output is not affected by the protection mechanism.

Chapter 7

Solutions for AES Modes of Operation

In this chapter, we propose two heuristically secure key-updating mechanisms for AES modes of operation. One uses a dedicated circuit for key-updating, while the other uses the underlying AES block cipher itself. The first one requires small area (for the additional circuit) but achieves negligible performance overhead. The second one has no area overhead, but requires small performance overhead. A system designer should be able to choose between the two alternatives according to the scarce resource.

7.1 AES Modes of Operation

AES modes of operation are algorithms used to extend capabilities of AES to cover plaintext of arbitrary length. Here, we propose solutions to protect the implementation of any standard mode. The considered modes are Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR) modes for data encryption and Counter with CBC-MAC (CCM), Galois/Counter (GCM) and Offset Codebook (OCB) modes for authenticated encryption [33, 5].

These modes assume that Alice and Bob are willing to exchange some data messages, and

that they have a shared secret key K. For every new message, aka session, they initiate the mode with a public nonce variable (also called initialization vector or counter). For CBC and CFB, the nonce needs to be unpredictable by Eve and unique. For the other modes, the nonce only needs to be unique. The length of nonce is fixed to 128 bits for CBC, CFB, OFB and CTR while it is variable for CCM, GCM and OCB. The maximum number of bytes to be encrypted in a single message is usually less than the birthday boundary of AES (2^{64}).

Every mode has a different way of connecting the input/output of the block cipher between different executions, however, they all have in common that they use the same secret key for all block cipher executions. Indeed, they employ a fixed secret key, so that the implementation requires only one execution of the key-schedule algorithm (e.g. [78]). Direct application of a key-updating scheme will require re-executing the key-schedule at every encryption, which is not compatible with the current implementations. Our key-update mechanisms are supported with an implementation trick to inject running keys directly instead of round keys. Hence, our solutions are compatible with current implementations and do not require re-executing the key-schedule.

7.2 System Overview

Fig. 7.1 shows a high-level representation of our solution. The secret key is used as a master key. The master key and the nonce (n) are processed with a leak-proof key-updating scheme. The key-updating scheme is composed of two phases. The stateless key-updating protects the master key against SCA and key-recovery attacks and generates a unique pseudorandom secret state. The stateful key updating starts from the secret state and generates session key and running keys. The session key is used in the key-schedule algorithm to generate round keys as shown in the figure. The running keys (in groups of two) are used to directly replace the first and last round keys of each encryption. In the figure, we did not show the connection between nonce, plaintext and ciphertext for specific mode as our scheme is

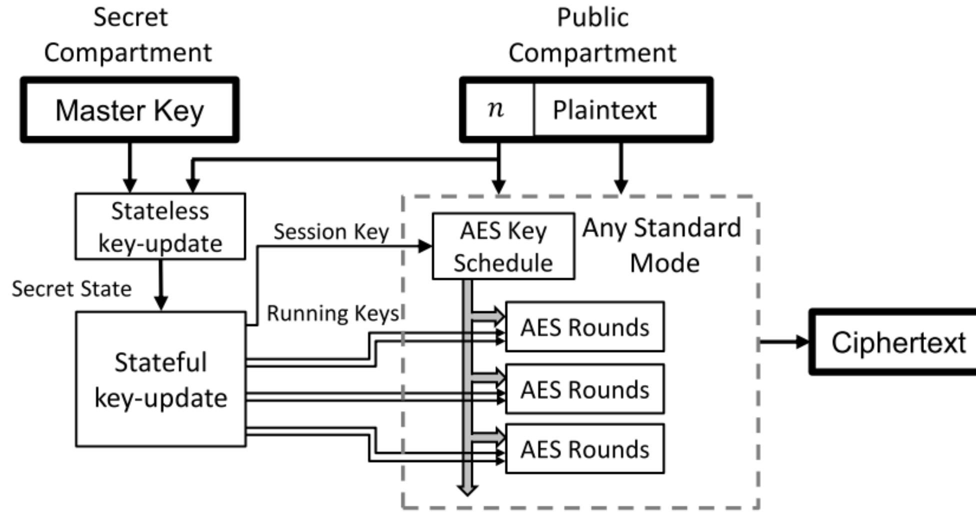

Figure 7.1: High-level representation of the proposed scheme.

compatible with any standard mode.

7.2.1 Interaction with the underlying mode of AES

The typical implementation of any standard AES mode of operation starts by running the key-schedule algorithm over the secret key to generate round keys. Then, the round keys are stored to be used in all AES encryptions.

Here, we use the first running key (which is the secret state) as a session key. Hence, the key-schedule will run over k_0 to generate round keys. Then, instead of directly using round keys in AES encryptions, each group of two running keys (k_i and k_{i+1} starting from $i = 1$) will replace the first and last round keys of each encryption as shown in Fig. 7.2.

Replacing two round keys with two running keys does not affect the black-box security of AES, as the running keys are pseudorandom and unknown. Moreover, it allows the Electronic Codebook (ECB) mode to generate indistinguishable ciphertexts.

Also, the proposed scheme is equivalent to an all-order provably secure masking scheme according to the definition of [22], because all the sensitive variables depend on pseudorandom

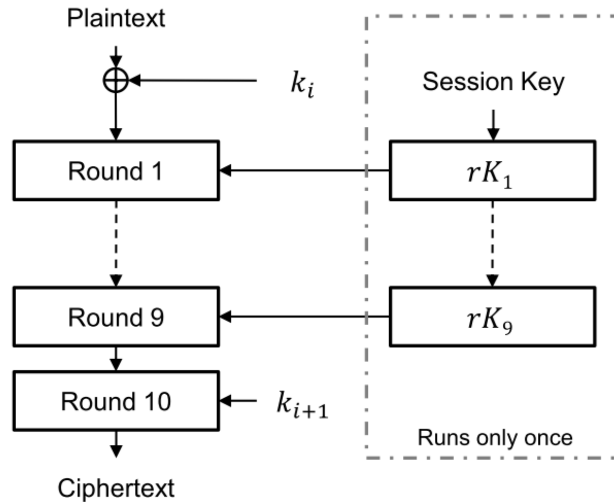

Figure 7.2: Replacing the first and last round keys by fresh running keys.

variables and they are statistically independent of the session key.

7.3 Previous Work

Previous contributions that used key-updating schemes with one public variable are shown in Table 7.1.

One of the early works that used key-updating is the work of Kocher [56], which is entirely based on DES. Unfortunately, the scheme has two drawbacks: it does not incorporate a nonce, and every key update requires two executions of the underlying DES. Without using nonce, the running keys will be generated in the same sequence in every session, which makes it vulnerable to SCA over different sessions. Two recent works proposed modular multiplication between the secret key and the nonce as an easy-to-protect key-updating primitive [69, 39]. They used practical countermeasures (e.g., hiding and masking) to protect the modular multiplication primitive.

The other contributions used GGM construction, which is the best practice in leakage resiliency. The randomization function at each step used was either a full-featured hashing

Table 7.1: Previous key-updating schemes.

Contribution	Stateless	Stateful
[56]	DES	DES
[69]	Modular MUL	—
[39]	Modular MUL + AES	AES
[57]	GGM (hashing)	Hashing
[70]	GGM (AES)	—
[12]	GGM (Minimum SP Net)	—
[95]	—	AES
This work (NLFSR)	Lightweight Tree (NLFSR)	NLFSR
This work (RR-AES)	Lightweight Tree (RR-AES)	RR-AES

function (SHA-256) [57], or full-featured Block cipher (AES) [70]. A recent contribution studied the minimum SP network that can provide sufficient protection and randomization [12].

Most key-updating contributions in the table focus only on the stateless key-updating. Under the conditions of using only one public variable, we found only few contributions for stateful key updating. Some contributions achieve heuristically secure constructions using either hashing functions or block ciphers [56, 39, 57], and one provable construction [95].

In this chapter, we propose two heuristically secure solutions for the stateless and stateful key-updating: one solution uses non-linear feedback shift registers (NLFSRs), and the other uses a round reduced version of AES (RR-AES).

7.4 Key-updating with Dedicated Circuit

An NLFSR is a common component in cryptographic stream ciphers. NLFSRs are known to be challenging targets for SCA [38], while having high performance at small implementation cost [86]. An n-stage NLFSR is a register of n binary storage units called stages. The value of the entire register is called the state. In each cycle, the register is shifted by one bit, while the new value of the first bit is the output of the feedback function $f(S)$; where $f(S)$ is a non-linear function computed over the state of the register with mapping $f : \{0,1\}^n \rightarrow \{0,1\}$. The output of the NLFSR is the sequence that shows at the last stage. The period of an NLFSR is the length of the longest cyclic output it can produce. The NLFSR that can generate the full period $(2^n - 1)$ is called a primitive NLFSR.

Unfortunately, there is no theory on how to construct a primitive NLFSR. However, there are many constructions in the literature that have been carefully designed for a guaranteed maximum period. Hence, we will not design a new NLFSR. Instead, we focus on how to use one of the established NLFSRs in the proposed construction.

Achterbahn [40] is a cryptographic stream cipher that was designed as part of the eSTREAM competition. An innovative part of Achterbahn stream cipher is the design of 13 primitive NLFSRs of different sizes (21 bits to 33 bits). Although Achterbahn did not advance to the eSTREAM portfolio due to some limitations in the combining function (which we are not using here), the NLFSRs are still a valuable contribution. As a re-keying scheme, the individual NLFSRs show a sufficient cryptographic security, as the output stream is not directly observable by Eve. Next, we use three of the Achterbahn NLFSRs to build the proposed re-keying scheme.

The proposed construction is composed of four NLFSRs, as shown in Fig. 7.3. We name the different registers by their feedback functions. Registers $f_1(S)$ is a 31-bit register, $f_2(S)$ and $f_3(S)$ are identical 32-bit registers and $f_4(S)$ is a 33-bit register. The feedback functions of these NLFSRs are taken from the Achterbahn stream cipher [40] (where they are named

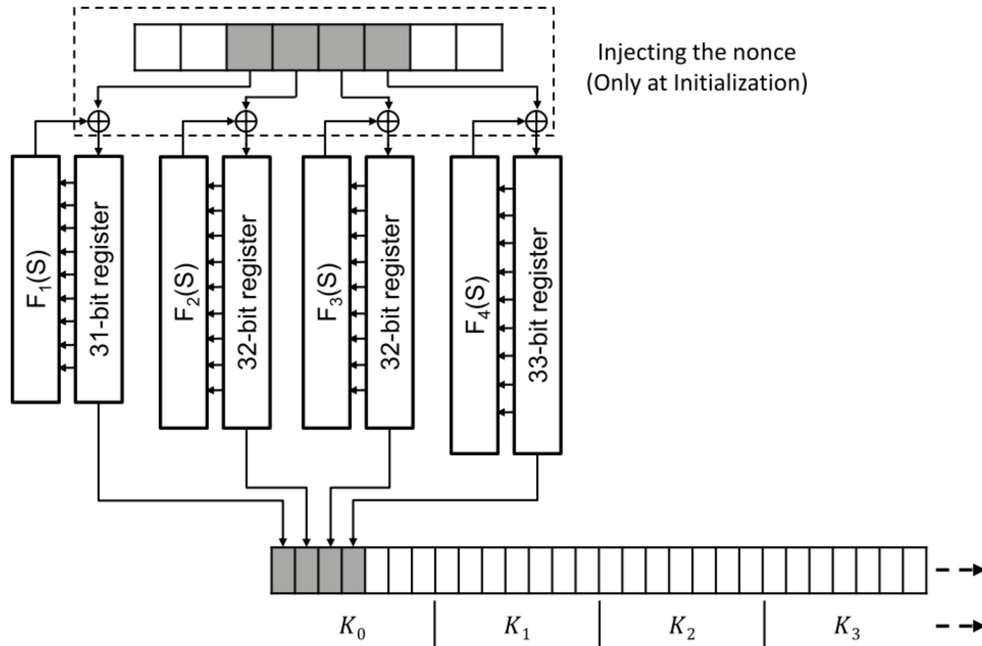

Figure 7.3: Architecture of the re-keying scheme.

A_{10}, A_{11}, A_{12}). Hence, the internal state of the construction is 128-bits, similar to the AES key length. The FSRs run independent from each other.

7.4.1 Key-Updating Functions

The key-updating mechanism follows the same framework discussed in the previous chapter. Here, we propose function Wt of the stateless key-updating (Fig. 6.1), and function Wc of the stateful key-updating (Fig. 6.3).

The stateless key-updating starts by loading all the master key bits into the internal state of the NLFSR in parallel. Then, every invocation of Wt is:

1. Adding four bits of the nonce, one bit for each register, to the NLFSRs by xoring with the feedback functions.
2. Running the NLFSRs without inputs for (33) clock cycle. Meanwhile, the output is

discarded.

At the end of executing the tree structure, the internal state of the registers will be unique and pseudorandom (the pseudorandom secret state). Then, every invocation of Wc is represented by running the NLFSRs for 32 clock cycles, enough to produce 128 bits of key.

7.4.2 Security Analysis

There are two main concerns about the security of the practical construction: security against mathematical (black-box) cryptanalysis, and security against SCA.

Mathematical Cryptanalysis

The NLFSR output is pseudorandom [40], hence the relation between the running keys will not match the required relation of any related-key attacks (e.g. [21, 20]), which sufficiently validates that the black-box security of AES is not affected.

Also, the feedback functions have correlation immunity (which is the number of state bits that if recovered, gives no additional information about the function output) of 6, 4, and 6 bits. This gives our construction high one-wayness property, which prevents key-recovery attacks.

Moreover, NLFSRs exhibit high linear complexity, which is the number of output bits that are required to re-construction NLFSR [31]. Hence, a single recovered running key cannot be used to recover the internal state of the NLFSR, which gives the construction both forward and backward security during the stateful key-updating.

Side-Channel Analysis

Protection against SCA attacks is achieved if Wt is secure against SPA and DPA attacks, with non-linear and high-diffusion properties (see Section 6.2.2).

First of all, parallel loading of the full master key provides sufficient protection against SPA attacks. Also, as stated in the previous chapter, the tree construction is leakage resilient iff each step can withstand its leakage. In each step of our construction, Eve has a maximum of two differential traces (at $n_i = 0$ and $n_i = 1$), and is supposed to recover 31, 32 or 33 bit of secret state from a hardware with four parallel-operating NLFSRs, which is practically an impossible task. For instance, practical attacks against Grain and Trivium, which are two stream ciphers from the eSTREAM portfolio that partially depend on NLFSRs, required as least 2600 and 550 power traces respectively [91].

In addition, the output bit of each register depends on the entire state hence, the high-diffusion condition is achieved. Also, the nonlinearity of feedback functions (which is the minimum hamming distance to affine functions) are 61440, 57344 and 114688 bits hence, the non-linearity condition is achieved.

7.4.3 Supported Number of Encryptions

The NLFSRs have different periods: $(2^{31} - 1)$, $(2^{32} - 1)$ and $(2^{33} - 1)$. This implies that, after any register completes its own cycle, the output byte value (which is the typical target for SCA) will not cycle in most of its bits. Hence, Eve will not be able to directly combine the leakage of the corresponding encryptions. The byte value will cycle only after the least common multiplier of the periods, which is on the order of 2^{96} bits. This output is enough for 2^{82} encryptions (256 bits per encryption) which exceed the birthday boundary of AES and exceed the current requirements for data encryption.

7.4.4 Implementation

The NLFSRs can run at different levels of parallelism in the feedback function. At no parallelism, the feedback function will produce one bit per register per clock. Hence, the tree structure will require $33 * (m/4)$ clock cycles. Each group of running keys (256 bits)

will require 64 clock cycles. The logic of the feedback function can be duplicated to produce 2, 4 or 8 bits per register per clock. Assuming that the level of parallelism is P; where $P \in [1, 2, 4, 8]$, the tree structure will take $(33/P) * (m/4)$ clocks, while each group of running keys will require $(64/P)$ clock cycles.

The hardware budget of the implementations at a low-Vt 1.5V standard cell library targeting 130 nm CMOS technology is:

$$Area = (864 + 125.5 * P)\ GE \tag{7.1}$$

The 864 GE covers the internal state with scan type flip-flop for parallel loading, and the 125.5 GE covers the feedback function of the four registers.

The master key and the nonce should be available at the start of every session. Thereafter, there are two methods to generate the running keys. First, all the required keys can be generated and stored off-line before the first encryption. The use of this method is limited by the availability of a sufficient storage. Second, the running keys can be generated on-line before every encryption. In this case, one running key should be ready in the same execution time of AES. Assuming the typical implementation of AES-128 in 10 clocks, parallel level of $P = 8$ can be used to generate one group of running keys every 8 clock cycles, with a hardware budget of 1,868 GE.

7.5 Key-updating with Round-Reduced AES

In this section, we propose another realization of the whitening function (Wt and Wc), which utilizes cryptographic properties of the underlying AES block cipher itself. Here, Wt and Wc functions are defined as follows.

7.5.1 Key-Updating Functions

Definition of Wt

Let $\mathsf{Encr}_k(p)$ denote the application of the first AddRoundKey and two rounds of AES to the plaintext p under the key k, i.e. a round-reduced version of AES. Let n denote a nonce, and $n(i)$ denote bit i of the nonce. Assuming K is the master key, the stateless key-updating starts by initializing $K^0 = K$. Then, one step of the tree will be defined as:

$$K^{i+1} = \mathsf{Wt}_{n(i)}(K^i) := \begin{cases} \mathsf{Encr}_{1^{128}}(K^i), & \text{if } n(i) = 1 \\ \mathsf{Encr}_{0^{128}}(K^i), & \text{if } n(i) = 0 \end{cases}$$

i.e. Wt is the application of a round reduced version of AES to the previous key under the key of all zeros or all ones (depending on the bit value of the nonce). Note that, the master key (and later keys) are used as the plaintext, and a fixed input is used as the key. Also, capital letters K denote the master key, or any key within the tree, while small letters k denote running keys. Finally, the pseudorandom secret state will be $s = K^{|n|-1}$, where $|n|$ is the bit-length of the nonce n.

Definition of Wc

The running key chain starts by initializing the first running key to the secret state: $k_0 = s$. Then, each new running key will be generated by applying the Wc function on the previous key. Wc will be a whitening function realized by Encr with the key fixed to all zeros:

$$k_{i+1} = \mathsf{Wc}(k_i) := \mathsf{Encr}_{0^{128}}(k_i)$$

7.5.2 Security Analysis

Stateless key-updating

First of all, although the master key is used in the data path and the fixed input is used as the key (which removes the need of key-schedule for the tree itself), this change is transparent to SCA analysis, as the two values are xored to each other.

Then, Wt function has two security features against SCA attacks:

- *Key-dependent algorithmic noise*: Similar to [70], the hardware implementation of Wt computes 16 S-boxes at the same time. All the S-boxes accept the same input (all 1's or all 0's) xored with different key bytes. Hence, while recovering any key byte, the effective signal to noise ratio will be (1/15).
- *Limited number of differential traces*: No matter how many traces are collected from Wt, there are only two fixed inputs (all 0's or all 1's). Hence, the total number of differential traces will be limited to only two.

Now, we need to examine if these security features can prevent SPA and DPA attacks. Hence, we study the mathematical security bounds under the best attack scenario: template subset sum attack.

Under parallel hardware implementations, the system power consumption of 16 parallel S-boxes at noiseless measurement is:

$$L_j = \sum_{i=0}^{16} l(\ S(p_j(i) \oplus k(i))\ ),$$

where j is the trace number, $p_j(i)$ is the fixed input byte at trace number j, and $k(i)$ is the secret key byte at location $i \in [1:16]$. Also, S is the S-box function, and l is the leakage function.

Here, template subset sum attack tries to recover all the secret key bytes at the same time, i.e. tries to find the combination of 16 key bytes that satisfies the above equation. For the

best attack scenario, we assume a perfect profiling phase where the leakage of every output of the S-box has its distinct value, i.e. $l(x) = x$. Considering SPA attacks (using only one equation), Eve's problem is to find a subset $\mathcal{K}$ of 16 elements from the set $[0 : 255]$, such that the previous equality holds. This problem is actually the well-known subset sum problem, which is NP-complete. Although many algorithms were proposed to find a correct solution (e.g. the LLL algorithm [59]), our problem is more complicated. Here, Eve is required to find all the correct solutions (not only any correct one), and test them all, including all the permutations, searching for the correct secret key.

The problem of testing all the correct solutions can be eased by considering two differential traces (DPA), i.e. at two different inputs similar to our tree construction. Now, Eve will only need to find a subset $\mathcal{K}$ that shows correct result for both traces. Given that the original problem is NP-complete, we could not find exact bounds for the number of correct solutions that show up in both traces. Hence, we simulated a small part of the key-space in a mathematical computing package, and computed the average number of correct solutions. Precisely, we did as follows:

1. Generate N random keys. We assume that the key-space is only N keys.
2. Compute all the corresponding power traces at two fixed inputs (all 0's and all 1's).
3. Loop over 500 experiments:
    - Randomly, select one key of the key-space.
    - Compute the corresponding power consumption at an input of all 0's.
    - Find all keys that result in the exact power consumption with the same input, and store the keys in $\mathcal{S}_0$.
    - Compute the corresponding power consumption at an input of all 1's.
    - Find all keys that result in the exact power consumption with the same input, and store the keys in $\mathcal{S}_1$.

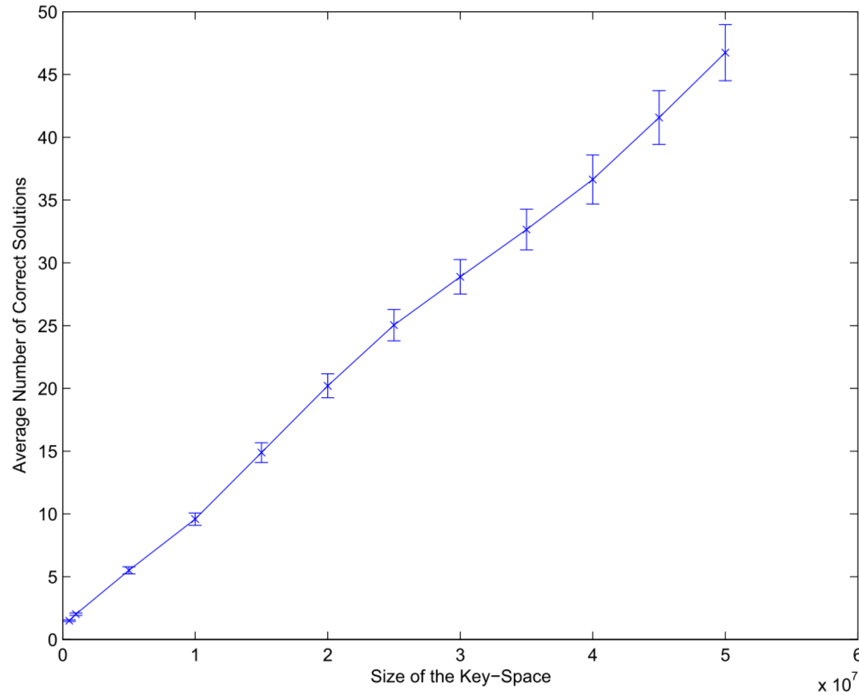

Figure 7.4: Average number of correct solutions at different key-space sizes.

- Count the number of keys that show up in both sets $\mathcal{S}_0$ and $\mathcal{S}_1$.

4. Find the average over all the experiments.

We tested the previous algorithm at different sizes of the key-space $N \in [100\text{K} : 50\text{M}]$ traces, only limited by the available memory at our workstation. The average number of correct solutions in the differential template subset sum attack is shown in Fig. 7.4. The figure shows an almost-linear relationship. We acknowledge that we cannot extrapolate these numbers to a key-space size of 2^{128}. However, the figure shows that the average number of correct solution at the full key-space is huge (in the order of 2^{108}). The 95% confidence intervals shown in the figure are computed with the bootstrapping method, as the probability density function of the average value did not match any standard distribution.

Our analysis shows that at the best possible attack, the parallel hardware is SCA-secured after collecting two differential traces. Indeed, one research showed that the number of

correct solutions will narrow to one (unique solution) only around 128 equations (or as we call them: differential trace) [66].

Stateful key-updating

As discussed in Sec. 6.2.2, the only requirement for Wc is to exhibit high non-linear diffusion. Here, the MixColumns operation of AES is responsible for diffusion [30]. It mixes four bytes of the state per round. ShiftRows operation changes the location of the bytes, so that the diffusion affects the entire state within two rounds. Hence, two rounds of AES is enough to achieve the high-diffusion requirement. Each round of AES processes each byte through the non-linear S-box transformation, hence, the non-linearity requirement is also achieved.

7.5.3 Implementation

To enable a round-reduced option in the hardware implementation, we add a mode input. If the mode input is set, the output is ready after two rounds, otherwise the output is ready after ten rounds. We implemented the two cores using Synopsys Design Compiler at UMC 130nm technology, where the difference was only two gates at 3.7 Gate Equivalent (GE).

All executions of Wt and Wc use only two keys (all 0's or all 1's), hence the key-schedule algorithm will run only two times to output, and store a total of four round keys.

The Wt function requires two clock cycles, plus two cycles to load the key and the fixed input (assuming that the fixed input changes at every step). Therefore, the complete performance overhead of the stateless key-updating is $|n| * 4$ clock cycles. Assuming the use of 128 bits nonce, which is a fixed value for most modes, the performance overhead will be 512 clock cycles.

Also, function Wc requires two clock cycles, plus one cycle to load the key (the input in fixed to all 0's). Every encryption requires two running keys, hence the total performance

overhead for the stateful key-updating is 6 clock cycles. This overhead is almost 60% less than the currently best direct constructions of [95, 39], where a full AES encryption (14 cycles) is required at each update.

It is clear that our construction achieves the highest level of security against SCA attacks, at small area and performance overheads.

7.6 Comparison

A comparison between the implementation overhead of the proposed schemes and that of the previous work is shown in Table 7.2. Here, we assume that the bit-length of the nonce is 128 bits. Note that, the performance overhead of the NLFSR structures, in the stateful key-updating, is lower than that considered in Section 7.4.4 by 12 cycles, as the NLFSR structure runs in parallel with the underlying AES. Also, we do not report any area overhead for AES related schemes, because they utilize the same underlying core. The results of [69] are taken at the first-order masked implementation, while the results of the minimum SP network are taken at the recommended implementation of [12]. For comparison at small area, we use the currently smallest implementation of AES in [77] and that of SHA-256 in [25]. For comparison at fast computation, we use the AES core in [70] and the SHA-256 core in [49].

Fig. 7.5 shows the implementation overhead for the stateless key-updating schemes. The key-updating schemes that use SHA-256 and AES-Small are not shown in the figure for having excessive implementation overhead. The results of [70] using AES-Fast is shown in a circle, as it targets a lower security level than our construction. Our RR-AES construction at the same security level (as will be cleared in the next section) is also shown for comparison.

The implementation overhead of different techniques used for the stateful key-updating is shown in Fig. 7.6. The scheme that uses SHA-256-Fast is not shown for having excessive area overhead. The figure also shows a state-of-are masking scheme. The smallest threshold implementation (to prevent leakage caused by glitches) of AES requires 8,393 GE of area

Table 7.2: Comparison between the implementation overhead of the key-updating schemes.

Contribution	Area (GE)	Clock cycles
Stateless key-update		
Modular Mul in [69]	7,300	562
Tree in [57] using SHA-256-Small	6,125	292*128
Tree in [57] using SHA-256-Fast	21,670	68*128
GGM in [70] using AES-Small	0	3,844
GGM in [70] using AES-Fast	0	206
Tree in [12] using Minimum SP Net	4,470	131
Our NLFSR, P=1	989.5	1056
Our NLFSR, P=2	1,115	528
Our NLFSR, P=4	1,366	264
Our NLFSR, P=8	1,868	132
Our RR-AES	3.7	512
Stateful key-update		
[39, 95] using AES-Small	0	226
[39, 95] using AES-Fast	0	12
[57] using SHA-256-Small	6,125	292
[57] using SHA-256-Fast	21,670	68
Our NLFSR, P=1	989.5	52
Our NLFSR, P=2	1,115	20
Our NLFSR, P=4	1,366	4
Our NLFSR, P=8	1,868	0
Our RR-AES	3.7	6

overhead, and works at 266 cycles per encryption [77]. The threshold implementation is shown on the stateful key-updating figure, as the performance overhead of stateless key-

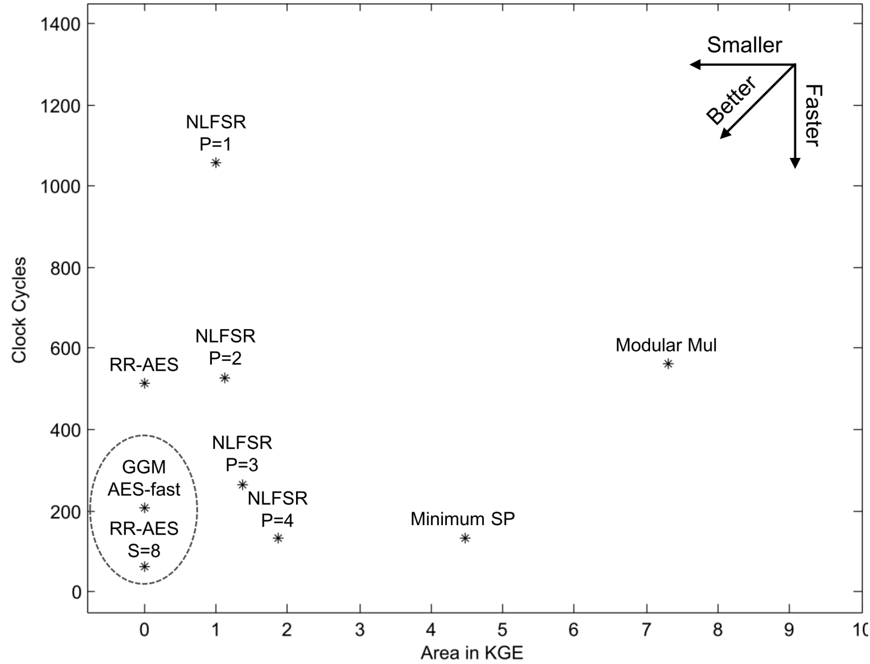

Figure 7.5: The implementation overhead of the different techniques used for the stateless key-updating.

updating is a one-time overhead (once per message), and can be trivialized at long message lengths.

The figures show that, if the scarce resource is the area, system designer should choose our construction with RR-AES. Our RR-AES scheme achieves the smallest performance overhead within the key-updating schemes with no area overhead (neglecting the 3.7 GE of our scheme). If the scarce resource is the execution time, system designer should choose the NLFSR construction with parallel level of $p = 4$ for having no performance overhead in the stateful key-updating. Also, our NLFSR construction achieves the smallest area of dedicated updating circuits.

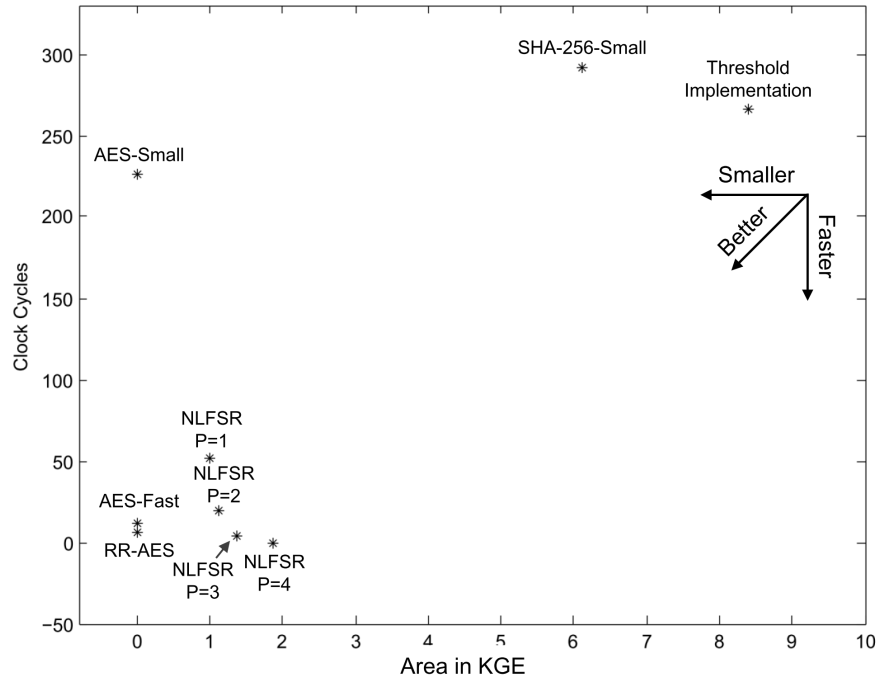

Figure 7.6: The implementation overhead of the different techniques used for the stateful key-updating.

7.7 Trading SCA-security for Performance

The reported performance overhead of the stateless key-updating structures targets the best possible protection against SCA. Both schemes use one bit of the nonce at each step of the tree for a maximum of two differential traces. Indeed, limiting Eve by two differential traces exhibit mathematically secure implementations, however, as discussed in section 5.3, most practical markets can trade some SCA security for a better performance.

The performance of our structures can be improved by using s bits of the nonce in each step of the tree. Here, s is a security parameter for trading marginal SCA-security for performance. Using a security level of s allows Eve to collect 2^s differential traces. So far, we designed our scheme for the best SCA-security ($s = 1$). However, lower security bounds can be adopted for low-cost applications, e.g. $s = 8$ was the security level tolerated in the design of [70]. The exact change in SCA-security can only be measured at a practical setup

as in [43]. We leave the exact measure of how s affects SCA-security as future work, because any results, although time consuming, will be applicable to only one implementation.

For the NLFSR structure, we insert s bits of the nonce to each register before clocking the structure, without inputs or outputs, for (33) cycles. This will result in reducing the performance overhead by s times. Hence, the entire tree structure will consume $(33/P) * (m/4s)$ clock cycles.

For the RR-AES structure, instead of repeating the nonce bit (0 or 1) over all the fixed input bits (result in all 0's or all 1's), we repeat blocks of n bits of the nonce. Similarly, this will result in reducing the performance overhead by s times. In this case, the entire tree structure will consume $(|n|/s) * 4$ clock cycles. The performance overhead of the RR-AES structure at $s = 8$ was shown in Fig. 7.6.

7.8 Summary

In this chapter, we proposed two solutions to protect the implementation of any AES mode of operation. One solution used a dedicated circuit for key-updating, hence achieved negligible performance overhead at small area overhead. The other solution utilized 2 rounds of the underlying AES, hence achieved negligible area overhead and small performance overhead. The two alternatives should cover the needs of secure hardware designers.

Chapter 8

Solution for Keyed Applications of SHA-3

Keccak is a hashing function selected by NIST to be the next SHA-3 standard for secure hashing functions. SHA-3 is the first hashing function to enable variable-length output, thanks to its Sponge construction. Hence, although the main application of SHA-3 is hashing, it can also perform random number generation, MAC generation, stream encryption, authenticated encryption and more [14]. SHA-3 may one day be the workhorse of cryptography for the beauty of having a single core that supports most security suites. However, this goal is currently humbled by the exclusive need of keyed applications to be protected against side-channel analysis.

Indeed, the authors of SHA-3 algorithm (Keccak) proposed threshold implementations, which is a provable way of applying masking countermeasure, using three and four shares [19]. Although the proposed masking technique is a sound countermeasure, it carries over a huge loss in performance (3x to 4x) when the application does not process any secret (e.g. hashing). Even for the keyed applications (e.g. MAC), masking is utilized for only one execution of Keccak-function f after the first message block as shown in Figure 8.1. Masking was developed to protect block ciphers, where the key is processed in every run. However,

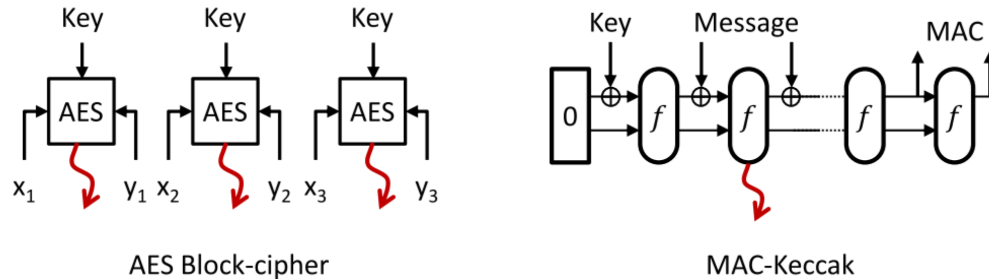

Figure 8.1: Difference between block-ciphers and hashing functions with respect to DPA attacks.

permutation-based techniques (including Keccak) process the key only once, while all the following runs continuously update the internal state, hence no SCA-protection is required. If we used the proposed masking countermeasure, we will face the problem of using 3x implementation cost that is useful for only few instances in the lifetime of the cryptographic module.

In this chapter, we solve this problem by proposing a novel countermeasure for SHA-3 using our key-updating framework. As SHA-3 is a new algorithm, to the best of our knowledge, there is no contribution in leakage resilient cryptography that can support the Sponge construction of SHA-3. Although one research proposed a leakage resilient MAC construction [67], the proposed function does not depend on any standard, and it cannot support other applications (e.g. authenticated encryption).

Our countermeasure depends on protecting the leaking execution of Keccak-function f at minimal hardware modifications so that, it does not affect the other runs of f in keyed applications or other unkeyed applications. Bearing in mind that the typical hardware implementation of Keccak processes all the internal state (1600 bits) at the same time, we used this massive parallelism as a hiding technique while reducing the active information within each trace to the minimum. To enable our countermeasure, we mandate the use of the nonce (which was optional in MAC's) in a special input format. For the unprotected executions of f, there will be no cost of any kind (except for a negligible area overhead).

For the protected executions of f (only few instances), there will be a one-time performance overhead equivalent to 15.875 extra runs (assuming nonce of 128 bits). This performance overhead targets the best possible SCA-protection while, if required by the market, the performance overhead can be reduced by trading some SCA-protection for performance.

8.1 Keyed Applications of SHA-3

Keccak has been introduced with sufficient details in Section 4.2. The applications that involve processing of a secret key are [14]:

- MAC-digest generation: The sponge operates with arbitrary-length input ($|K|+|n|+|M|$) and fixed-length output ($|MAC|$), where K is the secret key, n is the nonce, and M is the message. The hashing function is represented by H. In this application, the use of the nonce is optional.

$$MAC = H(K||[n]||M)$$

- Stream encryption: The sponge operates with fixed-length input ($|K| + |n|$) and arbitrary-length output ($|P|$), where P is the input plaintext.

$$C = H(K||n) \oplus P$$

- Authenticated encryption: The sponge operates in the SpongeWrap construction, where there is a new input and output in each execution of the Keccak-function. The authenticated encryption mode is shown in Figure 8.2, where the two previous modes are combined to generate ciphertext and MAC digest in a single pass. Here, the nonce can take the role of associated data.

Figure 8.2 can also serve to understand the other two modes. In the MAC mode, there is no ciphertext output, while there is no message or MAC output in the stream encryption mode.

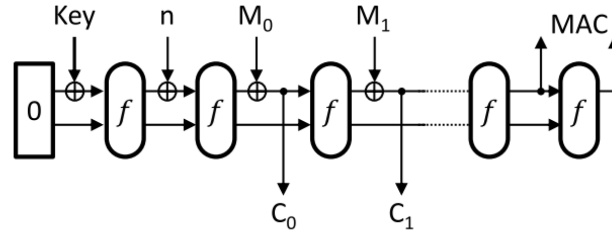

Figure 8.2: Authenticated Encryption mode using KECCAK.

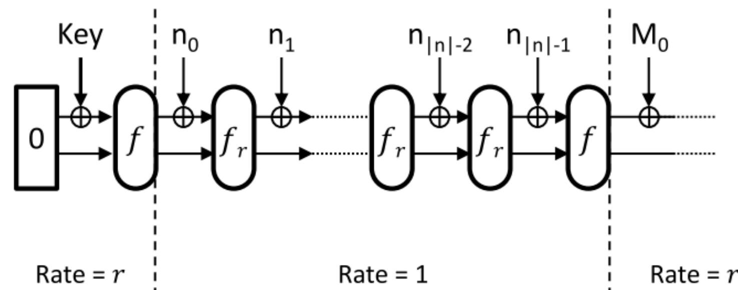

Figure 8.3: The new countermeasure, where $|n|$ is the bit-length of the nonce and f_r is a round-reduced version of Keccak f.

8.2 One Module for all Applications

Our countermeasure is highlighted in Figure 8.3, and works as follows:

First of all, we use one run of Keccak-function f to spread the limited size key (typically 128 bits) to the full internal state (1600 bits). Then, the rate of the Sponge is squeezed to only one bit, where every bit of the nonce, except the last one, goes to a separate input block followed by a round-reduced version of Keccak f_r. Here the tree construction shows out because every bit of the nonce determines the next leaf in the tree. The very last bit of the nonce will be followed by a normal full-round f. Then, the normal rate of the Sponge will be restored and the message will be processed normally.

This countermeasure matches the applications of stream and authenticated encryptions, however we have to mandate the use of nonce in the MAC-digest generation. We will not impose any special requirement on the nonce other than what is already required for these

applications, that it is nonce and uncontrolled by Eve [17]. Also, the bit-length of the nonce will depend on the required cryptographic strength, with no special requirements for applying our countermeasure.

8.2.1 Security Analysis

In fact, the countermeasure follows the key-updating framework proposed in Chapter 6. However, we start with applying a one-way function (f) before starting the tree. This step utilizes the large internal state of the Sponge in protection. Indeed, the first run of f increases the number of unknown bits, that Eve will have to recover, from the original key-size (assuming 128 bits) to 1600 bits.

Then, the tree structure starts. The Wt function is represented by a 3-rounds of Keccak function (round reduced version of Keccak: f_r), at an input rate of 1 bit. As discussed earlier, Wt should be protected against simple and differential power attacks.

First of all, having only one bit of known / controlled value (n_i) represents the most difficult attack of unprotected implementations, as shown previously in Figure 4.7. The detailed analysis of such attack is as follows.

There will be two points to mount DPA attack: during θ operation, similar to [92], and during the state update between rounds, similar to [16]. Recalling that θ is an xor gate with 11 inputs and one output, the known bit will take part in 11 different operations. If Eve mounts a DPA attack with a key guess of one bit, she will get positive results for both 0 and 1 if any input to these operations is 0 or 1. If she mounts a DPA attack with a key guess of 10 bits, she will get 11 different correct results with no way to put these results in the correct order. During the state update between rounds, the DPA attack will be worse because the known bit will affect 33 different state bits. Note that, divide and conquer principle of SCA is not applicable in these attacks, because all the considered state bits are affected by the same nonce-bit and get updated at the same time.

Still, Eve will have to attack the first three rounds, where the known bit affects the entire state. She will have to use the results from one round to attack the next round which increases the inter-dependency between the recovered key bits. In short, attacking unprotected hardware implementation of Keccak using one bit of known input is like searching for a needle in a haystack.

While processing the nonce bits (except the last one), the number of rounds is reduced to only 3 (instead of 24). This round-reduced version of Keccak will not affect its theoretical security because the intermediate outputs are not observable by Eve. These round-reduced f's are used only for the SCA-protection. Due to the high diffusion properties of Keccak, a single bit will affect the entire state after 3 rounds [9]. From SCA perspective, the remaining rounds do not add any more security to the implementation.

Then, the last nonce bit will be processed with a full 24 rounds of Keccak as a one-way function, which restores the theoretical security and prepares the output to be partially observed by Eve. The maximum size of the observable output is the rate.

Due to the use of nonce, the output at this point (after the tree) should be unique and pseudo random. From this point forward, the internal state will be updated in every execution. Eve will never be able to build a correct hypothesis, because she cannot observe part of the internal state (of size equal to the capacity). Hence, the stateful key-updating is not useful here.

8.3 Implementation

Fortunately, the input rate of Keccak can always be reduced by reformatting the input block. The input block at each execution gets xored with the previous state. Hence, by setting the first $r - 1$ bits to zero, we can reduce the effective rate to only one bit as required. The format of the Key and Message will not be changed, however the nonce will be re-formatted

as follows. Assuming that the Key is padded to fill complete input blocks, the nonce will be:

$$0^{r-1}||n_0 \; || \; 0^{r-1}||n_1 \; || \; .... \; || \; 0^{r-1}||n_{|n|-1}$$

Also, we updated the reference hardware implementation of Keccak (the high speed core) [45] with a mode input, similar to the round-reduced version of AES in the previous chapter. The difference in gate counts was only two gates at 3.7 Gate Equivalent (GE).

8.3.1 Performance and trading SCA-protection for performance

The performance of the new countermeasure is relatively slow. For the secured executions, we require $(|n| - 1) * 3$ extra Keccak rounds, which is equivalent to $(|n| - 1)/8$ extra runs of Keccak-function f (i.e. 15.875 runs at $|n| = 128$). This overhead targets the best possible SCA-protection and is required once per message.

If the market cannot withstand this performance overhead, the performance of our countermeasure can be improved by trading some SCA-protection. The 1-bit model that we used provides a top SCA-protection. Typically, higher numbers of known none bits can safely be used. Compared with the most studied algorithm AES, an unprotected FPGA implementation at the full 128 bits of known inputs could only be broken after almost 400 traces [84]. Hence, after a precise assignment of the target market and implementation, some SCA-protection can be traded for performance.

Here, we use (s) as an SCA-security parameter. In the context of SHA-3, s will represent the number of nonce input bits at every execution. s will have a flexible range $[1 : |n|]$, where $(s = 1)$ gives the best SCA-protection at the worst performance, while $(s = |n|)$ gives no SCA-protection at all. The performance overhead of our countermeasure will be reduced by s. The amount of lost / gained SCA-security by changing s can be approximated by the ratio between known bits and secret bits, as previously used in Section 4.3.

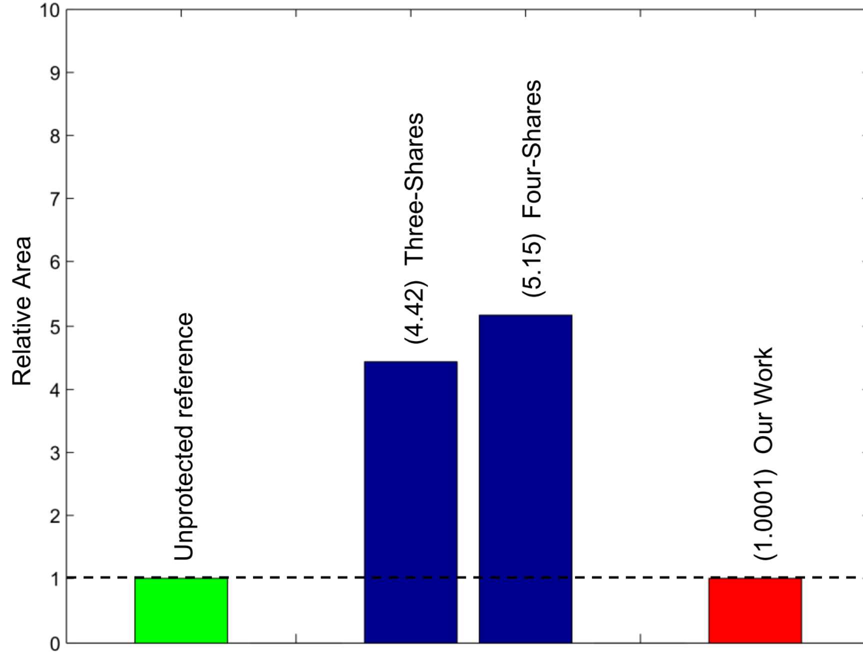

Figure 8.4: Relative area compared to the unprotected core.

8.3.2 Comparison

In this section, we compare the area and throughput of our countermeasure to the parallel threshold implementation of [19].

We used the following equation to measure the relative throughput of our countermeasure, where the effect of the number of original input blocks (m) is included ($m = (|K|+|n|+|M|)/r$). The equation can easily be driven using $(|n|-1)/(8s)$ extra executions.

$$T = \frac{m}{m + (|n|-1)/(8s)} \tag{8.1}$$

A graphical representation of the relative area is shown in Figure 8.4, while that of the relative throughput is shown in Figure 8.5. The throughput is plotted with $|n| = 128$ bits, and for $s =$ 1,2 and 4 bits.

It is clear that we achieve negligible area overhead. Given that our countermeasure uses a fixed performance overhead, the average throughput increases by increasing the message

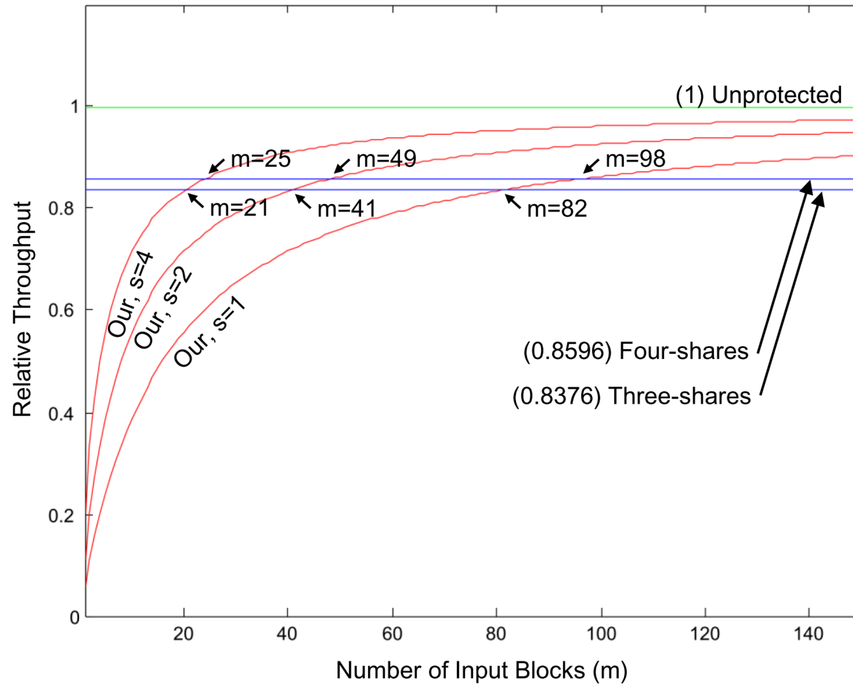

Figure 8.5: Relative throughput at 128 bits of nonce compared to the unprotected core.

length. The throughput of our countermeasure at the maximum SCA-protection ($s = 1$) exceeds the three and four shares threshold implementations for message length exceeding 82 and 98 blocks. A reduced SCA-protection (e.g $s = 4$) can achieve an average throughput higher than the considered threshold implementations for message length exceeding 21 and 25 blocks.

Flexibility, compatibility and portability There are three unique features in our countermeasure that could not be captured in the area / throughput comparison above.

First, our countermeasure is flexible. Once masking is built into the hardware, it can only be changed with reconfigurability. However, even after the proposed solution is implemented in hardware, the trade-off between performance and SCA-resistance can be tuned with the driver software.

Second, our solution is compatible with already-build modules. For example, the ASIC

implementation of [48] can only be protected against SCA-attacks using our countermeasure. Although the SCA-security parameter discussed above is still applicable, the overall performance overhead in this case will be $(|n| - 1)$ extra executions.

Also, the proposed concept is portable. Our countermeasure can protect the Sponge construction itself using any underlying permutation function. Masking can only protect one specific permutation function however, our countermeasure can also protect PHOTON [47], QUARK [8], SPONGENT [23], or any other permutation function.

8.4 Summary

In this chapter, we proposed a novel, flexible SCA-countermeasure for SHA-3 that. Our countermeasure enables producing single hardware core for all the keyed and unkeyed applications of SHA-3. For the unkeyed applications, there will be no loss of any kind. For the keyed applications, there will be a one-time performance loss that can be trivialized at long message lengths.

Chapter 9

Conclusion

We conclude our research with a discussion about the power of leakage resiliency as an SCA-countermeasure. Then we state the overall conclusion.

9.1 Conclusion about Leakage Resiliency

We believe that practical leakage resiliency is a very powerful and generic tool to protect against SCA-attacks. This argument is supported with two solutions for the block cipher AES and one solution for the hashing function SHA-3, that achieve better security, flexibility and performance than the state-of-art masking and hiding techniques.

However, there are three limitations in practical leakage resiliency:

- New Design to New Crypto: As observed in the dissertation, solutions for block-ciphers are different from those for hashing functions. Hence, we need a new design for every new cryptographic protocol. If a system designer requires one solution for all cryptographic protocols, only the techniques of hiding will work. Hiding enables the design of secure processors that can perform generic computations [94, 28]. However, hiding requires at least double the area, and cannot prevent an attack. It only increases

the required number of traces to break the module (in orders of magnitude).

- Protocol Level: As previously discussed (in Sec. 6.2.3), leakage resiliency is a protocol level protection against SCA. The final output depends on the key-updating mechanism. Hence, the two communicating parties have to follow the same key-updating mechanism, even if one of them is physically secured (e.g. server). This is not the case for hiding or masking, where the final output is not affected by the protection mechanism. If the overhead required by the techniques of practical leakage resiliency is not scalable in the server side, we recommend the use of masking. Masking is better than hiding as discussed in Sec. 5.1. Still, masking cannot prevent an attack. It can only make it harder, by requiring a higher-order DPA.

- Overhead of the Tree: Finally, the techniques of practical leakage resiliency has a one-time performance overhead while processing the tree. As observed in Fig. 8.5, the throughput is promising only at long message lengths. The performance overhead of practical leakage resiliency will be significant if the cryptographic protocol is used for only one message block, e.g. challenge response application. In such application, we also recommend the use of masking, once the designer is aware of its limitations.

In general, we believe that hiding and masking have gained significant research from the community over more than ten years. In this dissertation, we started the new track of practical leakage resiliency, with solutions for block-ciphers and hashing functions. We hope that the research in practical leakage resiliency continue to support other cryptographic primitives, e.g. public-key crypto and signature schemes.

9.2 Overall Conclusion

In this dissertation, we proposed several advances in the side-channel analysis of symmetric cryptographic primitives. In the attack side, we proposed a novel method to profile the side-

channel leakage of parallel hardware modules. We also proposed the first comprehensive analysis of the new secure hashing function SHA-3. We studied the effect of changing the key-length on the difficulty of mounting attacks. We supported our analysis with practical attacks of a 32-bit soft-core processor.

In the protection side, we proposed a framework of lightweight key-updating for efficient leakage resiliency. We used this framework to propose two solutions for AES modes of operation. One solution using a dedicated circuit for key-updating. The other solution utilized the cryptographic properties of the underlying AES itself. Also, we proposed a novel SCA-countermeasure for the keyed applications of SHA-3. Our countermeasure let the module run at full speed in unkeyed applications, and require one-time performance overhead in keyed applications.

The focus of our research is to provide a better understanding of Side-Channel Analysis, and to support the cryptographic community with lightweight and efficient countermeasures.

List of Publications

- *Attacks on Block-Ciphers:*

  -*Power Attacks:*

  [1] Mostafa Taha, Patrick Schaumont. A Novel Profiled Attack in the Presence of High Algorithmic Noise. *International Conference on Computer Design (ICCD-2012)*, Montreal, Canada, September 2012.

  -*Fault Attacks:*

  [2] Nahid Ghalaty, Mostafa Taha, Patrick Schaumont "'Differential Fault Intensity Analysis,"' *Under review.*

- *Countermeasures for Block-Ciphers:*

  -*Hiding:*

  [3] Suvarna Mane, Mostafa Taha, Patrick Schaumont. Efficient and Side-Channel-Secure Block Cipher Implementation with Custom Instructions on FPGA. *International Conference on Field Programmable Logic and Applications (FPL-2012)*, Oslo, Norway, August 2012.

  -*Masking:*

  [4] Hassan Eldib, C. Wang, Mostafa Taha and Patrick Schaumont. QMS: Evaluating the side-channel resistance of masked software from source code, *ACM/IEEE Design Automation Conference (DAC-2014).* San Francisco, CA, June 2014.

-*Leakage Resiliency:*

[5] Mostafa Taha, Patrick Schaumont. A Key Management Scheme for DPA-Protected Authenticated Encryption, *Directions in Authenticated Ciphers (DIAC-2013), Chicago, IL, August 2013.*

[6] Mostafa Taha, Patrick Schaumont. Lightweight Key-Updating For Efcient Leakage Resiliency, *Under review at the IEEE Transactions on Information Forensics and Security.*

[7] Mostafa Taha, Sebastian Faust, Patrick Schaumont. Low Cost Side-Channel Countermeasure for AES Modes of Operation, *Under review at the Journal of Cryptographic Engineering.*

- *Attacks on Hashing Functions:*

[8] Mostafa Taha, Patrick Schaumont, "Differential Power Analysis of MAC-Keccak at Any Key-Length," 8th International Workshop on Security (IWSEC-2013), Okinawa, Japan, November 2013.

[9] Mostafa Taha, Patrick Schaumont, "Side-channel Analysis of MAC-Keccak",IEEE International Symposium on Hardware-Oriented Security and Trust (HOST-2013), Austin, TX, June 2013.

- *Countermeasures for Hashing Functions:*

-*Leakage Resiliency:*

[10] Mostafa Taha and Patrick Schaumont, "Side-Channel Countermeasure for SHA-3 at Almost-Zero Area Overhead", IEEE Symposium on Hardware Oriented Security and Trust (HOST-2014), Arlington, VA, May 2014.

Bibliography

[1] Keccak reference code submission to NIST (round 3). http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/documents/Keccak_FinalRnd.zip.

[2] Xilinx microblaze soft processor core. http://www.xilinx.com/tools/microblaze.htm.

[3] *DPA Contest v2 2009/2010*. Telecom ParisTech french University, 2009.

[4] RSA cryptography standard PKCS# 1 v2.2. *RSA Laboratories*, page 63, 2012.

[5] Information technology, security techniques, authenticated encryption. In *ISO/IEC 19772:2009*. Retrieved March 12, 2013.

[6] Michel Abdalla, Sonia Belaïd, and Pierre-Alain Fouque. Leakage-resilient symmetric encryption via re-keying. In *Cryptographic Hardware and Embedded Systems-CHES*, pages 471–488. Springer, 2013.

[7] Dakshi Agrawal, Bruce Archambeault, Josyula Rao, and Pankaj Rohatgi. The EM SideChannel(s). In *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer Berlin / Heidelberg, 2003.

[8] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Mara Naya-Plasencia. Quark: A lightweight hash. *Journal of Cryptology*, 26(2):313–339, 2013.

[9] Jean-Philippe Aumasson and Dmitry Khovratovich. First analysis of keccak, 2009.

[10] Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, and Ingrid Verbauwhede. Power analysis of atmel cryptomemory–recovering keys from secure eeproms. *Topics in Cryptology–CT-RSA 2012*, pages 19–34, 2012.

[11] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential cluster analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 112–127. Springer Berlin Heidelberg, 2009.

[12] Sonia Belaid, Fabrizio De Santis, Johann Heyszl, Stefan Mangard, Marcel Medwed, Jorn-Marc Schmidt, Francois-Xavier Standaert, and Stefan Tillich. Towards fresh re-keying with leakage-resilient PRFs: Cipher design principles and analysis. Cryptology ePrint Archive, Report 2013/305, 2013.

[13] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology CRYPTO 96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 1996.

[14] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic sponge functions. working draft v0.1, 2011. `http://sponge.noekeon.org/CSF-0.1.pdf`.

[15] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak reference. *Submission to NIST (Round 3)*, 3.0, 2011.

[16] Guido Bertoni, Joan Daemen, Nicolas Debande, Thanh-Ha Le, Michael Peeters, and Gilles Van Assche. *Power Analysis of Hardware Implementations Protected with Secret Sharing.* 2013. Published: Cryptology ePrint Archive, Report 2013/067 `http://eprint.iacr.org/2013/067.pdf`.

[17] Guido Bertoni, Joan Daemen, Michal Peeters, and Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption and other applications. Cryptology ePrint Archive, Report 2011/499, 2011.

[18] Guido Bertoni, Joan Daemen, Michal Peeters, and Gilles Van Assche. The keccak sha-3 submission. *Submission to NIST (Round 3)*, 2011.

[19] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and first-order DPA resistant implementations of keccak. In *Smart Card Research and Advanced Application - CARDIS*. Springer Berlin Heidelberg, 2013.

[20] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key recovery attacks of practical complexity on AES variants with up to 10 rounds. Cryptology ePrint Archive, Report 2009/374, 2009.

[21] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. Cryptology ePrint Archive, Report 2009/317, 2009.

[22] Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably secure masking of AES. In *Selected Areas in Cryptography*, volume 3357, pages 69–83. Springer, 2005.

[23] Andrey Bogdanov, Miroslav Kneevi, Gregor Leander, Deniz Toz, Kerem Varc, and Ingrid Verbauwhede. Spongent: A lightweight hash function. In *Cryptographic Hardware and Embedded Systems CHES*, volume 6917, pages 312–325. Springer Berlin Heidelberg, 2011.

[24] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 135–152. Springer Berlin / Heidelberg, 2004.

[25] Xiaolin Cao and Maire O'Neill. Application-oriented SHA-256 hardware design for low-cost RFID. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, page 1412:1415. IEEE, 2012.

[26] Suresh Chari, Josyula Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 51–62. Springer Berlin / Heidelberg, 2003.

[27] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 191–206, May.

[28] Zhimin Chen, A. Sinha, and P. Schaumont. Using virtual secure circuit to protect embedded software from side-channel attacks. *Computers, IEEE Transactions on*, 62(1):124–136, 2013.

[29] Joan Daemen, G Bertoni, M Peeters, G Van Assche, and R Van Keer. Keccak implementation overview. Technical report, Technical report, NIST, 2012.

[30] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[31] Elena Dubrova, Maxim Teslenko, and Hannu Tenhunen. On analysis and synthesis of (n,k)-non-linear feedback shift registers. In *Design, Automation and Test in Europe, 2008. DATE'08*, page 1286:1291. IEEE, 2008.

[32] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on aes. In *Applied Cryptography and Network Security*, pages 293–306. Springer, 2003.

[33] Morris Dworkin. NIST special publication 800-38A, recommendation for block cipher modes of operation: Methods and techniques.

[34] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 293–302. IEEE, 2008.

[35] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad Shalmani. On the power of power analysis in the real world: A

complete break of the KeeLoq code hopping scheme. *Advances in Cryptology–CRYPTO 2008*, pages 203–220, 2008.

[36] Hassan Eldib, Chao Wang, Mostafa Taha, and Patrick Schaumont. QMS: Evaluating the side-channel resistance of masked software from source code. In *ACM/IEEE Design Automation Conference (DAC)*, June 2014.

[37] Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. Practical leakage-resilient symmetric cryptography. In *Cryptographic Hardware and Embedded Systems-CHES*, pages 213–232. Springer, 2012.

[38] Wieland Fischer, Berndt M Gammel, Oliver Kniffler, and Joachim Velten. Differential power analysis of stream ciphers. In *Topics in Cryptology, CT-RSA 2007*, page 257270. Springer, 2006.

[39] Berndt Gammel, Wieland Fischer, and Stefan Mangard. Generating a session key for authentication and secure data transfer. Google Patents, 2010. US Patent App. 12/797,704.

[40] Berndt M Gammel, Rainer Göttfert, and Oliver Kniffler. Achterbahn-128/80. In *eSTREAM, ECRYPT Stream Cipher Project*. 2006.

[41] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. Cryptology ePrint Archive, Report 2013/857, 2013. `http://eprint.iacr.org/`.

[42] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer Berlin / Heidelberg, 2008.

[43] Benjamin Jun Gilbert Goodwill, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST Non-invasive attack testing workshop*, 2011.

[44] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[45] Michal Peeters Guido Bertoni, Joan Daemen and Gilles Van Assche. Keccak hardware implementation in vhdl version 3.1.

[46] Tim Güneysu and Amir Moradi. Generic side-channel countermeasures for reconfigurable devices. In *Cryptographic Hardware and Embedded Systems-CHES*, pages 33–48. Springer, 2011.

[47] Jian Guo, Thomas Peyrin, and Axel Poschmann. The photon family of lightweight hash functions. In *Advances in Cryptology CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer Berlin Heidelberg, 2011.

[48] Xu Guo, M. Srivastav, Sinan Huang, D. Ganta, M.B. Henry, L. Nazhandali, and P. Schaumont. Asic implementations of five sha-3 finalists. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1006–1011, 2012.

[49] Xu Guo, Meeta Srivistav, Sinan Huang, Dinesh Ganta, Michael Henry, Leyla Nazhandali, and Patrick Schaumont. ASIC Implementations of Five SHA-3 Finalists. In *Design, Automation Test in Europe Conference Exhibition, DATE.*, March 2012.

[50] Annelie Heuser, Michael Kasper, Werner Schindler, and Marc Stttinger. A new difference method for Side-Channel analysis with High-Dimensional leakage models. In *Topics in Cryptology CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 365–382. Springer Berlin / Heidelberg, 2012.

[51] Annelie Heuser, Werner Schindler, and Marc Stoettinger. Revealing Side-Channel issues of complex circuits by enhanced leakage models. In *ACM/IEEE Design Automation and Test in Europe (DATE'12)*, March 2012.

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

[53] Lyndon Judge, Michael Cantrell, Cagil Kendir, and Patrick Schaumont. A modular testing environment for implementation attacks. In *BioMedical Computing (BioMedCom), 2012 ASE/IEEE International Conference on*, pages 86–95. IEEE, 2012.

[54] Toshihiro Katashita, Yohei Hori, Hirofumi Sakane, and Akashi Satoh. Side-channel attack standard evaluation board sasebo-w for smartcard testing. *Power*, 3:400, 2011.

[55] Paul Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology CRYPTO 96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin / Heidelberg, 1996.

[56] Paul Kocher. Leak-resistant cryptographic indexed key update, 2003. US Patent 6,539,092.

[57] Paul Kocher. Complexity and the challenges of securing SoCs. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, page 328331, 2011.

[58] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 789–789. Springer Berlin / Heidelberg, 1999.

[59] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[60] Yang Li, Daisuke Nakatsu, Qi Li, Kazuo Ohta, and Kazuo Sakiyama. Clockwise collision analysis – overlooked side-channel leakage inside your measurements. Cryptology ePrint Archive, Report 2011/579, 2011. `http://eprint.iacr.org/`.

[61] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 320–334. Springer, 2010.

[62] Yingxi Lu, M.P. O'Neill, and J.V. McCanny. Fpga implementation and analysis of random delay insertion countermeasure against dpa. In *ICECE Technology, 2008. FPT 2008. International Conference on*, pages 201–208, Dec 2008.

[63] S. Mane, M. Taha, and P. Schaumont. Efficient and side-channel-secure block cipher implementation with custom instructions on FPGA. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 20–25, Aug 2012.

[64] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks revealing the secrets of smart cards*. Springer, New York, N.Y., 2007.

[65] Stefan Mangard, Thomas Popp, and BerndtM. Gammel. Side-channel leakage of masked cmos gates. In *Topics in Cryptology CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer Berlin Heidelberg, 2005.

[66] Olvi L Mangasarian and Benjamin Recht. Probability of unique integer solution to a system of linear equations. *European Journal of Operational Research*, 214(1):27–30, 2011.

[67] Dan Martin, Elisabeth Oswald, and Martijn Stam. A leakage resilient mac. Cryptology ePrint Archive, Report 2013/292, 2013. `http://eprint.iacr.org/`.

[68] Robert McEvoy, Michael Tunstall, Colin C Murphy, and William P Marnane. Differential power analysis of HMAC based on SHA-2, and countermeasures. In *Information Security Applications*, pages 317–332. Springer, 2007.

[69] 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*, page 279:296. Springer, 2010.

[70] 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*, page 193:212. Springer, 2012.

[71] Olivier Meynard, Sylvain Guilley, Jean-Luc Danger, and Laurent Sauvage. Far correlation-based ema with a precharacterized leakage model. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 977–980. European Design and Automation Association, 2010.

[72] A. Moradi, O. Mischke, and C. Paar. One attack to rule them all: Collision timing attack versus 42 aes asic cores. *Computers, IEEE Transactions on*, 62(9):1786–1798, 2013.

[73] 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 *Proceedings of the 18th ACM conference on Computer and communications security*, pages 111–124. ACM, 2011.

[74] Amir Moradi, Markus Kasper, and Christof Paar. Black-box side-channel attacks highlight the importance of countermeasures. *Topics in Cryptology–CT-RSA 2012*, pages 1–18, 2012.

[75] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 125–139. Springer Berlin / Heidelberg, 2010.

[76] Amir Moradi, David Oswald, Christof Paar, and Pawel Swierczynski. Side-channel attacks on the bitstream encryption mechanism of altera stratix ii: facilitating black-

box analysis using software reverse-engineering. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 91–100. ACM, 2013.

[77] 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*, page 69:88. Springer, 2011.

[78] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. Efficient and high-performance parallel hardware architectures for the AES-GCM. *Computers, IEEE Transactions on*, 61(8):1165–1178, 2012.

[79] M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger. Rsm: A small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1173–1178, 2012.

[80] Colin OFlynn. Power analysis for cheapskates. Rev 15, 2012. `http://www.newae.com/tiki-index.php?page=OpenADC`.

[81] Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. Cryptology ePrint Archive, Report 2014/204, 2014. `http://eprint.iacr.org/`.

[82] David Oswald and Christof Paar. Breaking mifare desfire mf3icd40: power analysis and templates in the real world. *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 207–222, 2011.

[83] E Oswald et al. Opensca, an open source toolbox for matlab, 2010.

[84] TELECOM Paristech. DPA contest v4, 2010.

[85] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *Advances in Cryptology-EUROCRYPT 2009*, pages 462–482. Springer, 2009.

[86] Matthew Robshaw and Olivier Billet. New stream cipher designs: the estream finalists. volume 4986. Springer, 2008.

[87] AKASHI Satoh. Side-channel attack standard evaluation board, sasebo. *Project of the AIST–RCIS (Research Center for Information Security), http://www. rcis. aist. go. jp/special/SASEBO*, 135, 2010.

[88] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *Cryptographic Hardware and Embedded Systems CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer Berlin / Heidelberg, 2005.

[89] Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. Simple photonic emission analysis of aes. In *Cryptographic Hardware and Embedded Systems–CHES 2012*, pages 41–57. Springer, 2012.

[90] 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*, pages 99–134. 2010.

[91] Daehyun Strobel, Ing Christof Paar, and M Kasper. *Side channel analysis attacks on stream ciphers*. PhD thesis, master thesis, 2009.

[92] M. Taha and P. Schaumont. Side-channel analysis of MAC-Keccak. In *Hardware-Oriented Security and Trust (HOST), IEEE International Symposium on*, June 2013.

[93] Kris Tiri, David Hwang, Alireza Hodjat, Bo-Cheng Lai, Shenglin Yang, Patrick Schaumont, and Ingrid Verbauwhede. Prototype ic with wddl and differential routing–dpa resistance assessment. In *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 354–365. Springer, 2005.

[94] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. In *Proceedings of the conference on Design,*

automation and test in Europe - Volume 1, DATE '04, page 10246. IEEE Computer Society, 2004.

[95] Yu Yu and François-Xavier Standaert. Practical leakage-resilient pseudorandom objects with minimum public randomness. In *Topics in Cryptology–CT-RSA 2013*, pages 223–238. Springer, 2013.

[96] Yu Yu, François-Xavier Standaert, Olivier Pereira, and Moti Yung. Practical leakage-resilient pseudorandom generators. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 141–151. ACM, 2010.

[97] M. Zohner, M. Kasper, M. Stottinger, and S.A. Huss. Side channel analysis of the SHA-3 finalists. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1012 –1017, March 2012.