HECTOR

D3.3

Report on the Security Evaluation of Cryptographic Algorithms and Countermeasures when non Ideal Hardware Building Blocks are Used

Project number:	ICT-644052
Project acronym:	**HECTOR**
Project title:	Hardware Enabled Crypto and Randomness
Project Start Date:	1 March, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Report
Reference Number:	ICT-644052-D3.3
Workpackage:	WP3
Due Date:	31 August, 2017
Actual Submission Date:	31 August, 2017
Responsible Organisation:	KU Leuven
Editor:	Dave Singelée, Josep Balasch
Dissemination Level:	Public
Revision:	1.0
Abstract:	This report is one of the main scientific outcomes of the HECTOR project and represents the final version of deliverable D3.3 of work package WP3. Together with deliverable D3.1, it is part of the WP3 proceedings and extensively discusses the research results of WP3. It covers four main activities. First, the cryptanalysis of cryptographic primitives with respect to non-ideal keys. Second, it proposes an optimization strategy for cryptographic post-processing of PUFs and TRNGs, based on a duplex-sponge construction. Third, the study of the security degradation of countermeasures in the presence of non-ideal random numbers. Fourth, it reports several new results on side channel countermeasures and the evaluation of side channel robustness at design-time, which not yet have been reported in deliverables D3.1 and D3.2.
Keywords:	cryptanalysis, post-processing, side channel countermeasures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 644052.

Editor

Dave Singelée, Josep Balasch (KU Leuven)

Contributors (ordered according to beneficiary numbers)

Josep Balasch, Dave Singelée (KUL)
Maria Eichlseder (TUG)
Pierre Bayon (BRT)
Ruggero Susella, Filippo Melzani (STI)

Executive Summary

This deliverable concerns the research activities carried out within tasks T3.1 and T3.2 of the HECTOR project. It covers two main topics: the security evaluation of cryptographic primitives with non-ideal keys and the analysis of the security degradation of countermeasures against physical attacks in presence of non-ideal random numbers. This deliverable also reports on the latest research progress in task T3.4, which was not yet described in deliverable D3.1.

To assess how security degrades when keying material (keys, initialization vectors, nonces, etc.) deviate from ideal, we have applied the related-key attacker model, known-key security model and TWEAKEY framework to various cryptographic primitives. This has resulted in improved cryptanalysis for each of these block ciphers, hash functions and authenticated encryption schemes. Moreover, we have proposed a novel security model to study how known-key attacks on block ciphers and permutations impacts the security of cryptographic hash functions built upon these primitives. As an illustration on how to improve the efficiency of security schemes which combine TRNGs or PUFs with lightweight cryptography, we have proposed a sponge construction which integrates cryptographic post-processing and (authenticated) encryption into a single primitive.

To determine the security degradation of side channel countermeasures in the presence of non-ideal random numbers, we have devised a generic evaluation framework that outputs as quality metric the minimum number of side channel measurements for an attack to succeed. We have applied this framework to different families of countermeasures (masking and hiding), using multiple sets of non-ideal random number sequences (synthetic and real) and in two different yet relevant scenarios (under simulation and experimentally). The analysis of all these different configurations allows us to gain insight on which countermeasures are more susceptible to potential non-idealities in the random number sequences. At the same time, it also serves to quantify the adversarial effort gains necessary to break the protected implementations.

This deliverable is organized as follows. The overall topic of the report is introduced in Chapter 1. Novel cryptanalysis results based on non-ideal keys are presented in Chapter 2. In the same line of work, Chapter 3 proposes the Weak Cipher Model (WCM) and applies this to various Block cipher-Based and Permutation-Based Hash Functions. Chapter 4 explores the concept of lightweight post-processing, and illustrates how sponge constructions could be applied to reduce the implementation cost. The next part of the deliverable presents the research results on evaluating the security degradation of side channel countermeasures when using non-ideal random numbers. The methodology followed to analyze this security degradation has been outlined in Chapter 5, while Chapter 6 and 7 respectively describe the evaluation results using side channel measurement simulations and real measurements from implementations on the HECTOR boards. In Chapter 8, novel research results on efficient countermeasures (task T3.4 of the HECTOR project) are presented. Finally, Chapter 9 concludes this deliverable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Random numbers are essential in cryptography, as they are widely used for confidential keys, initialization vectors and padding values. They are also used in cryptographic authentication protocols and even in countermeasures against side channel attacks. Therefore, Random number generators (RNG) are one of the most essential building blocks in a security system. Physical unclonable functions (PUFs) are a much younger primitive. One of their use cases is the generation of uncloneable cryptographic keys with highly secured storage.
Since security is based on confidentiality of cryptographic keys, key generators applied in cryptographic applications must be cryptographically secure, they must generate keys (random numbers) that have good statistical properties and the generated sequences must not be predictable or vulnerable to manipulation. However, requirements of unpredictability and robustness against manipulation can be generalized to all applications of random numbers and PUF responses in data security applications. For this reason, the expected functioning of physical random number generators and physical unclonable functions directly impacts the security of the whole cryptographic system.
In this deliverable, we actually look at two cases where random numbers are used as input: cryptographic primitives (keys, nonces, etc.) and side channel countermeasures (masking, noise, etc.). Security in the context of limited randomness has already been investigated at the protocol level and for public-key encryption. However, it has not yet been studied at the symmetric-key primitive level. In task T3.1, a first step into this direction is made by building upon the related-key and known-key security models for block ciphers. Similarly, the security of most side channel countermeasures - in particular masking - relies on random numbers that are drawn from a uniform distribution. It is understood that if this property does not hold, leakage will appear that may enable attacks. However, it has not been studied in detail what types of non-uniformity are more harmful to which countermeasures. This is the main research challenge tackled in task T3.2.
This deliverable discusses the scientific outcomes of work package WP3 of the HECTOR project, together with deliverable D3.1 [82] which has been submitted before. It consists of four main parts. The first two parts present the research results of task T3.1, which aims to evaluate the security degradation of cryptographic primitives when using non-ideal keys. This problem has been studied from two different angles. The first part of this work, which is discussed in Chapter 2 and 3 of this deliverable, focuses on cryptanalytic techniques to study the effect of non-ideal keys (for example, relations between subkeys, non-independent round keys, etc.) on the mathematical strength of a cipher or hash function. The goal of this research is to get a better understanding of the relation between structural properties of cryptographic primitives and randomly drawn variables (keys, nonces, etc.). The second part of this work, which is discussed

in Chapter 4, connects the work of WP2 and WP3 of the HECTOR project. This research starts from the observation that proper TRNG and PUF design requires post-processing to improve the "quality" of their output, and that this post-processing might be consuming more resources than the cryptographic primitive that will be using these TRNG/PUF outputs. Therefore, the research challenge tackled in this work is to investigate how the overall system can be made more lightweight, for example by combining part of the functionality of the post-processing and the cryptographic primitive. The third part of this deliverable covers the research results of task T3.2, which aims to evaluate the security degradation of countermeasures against physical attacks in the presence of non-ideal random numbers. Chapter 5 gives an overview of the framework and tools we used to perform this evaluation, while the results of the evaluation of the countermeasures are discussed in Chapter 6 and 7. Our study employs both synthetic sets of non-ideal random numbers as well as real sets obtained from experiments in WP2, and uses both simulations of side channel measurements as real measurements from implementations on the HECTOR board to assess the security degradation of the side channel countermeasures. Lastly, the fourth and last part of this deliverable gives an update on the novel research results obtained in the context of HECTOR task T3.4, but which not yet have been reported in deliverable D3.1. The overview of these scientific results can be found in Chapter 8 of the deliverable.

Chapter 2

Cryptanalysis based on non-ideal keys

2.1 Introduction

When designing cryptographic primitives, it is important to understand the robustness of the primitives in a mathematical context of a cryptanalytic scenario. This cryptanalysis is a complex task and has evolved over the years. The two most well-known techniques are linear cryptanalysis, whose goal is to approximate the cipher by affine equations, and differential cryptanalysis, whose goal is to study how differences in the input of the cipher affect differences at the output. However, there are other cryptanalysis techniques as well to assess the mathematical strength of the cipher. Some of these explicitly study the relation between cryptographic (sub)keys - for example generated by an RNG - and the cipher. These cryptanalytic techniques are not only important from a theoretical point of view, because protocol flaws or weaknesses in random numbers could create unintentional relations between nonces and/or keys, which could significantly influence the cryptographic strength of the cipher. Within the HECTOR project, we particularly focused on the cryptanalytic techniques that capture this concept of "non-ideal" keys: related-key attacks, known-key attacks and related-tweaky (differential) attacks.

Related-key attack: In the related-key model, the attacker can decrypt/encrypt not only under the master key K, but also under the keys $f_1(K), f_2(K), \ldots, f_m(K)$, which are called *related-keys*. The relations f_i are chosen by the attacker in advance. This relation could be a simple mapping such as rotations or bit flips, or a more advanced function. The aim of the attacker is to recover the master keys.

Known-key attack: In the classical security models for block ciphers the key is secret and randomly drawn, and the adversary's target is to distinguish the instantiation of the cipher from a random permutation. In the setting of known-key security, the key is presumed known, and the adversary succeeds in distinguishing if it identifies a structural property of the cipher.

Related tweaks: Tweakable block ciphers (TBCs) [76] generalize block ciphers by adding an additional public input, the tweak, which plays a role similar to the nonces of AEAD schemes, and is under full control of the adversary. Tweakable constructions such as the TWEAKEY framework [60] incorporate the tweak as part of the key schedule. Thus, choosing related tweaks creates a cryptanalytic setting closely tied to related keys.

Within the HECTOR project, we applied each of these cryptanalysis techniques to one or more symmetric key primitives, such as block ciphers or authenticated encryption primitives, and improved the currently best-known cryptanalytic attacks on these algorithms.

2.2 Related-Key Rectangle Cryptanalysis of Rijndael-160 and Rijndael-192

2.2.1 Introduction

Rijndael [32] is a block cipher designed by Daemen and Rijmen and is a substitution-permutation network following the wide-trail strategy. Both the block length and the key length can be any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits, independently of each other with key size greater than or equal to block size. The 128-bit block variant of Rijndael has been chosen as the Advanced Encryption Standard (AES) [95]. Our research deals with non-AES Rijndael variants, that is, Rijndael-b/k where b indicates the block size and k indicates the key size in bits.
AES is one of the most well-studied block ciphers: since its introduction 15 years ago there has been extensive analysis of AES. Some prominent examples include square attacks, impossible differential attacks, boomerang attacks, rectangle attacks and meet-in-the-middle attacks in both the single-key and related-key settings. On the other hand, the variants of Rijndael with larger block sizes have got arguably less attention from the cryptographic community. Current analysis includes several multiset and integral attacks [40, 43, 74, 91], as well as impossible differential cryptanalysis [92, 112, 117]. These target 6 or 7 rounds of Rijndael-160/160 and 6, 7, 8 or 9 rounds of Rijndael-192/192 respectively. We extend this work by presenting the first related-key rectangle cryptanalysis of Rijndael-160/160 and Rijndael-192/192. The results of this section were published in the journal IET Information Security [113].

2.2.2 Description of Rijndael

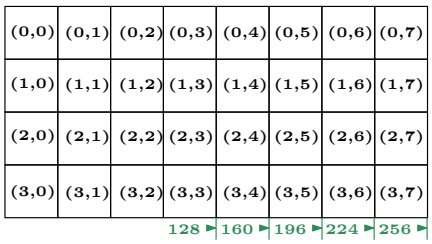
In Rijndael, each data block (plaintext, ciphertext, subkey, or intermediate step) is represented by a $4 \times N_b$ `state matrix` of bytes, where N_b is the block size divided by 32. The state is then transformed by iterating a round function. The round function is composed of the following four operations:

- `SubBytes (SB)` : a non-linear byte substitution (8×8-bit S-box) that acts on every byte of the state.
- `ShiftRows (SR)`: a cyclic shift of bytes in a row that acts individually on each of the last three rows of the state. The shift offset C_i of row i depends on the block length N_b (See Figure 2.1).
- `MixColumns (MC)`: a linear transformation (based on an $[8, 4, 5]$ MDS code over $GF(2^8)$) that acts independently on every column of the state
- `AddRoundKey (AK)`: the exclusive-or of the round key with the intermediate state.

The number of rounds for the cipher N_r varies with N_b and N_k (the key size divided by 32). Before the first round, there exists a whitening layer consisting of `AK` only, and in the last round the `MC` operation is omitted. We assume that this is also the case for the reduced round versions of Rijndael.

Key Scheduling

The key schedule derives $(N_r + 1)$ b-bit round keys RK_0, $RK_1 \ldots RK_{N_r}$ from the master key. It consists of a linear array of 4-byte words denoted by $W[i]$ for $0 \leq i \leq N_b \cdot (N_r + 1)$. The



(a) Byte index

N_k	N_b 4	5	6	7	8
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

(b) Number of rounds

N_b	C_0	C_1	C_2	C_3
4	0	1	2	3
5	0	1	2	3
6	0	1	2	3
7	0	1	2	4
8	0	1	3	4

(c) Shift offsets

Figure 2.1: Byte index of the state matrix and the shift offsets for each block length N_b

first N_k words $W[0]\|W[1]\|\cdots\|W[N_k-1]$ are directly initialized with the words of the master key, while the remaining key words, $W[i]$ for $i \in [N_k, N_k \cdot (N_r+1)-1]$ are generated by the following algorithm:

$$\texttt{if } (i \bmod N_k) = 0 \texttt{ then } W[i] = W[i-N_k] \oplus SB(W[i-1] \lll 8) \oplus Rcon[i/N_k]$$

$$\texttt{else if } (N_k > 6 \text{ and } i \bmod N_k = 4) \texttt{ then } W[i] = W[i-N_k] \oplus SB(W[i-1])$$

$$\texttt{else } W[i] = W[i-N_k] \oplus W[i-1]$$

where $\lll$ denotes the rotation of the word to the left and $Rcon[\cdot]$ denotes the fixed constants. Then the round key RK_i is given by the words $W[N_b \cdot i]$ to $W[N_b \cdot (i+1)]$.

Notation

The notation that we will use throughout this section is as follows:

P_a, P_b, P_c, P_d	the plaintexts
$(P_a)_{i,j}$,$(P_b)_{i,j}$,$(P_c)_{i,j}$,$(P_d)_{i,j}$	the byte at row i column j of the plaintext state
C_a, C_b, C_c, C_d	the ciphertexts
K_a, K_b, K_c, K_d	secret related keys
$(K_a)_{i,j}$, $(K_b)_{i,j}$, $(K_c)_{i,j}$, $(K_d)_{i,j}$	the byte at row i column j of the secret related keys
K_a^r, K_b^r, K_c^r, K_d^r	secret subkey of K_a, K_b, K_c and K_d in round r
ΔK_{ab}^r, ΔK_{ac}^r, ΔK_{cd}^r, ΔK_{bd}^r	the difference of the related keys in round r
$(\Delta K_{ab}^r)_{i,j}$	difference byte of state ΔK_{ab}^r, at position row i and column j
$\texttt{SB}[(i,j)]$	the SB operation on the byte at row i column j of the state matrix
$E(\cdot,\cdot)$	encryption operation defined as $\{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$

2.2.3 Rectangle Attack

The rectangle attack [12] introduced by Biham et al. is a special type of differential cryptanalysis. The main idea of the attack is to divide the cipher E into two sub-ciphers E_0 and E_1 such that $E = E_1 \circ E_0$. The attacker then constructs two relatively short differentials for E_0 and E_1 instead of finding a long differential for the block cipher E. After that, a rectangle distinguisher for E can be established by combining these two short differentials delicately. This technique is useful when we have good short differential characteristics and very bad long ones.

It is also possible to combine rectangle attack with related-key attack to derive the related-key rectangle attack [15] in which the attacker can query to the cipher with other keys that have a specified relation (often an xor-difference) with the original key. Actually, this kind of combined approach has been applied to various block ciphers and some intriguing results have been achieved for AES [18, 20] and KASUMI [13, 38].

Description. Let the encryption function E of the block cipher be considered as a cascade of two sub-ciphers $E = E_1 \circ E_0$. Assume that there exists a related-key differential $\alpha \rightarrow \beta$ for E_0 under the key difference ΔK_{ab} with probability p, i.e., $(\Pr[E_0(P,K) \oplus E_0((P \oplus \alpha), (K \oplus \Delta K_{ab})) = \beta] = p)$. Similarly, assume that there exists a related-key differential $\gamma \rightarrow \delta$ for E_1 under the key difference ΔK_{ac} with probability q, where ΔK_{ab} and ΔK_{ac} are the key differences known by the attackers. A related-key rectangle distinguisher is then as follows:

1. Choose N plaintext pairs (P_a, P_b) with $P_b = P_a \oplus \alpha$ at random. Ask for the encryption of P_a under K_a and of P_b under K_b respectively, where $K_b = K_a \oplus \Delta K_{ab}$.

2. Choose N plaintext pairs (P_c, P_d) with $P_d = P_c \oplus \alpha$ at random. Ask for the encryption of P_c under K_c and of P_d under K_d respectively, where $K_c = K_a \oplus \Delta K_{ac}$ and $K_d = K_c \oplus \Delta K_{ab} = K_b \oplus \Delta K_{ac}$.

3. For a quartet of plaintexts (P_a, P_b, P_c, P_d) with corresponding ciphertexts (C_a, C_b, C_c, C_d), check whether $C_a \oplus C_c = C_b \oplus C_d = \delta$ holds or not. If yes, we call it a *right rectangle quartet*. Figure 2.2 illustrates the related-key rectangle distinguisher.

The related-key rectangle attack can be mounted for all possible β's and γ's simultaneously. Starting with N plaintext pairs with difference α, we expect to find $N^2 2^{-n} (\hat{p}\hat{q})^2$ right quartets, where

$$\hat{p} = \sqrt{\sum_{\beta} Pr^2[\alpha \rightarrow \beta]}, \qquad \hat{q} = \sqrt{\sum_{\gamma} Pr^2[\gamma \rightarrow \delta]},$$

and n is the block size. For an ideal cipher, Step 3 is expected to hold with probability 2^{-2n}. Therefore, if $\hat{p}\hat{q} \gg 2^{-n/2}$, the algorithm above allows to distinguish E from an ideal cipher. We refer to [12, 14, 15, 64] for more detail.
Recently, in [90], Murphy showed that differential characteristics used for each sub-cipher must be independent, otherwise the complexity estimation of the attack given above does not hold. Following this result, Kim et al. [67] revisited the validity of related-key boomerang and rectangle attacks, and pointed out that if the probabilities of any round in the differentials used in the rectangle-type distinguisher are not extremely low, it is reasonable to assume that the independence assumptions underlying the rectangle-type attacks are valid. Therefore, in our analysis, we establish related-key rectangle distinguishers by choosing differentials for each sub-cipher delicately to avoid very low probabilities of any rounds in these differentials.

Local Collisions

Figure 2.2 illustrates the related-key rectangle distinguisher. The idea of local collisions has been first introduced by Joux and Chabaud [29] to attack hash functions. It aims to inject a difference into an intermediate step and then to cancel the resulting differences (called *disturbance*) with the injections in the next steps to obtain a collision. The goal is to reduce the complexity of the attack by having as few disturbances as possible. This idea has been later successfully applied to block ciphers in [18]. A local collision for one round of Rijndael-160/160 is illustrated in Figure 2.3. We will now apply this approach in our analysis of Rijndael-160/160 and Rijndael-192/192.

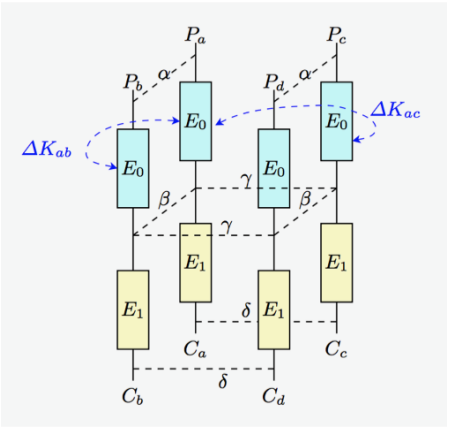


Figure 2.2: The related-key Rectangle distinguisher

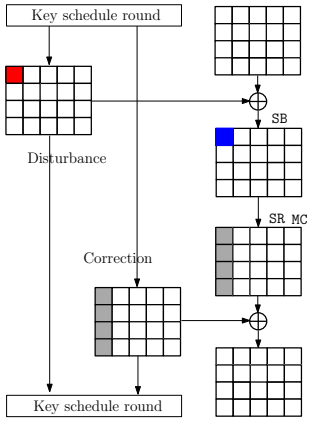


Figure 2.3: A local collision of Rijndael-160/160

2.2.4 Designing the rectangle distinguisher

The Related Keys

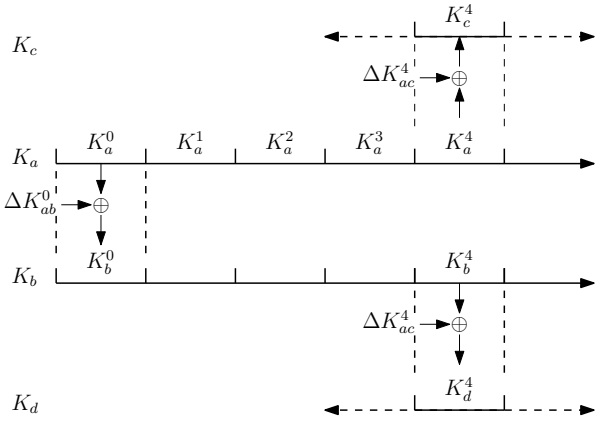


Figure 2.4: The related-key computation of Rijndael-160/160

For Rijndael-160/160, we define the relation between four keys as follows (see Table 2.1). For a secret key K_a, which the attacker tries to find, apply the subkey difference ΔK_{ab}^0 to obtain K_b. Then we compute the subkeys K_a^4 and K_b^4, based on which, the subkeys K_c^4 and K_d^4 can be calculated by using the subkey difference ΔK_{ac}^4. After that, according to the key schedule of Rijndael-160/160, we can derive K_c and K_d from the subkeys K_c^4 and K_d^4 respectively. This is depicted in Figure 2.4. Similarly, we can define the relation between four keys for Rijndael-192/192 (see Table 2.2).

Rectangle Switch

Let us now focus on the transition between the top characteristic E_0 and the bottom characteristic E_1 of the rectangle. This method is called *switch technique* [111] and it has been used to improve the probability of boomerang distinguisher. Here, we apply this switch technique to our rectangle distinguishers on Rijndael. Let E be $(m+n)$-round Rijndael cipher. Then, the

Table 2.1: Related-key differences for Rijndael-160

	ΔK^r_{ab}														
0	3f	3e	3f	3e	00	1	3f	01	3e	00	00	2	3f	3e	00	00	00	3	3f	01	01	01	01
	1f	1f	1f	1f	00		1f	00	1f	00	00		1f	1f	00	00	00		1f	00	00	00	00
	1f	1f	1f	1f	00		1f	00	1f	00	00		1f	1f	00	00	00		1f	00	00	00	00
	21	21	21	21	00		21	00	21	00	00		21	21	00	00	00		21	00	00	00	00
	ΔK^r_{ac}														
4	21	21	21	21	00	5	21	00	21	00	00	6	21	21	00	00	00	7	21	00	00	00	00
	3e	3f	3e	3f	00		3e	01	3f	00	00		3e	3f	00	00	00		3e	01	01	01	01
	1f	1f	1f	1f	00		1f	00	1f	00	00		1f	1f	00	00	00		1f	00	00	00	00
	1f	1f	1f	1f	00		1f	00	1f	00	00		1f	1f	00	00	00		1f	00	00	00	00
8	21 ⊕ x^*	21 ⊕ x	21 ⊕ x	21 ⊕ x	21 ⊕ x																		
	3e	3f	3e	3f	3e																		
	1f	1f	1f	1f	1f																		
	1f	1f	1f	1f	1f																		

* x might differ for ΔK^8_{ac} and ΔK^8_{bd}

Table 2.2: Related-key differences for Rijndael-192

	ΔK^r_{ab}																						
0	?	3e	00	00	3f	3e	1	3f	01		01		01		3e	00	2	3f	3e	3f	3e	00	00
	?	1f	00	00	1f	1f		1f	00		00		00		1f	00		1f	1f	1f	1f	00	00
	?	1f	00	00	1f	1f		1f	00		00		00		1f	00		1f	1f	1f	1f	00	00
	?	21	00	00	21	21		21	00		00		00		21	00		21	21	21	21	00	00
3	3f	01	3e	00	00	00	4	3f	3e		00		00		00	00	5	3f	01	01	01	01	01
	1f	00	1f	00	00	00		1f	1f		00		00		00	00		1f	00	00	00	00	00
	1f	00	1f	00	00	00		1f	1f		00		00		00	00		1f	00	00	00	00	00
	21	00	21	00	00	00		21	21		00		00		00	00		21	00	00	00	00	00
	ΔK^r_{ac}																						
6	00	21	21	21	21	00	7	00	21		00		21		00	00	8	00	21	21	00	00	00
	00	3e	3f	3e	3f	00		00	3e		01		3f		00	00		00	3e	3f	00	00	00
	00	1f	1f	1f	1f	00		00	1f		00		1f		00	00		00	1f	1f	00	00	00
	00	1f	1f	1f	1f	00		00	1f		00		1f		00	00		00	1f	1f	00	00	00
9	00	21	00	00	00	00	10	x^*	21⊕x		21⊕x		21⊕x		21⊕x	21⊕x							
	00	3e	01	01	01	01		00	3e		3f		3e		3f	3e							
	00	1f	00	00	00	00		00	1f		1f		1f		1f	1f							
	00	1f	00	00	00	00		00	1f		1f		1f		1f	1f							

* x might differ for ΔK^{10}_{ac} and ΔK^{10}_{bd}

common application is to choose

$$E_0 = (AK \circ MC \circ SR \circ SB)^m$$
$$E_1 = (AK \circ MC \circ SR \circ SB)^n$$

However, due to the flexibility of the SB operation (i.e., it is applied to each byte in the state independently), this choice of E_0, E_1 can be done in a more clever way. For instance, to minimize the number of active S-boxes, all the active S-boxes can be discarded in the $(m+1)^{th}$ round except those that are active for both differentials in E_0 and E_1. Let SB$[(i,j)]$, SB$[(i',j')]$ be the

SB operations on the bytes (i, j) and (i', j') respectively. Then E_0 and E_1 can alternatively be defined as follows:

$$\begin{aligned} E_0 &= SB[(i,j)] \circ (AK \circ MC \circ SR \circ SB)^m, \\ E_1 &= (AK \circ MC \circ SR \circ SB)^{n-1} \circ AK \circ MC \circ SR \circ SB[(i',j')], \end{aligned} \tag{2.1}$$

where $(i, j) \neq (i', j')$, and the bytes (i, j) and (i', j') are passive in E_0 and E_1 respectively.
For example, in our rectangle distinguisher of Rijndael-160/160, we take $m = 2, n = 4$. The first subcipher E_0 covers rounds 2–3 of Rijndael-160/160 and SB operations on 12 bytes (i, j) in round 4, where $1 \leq i \leq 3$ and $1 \leq j \leq 4$; The second subcipher E_1 starts with the SB operations on 8 bytes $(i', 0)$, $(0, j')$ in round 4 ($1 \leq i' \leq 3$, $0 \leq j' \leq 4$), followed by $AK \circ MC \circ SR$ and rounds 5–7. Our 6-round rectangle distinguisher is illustrated in Figure 2.5.

The Rectangle Distinguisher

In the analysis of Rijndael-160/160, we use a 6-round rectangle distinguisher, and extend one round before and after the distinguisher respectively (see Figure 2.5). Our rectangle distinguisher covers rounds 2–7 and we use the switch technique in round 4 to avoid the active S-boxes in the key schedule and hence to reduce the complexity of our attack. We take $m = 2$ and $n = 4$ in Equation (2.1) to obtain E_0 and E_1.

The plaintext difference α is specified in 16 bytes, two of them (denoted in dark green) can take any value whereas the remaining ones (denoted in gray) are fixed to $(0x3e, 0x1f, 0x1f, 0x21)^T$. The key difference is chosen such that when it is xored to the state, all differences cancel each other except the two bytes at $(0, 0)$ and $(0, 2)$ of the top characteristic. For the differences in these two active bytes we have:

$$(0x01 \oplus \alpha_{0,i}) \xrightarrow{SB} 0x1f, \qquad i \in \{0, 2\} \tag{2.2}$$

This guarantees that the input differences to the S-box operations in all the internal states (except the ones specified in Equation 2.2) are 0x01. For the active bytes in round 2 of the top characteristic, we adopt 0x1f as the output difference of SB operation in order to achieve the optimal differential probability 2^{-6}. We develop the bottom characteristic by taking an similar approach. As to the active byte in round 3 of the top characteristic, there are 127 possibilities of the output difference for the input difference 0x01 according to the differential distribution table of SB operation, among which one happens with the probability 2^{-6}, the others happen with probability 2^{-7}. Then we construct the 6-round related-key rectangle distinguisher by combining the 127 top characteristics and one bottom characteristic mentioned above.
The probability of the 6-round distinguisher can be computed as follows.

- There are three active S-boxes in rounds 2–3, thus the probability of the 127 differentials for E_0 can be calculated as $\hat{p} = \sqrt{(2^{-6})^{2 \cdot 2}[1 \cdot (2^{-6})^2 + 126 \cdot (2^{-7})^2]} \approx 2^{-10.5}$.
- There are five active S-boxes in rounds 4–7, thus the probability of the differential for E_1 is $\hat{q} = (2^{-6})^5 = 2^{-30}$.
- In total, the probability of this distinguisher can be calculated as $(2^{-10.5} \cdot 2^{-30})^2 \cdot 2^{-160} = 2^{-251}$.

Similarly, we find a 8-round rectangle distinguisher for Rijndael-192/192 which covers rounds 2–9, and the rectangle switch technique is applied at round 6 (see Figure 2.6). There are 9 active S-boxes in the differential characteristics of E_0 (Note that for the active S-box in round 5, all the 127 possible output differences are used to derive 127 characteristics), and the probability can be computed as $\hat{p} = \sqrt{(2^{-6})^{2\cdot 8}[1 \cdot (2^{-6})^2 + 126 \cdot (2^{-7})^2]} \approx 2^{-51.5}$. For the differential characteristic of E_1, there are 5 active S-boxes and the probability is $\hat{q} = (2^{-6})^5 = 2^{-30}$. In total, the distinguisher holds with probability $(2^{-51.5} \cdot 2^{-30})^2 \cdot 2^{-192} = 2^{-355}$.
Moreover, the differences after the `MC` operations (denoted in gray) are given as:

$$\begin{pmatrix} \texttt{0x1f} \\ 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} \texttt{0x3e} \\ \texttt{0x1f} \\ \texttt{0x1f} \\ \texttt{0x21} \end{pmatrix}; \quad \begin{pmatrix} 0 \\ \texttt{0x1f} \\ 0 \\ 0 \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} \texttt{0x21} \\ \texttt{0x3e} \\ \texttt{0x1f} \\ \texttt{0x1f} \end{pmatrix}$$

2.2.5 Attack on 8-Round Rijndael-160/160

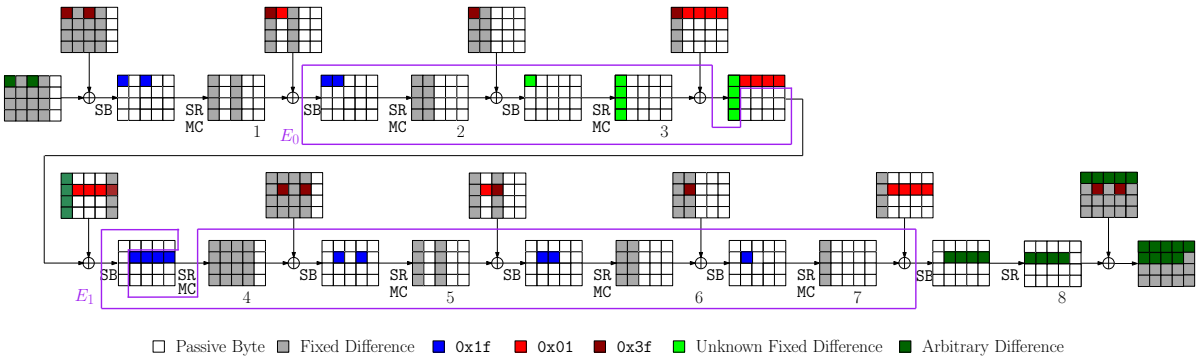


Figure 2.5: The related-key Rectangle attack on 8-Round Rijndael-160/160. Switch is applied in Round 4

Based on the 6-round distinguisher for round 2–7 given in Section 2.2.4, we can now present a key recovery attack on 8-round Rijndael-160/160 (round 1–8). The attack procedure is divided into two phases: *data collection* phase and *key recovery* phase. We will now discuss these two phases more in detail.

Data Collection

1. Generate $2^{109.5}$ structures $G_i = \{U_i, V_i\}$ of 2^{17} plaintexts each, where $1 \leq i \leq 2^{109.5}$, U_i, V_i are the sets of 2^{16} plaintexts of the form respectively, $c_i's$ $(1 \leq i \leq 18)$ are fixed 8-bit

?	c_1	?	c_2	c_3
c_4	c_5	c_6	c_7	c_8
c_9	c_{10}	c_{11}	c_{12}	c_{13}
c_{14}	c_{15}	c_{16}	c_{17}	c_{18}

and

?	$c_1 \oplus$ 3e	?	$c_2 \oplus$ 3e	c_3
$c_4 \oplus$ 1f	$c_5 \oplus$ 1f	$c_6 \oplus$ 1f	$c_7 \oplus$ 1f	c_8
$c_9 \oplus$ 1f	$c_{10} \oplus$ 1f	$c_{11} \oplus$ 1f	$c_{12} \oplus$ 1f	c_{13}
$c_{14} \oplus$ 21	$c_{15} \oplus$ 21	$c_{16} \oplus$ 21	$c_{17} \oplus$ 21	c_{18}

values and '?' takes all possible values.

2. For each structure G_i

   (a) Ask for the encryption of U_i, V_i under K_a and K_b respectively to obtain $G_i^1 = \{X_i^1, Y_i^1\}$.

   (b) Ask for the encryption of V_i, U_i under K_a and K_b respectively to obtain $G_i^2 = \{X_i^2, Y_i^2\}$.

   (c) Ask for the encryption of U_i, V_i under K_c and K_d respectively to obtain $H_i^1 = \{Z_i^1, W_i^1\}$.

   (d) Ask for the encryption of V_i, U_i under K_c and K_d respectively to obtain $H_i^2 = \{Z_i^2, W_i^2\}$.

   (e) Let $T^{1a} = \{X_i^1\}_{1\le i\le 2^{109.5}}$, $T^{1b} = \{Y_i^1\}_{1\le i\le 2^{109.5}}$, $T^{2a} = \{X_i^2\}_{1\le i\le 2^{109.5}}$, $T^{2b} = \{Y_i^2\}_{1\le i\le 2^{109.5}}$, $T^{1c} = \{Z_i^1\}_{1\le i\le 2^{109.5}}$, $T^{1d} = \{W_i^1\}_{1\le i\le 2^{109.5}}$, $T^{2c} = \{Z_i^2\}_{1\le i\le 2^{109.5}}$ and $T^{2d} = \{W_i^2\}_{1\le i\le 2^{109.5}}$.

3. Initialize two vectors L^{ac} and L^{bd} consisting of 2^{88} lists L_η^{ac} and L_η^{bd} respectively, where η corresponds to a 11-byte value (i.e., the bytes (1,4) and (i, j) of a ciphertext, where $2 \le i \le 3$, $0 \le j \le 4$), $L_\eta^{ac} = \{S_\eta^a, S_\eta^c, N_\eta^a, N_\eta^c\}$, $L_\eta^{bd} = \{S_\eta^b, S_\eta^d, N_\eta^b, N_\eta^d\}$, $S_\eta^a, S_\eta^b, S_\eta^c, S_\eta^d$ are the sets of ciphertexts under K_a, K_b, K_c and K_d respectively as well as their structure indices, and $N_\eta^a, N_\eta^b, N_\eta^c, N_\eta^d$ denote the cardinalities of the sets $S_\eta^a, S_\eta^b, S_\eta^c$ and S_η^d respectively.

4. For each ciphertext in T^{1a}, extract the 88-bit value η, then insert the ciphertext and its structure index into the set S_η^a of the corresponding list L_η^{ac} and increase N_η^a by 1. For each ciphertext in T^{1c}, xor it with

00	00	00	00	00
00	00	00	00	3e
1f	1f	1f	1f	1f
1f	1f	1f	1f	1f

   and then extract the 88-bit value η. After that, insert the ciphertext and its structure index into the set S_η^c of the corresponding list L_η^{ac} and increase N_η^c by 1. Do similarly for the ciphertexts in T^{1b}, T^{1d} and update the lists L_η^{bd}.

5. Keep the lists L_η^{ac} in which both N_η^a and N_η^c are non-zero, and keep the lists L_η^{bd} in which both N_η^b and N_η^d are non-zero. Then derive the ciphertext quartets (C_a, C_b, C_c, C_d) from the remaining lists L_η^{ac} and L_η^{bd} by using following criteria:

   (a) C_a, C_c are chosen from the same list L_η^{ac}, and C_b, C_d come from the same list $L_{\eta'}^{bd}$.

   (b) The structure indices of C_a and C_b are the same, and the structure indices of C_c and C_d are the same.

   We obtain around $(2^{16} \cdot 2^{16} \cdot 2^{109.5})^2 \cdot (2^{-88})^2 = 2^{107}$ ciphertext quartets (C_a, C_b, C_c, C_d) and their plaintext quartets (P_a, P_b, P_c, P_d) in this step.

6. For each of the 2^{107} quartets (C_a, C_b, C_c, C_d), check whether the following conditions

$$(C_a \oplus C_c)_{0,0} = (C_a \oplus C_c)_{0,1} = \ldots = (C_a \oplus C_c)_{0,4}$$

and

$$(C_b \oplus C_d)_{0,0} = (C_b \oplus C_d)_{0,1} = \ldots = (C_b \oplus C_d)_{0,4}$$

hold or not. If not, discard the quartet. The expected number of remaining quartets after this step is about $2^{107} \cdot (2^{-32})^2 = 2^{43}$.

7. Repeat Steps 4−6 three times for $(T^{1a}, T^{1b}, T^{2c}, T^{2d})$, $(T^{2a}, T^{2b}, T^{1c}, T^{1d})$ and $(T^{2a}, T^{2b}, T^{2c}, T^{2d})$.

With the above procedure we get about $2^{43} \cdot 4 = 2^{45}$ ciphertext quartets (C_a, C_b, C_c, C_d) and their plaintext quartets (P_a, P_b, P_c, P_d).

Key Recovery

8. Guess the subkey bytes $(K_a)_{0,j}$, $(K_c)_{0,j}$ $(j \in \{0, 2\})$ as follows:

   (a) guess $(K_a)_{0,0}$, $(K_c)_{0,0}$ and calculate the values of $(K_b)_{0,0}$, $(K_d)_{0,0}$ by using Table 2.1.

   (b) guess $(K_a)_{0,2}$, $(K_c)_{0,2}$ and derive the values of $(K_b)_{0,2}$, $(K_d)_{0,2}$ from Table 2.1.

   For each of the remaining quartets in substeps (a)–(b), test whether the corresponding equations

$$\begin{aligned} SB((P_a)_{0,j} \oplus (K_a)_{0,j}) \oplus SB((P_b)_{0,j} \oplus (K_b)_{0,j}) &= \texttt{1f} \\ SB((P_c)_{0,j} \oplus (K_c)_{0,j}) \oplus SB((P_d)_{0,j} \oplus (K_d)_{0,j}) &= \texttt{1f} \end{aligned}$$

   are satisfied or not. If not, discard the quartet. After this step, the number of remaining quartets is about $2^{45} \cdot (2^{-8})^4 = 2^{13}$.

9. Guess the subkey bytes $(K_a^7)_{1,4}$, $(K_b^7)_{1,4}$, $(K_a^8)_{1,j}$, $(K_b^8)_{1,j}$ $(0 \le j \le 3)$ and obtain the values of $(K_c^7)_{1,4}$, $(K_d^7)_{1,4}$, $(K_c^8)_{1,j}$, $(K_d^8)_{1,j}$ according to Table 2.1. Then for each remaining quartet (C_a, C_b, C_c, C_d), verify whether the following equations

$$\begin{aligned} SB((K_a^7)_{1,4}) \oplus SB((K_c^7)_{1,4}) \oplus (C_a)_{0,0} \oplus (C_c)_{0,0} &= \texttt{21} \\ SB((K_b^7)_{1,4}) \oplus SB((K_d^7)_{1,4}) \oplus (C_b)_{0,0} \oplus (C_d)_{0,0} &= \texttt{21} \\ SB^{-1}((C_a)_{1,j} \oplus (K_a^8)_{1,j}) \oplus SB^{-1}((C_c)_{1,j} \oplus (K_c^8)_{1,j}) &= \texttt{01} \\ SB^{-1}((C_b)_{1,j} \oplus (K_b^8)_{1,j}) \oplus SB^{-1}((C_d)_{1,j} \oplus (K_d^8)_{1,j}) &= \texttt{01} \end{aligned}$$

   hold or not. If not, remove the quartet.

10. If the number of the remaining quartets after above steps is two or more, output the corresponding 14 guessed subkey bytes $(K_a)_{0,j_1}$, $(K_c)_{0,j_1}$, $(K_a^7)_{1,4}$, $(K_b^7)_{1,4}$, $(K_a^8)_{1,j_2}$ and $(K_b^8)_{1,j_2}$ $(j_1 \in \{0, 2\},\ 0 \le j_2 \le 3)$ as the correct key information. Otherwise, return to Step 8 and repeat the procedure.

11. If the above 14 subkey bytes are retrieved after Step 10, perform an exhaustive search over all possible values of the remaining 128 bits of K_a^8 so as to recover the secret key.

Analysis of the attack on 8-Round Rijndael-160/160

In Step 9, ten equations (each with probability 2^{-8}) need to be satisfied. Therefore, for a wrong guess of the above 14 subkey bytes, the expected number of quartets after Step 9 is $2^{13} \cdot (2^{-8})^{10} = 2^{-67}$. On the other hand, for a right guess of the key, the expected number of right quartets is about

$$(2^{16} \cdot 2^{16} \cdot 2^{109.5})^2 \cdot 4 \cdot (2^{-16})^2 \cdot 2^{-251} = 2^2.$$

This means that we can discard all the wrong subkeys (since the expected number of remaining quartets for a wrong subkey is 2^{-67}) and find the right 14 subkey bytes.
The probability of outputting a wrong key guess in Step 10 is derived by the following Poisson distribution:

$$X \sim Poi(\lambda = 2^{-67}).$$

As $\Pr[X \geq 2] \approx 2^{-135}$, the expected number of wrong key guesses suggested in Step 10 is about $(2^8)^{14} \cdot 2^{-135} = 2^{-23}$, and the wrong key information can be easily removed in Step 11. Similarly, the probability that two or more quartets remain after Step 10 for the correct key guess is also computed by the Poisson distribution:

$$X \sim Poi(\lambda = 4).$$

Since $\Pr[X \geq 2] \approx 0.91$, the success probability of the attack on 8-round Rijndael-160/160 is approximately 91%.
The data complexity of this attack is $2^{109.5} \cdot 2^{17} = 2^{126.5}$ chosen plaintexts which are encrypted under K_a, K_b, K_c and K_d respectively (resulting in $2^{126.5} \cdot 4 = 2^{128.5}$ ciphertexts). The memory complexity is primarily owing to keeping T^{1a}, T^{1b}, T^{1c}, T^{1d}, T^{2a}, T^{2b}, T^{2c} and T^{2d}, thus it can be estimated as $8 \cdot 2^{125.5} \cdot 20 \approx 2^{132.82}$ bytes. The time complexity of the attack can be derived as follows:

- The time complexity of the *data collection* phase is mainly dominated by Step 2 and Step 4. For Step 2, the time complexity is $2^{126.5} \cdot 4 = 2^{128.5}$ 8-round Rijndael-160 encryptions. As to Step 4, the time complexity is about $2^{125.5} \cdot 8 = 2^{128.5}$ memory accesses, which can be measured as $2^{128.5} \cdot \frac{1}{20 \cdot 8} \approx 2^{121.18}$ 8-round Rijndael-160 encryptions.
- The time complexity of the *key recovery* phase is mainly dominated by Step 9 and Step 11. For Step 9, the time complexity can be estimated as $2^{112} \cdot 2^{13} \cdot \frac{20}{20 \cdot 8} = 2^{122}$ 8-round Rijndael-160 encryptions. Regarding Step 11, the time complexity is about 2^{128} 8-round Rijndael-160 encryptions.

As a result, the total time complexity of the attack is approximately $2^{129.28}$ 8-round Rijndael-160 encryptions.

2.2.6 Attack on 10-Round Rijndael-192/192

Based on the 8-round distinguisher for round 2–9 given in Section 2.2.4, we now present a key recovery attack on 10-round Rijndael-192/192 (round 1–10). Similarly as the attack on 8-round Rijndael-160/160, the attack procedure is divided into two phases: *data collection* phase and *key recovery* phase.

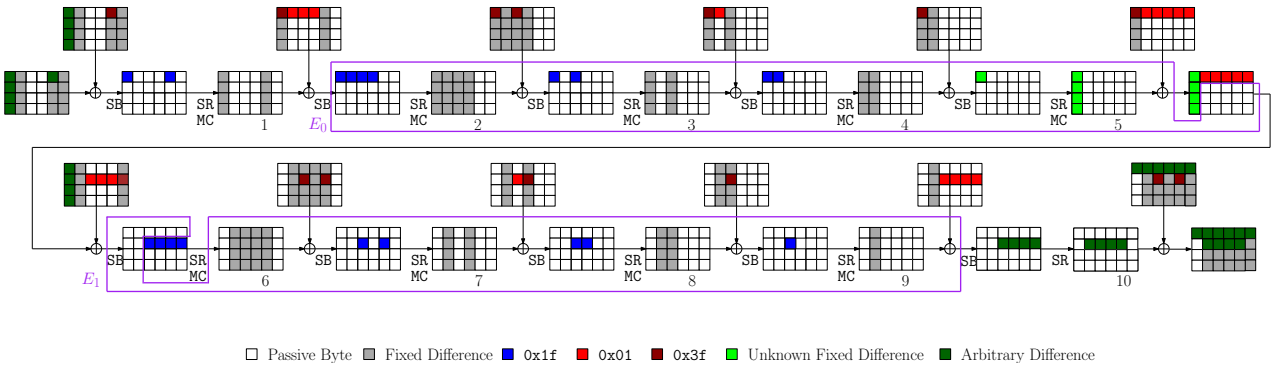


Figure 2.6: The related-key Rectangle attack on 10-Round Rijndael-192/192. Switch is applied in Round 6

?	c_1	c_2	c_3	?	c_4
?	c_5	c_6	c_7	c_8	c_9
?	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}
?	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}

and

?	$c_1 \oplus$ 3e	c_2	c_3	?	$c_4 \oplus$ 3e
?	$c_5 \oplus$ 1f	c_6	c_7	$c_8 \oplus$ 1f	$c_9 \oplus$ 1f
?	$c_{10} \oplus$ 1f	c_{11}	c_{12}	$c_{13} \oplus$ 1f	$c_{14} \oplus$ 1f
?	$c_{15} \oplus$ 21	c_{16}	c_{17}	$c_{18} \oplus$ 21	$c_{19} \oplus$ 21

Data Collection

1. Generate 2^{138} structures $G_i = \{U_i, V_i\}$ of 2^{41} plaintexts each, where $1 \leq i \leq 2^{138}$, U_i, V_i are the sets of 2^{40} plaintexts of the form respectively, $c_i's$ $(1 \leq i \leq 16)$ are fixed 8-bit values and '?' takes all possible values.

2. For each structure G_i

   (a) Ask for the encryption of U_i, V_i under K_a and K_b respectively to obtain $G_i^1 = \{X_i^1, Y_i^1\}$.

   (b) Ask for the encryption of V_i, U_i under K_a and K_b respectively to obtain $G_i^2 = \{X_i^2, Y_i^2\}$.

   (c) Ask for the encryption of U_i, V_i under K_c and K_d respectively to obtain $H_i^1 = \{Z_i^1, W_i^1\}$.

   (d) Ask for the encryption of V_i, U_i under K_c and K_d respectively to obtain $H_i^2 = \{Z_i^2, W_i^2\}$.

   (e) Let $T^{1a} = \{X_i^1\}_{1 \leq i \leq 2^{138}}$, $T^{1b} = \{Y_i^1\}_{1 \leq i \leq 2^{138}}$, $T^{2a} = \{X_i^2\}_{1 \leq i \leq 2^{138}}$, $T^{2b} = \{Y_i^2\}_{1 \leq i \leq 2^{138}}$, $T^{1c} = \{Z_i^1\}_{1 \leq i \leq 2^{138}}$, $T^{1d} = \{W_i^1\}_{1 \leq i \leq 2^{138}}$, $T^{2c} = \{Z_i^2\}_{1 \leq i \leq 2^{138}}$ and $T^{2d} = \{W_i^2\}_{1 \leq i \leq 2^{138}}$.

3. Initialize two vectors L^{ac} and L^{bd} consisting of 2^{112} lists L_η^{ac} and L_η^{bd} respectively, where η corresponds to a 14-byte value (i.e., the bytes (1,0), (1,5) and (i, j) of a ciphertext, where $2 \leq i \leq 3$, $0 \leq j \leq 5$), $L_\eta^{ac} = \{S_\eta^a, S_\eta^c, N_\eta^a, N_\eta^c\}$, $L_\eta^{bd} = \{S_\eta^b, S_\eta^d, N_\eta^b, N_\eta^d\}$, $S_\eta^a, S_\eta^b, S_\eta^c, S_\eta^d$ are the sets of ciphertexts under K_a, K_b, K_c and K_d respectively as well as their structure indices, and $N_\eta^a, N_\eta^b, N_\eta^c, N_\eta^d$ denote the cardinalities of the sets $S_\eta^a, S_\eta^b, S_\eta^c$ and S_η^d respectively.

4. For each ciphertext in T^{1a}, extract the 112-bit value η, then insert the ciphertext and its structure index into the set S_η^a of the corresponding list L_η^{ac} and increase N_η^a by 1. For each ciphertext in T^{1c}, xor it with

00	00	00	00	00	00
00	00	00	00	00	3e
00	1f	1f	1f	1f	1f
00	1f	1f	1f	1f	1f

and then extract the 112-bit value η. After that, insert the ciphertext and its structure index into the set S^c_η of the corresponding list L^{ac}_η and increase N^c_η by 1. Do similarly for the ciphertexts in T^{1b}, T^{1d} and update the lists L^{bd}_η.

5. Discard the lists L^{ac}_η in which N^a_η or N^c_η is 0, and remove the lists L^{bd}_η in which N^b_η or N^d_η is 0. For each remaining list L^{ac}_η, initialize 2^{48} lists $L^{ac}_{\eta,\theta} = \{S^a_{\eta,\theta}, S^c_{\eta,\theta}, N^a_{\eta,\theta}, N^c_{\eta,\theta}\}$ defined similarly to L^{ac}_η, where θ corresponds to a 6-byte value (i.e., the bytes $(0, j)$, $0 \leq j \leq 5$), then do the following:

   (a) For each ciphertext in S^a_η, extract the 48-bit value θ, then insert the ciphertext and its structure index into the set $S^a_{\eta,\theta}$ of the corresponding list $L^{ac}_{\eta,\theta}$ and increase $N^a_{\eta,\theta}$ by 1.

   (b) Let $\delta_1, \delta_2, \ldots, \delta_{127}$ denote all possible output differences of the S-Box for the input difference 01. For each ciphertext in S^c_η, xor it with

00	21	21	21	21	21
00	00	00	00	00	00
00	00	00	00	00	00
00	00	00	00	00	00

   and then extract the 48-bit value θ. After that, insert the ciphertext and its structure index into the set S^c_{η,θ_1}, S^c_{η,θ_2}, ..., $S^c_{\eta,\theta_{127}}$ of the lists L^{ac}_{η,θ_1}, L^{ac}_{η,θ_2}, ..., $L^{ac}_{\eta,\theta_{127}}$ and increase N^c_{η,θ_1}, N^c_{η,θ_2}, ..., $N^c_{\eta,\theta_{127}}$ by 1 respectively, where θ_1, ..., θ_{127} denote $\theta \oplus (\delta_1\|\delta_1\|\delta_1\|\delta_1\|\delta_1\|\delta_1)$, ..., $\theta \oplus (\delta_{127}\|\delta_{127}\|\delta_{127}\|\delta_{127}\|\delta_{127}\|\delta_{127})$ respectively.

   For each remaining list L^{bd}_η, do similarly to get the lists $L^{bd}_{\eta,\theta}$.

6. Keep the lists $L^{ac}_{\eta,\theta}$ in which both $N^a_{\eta,\theta}$ and $N^c_{\eta,\theta}$ are non-zero, and keep the lists $L^{bd}_{\eta,\theta}$ in which both $N^b_{\eta,\theta}$ and $N^d_{\eta,\theta}$ are non-zero. Then derive the ciphertext quartets (C_a, C_b, C_c, C_d) from the remaining lists $L^{ac}_{\eta,\theta}$ and $L^{bd}_{\eta,\theta}$ by using following criteria:

   (a) C_a, C_c are chosen from the same list $L^{ac}_{\eta,\theta}$, and C_b, C_d come from the same list $L^{bd}_{\eta',\theta'}$.

   (b) The structure indices of C_a and C_b are the same, and the structure indices of C_c and C_d are the same.

7. Repeat Steps 4−6 three times for $(T^{1a}, T^{1b}, T^{2c}, T^{2d})$, $(T^{2a}, T^{2b}, T^{1c}, T^{1d})$ and $(T^{2a}, T^{2b}, T^{2c}, T^{2d})$.

With the above procedure we obtain around

$$(2^{40} \cdot 2^{40} \cdot 2^{138})^2 \cdot (2^{-112})^2 \cdot (\frac{127}{2^{48}})^2 \cdot 4 \approx 2^{132}$$

ciphertext quartets (C_a, C_b, C_c, C_d) and their plaintext quartets (P_a, P_b, P_c, P_d).

Key Recovery

8. Let t_1, t'_1 denote $SB((K_a)_{2,5}) \oplus SB((K_a)_{2,5} \oplus 1\text{f}) \oplus 1\text{f}$ and $SB((K_c)_{2,5}) \oplus SB((K_c)_{2,5} \oplus 1\text{f}) \oplus 1\text{f}$ respectively. Guess the subkey bytes $(K_a)_{2,5}$, $(K_c)_{2,5}$ and calculate the values of t_1 and t'_1. Then for each quartet (P_a, P_b, P_c, P_d), check whether the equations

$$\begin{aligned} (P_a)_{1,0} \oplus (P_b)_{1,0} \oplus t_1 &= 0 \\ (P_c)_{1,0} \oplus (P_d)_{1,0} \oplus t'_1 &= 0 \end{aligned}$$

hold or not. If not, discard the quartet.

9. Let t_2, t'_2 denote $SB((K_a)_{3,5}) \oplus SB((K_a)_{3,5} \oplus 21) \oplus 1\text{f}$ and $SB((K_c)_{3,5}) \oplus SB((K_c)_{3,5} \oplus 21) \oplus 1\text{f}$ respectively. Guess the subkey bytes $(K_a)_{3,5}$, $(K_c)_{3,5}$ and compute the values of t_2 and t'_2. Then for each remaining quartet (P_a, P_b, P_c, P_d), test whether the equations

$$\begin{aligned} (P_a)_{2,0} \oplus (P_b)_{2,0} \oplus t_2 &= 0 \\ (P_c)_{2,0} \oplus (P_d)_{2,0} \oplus t'_2 &= 0 \end{aligned}$$

hold or not. If not, remove the quartet.

10. Let t_3, t'_3 denote $SB((K_a)_{0,5}) \oplus SB((K_a)_{0,5} \oplus 3\text{e}) \oplus 21$ and $SB((K_c)_{0,5}) \oplus SB((K_c)_{0,5} \oplus 3\text{e}) \oplus 21$ respectively. Guess the subkey bytes $(K_a)_{0,5}$, $(K_c)_{0,5}$ and derive the values of t_3 and t'_3. Then for each remaining quartet (P_a, P_b, P_c, P_d), verify whether the equations

$$\begin{aligned} (P_a)_{3,0} \oplus (P_b)_{3,0} \oplus t_3 &= 0 \\ (P_c)_{3,0} \oplus (P_d)_{3,0} \oplus t'_3 &= 0 \end{aligned}$$

hold or not. If not, discard the quartet.

11. Guess the subkey bytes $(K_a)_{0,4}$, $(K_c)_{0,4}$. Then for each remaining quartet (P_a, P_b, P_c, P_d), check whether the equations

$$\begin{aligned} SB((P_a)_{0,4} \oplus (K_a)_{0,4}) \oplus SB((P_b)_{0,4} \oplus (K_a)_{0,4} \oplus 3\text{f}) &= 1\text{f} \\ SB((P_c)_{0,4} \oplus (K_c)_{0,4}) \oplus SB((P_d)_{0,4} \oplus (K_c)_{0,4} \oplus 3\text{f}) &= 1\text{f} \end{aligned}$$

hold or not. If not, remove the quartet.

12. Guess the subkey bytes $(K_a)_{0,0}$, $(K_b)_{0,0}$, $(K_c)_{0,0}$, $(K_d)_{0,0}$. Then for each remaining quartet (P_a, P_b, P_c, P_d), test whether the equations

$$\begin{aligned} SB((P_a)_{0,0} \oplus (K_a)_{0,0}) \oplus SB((P_b)_{0,0} \oplus (K_b)_{0,0}) &= 1\text{f} \\ SB((P_c)_{0,0} \oplus (K_c)_{0,0}) \oplus SB((P_d)_{0,0} \oplus (K_d)_{0,0}) &= 1\text{f} \end{aligned}$$

hold or not. If not, remove the quartet.

13. Now we still have about $2^{132} \times (2^{-8})^{10} = 2^{52}$ plaintext quartets and their ciphertext quartets. Let t_4, t'_4 denote $SB((K^9_a)_{1,5}) \oplus SB((K^9_a)_{1,5} \oplus 01)$ and $SB((K^9_b)_{1,5}) \oplus SB((K^9_b)_{1,5} \oplus 01)$ respectively, then t_4, $t'_4 \in \{\delta_1, \ldots, \delta_{127}\}$. Guess the subkey bytes $(K^9_a)_{1,5}$, $(K^9_b)_{1,5}$ and

calculate the values of t_4 and t'_4. Then for each remaining quartet (C_a, C_b, C_c, C_d), verify whether the equations

$$\begin{aligned}(C_a)_{0,0} \oplus (C_c)_{0,0} \oplus t_4 &= 0\\ (C_b)_{0,0} \oplus (C_d)_{0,0} \oplus t'_4 &= 0\end{aligned}$$

hold or not. If not, remove the quartet. Otherwise, $(C_a)_{0,i} \oplus (C_c)_{0,i} = (\Delta K^{10}_{ac})_{0,i}$ and $(C_b)_{0,i} \oplus (C_d)_{0,i} = (\Delta K^{10}_{bd})_{0,i}$ hold for $1 \le i \le 5$ according to Steps 5-6. Note that $(C_a)_{0,0} \oplus (C_c)_{0,0}$, $(C_b)_{0,0} \oplus (C_d)_{0,0} \in \{\delta_1, \ldots, \delta_{127}\}$, thus the expected number of remaining quartets after this step is $2^{52} \cdot (2^{-7})^2 = 2^{38}$.

14. Guess the subkey bytes $(K^{10}_a)_{1,j}$, $(K^{10}_b)_{1,j}$ $(1 \le j \le 4)$ as follows:

    (a) guess $(K^{10}_a)_{1,1}$, $(K^{10}_b)_{1,1}$ and calculate the values of $(K^{10}_c)_{1,1}$, $(K^{10}_d)_{1,1}$ by using Table 2.2.

    (b) guess $(K^{10}_a)_{1,2}$, $(K^{10}_b)_{1,2}$ and derive the values of $(K^{10}_c)_{1,2}$, $(K^{10}_d)_{1,2}$ from Table 2.2.

    (c) guess $(K^{10}_a)_{1,3}$, $(K^{10}_b)_{1,3}$, $(K^{10}_a)_{1,4}$, $(K^{10}_b)_{1,4}$ and use Table 2.2 to obtain the values of $(K^{10}_c)_{1,3}$, $(K^{10}_d)_{1,3}$, $(K^{10}_c)_{1,4}$, $(K^{10}_d)_{1,4}$.

    For each of the remaining quartets in substeps (a)–(c), check whether the corresponding equations

$$\begin{aligned}SB^{-1}((C_a)_{1,j} \oplus (K^{10}_a)_{1,j}) \oplus SB^{-1}((C_c)_{1,j} \oplus (K^{10}_c)_{1,j}) &= \texttt{01}\\ SB^{-1}((C_b)_{1,j} \oplus (K^{10}_b)_{1,j}) \oplus SB^{-1}((C_d)_{1,j} \oplus (K^{10}_d)_{1,j}) &= \texttt{01}\end{aligned}$$

    are satisfied or not. If not, discard the quartet.

15. If the number of the remaining quartets after above steps is six or more, output the corresponding 22 guessed subkey bytes $(K_a)_{i,5}$, $(K_c)_{i,5}$, $(K_a)_{0,4}$, $(K_c)_{0,4}$, $(K_a)_{0,0}$, $(K_b)_{0,0}$, $(K_c)_{0,0}$, $(K_d)_{0,0}$, $(K^9_a)_{1,5}$, $(K^9_b)_{1,5}$, $(K^{10}_a)_{1,j}$ and $(K^{10}_b)_{1,j}$ $(i \in \{0, 2, 3\},\ 1 \le j \le 4)$ as the correct key information. Otherwise, return to Step 8 and repeat the procedure.

16. If the above 22 subkey bytes are retrieved after Step 15, perform an exhaustive search over all possible values of the remaining 152 bits of K_a so as to recover the secret key.

Analysis of the attack on 10-Round Rijndael-192/192

The expected number of remaining quartets after each substep in Step 14 is given as (a) $2^{38} \cdot (2^{-8})^2 = 2^{22}$, (b) $2^{22} \cdot (2^{-8})^2 = 2^6$ and (c) $2^6 \cdot (2^{-8})^4 = 2^{-26}$, respectively. Thus for a wrong guess of the above 22 subkey bytes, the expected number of quartets after Step 14 is 2^{-26}. On the other hand, for a right guess of the key, the expected number of right quartets is about

$$(2^{40} \cdot 2^{40} \cdot 2^{138})^2 \cdot 4 \cdot (2^{-40})^2 \cdot 2^{-355} = 2^3.$$

The probability of outputting a wrong key guess in Step 15 is derived by the following Poisson distribution:

$$X \sim Poi(\lambda = 2^{-26}).$$

As $\Pr[X \ge 6] \approx 2^{-165.49}$, the expected number of wrong key guesses suggested in Step 15 is about $(2^8)^{22} \cdot 2^{-165.49} = 2^{10.51}$, and the wrong key information can be removed in Step 16. Similarly,

the probability that six or more quartets remain after Step 14 for the correct key guess is also computed by the Poisson distribution:

$$X \sim Poi(\lambda = 8).$$

Since $\Pr[X \geq 6] \approx 0.81$, the success probability of the attack on 10-round Rijndael-192 is approximately 81%.
The data complexity of this attack is $2^{138} \cdot 2^{41} = 2^{179}$ chosen plaintexts which are encrypted under K_a, K_b, K_c and K_d respectively (resulting in $2^{179} \cdot 4 = 2^{181}$ ciphertexts). The memory complexity is primarily owing to keeping T^{1a}, T^{1b}, T^{1c}, T^{1d}, T^{2a}, T^{2b}, T^{2c} and T^{2d}, thus it can be estimated as $8 \cdot 2^{178} \cdot 24 \approx 2^{185.59}$ bytes. The time complexity of the attack can be derived as follows:

- During the *data collection* phase, the time complexity is mainly dominated by Step 2, which is $2^{179} \cdot 4 = 2^{181}$ 10-round Rijndael-192 encryptions.
- For the *key recovery* phase, the time complexity is mainly dominated by Step 14, which can be measured as $2^{176} \cdot 2^6 \cdot \frac{8}{24 \cdot 10} \approx 2^{177.09}$ 10-round Rijndael-192 encryptions.

As a result, the total time complexity of the attack is approximately $2^{181.09}$ 10-round Rijndael-192 encryptions.

2.3 Subkey Recovery on CAESAR candidate iFeed

2.3.1 Introduction

iFeed is one of first round CAESAR submissions, proposed by Zhang et al. [118]. It is an AES block cipher-based design which combines PMAC-style authentication with dedicated encryption. iFeed processes the data in an on-line manner and it is inverse-free, meaning that both encryption and decryption only use the block cipher in forward direction. The design is inherently nonce-based: the authors claim *and* prove confidentiality and authenticity of iFeed in a setting where the adversary is not allowed to make repeated queries under the same nonce. We refer to this setting and type of adversary as *nonce-respecting*. The design is moreover claimed to achieve confidentiality under some conditions also in the *nonce-reuse* setting.
Chakraborti et al. [30] recently presented forgeries on iFeed both in the nonce-reuse setting, and in a setting where the adversary is granted access to ciphertext decryptions irrespective of the verification result, also known as the *release of unverified plaintext* (RUP) setting from [2]. The iFeed designers, however, do not claim any security against these properties, and the attacks from Chakraborti et al. do not invalidate the security of iFeed. The work presented in this section exploits the unfortunate repetition of subkeys at the finalization of the associated data and plaintext. This leads to a total security break and also invalidates the security claims and proofs posited by the designers of iFeed. The attack uses only one encryption query with no associated data and an arbitrary n-bit (or single block) plaintext. The attack also allows to recover the values of two subkeys, which then can be used to recover plaintext from old encryptions, even though they were performed under a different nonce. The results of this section were published at SAC 2015 [105].

2.3.2 Description of iFeed

iFeed uses the secret key K to derive two subkeys: a nonce-independent $Z_0 = E_K(0^{128})$ and its multiples $Z_i = 2^i \cdot Z_0$, and a nonce-dependent $U = E_K(PMN \| 10^*)$ where PMN is a variable

Algorithm 1 iFeed $\mathcal{E}$ for $|PMN| = 127$, $|A| = 0$, and $|P_1| = n$

$Z_0 = E_K(0^{128})$
for $i = 1, \ldots, 3$ **do**
    $Z_i = 2 \cdot Z_{i-1}$
$U = E_K(PMN \| 1)$
$T_A = 0^{128}$
$C_1 = E_K(Z_3 \oplus U) \oplus P_1$
$C_2 = E_K(P_1 \oplus Z_2 \oplus U)$
return $(C, T) = (C_1, T_A \oplus C_2)$

public message number (nonce) and 10^* is the padding to a full 128-bit block with one 1 bit and appropriate number of 0 bits. The processing of the associated data and of the message are done independently of each other, both resulting in a subtag. The XOR of the two subtags produces the final tag. The general encryption and decryption procedures of iFeed are given in Figures 2.7 and 2.8, respectively. The processing of the associated data is distinctively independent of the nonce: the inputs to the block cipher E are masked only using Z_i. The data is encrypted using both the Z_i and U. One design choice of iFeed is that the computation of the subtags is performed using the same subkeys: Z_1 or Z_2 for the associated data, and $Z_1 \oplus U$ and $Z_2 \oplus U$ for the plaintext.

The attacks we will describe below, are generic and do not exploit any weakness of AES-128. Therefore, from now we simply describe iFeed based on any block cipher $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, where $n = 128$. iFeed is on-line, it only uses E in forward direction (inverse-free), and it allows for parallelized encryption.

The scheme is keyed via a key $K \in \{0,1\}^n$. It operates on public message numbers PMN of size between 1 and 127 bits, associated data A in $\{0,1\}^{\leq |A|_{\max}}$, and plaintexts/ciphertexts P/C from $\{0,1\}^{\leq |P|_{\max}}$, where $|A|_{\max}$ and $|P|_{\max}$ are some large values which sum to at most $(2^{71} - 512)$. The tag T is of size $32 \leq \tau \leq n = 128$. In the nonce-respecting setting, the public message number is required to be unique for every query to the iFeed encryption function $\mathcal{E}$. The iFeed encryption $\mathcal{E}$ and decryption $\mathcal{D}$ functions are depicted in Figure 2.7 and 2.8, respectively. Here, $\mathrm{Pad}(X)$ equals X if $|X| = n$ and $X \| 10^{n-1-|X|}$ if $|X| < n$, and the usage of Z_1 versus Z_2 (resp. Z'_1 versus Z'_2) depends on the last block of A (resp. P): Z_2/Z'_2 is used if the last data block is of size n bits, Z_1/Z'_1 is used if the last block is fractional. We remark that our attacks use integral blocks only, for which $\mathrm{Pad}(X) = X$ and Z_2 is used instead of Z_1.

For the presentation of the attacks, however, it suffices to only discuss a simplified version of iFeed. In more detail, in our attacks we will only query $\mathcal{E}$ on input of n-bit plaintext and no associated data. We also make the forgery queries to $\mathcal{D}$ for either n or $2n$-bit associated data and $2n$-bit ciphertext. All queries consider 127-bit PMN and tag size $\tau = n = 128$. In Algorithms 1 and 2, we give a formal description of iFeed's $\mathcal{E}$ and $\mathcal{D}$, respectively, for the input sizes relevant for our attacks. We refer to [118] for the general description of iFeed, and stress that our attacks easily translate to the general case.

The iFeed mode makes use of secret subkey $Z_0 = E_K(0^{128})$ which is used to derive additional subkeys $Z_i = 2 \cdot Z_{i-1}$. Here, the multiplication is performed in the binary Galois Field $\mathrm{GF}(2^{128})$ defined by the primitive polynomial $x^{128} + x^7 + x^2 + x + 1$.

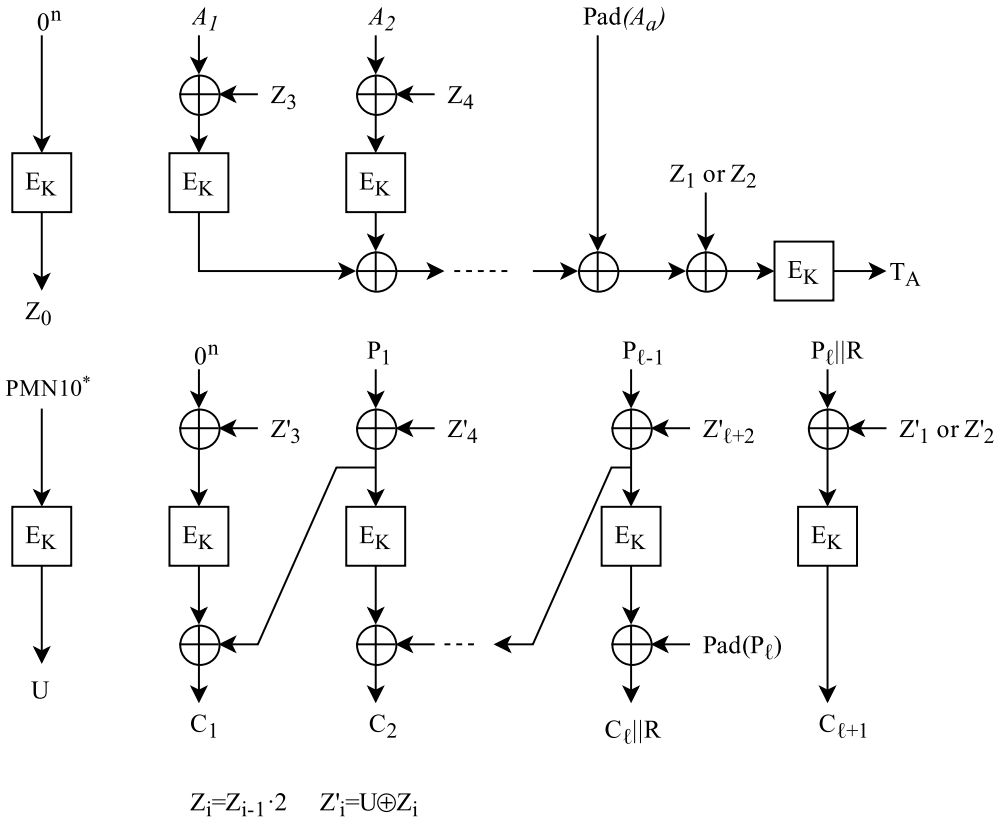


Figure 2.7: iFeed encryption $\mathcal{E}$. All wires represent n-bit values. The output is the ciphertext $C_1 \cdots C_\ell$ and the tag $T = \text{left}_\tau(T_A \oplus C_{\ell+1})$

2.3.3 Forgery and Subkey Recovery Attack on iFeed

Let $K \xleftarrow{\$} \{0,1\}^n$ be the secret key and consider $\tau = n$. We present our forgery attack on iFeed. It consists of one encryption query and the forgery itself. Upon successful verification, the forgery will disclose the subkeys $Z_0 = E_K(0^{128})$ and $U = E_K(PMN\|1)$.

- Fix arbitrary $PMN \in \{0,1\}^{127}$ and arbitrary $P_1 \in \{0,1\}^{128}$, and make **encryption query** with no associated data $A = \varepsilon$:

$$(C_1, T) = \mathcal{E}_K(PMN, \varepsilon, P_1)\,;$$

- Write $C_1' = C_1 \oplus P_1 \oplus PMN\|1$, and fix an arbitrary $C_2' \in \{0,1\}^{128}$. Set $A = C_2'$, $T' = 0^{128}$, and output **forgery**:

$$\mathcal{D}_K(PMN, A, C_1'C_2', T')\,.$$

We next demonstrate that the forgery attempt is successful. First, regarding the encryption query, note that

$$C_1 = E_K(Z_3 \oplus U) \oplus P_1\,. \tag{2.3}$$

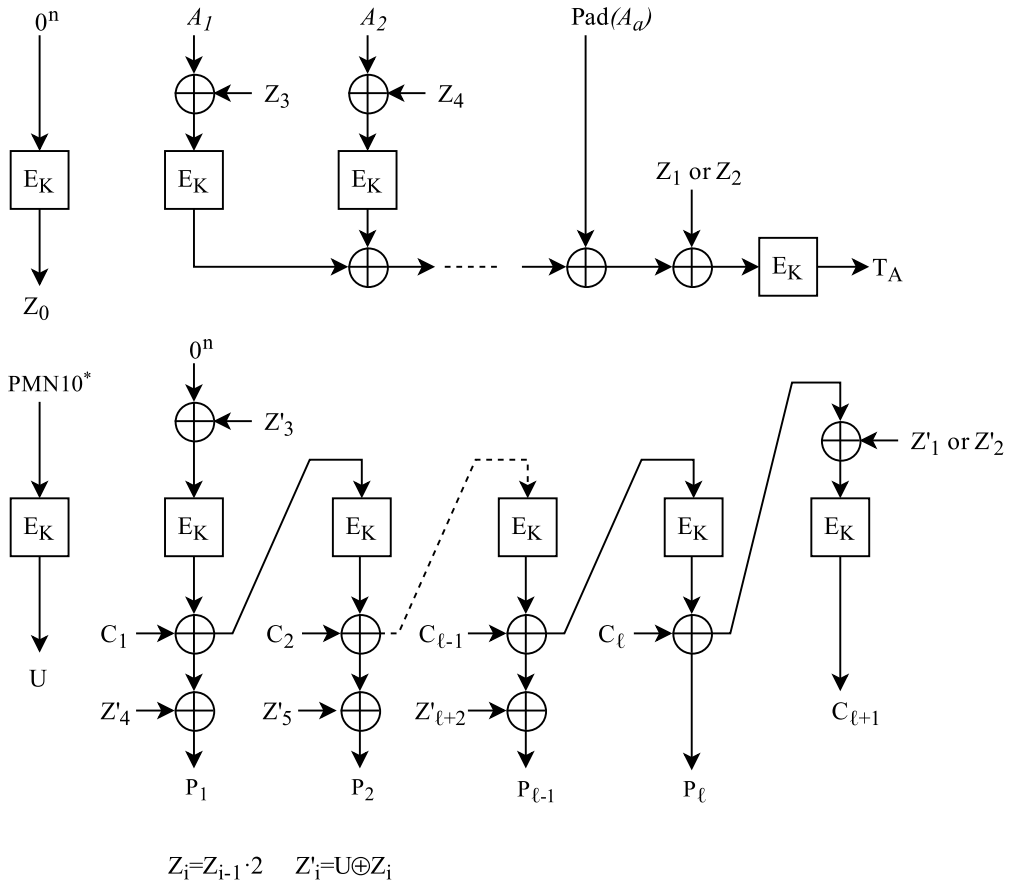


Figure 2.8: iFeed decryption $\mathcal{D}$. All wires represent n-bit values. The output is the plaintext $P_1 \cdots P_\ell$ when T is $\text{left}_\tau(T_A \oplus C_{\ell+1})$

Now, verification of the forgery (cf. Algorithm 2) succeeds if $T' = T'_A \oplus C'_3 = 0^{128}$. Note that we have

$$\begin{aligned} P'_1 &= E_K(Z_3 \oplus U) \oplus C'_1 \oplus Z_4 \oplus U \\ &= PMN\|1 \oplus Z_4 \oplus U\,, \\ P'_2 &= E_K(P'_1 \oplus Z_4 \oplus U) \oplus C'_2 \\ &= E_K(PMN\|1) \oplus C'_2 = U \oplus C'_2\,, \\ C'_3 &= E_K(P'_2 \oplus Z_2 \oplus U) \\ &= E_K(C'_2 \oplus Z_2)\,. \end{aligned}$$

As we defined $A = C'_2$, this yields successful verification:

$$T'_A = E_K(A \oplus Z_2) = E_K(C'_2 \oplus Z_2) = C'_3\,.$$

As verification is successful, $\mathcal{D}$ returns $P'_1 P'_2$. The values $U = P'_2 \oplus C'_2$ and $Z_0 = 2^{-4}(P'_1 \oplus PMN\|1 \oplus U)$ are directly obtained.

2.3.4 Finding $E_K(P^*)$ for Any Plaintext P^*

Once the subkeys U, Z_0 are known, by performing the main forgery attack described above, one is able to learn $E_K(P^*)$ for any plaintext data $P^* \in \{0,1\}^n$ and to perform additional forgeries.

Algorithm 2 iFeed $\mathcal{D}$ for $|PMN| = 127$, $|A'| = n$ or $2n$, and $|C'| = 2n$

$Z_0 = E_K(0^{128})$
for $i = 1, \ldots, 4$ **do**
    $Z_i = 2 \cdot Z_{i-1}$
$U = E_K(PMN\|1)$
if $|A'| = n$ **then**
    define $A'_1 = A'$
    $T'_A = E_K(A'_1 \oplus Z_2)$
else
    parse $A'_1 A'_2 = A'$
    $T'_A = E_K(E_K(A'_1 \oplus Z_3) \oplus A'_2 \oplus Z_2)$
parse $C'_1 C'_2 = C'$
$P'_1 = E_K(Z_3 \oplus U) \oplus C'_1 \oplus Z_4 \oplus U$
$P'_2 = E_K(P'_1 \oplus Z_4 \oplus U) \oplus C'_2$
$C'_3 = E_K(P'_2 \oplus Z_2 \oplus U)$
if $T' = T'_A \oplus C'_3$ **then**
    return $P' = P'_1 P'_2$
else
    return $\perp$

- Let PMN be as before, define $(A''_1, A''_2) = (P^* \oplus Z_3, U)$, $(C''_1, C''_2) = (E_K(Z_3 \oplus U) \oplus P^*, 0^{128})$, and $T = 0^{128}$, and output **forgery**:

$$\mathcal{D}_K(PMN, A''_1 A''_2, C''_1 C''_2, T'') \,.$$

The verification (cf. Algorithm 2) is successful if $T'' = T''_A \oplus C''_3 = 0^{128}$. Note that we have

$$\begin{aligned} P''_1 &= E_K(Z_3 \oplus U) \oplus C''_1 \oplus Z_4 \oplus U \\ &= P^* \oplus Z_4 \oplus U \,, \\ P''_2 &= E_K(P''_1 \oplus Z_4 \oplus U) \oplus C''_2 \\ &= E_K(P^*) \,, \\ C''_3 &= E_K(P''_2 \oplus Z_2 \oplus U) \\ &= E_K(E_K(P^*) \oplus Z_2 \oplus U) \,. \end{aligned}$$

On the other hand, T''_A is computed from the two-block $(A''_1 A''_2)$ as follows

$$\begin{aligned} T''_A &= E_K(E_K(A''_1 \oplus Z_3) \oplus A''_2 \oplus Z_2) \\ &= E_K(E_K(P^*) \oplus Z_2 \oplus U) \,, \end{aligned}$$

thus $C''_3 = T''_A$, and the verification is successful. The resulting plaintext satisfies $P''_2 = E_K(P^*)$. This attack works for any n-bit data block P^*, it can for instance be used to recover the subkeys of older iFeed encryptions (by putting $P^* = PMN'\|10^* \neq PMN\|1$), and indirectly to decrypt earlier encryptions without having possession of the key K.

2.3.5 Discussion

The attack of Section 2.3.3 is possible due to two main iFeed properties:

1. iFeed uses Z_2 as masking in both the T'_A and C'_{l+1};
2. The second last ciphertext block (C_1 in our encryption example) is not masked with $Z_4 \oplus U$ as other ciphertext blocks.

Using these properties, the forgery is constructed in such a way upon decryption, $PMN\|1$ is directly fed into E_K and the term U in the final mask $Z_2 \oplus U$ is canceled out.
At a high level, the flaw is caused by an oversight that the subkeys for the associated data and the plaintext are dependent. Indeed, the associated data is masked with $Z_1, \ldots, Z_{a+2}$ and the plaintext with $Z_1 \oplus U, \ldots, Z_{\ell+2} \oplus U$ (where a and ℓ denote the number of associated data and plaintext blocks). For the decryption query, the proof claims that both T_A and $C_{\ell+1}$ is randomly generated except with a small probability.[1] These two cases independently of each other rely on the randomness of Z_0. In our attacks, T_A and $C_{\ell+1}$ are, indeed, both newly and randomly generated. However, their drawing is not independent, in fact, they satisfy $T_A = C_{\ell+1}$.
This security analysis clearly illustrates the serious security problems that could be caused by relations that hold between secret values used in the cryptographic algorithm, while implicitly the security claims rely on the (wrong) assumption that these are independent and random.

2.4 Related-Tweakey Differential Attack on MANTIS-5

2.4.1 Introduction

Related Keys and Tweakable Block Ciphers

Tweakable block ciphers (TBCs) [76] generalize block ciphers by adding an additional public input, the tweak. This tweak serves as an additional "diversifier" and thus plays a role similar to the nonces of authenticated encryption (AEAD) schemes: Each value of the tweak defines a different family of permutations, i.e., a different block cipher. Recent advances in the design of authenticated ciphers have shown that this additional input makes TBCs a very useful building block for AEAD schemes with high performance and simpler, more elegant proofs.
Several generic constructions of TBCs from block ciphers have been proposed, but unfortunately those are usually either not very efficient (e.g., needing multiple block cipher calls) or provide only birthday-bound security. Practical ad-hoc constructions try to achieve the best of both (efficiency and security) by tightly integrating the tweak with the block cipher operations.
The TWEAKEY framework [60] is one of the most widely used ad-hoc constructions and incorporates the tweak as part of the key schedule. Since the tweak input is considered under full control of the adversary, *choosing related tweaks creates a cryptanalytic setting closely tied to related keys*.
In this section, we investigate the security implications of these "related tweakeys". We show that the classical metrics for security against differential cryptanalysis are not sufficient. We propose a class of differential characteristics that targets the "related tweakeys" by combining advantages of both classical differential characteristics (e.g., profiting from suboptimal S-box properties) and truncated differential characteristics (e.g., clustering to improve the differential probability and decrease the data complexity).

[1] In fact, it is claimed that P_{w+1} is random, where w is the first block in the forgery that is different from the older encryption query with the same nonce, and that all subsequent values $P_{w+2}, \ldots, P_\ell, C_{\ell+1}$ are random.

Application to MANTIS

MANTIS is a tweakable block cipher published at CRYPTO 2016 by Beierle et al. [8]. The designers' goal is to optimize this versatile building block for low-latency implementations. To this end, they use the same α-reflective structure as PRINCE by Borghoff et al. [25], but combine it with the round function of Midori by Banik et al. [6]. According to their analysis [8], this improves both the latency and the security compared to the original PRINCE, since Midori's variant of ShiftRows leads to a higher bound on the minimum number of active S-boxes. The tweak is incorporated using an adapted version of the TWEAKEY framework by Jean et al. [60]. The full version MANTIS_7 has 14 rounds, but the authors also give a reduced security claim for the 10-round version, MANTIS_5. They claim security against practical attacks, which they define as related-tweak attacks with data complexity 2^d less than 2^{30} chosen plaintexts (or 2^{40} known plaintexts), and computational complexity at most 2^{126-d} block cipher calls, similar to the PRINCE challenge. We present a key-recovery attack against MANTIS_5 with 2^{28} chosen plaintexts and a computational complexity of about 2^{38} block cipher calls, which violates this claim.
Our attack exploits the lightweight near-MDS mixing layer and certain differential properties of the involutive S-box, both inherited from Midori. These properties make it relatively easy to find a differential characteristic with the claimed optimal probability in the related-tweak setting. Using the same properties, this differential characteristic can then be expanded to a family of characteristics with a corresponding initial structure that makes efficient use of the low data complexity limit of only 2^{30} chosen plaintexts. Furthermore, the choice to keep the original Midori order of linear operations (first permute, then mix) makes the PRINCE-like middle rounds differentially less effective than the ordering used by PRINCE (first mix, then permute). Midori's order preserves a Superbox structure over 4 S-box layers in the middle rounds, instead of 2.
We verified the validity of the attack in a practical implementation. The implementation revealed an additional differential property of the Midori S-box that complicates some steps of the attack due to differentially equivalent keys. In particular, we found that slightly increasing either the memory requirements or the data complexity (still respecting the data limit) significantly increases the robustness of the attack. An adapted version of the attack recovers the full key in about 1 core hour using about 2^{30} chosen plaintexts.
The results of this section were published at FSE 2017 [34].

2.4.2 Description of MANTIS

The Tweakable Block Cipher

MANTIS is a tweakable block cipher published at CRYPTO 2016 by Beierle et al. [8]. The designers propose several variants MANTIS_r that differ only in the number of rounds. All variants operate on a 64-bit message block $M = M_0 \| M_1 \| \cdots \| M_{15}$ and work with a 64-bit tweak $T = T_0 \| T_1 \| \cdots \| T_{15}$ and $(64 + 64)$-bit key $K = (k_0, k_1)$. All 64-bit values are mapped to 4×4 states S of 4-bit cells S_j:

$$S = \begin{array}{|c|c|c|c|} \hline S_0 & S_1 & S_2 & S_3 \\ \hline S_4 & S_5 & S_6 & S_7 \\ \hline S_8 & S_9 & S_{10} & S_{11} \\ \hline S_{12} & S_{13} & S_{14} & S_{15} \\ \hline \end{array} .$$

The cipher's structure is similar to PRINCE, with r forward rounds $\mathcal{R}_i$ and r backward rounds $\mathcal{R}_{2r+1-i} = \mathcal{R}_i^{-1}$, separated by an involutive, unkeyed middle layer $\mathsf{S} \circ \mathsf{M} \circ \mathsf{S}$. The 64-bit subkey k_1

is used as round key for the outer forward and backward rounds, while the other 64-bit subkey k_0 and the derived $k_0' = (k_0 \ggg 1) + (k_0 \gg 63)$ serve as whitening keys. The tweak T is added together with k_1 in every round according to the TWEAKEY construction, with a simple cell permutation h as a tweak schedule. The construction is illustrated in Figure 2.9.

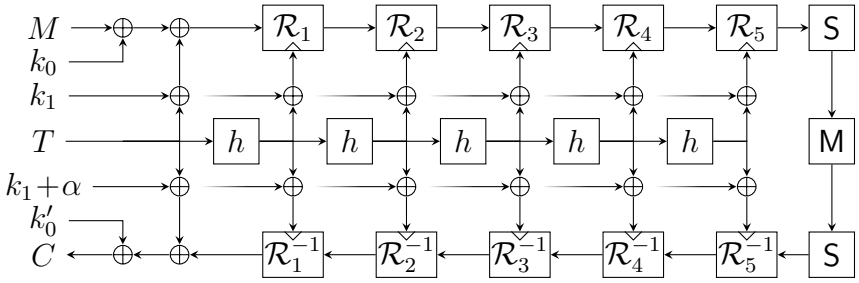


Figure 2.9: PRINCE-like structure of MANTIS_r, illustrated for MANTIS_5.

The Round Functions $\mathcal{R}_i$ and $\mathcal{R}_i^{-1}$

The round function $\mathcal{R}_i$ is very closely related to that of Midori [6]. It updates the 4×4 state of 4-bit cells by means of the sequences of transformations

$$\mathcal{R}_i = \mathsf{MixColumns} \circ \mathsf{PermuteCells} \circ \mathsf{AddTweakey}_i \circ \mathsf{AddConstant}_i \circ \mathsf{SubCells},$$
$$\mathcal{R}_i^{-1} = \mathsf{SubCells} \circ \mathsf{AddConstant}_i \circ \mathsf{AddTweakey}_i \circ \mathsf{PermuteCells}^{-1} \circ \mathsf{MixColumns},$$

as illustrated in Figure 2.10. In the following, we briefly describe the individual operations. For a more detailed description of the MANTIS family, we refer to the design paper [8].

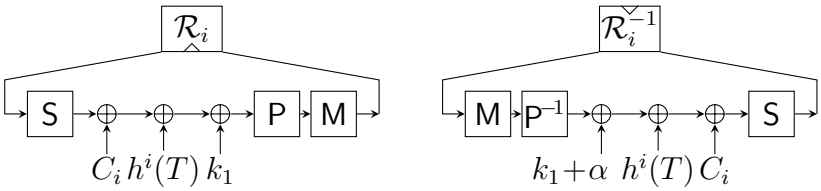


Figure 2.10: The MANTIS round functions $\mathcal{R}_i$ and $\mathcal{R}_i^{-1}$.

SubCells (S). The involutive 4-bit S-box $\mathcal{S}$ given below is applied to each cell of the state. For our attack, we are primarily interested in the differential behaviour of $\mathcal{S}$, which is illustrated below.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

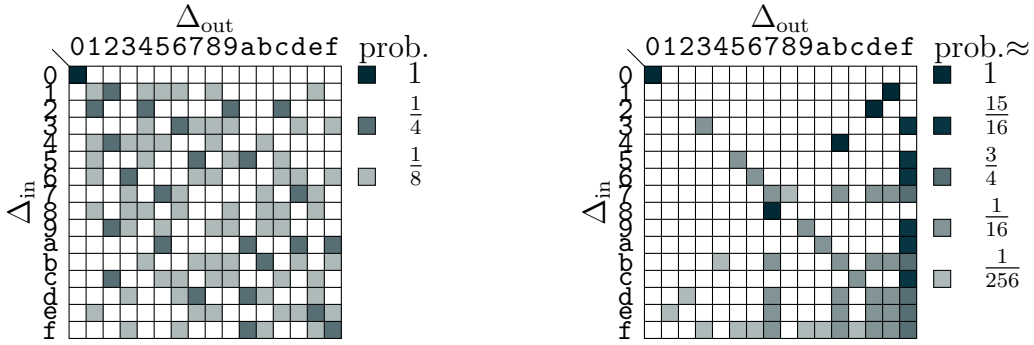
AddTweakey$_i$ (A) and AddConstant$_i$ (C). Several round-dependent values are added to the state: The round constant C_i, the subkey k_1 (for $\mathcal{R}_i$) or $k_1 + \alpha$ (for $\mathcal{R}_i^{-1}$), and the round tweakey $h^i(T)$. The tweakey update function h simply permutes the order of cells using the permutation h, specified in Figure 2.11a.

PermuteCells (P). The cells of the state are permuted by P, specified in Figure 2.11b.

(a) The tweak update function h.

(b) The state transformation P.

Figure 2.11: The MANTIS permutations h and P.



(a) DDT of SubCells.

(b) Truncated DDT of MixColumns.

Figure 2.12: Differential distribution tables (DDT) of the MANTIS round operations.

MixColumns (M). Each column of the state is multiplied with the following involutive near-MDS matrix M over GF(2^4), whose truncated differential behaviour per column is illustrated below:

$$\mathsf{M} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

2.4.3 Differential Characteristic

Bounds and Security Claim

The designers of MANTIS analyze the security of the cipher against differential cryptanalysis by modelling the differential behaviour (truncated to state cells) as a mixed-integer linear program [8]. They analyzed the minimum number of active S-boxes for different round numbers, both in a fixed-tweak and a related-tweak setting. The design document provides lower bounds for full and round-reduced MANTIS.

For MANTIS_5, the minimum number of active S-boxes in the related-tweak setting is 34 (for the full MANTIS_7: 50), and the maximum differential probability of the S-box is 2^{-2}. The designers conclude that "no related tweak linear or differential distinguisher based on a characteristics is possible for MANTIS_5" [8]. In particular, they claim that MANTIS_5 is secure against "practical attacks", here defined as related-tweak attacks with data complexity 2^d at most 2^{30} chosen plaintexts (or 2^{40} known plaintexts), and computational complexity at most 2^{126-d}.

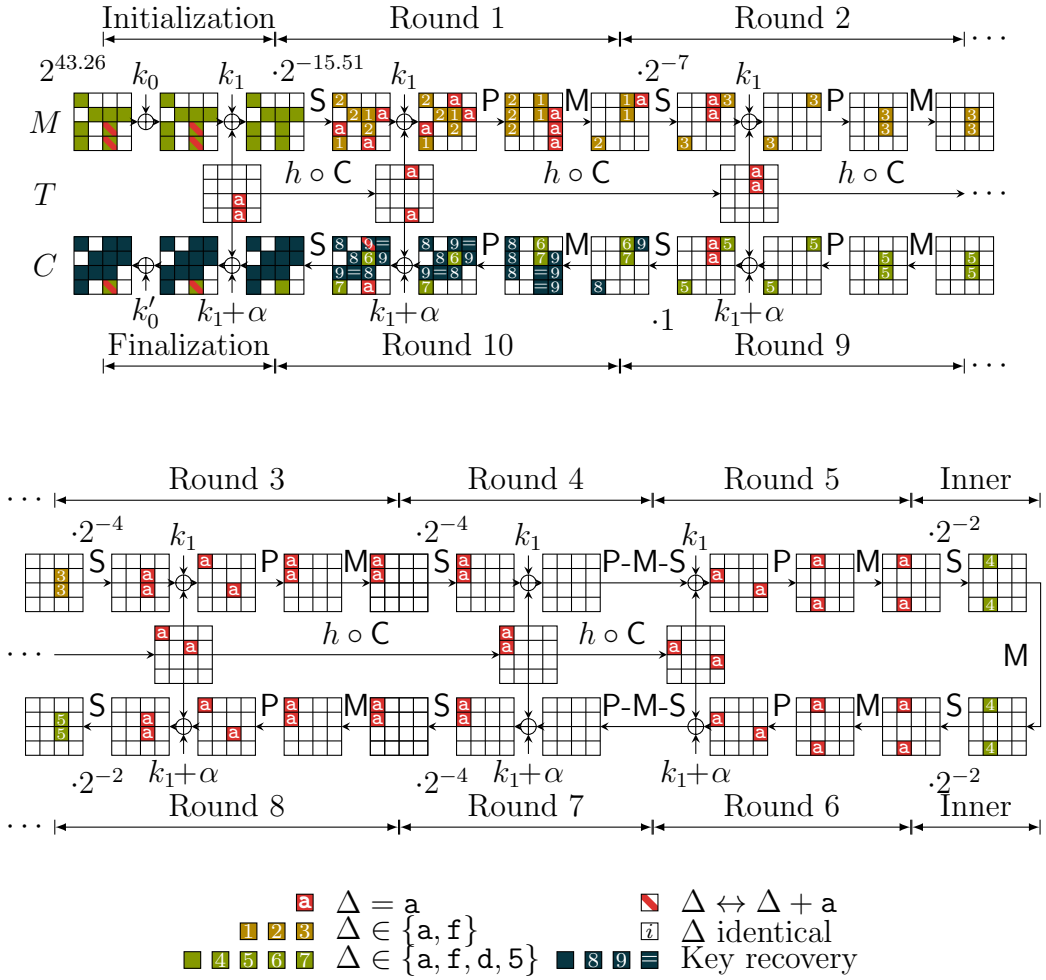


Figure 2.13: Family of differential characteristics for MANTIS_5.

A Family of Differential Characteristics

Our attack is based on a truncated differential characteristic for the related-tweak setting that meets this lower bound of 34 active S-boxes. The truncated characteristic is illustrated in Figure 2.13. Instead of considering only a single fixed input difference and differential characteristic, we will cluster several related differential characteristics following the same truncated differential characteristic, thus obtaining a much better probability.

An Optimal Differential Characteristic. To analyze the probability, we first construct a differential characteristic that matches the claimed optimal differential probability of $2^{-34 \cdot 2} = 2^{-68}$. Consider the differential distribution table of SubCells, given in Figure 2.12a. Observe that SubCells is an involution, so the table is symmetric. There is one input/output difference, a, such that all transitions from or to difference a have the maximum probability of $\frac{1}{4}$. Furthermore, these possible transitions include $\mathsf{a} \mapsto \mathsf{a}$. Since MixColumns only has binary coefficients, all transitions that match the branch number of 4 for MixColumns ($1 \to 3$, $2 \to 2$, $3 \to 1$) are valid when all active cells have a fixed difference of a.
Since all non-trivial MixColumns transitions of the truncated differential characteristic in Figure 2.13 match its branch number, setting all active cells to a results in a valid differential characteristic with the claimed optimal probability of $2^{-34 \cdot 2} = 2^{-68}$.

Clustering Differential Characteristics. We will now relax some of these constraints, and also consider characteristics with cell differences other than a in selected sections of the characteristic. Interesting candidates include all differences that can be mapped from and to a by SubCells, that is, $\{5, a, d, f\}$.

Rounds 9 and 2. First, consider Round 9. The SubCells layer at the end of Round 8 has 2 active S-boxes, at positions S_6 and S_{10}. Assume we allow all possible output differences $\{5, a, d, f\}$ for the two S-boxes, marked 5 in Figure 2.13. Then, the characteristic will follow the same truncated differential, with the same probability of $2^{-4 \cdot 2}$ to transition to the all-a state at the end of Round 9, as long as both S-boxes map to the same difference. The probability for this is 2^{-2}, instead of the original 2^{-4} of the all-a differential characteristic.
A similar observation applies for the two S-boxes S_3 and S_{12} of Round 2, marked 3 in Figure 2.13. However, as we want to relax also the input differences to Round 2, we will consider only output differences $\{a, f\}$. These have the additional advantage of allowing transitions with probability 2^{-2} not only to a, but each to both a and f, so this relaxation can be used in multiple consecutive rounds. The probability for Round 2 improves from 2^{-8} to $2^{-2 \cdot 2} \cdot 2^{-1} \cdot 2^{-2} = 2^{-7}$.

Inner Part. Second, consider the inner part. Similar as for Round 9, we can allow all 4 output differences for the first SubCells operation of the inner part, as long as both S-boxes map to the same difference, marked 4 in Figure 2.13. This seems to improve the probability for the inner part from $2^{-4} \cdot 2^{-4}$ to $2^{-2} \cdot 2^{-4}$. However, note that there is no tweakey addition between the two SubCells layers of the inner part, so the probabilities for the S-box transitions are certainly not independent. Since there is also no PermuteCells operation, we can simply compute the exact Superbox transition probability for the entire second column of the state. This reveals that the probability for the inner part is in fact 2^{-4}.

Initialization and Round 1. Like Round 2, we relax some of the differences of Round 1 to $\{a, f\}$. The estimated probability for Round 2 will remain valid for the output cells of Round 1 (1, 2, a). Again, MixColumns adds several constraints for the output differences of the SubCells layer of Round 1.
Finally, we relax the input differences. In addition to $\{a, f\}$, we also allow $\{5, d\}$ in order to generate more message pairs, while retaining a reasonable differential probability. For message cells S_{10} and S_{14}, marked in Figure 2.13, we need to compensate the AddTweakey operation of the initialization part by considering input differences Δ such that $\Delta + a \in \{a, f, 5, d\}$, or equivalently, $\Delta \in \{0, 5, f, 7\}$. The probability for the SubCells layer of Round 1, assuming uniformly distributed input differences, is then

$$\underbrace{2^{-3 \cdot 2}}_{\blacksquare \to a,\ \blacksquare \to a,\ \blacksquare \to a} \cdot \underbrace{\left(\frac{1}{4} \cdot 2^{-3} + \frac{3}{4} \cdot 2^{-4}\right)}_{\blacksquare, \blacksquare \to 1, 1} \cdot \underbrace{\left(\frac{1}{8} \cdot 2^{-5} + \frac{7}{8} \cdot 2^{-6}\right)}_{\blacksquare, \blacksquare, \blacksquare \to 2, 2, 2} \approx 2^{-15.51}.$$

Consequently, the overall probability of the family of characteristics up to Round 9 (or more precisely, up to AddTweakey of Round 10) is at least about

$$2^{-15.51-7-4-4-2-2-4-2} = 2^{-40.51}.$$

Round 10. If a pair followed the family of characteristics up to Round 9, the output of the AddTweakey operation of Round 10 will have several properties that can be used as a filter for key recovery:

(a) Differences $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ (■). (b) Differences $\{\mathtt{0}, \mathtt{5}, \mathtt{7}, \mathtt{f}\}$ (■).

Figure 2.14: Initial structure with $8 \cdot 4$ pairs from $2 \cdot 8$ queries per cell.

- Cells $S_1, S_4, S_{11}, S_{13}, S_{15}$ have zero difference, which will also be immediately visible in the ciphertexts (though not useful for key recovery).
- Cell S_{14} (marked a) has difference $\mathtt{a}$ (2-bit filter).
- Cells S_0, S_5, S_{10} (marked 8) will have the same difference (8-bit filter), as will cells S_2, S_7, S_8 (marked 9) after compensating for the tweak difference (8-bit filter).
- Cells S_6 and S_{12} (marked 6, 7) will have differences $\{\mathtt{a}, \mathtt{f}, \mathtt{5}, \mathtt{d}\}$, and additionally, due to the properties of MixColumns, cells S_3 and S_9 (marked ■) will have the same difference, which is the sum of the differences of S_6, S_{12} (12-bit filter).

Overall, the family of characteristics provides a 30-bit filter with probability $2^{-40.51}$.

Initial Structure

We now want to generate enough message pairs to expect at least one valid pair, while staying well below the data complexity limit of 2^{30} chosen plaintexts. Obviously, the characteristic's probability is not good enough for a straightforward solution with 2^{29} suitable pairs. However, we can use the set $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ of valid differences for each cell to our advantage.
We repeat the following for two random base plaintext-tweak pairs. For each of the two plaintext-tweak pairs, we query two sets of derived plaintext-tweak pairs: one for the base tweak, and one for the modified tweak with a difference of $\mathtt{a}$ in two cells, as specified by the truncated differential characteristic in Figure 2.13. The first set for the base tweak contains the following 8^8 modified messages. Each of the 8 active cells (■, ■) varies over 8 values: the base plaintext plus differences $\{\mathtt{0}, \mathtt{a}, \mathtt{f}, \mathtt{5}, \mathtt{d}, \mathtt{8}, \mathtt{7}, \mathtt{2}\}$. The second set for the modified tweak contains the same 8^8 messages. In total, the number of chosen plaintext-tweak pairs we query is

$$2 \cdot 2 \cdot 8^8 = 2^{26}.$$

Thus, we could repeat this up to $2^4 = 16$ times and still stay below the data complexity limit. To see how many suitable pairs we can generate from these queries, note that for each value of a cell in the first set, there are exactly 4 (out of 8) values for this cell in the second set that give a valid difference $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ (■) or $\{\mathtt{0}, \mathtt{5}, \mathtt{7}, \mathtt{f}\}$ (■), as illustrated in Figure 2.14. Here, we exploited that $\mathtt{a} + \mathtt{5} = \mathtt{f}$, where all these three values are suitable for our family of characteristics. Thus, the number of pairs we get is

$$2 \cdot 8^8 \cdot 4^8 = 2^{41},$$

and the expected number of valid pairs is at least

$$2^{41} \cdot 2^{-40.51} = 2^{0.49} \approx 1.40\,.$$

By repeating this up to 2^4 times, we can increase the expected number of valid pairs up to $2^{4.49} \approx 22.47$. We evaluated the initial structure practically for 1024 random keys, and found that the average number of valid pairs is significantly higher than the estimated 22.47, around $2^{6.28} \approx 78$.

2.4.4 Key Recovery

We can now use the family of characteristics and initial structure from subsection 2.4.3 to recover the two 64-bit secret keys k_0 and k_1. In the following, we will use 4 repetitions $r = 1, \ldots, 4$ of the initial structure. Thus, we need to query $4 \cdot 2^{26} = 2^{28}$ chosen plaintexts with chosen tweaks in order to generate the $4 \cdot 2^{41} = 2^{43}$ plaintext pairs. This is well below the complexity limit of 2^{30} chosen plaintexts for MANTIS_5.

Pre-Filtering Ciphertexts for Wrong Pairs

Before starting with the key guessing, we can filter for pairs which definitely do not follow the family of characteristics given in Figure 2.13. The necessary conditions for valid ciphertext pairs are that 5 cells $(S_1, S_4, S_{11}, S_{13}, S_{15})$ have a zero difference (marked □), while the difference in cell S_{14} is $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ after removing the last tweak addition (marked ◪). The reason for the restriction of the differences in cell S_{14} lies in the tweak addition in this cell before the last S-box application.
If we assume that plaintext pairs which do not follow our family of characteristics produce a randomly distributed difference pattern for corresponding ciphertext pairs, these conditions are fulfilled with a probability of 2^{-22}. Hence, we reduce the set of 2^{41} pairs per repetition r from the initial structure to a set I_r of about $2^{41-22} = 2^{19}$ pairs. Each set I_r is still expected to contain $2^{0.49} > 1$ valid pairs that follow the family of characteristics of Figure 2.13.

Complexity and Optimizations. A naive implementation of generating and pre-filtering pairs costs $4 \cdot 2^{41}$ state xor operations. However, instead of enumerating all valid pairs and then filtering for matches on 5 cells, it is much more efficient to reverse the process and only generate the relevant pairs as follows. Store each plaintext-tweak-ciphertext of Set 1 in a data structure of 2^{20} partitions, partitioned according to the value of the 5 pre-filter cells $S_1, S_4, S_{11}, S_{13}, S_{15}$. The expected size of each partition is about 2^5. Then, for each plaintext-tweak-ciphertext of Set 2, iterate only over the 2^5 candidates in the correct partition, and check whether the input difference is valid and the difference of output cell S_{14} is valid. The set I_r of remaining filtered pairs is the same, but the computational complexity is reduced to less than 2^{30} state xor operations.

Recovery of 44-bit $k_0' + k_1$

The first step of the attack is the partial recovery of 44 bits of the final whitening key $k_0' + k_1$. We want to check our key guesses against the differential pattern we get before the last application of MixColumns in Round 10 for our filtered ciphertext pairs. The probability that a 44-bit key guess leads to this pattern before the application of MixColumns is 2^{-30}:

- **Column 1:** Here, only cell S_{12} has a difference at the input of MixColumns, while the others have none. The requirements that lead to this pattern are that a key guess on the ciphertext cells S_0, S_5, S_{10} (■) leads to an equal difference after an S-box application, which happens with a probability of 2^{-8} per ciphertext pair and key guess.
- **Column 2:** This column is inactive. The only condition we have to fulfill here is that the difference introduced in cell S_{14} (◪) of the ciphertext is canceled by the tweak addition that happens before the S-box application of the last round (right after the last application of PermuteCells). Since our filtering ensures that only ciphertext pairs with differences

$\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ in cell S_{14} (■) after the last SubCells are considered, this happens with a probability of 2^{-2}.

- **Column 3:** For this column, cells S_2 (6) and S_6 (7) must have a difference $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$, while cells S_{10}, S_{14} have zero difference (□). The necessary conditions for this to happen are that a key guess on cells S_3, S_6, S_9, S_{12} of the ciphertext pair leads to an input difference $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\}$ on cells S_6, S_{12} (6, 7) before the last SubCells (2^{-2} per cell), and that the differences in S_3, S_9 (▬) each equal the difference between S_6 and S_{12} (2^{-4} per cell). The overall probability for this is 2^{-12} per ciphertext pair and key guess.
- **Column 4:** For this column, the same reasoning as for column 1 applies, now for ciphertext and key cells S_2, S_7, S_8 (9) after compensating for the last tweak addition. Again, the probability is 2^{-8}.

If we now decrypt one ciphertext pair $i \in I_r$ backwards for one SubCells layer under $2^{11 \cdot 4} = 2^{44}$ key guesses, $2^{44-30} = 2^{14}$ key guesses remain which satisfy all these conditions for this ciphertext pair i. We expect the correct key guess to satisfy the conditions for at least one of the ciphertext pairs $i \in I_r$, which follows the family of characteristics in Figure 2.13. Thus, we repeat the procedure for all 2^{19} pairs and consider the union of all resulting potential key candidates. We expect at most $2^{14} \cdot 2^{19} = 2^{33}$ candidates for the right key guess, which effectively reduces our keyspace by 2^{-11}. So, repeating the attack a total of 4 times with fresh initial structures is sufficient to recover the correct value of 44 bits of $k'_0 + k_1$.

Complexity and Optimizations. To get the possible key candidates per ciphertext pair, we need $2 \cdot (2^{16} \cdot 4 + 2 \cdot 2^{12} \cdot 3 + 2^4) \approx 2^{19.13}$ S-box look-ups, which corresponds roughly to $2^{11.54}$ MANTIS_5 encryptions (based on the total number of $16 \cdot 12$ S-boxes in MANTIS_5). In total, we have to generate key candidates for $4 \cdot 2^{19}$ pairs, corresponding to a total of about $2^{32.54}$ MANTIS_5 encryptions.
In a straightforward implementation, we get 4 lists, each containing 2^{33} key candidates, which dominates our memory requirements. We need to find matches between the 4 lists, which adds a computational complexity of roughly 2^{33} operations, depending on the implementation.
Note that it is not actually necessary to guess all 44 bit of the subkey at once per ciphertext pair $i \in I_r$. Instead, we can split up the key guesses column-wise into a 12-bit subkey for column 1 (with a set of valid subkey candidates of expected size $|\mathcal{C}^{(r,i)}_{0,5,10}| = 2^4$), a 4-bit subkey for column 2 ($|\mathcal{C}^{(r,i)}_{14}| = 2^2$), a 16-bit subkey for column 3 ($|\mathcal{C}^{(r,i)}_{3,6,9,12}| = 2^4$), and a 12-bit subkey for column 4 ($|\mathcal{C}^{(r,i)}_{2,7,8}| = 2^4$). The expected set of 2^{14} full key candidates per pair i is then the product set of these sub-candidates. We refer to this structured set of key candidates from repetition r and pair $i \in I_r$ as a bundle $\mathcal{B}^{(r,i)}$, where

$$\mathcal{B}^{(r,i)} = \mathcal{C}^{(r,i)}_{0,5,10} \times \mathcal{C}^{(r,i)}_{14} \times \mathcal{C}^{(r,i)}_{3,6,9,12} \times \mathcal{C}^{(r,i)}_{2,7,8}.$$

Storing all bundles requires only about $4 \cdot 2^{19} \cdot 10.25 < 2^{25}$ MANTIS states. To find the correct value of all 44 bits, we now need to compute

$$\bigcap_{r=1}^{4} \bigcup_{\substack{i \in I_r \\ |I_r| \approx 2^{19}}} \mathcal{C}^{(r,i)}_{0,5,10} \times \mathcal{C}^{(r,i)}_{14} \times \mathcal{C}^{(r,i)}_{3,6,9,12} \times \mathcal{C}^{(r,i)}_{2,7,8}.$$

The computational complexity of matching the bundles of key candidates is similar to before if the list of bundles per repetition is indexed efficiently per subkey candidate. Then, the bundles can be intersected subkey by subkey, starting with the most restrictive subkey, $\mathcal{C}^{(r,i)}_{3,6,9,12}$.

Recovery of 32-bit $k_0 + k_1$

With the help of the recovered 44 bits of $k'_0 + k_1$, we can filter our plaintext pairs $i \in I_r$ so that only the valid plaintext pairs following the family of characteristics in Figure 2.13 remain. The probability that the right key identifies a wrong pair as correct one is 2^{-30}. Therefore, it is likely that only correct pairs (approximately 4) remain after filtering $4 \cdot 2^{19}$ pairs. We now use those 4 valid pairs to recover 32 bits of the initial whitening key $k_0 + k_1$. We guess the key bits for all plaintext cells with differences, $S_0, S_5, S_6, S_7, S_8, S_{10}, S_{12}, S_{14}$. Then we can compute forward through the SubCells layer of Round 1, and check if the resulting difference pattern matches the family of characteristics. As shown in Figure 2.13, a wrong key matches the pattern with a probability of $2^{-15.51}$. So, the probability that a wrong key matches for all 4 correct pairs is $2^{-62.04}$. Therefore, we expect that only the correct subkey out of the 2^{32} possible candidates remains.

Complexity. We make a 32-bit key guess for each of 4 pairs, leading to a total of $2 \cdot 4 \cdot 8 \cdot 2^{32} = 2^{38}$ S-box look-ups. This corresponds to about $2^{30.42}$ MANTIS_5 encryptions.

Recovery of k_0 and k_1

Up to this point, we have recovered 32 bits of information about $k_0 + k_1$ and 44 bits of information about $k'_0 + k_1 = (k_0 \ggg 1) + (k_0 \gg 63) + k_1$. This gives us a system of 76 linearly independent linear equations for k_0 and k_1. To recover the full key, we have to guess 52 remaining bits and identify the right key using trial encryptions.
Instead of guessing all 52 bits, we can also use the SubCells layers of Rounds 2 and 3 (or 9 and 8) to first recover more bits of k_1, based on the previously recovered information. Similar to recovering $k_0 + k_1$, we can apply a guess-and-determine approach to only the 4 valid pairs, for example:

(1) Recover $S_0 + S_5 + S_{10}$ of k_1: We target cell S_{12} at the beginning of Round 2 (transition 2 → 3 in Figure 2.13). From our previously recovered key bits, we know the values of cells S_0, S_5, S_{10} at the beginning of Round 1 (2). Our target cell is the sum of these known values, plus an unknown cell $S_0 + S_5 + S_{10}$ of k_1. Checking the correct S-box transition for all 4 valid pairs is expected to eliminate all but the correct cell value (otherwise, we can additionally check the transition 8 ← 5 in Round 9). This adds 1 linearly independent equation to the system.

(2) Recover $S_6 + S_{12}$ of k_1: We target cells S_2, S_6 at the beginning of Round 2 (transitions 1 → a). Each of the two is the sum of the same two unknown, constant values (cell S_3 and cell S_9 after AddTweakey of Round 1), a known, variable value, and a cell of k_1 (S_6 or S_{12}, respectively). By checking the S-box transitions and then eliminating the two unknown constants, we recover the cell sum $S_6 + S_{12}$ of k_1. This adds 1 linearly independent equation to the system.

(3) Recover $S_2 + S_7 + S_8$ of k_1: We target cell S_3 at the end of Round 9 (transition 9 ← 5). Similar to (1), the transition depends on a sum of k_1 cells, $S_2 + S_7 + S_8$. From (1), we can derive the exact target difference in 5, so the transition probability is at most 2^{-2}, and we expect only the one correct cell value to remain. This adds 4 linearly independent equations to the system.

(4) Guess 1 bit: If we guess only 1 bit of k_0 now (e.g., in cell S_{12}), this will fully determine the values of cells $S_2, S_5, S_6, S_7, S_8, S_{12}$ of k_0 and k_1.

(5) Recover S_3 of k_1: We target cells S_6 and S_{10} at the beginning of Round 3 (transitions 3 → a). Due to the previous MixColumns operation, the internal difference between cells S_6 and S_{10} is equal to the internal difference between cells S_3 and S_{12} after the previous AddTweakey operation, which is known except for the addition of key cell S_3 of k_1. On the other hand, since we require that both target cells belong to the same set of 4 possible values for a valid transition, this cuts down the possible values for S_3 of k_1 to less than half. After repeating for all 4 valid pairs and, if necessary, similarly for the transition 5 ← a in Round 8, we expect only the correct candidate to remain. This adds 4 linearly independent equations to the system.

(6) Recover S_9 of k_1: We target cells S_2 and S_6 at the end of Round 9 (transitions 6 → a and 7 → a). The transition depends on the values of cells S_3, S_6, S_9, S_{12} before AddTweakey of Round 10, which are all known by now except for the addition of key cell S_9 of k_1. Determining S_9 adds another 4 linearly independent equations to the system.

Complexity. The guess-and-determine approach recovers 14 of the missing bits of the original 64-bit keys k_0 and k_1. This reduces the remaining bits that need to be guessed to 38. To complete the key, we have to compute 2^{38} trial encryptions, which dominates our attack complexity.

Practical Verification

We implemented the key recovery attack in C/C++ in order to verify the probability estimates and attack complexity. A first straightforward implementation revealed some additional structural properties of MANTIS that negatively affect the success probability of the attack. For this reason, we adapted some aspects of the attack in order to obtain a good success probability in practice.

The first issue is that while the estimated number of about 1 to 10 valid pairs per repetition appears to be a reasonable estimate on average, the variance is relatively high. We observed several repetitions with no valid pairs, while other repetitions produced a dozen or more pairs. This is a problem for the 44-bit key recovery of section 2.4.4, which relies on finding at least 1 valid pair per repetition. There are several options to compensate for this. If memory requirements and higher runtime are not an issue, we can simply expand all bundles of key candidates and count the number of occurrences of each candidate, which will reveal the correct candidate with very high probability. A more practical alternative is to change the initial structures per repetition to contain more structures for different plaintexts, but with fewer queries per structure, in order to decrease the variance. For example, if we use 2^6 base plaintexts per repetition, but vary only 7 instead of 8 cells, the resulting expected number of pairs per repetition remains the same at $2^6 \cdot 8^7 \cdot 4^7 = 2^{41}$, but the data complexity increases slightly to $2 \cdot 2^6 \cdot 8^7 = 2^{28}$, or $4 \cdot 2^{28} = 2^{30}$ in total for all repetitions.

The second issue is that during the 32-bit key recovery of section 2.4.4, we always find at least 2^8 possible key candidates instead of just 1, and 2 key candidates for the 44-bit subkey. Both this and the previous issue are caused by the same structural property of the MANTIS S-box. We filter our keys by checking whether the valid pairs follow the correct differential S-box transitions in Round 1, that is, $\{\mathtt{a}, \mathtt{f}, \mathtt{d}, \mathtt{5}\} \mapsto \{\mathtt{a}, \mathtt{f}\}$ for each cell. However, it turns out that whenever a pair of cells (x, x') follows one of these transitions, then so does $(x + \mathtt{a}, x' + \mathtt{a})$. This means that for each cell k of the correct subkey, there is an equivalent value $k + \mathtt{a}$ which also satisfies all the constraints of Round 1, leading to a total of at least 2^8 candidates. This would also increase the complexity of section 2.4.4 accordingly. Instead of the expensive brute-force

approach of section 2.4.4, we encoded the recovery of the remaining key information as a Boolean satisfiability (SAT) problem.
The final adapted attack successfully recovered the full key for several tested random challenges. A sample test run takes about 16 minutes to query 2^{30} plaintexts and generate the pre-filtered list of about 4×2^{21} pairs (section 2.4.4). Creating the bundles of key guesses takes 22 minutes and produces about $4 \times 2^{14.6}$ bundles in total, corresponding to about 2^{32} key candidates per repetition if fully enumerated (section 2.4.4). Intersecting these lists takes 18 minutes and produces more key candidates than expected, about 2^7. However, counting the frequency of each of these candidates across repetitions clearly identifies the correct 44-bit final whitening subkey (except for 1 bit, due to differentially equivalent keys as discussed above, which can be filtered by the SAT solver). In the sample test run, this correct key identified 14 valid pairs, slightly less than the observed average of roughly 2^5 pairs. The high number of valid pairs means that it is relatively unlikely that one repetition contains no valid pairs, and that these cases could also usually be easily fixed by reshuffling the random plaintexts between repetitions. We also observed no false positives among the valid pairs, except for several cases with differences $\{\mathtt{d}, \mathtt{5}\}$ in $S_0, S_5, S_6, S_{10}, S_{12}$ after the SubCells layer of Round 1 (1, 2). We included these cases in the family of target characteristics. For the initial whitening subkey recovery (section 2.4.4), as discussed above, the recovered information is only about $32 - 9$ bits due to differentially equivalent subkeys, and takes about 3 minutes. Finally, the SAT solver takes another 1.5 minutes to successfully recover the rest of the key (section 2.4.4). Overall, the full correct key is recovered in about 1 hour on a single core, and the process is trivially parallelizable.

2.4.5 Discussion

We recover the full 128-bit key for MANTIS_5 with a complexity of 2^{38} encryptions, memory requirements of 2^{25} MANTIS states, and a data complexity of 2^{28} chosen plaintexts with chosen tweaks. A practical implementation recovered the correct key in an hour based on 2^{30} chosen plaintexts. This violates the security claim for MANTIS_5.
We did not analyze the full-round MANTIS_7 proposal. Many of the observations and methods for MANTIS_5 also apply to MANTIS_7: It is relatively easy to find a very similar optimal differential characteristic with probability 2^{-100} (compared to 2^{-68} for MANTIS_5), and to apply the same observations for clustering characteristics. However, a straightforward adaptation of the full key recovery attack is made more difficult by several factors. For example, it is hard to find characteristics for MANTIS_7 which on the one hand have a sufficiently low number of active S-boxes, and on the other hand have enough active cells at the input and output to be useful for key recovery. Also, due to the small state size, the probability must be relatively high to avoid false positives among the valid pairs.
Our attack takes advantage of several lightweight building blocks of MANTIS, mostly inherited from the Midori block cipher. This includes the involutive S-box with its high-probability differential fixed points a and f, the lightweight near-MDS matrix with its binary coefficients, and the lightweight tweakey schedule. Throughout the analysis, the symmetries of the PRINCE-like design facilitate the repeated exploitation of these properties. Another major issue is the interaction of the Midori round function with PRINCE-like inner rounds, which leads to a Superbox structure over 4 S-box layers in the inner rounds. Considering all these properties, the security margin of MANTIS may be too optimistic.

2.5 Known-Key Differential Attack on Simpira v1

2.5.1 Introduction

Known Keys and Permutations

Permutations represent another popular trend in the design of AEAD schemes. Whereas TBCs provide a distinctly assigned input for each piece of information (tweak for positional metadata, key as secret, and plaintext for the actual data block), permutation-based designs simply operate on one large, secret state that successively accumulates data, secrets, and (implicit and explicit) positional metadata. While their approaches are more or less orthogonal, their goals are similar: Both TBC-based and permutation-based designs claim to provide cleaner, more efficient modes of operation than classic block cipher modes.
The burden is, however, shifted to the design of the underlying primitive. Designing a permutation, which simply maps a large input state to an output state of the same size, is a very challenging task: The permutation must be both efficient and "secure". The latter is hard to specify as well as hard to achieve due to the lack of a clearly assigned secret key input as a "root of security".
There have been several efforts to build permutations from well-analyzed, well-understood block ciphers, in particular AES. These efforts hope to profit from the vast body of literature on performance optimizations, implementations, and cryptanalytic insights for the original block cipher. In place of the round keys, other, possibly known values are added in each round, such as round constants or other input-dependent values from other branches of a Feistel network.
In this section, we investigate the latter case of an unkeyed Feistel network based on the AES round function. The designers invoke well-known results and techniques from the analysis of AES-like block ciphers to argue the security against differential cryptanalysis. However, we show that the lack of independent, unknown round keys invalidates this analysis at several stages, including the number of active S-boxes, the maximum differential probability, the complexity per solution, and the necessary security margin in the number of rounds.

Application to Simpira v1

The AES and its underlying wide-trail design strategy are among the most popular building blocks for new symmetric designs. There are several good reasons for this. New AES-like designs profit both from the insights in efficient implementations and from the extensive cryptanalysis and well-understood security bounds of AES. In particular, if new designs not only reuse the general design ideas, but the AES block cipher itself or its round function, then Intel's AES-NI instruction set can provide high software performance on modern CPUs. However, while block ciphers are a versatile building block for other cryptographic primitives, the fixed block size of AES of 128 bits implies a certain limitation. Modern designs often require larger states for efficiency or security. Examples include permutation-based cryptography (hash functions, authenticated encryption, etc.), wide-block encryption, security beyond 2^{64} inputs without resorting to beyond-birthday-security schemes, and more.
These considerations have motivated the design of numerous cryptographic algorithms based on the AES round function. Notable recent examples of dedicated designs include several authenticated encryption algorithms with excellent software performance, such as the CAESAR round-2 candidates AEGIS [114] and Tiaoxin [93], but also more specialized primitives like the Haraka hash function for short inputs [71]. Very recently, Jean and Nikolić [59] analyzed a more general family of AES-round-based building blocks that generalizes several of the previous

dedicated designs. However, except for the last work, these dedicated designs target only specific state sizes, and do not offer scalable, easily reusable building blocks for other cryptographic applications.
Simpira is a recently proposed family of permutations designed by Gueron and Mouha [52] that aims to fill this gap. The design goal is to provide very efficient permutations for arbitrarily large input sizes of $b \cdot 128$ bits, $b \in \mathbb{N}^+$, while taking advantage of the Intel AES-NI instruction set for optimized software implementations. To achieve these goals, Simpira plugs the AES round function into a generalized Feistel construction. Additionally, the designers provide computer-aided bounds for the minimum number of active S-boxes, and argue that these bounds provide security against a wide range of attack vectors. To showcase the versatility of the Simpira permutations, the designers propose a number of application scenarios, including Even-Mansour block cipher constructions, or a keyless Davies-Meyer variant for hash functions with limited-length inputs.

Our contribution. We analyze members of the original Simpira v1 family [52]. We show that the underlying assumptions of independence, and thus the derived bounds on the minimum number of active S-boxes, are incorrect. We focus our analysis on family member Simpira-4 with its 512-bit state, but similar observations also apply to other family members with larger state sizes. For Simpira-4, we provide differential trails with only 40 (instead of 75) active S-boxes for the recommended 15 rounds. Based on these trails, we propose collision attacks on the proposed Simpira-4 Davies-Meyer hash construction. For 16 rounds of the permutation, we obtain collisions for the full 512-bit hash output with complexity $2^{110.16}$. We also adapted the attack to the originally recommended 15 rounds, but omit the details in this report. These attacks violate the designers' security claims that there are no structural distinguishers below 2^{128}.

Related work. Rønjom [101] independently analyzed Simpira v1, and identified invariant subspaces for any even number of rounds of Simpira-4. Both attacks on Simpira v1 exploit properties of the underlying Type-1.x Generalized Feistel Structure by Yanagihara and Iwata [115] and the sparse, structured round constants. In response to Rønjom's and our attacks, Gueron and Mouha proposed a new version of the design, Simpira v2 [53], published at ASIACRYPT 2016. Simpira v2 replaces both the Feistel construction and the round constant schedule. In the remaining document, Simpira always refers to Simpira v1.
Simpira is not the first AES-round-based design with problematic round constants. Other examples include the analysis of the hash function Haraka [71] by Jean [58], the analysis of the withdrawn CAESAR round-1 candidate PAES [116] by Jean et al. [61, 62], or the analysis of SHAvite-3 [11] by Peyrin [96]. In all three cases, the structure of the round constants failed to break the symmetry properties of the unkeyed AES round function. However, our attack exploits different properties, in particular the incomplete diffusion of differences in the structured round constants. The results discussed below were published at SAC 2016 [35].

2.5.2 Description of Simpira

Simpira is a family of permutations designed by Gueron and Mouha [52]. By using the AES round function in a generalized Feistel construction, it can be adapted to any input size of $b \cdot 128$ bits, $b \in \mathbb{N}^+$. We refer to Simpira family members as Simpira-b.

F-Function

The Feistel update function $F = F_{c,b}$ applies two rounds of AES, where the Simpira family member b and the round counter c define the round constants. Like for AES, the 128-bit intermediate state of F is represented as a 4×4-matrix of bytes, labelled $s_0, \ldots, s_{15}$:

$$S = \begin{array}{|c|c|c|c|} \hline s_0 & s_4 & s_8 & s_{12} \\ \hline s_1 & s_5 & s_9 & s_{13} \\ \hline s_2 & s_6 & s_{10} & s_{14} \\ \hline s_3 & s_7 & s_{11} & s_{15} \\ \hline \end{array} .$$

We also refer to the value at byte position s_i in state S as $S[i]$.
The operations SubBytes, ShiftRows, and MixColumns are defined identically to AES, whereas AddConstant adds counters that define an invocation counter and the value b:

- SubBytes (SB): Applies the 8-bit AES S-box $\mathcal{S}$ to each of the 16 state bytes.
- ShiftRows (SR): Rotates row i of the state, $0 \leq i \leq 3$, by i bytes to the left.
- MixColumns (MC): Multiplies each byte column of the state by the MDS-matrix M over $\mathbb{K} = \mathbb{F}_2[\alpha]/(\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1)$,

$$M = \begin{pmatrix} \alpha & \alpha+1 & 1 & 1 \\ 1 & \alpha & \alpha+1 & 1 \\ 1 & 1 & \alpha & \alpha+1 \\ \alpha+1 & 1 & 1 & \alpha \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

- AddConstant (AC): In the c-th invocation of F for Simpira-b, xors the following round constant $C_{c,b}$ to the state:

$$C_{c,b} = \begin{array}{|c|c|c|c|} \hline c_0 & b_0 & 0 & 0 \\ \hline c_1 & b_1 & 0 & 0 \\ \hline c_2 & b_2 & 0 & 0 \\ \hline c_3 & b_3 & 0 & 0 \\ \hline \end{array} .$$

  In the remaining section, we focus on Simpira-4, so $b_0 = 04$ and $b_1 = b_2 = b_3 = 00$. Also, since the number of invocations of F is limited to 30 in Simpira-4, $c_1 = c_2 = c_3 = 00$. This constant is only added in the first of the two AES rounds of F, while the second round adds 0.

To refer to intermediate states of F for an input S, we use the following notation:

$$S \xmapsto{\mathsf{SB}} S^{\mathsf{SB1}} \xmapsto{\mathsf{SR}} S^{\mathsf{SR1}} \xmapsto{\mathsf{MC}} S^{\mathsf{MC1}} \xmapsto{\mathsf{AC}} S^{\mathsf{AC}} \xmapsto{\mathsf{SB}} S^{\mathsf{SB2}} \xmapsto{\mathsf{SR}} S^{\mathsf{SR2}} \xmapsto{\mathsf{MC}} S^{\mathsf{MC2}} = F(S) .$$

Round Function and Permutation

The permutation Simpira-b keeps a state of $b \cdot 128$ bits. The generalized Feistel round function for $b \geq 4$, where $b \neq 6, 8$, is illustrated in Figure 2.15. The final output of Simpira-b for $b \geq 4$, $b \neq 6, 8$, is the state after $6b - 9$ such rounds. Note that if the number of rounds is not a multiple of b, the state words are output in a permuted order to allow for more efficient implementations.

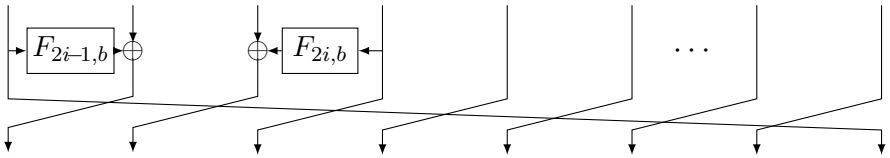


Figure 2.15: Round function for round i of Simpira-b for $b \geq 4$, $b \neq 6, 8$.

In case of Simpira-4, we denote the 4 state words before round $i \geq 1$ by $S_i^A, S_i^B, S_i^C, S_i^D$, so the state update rule corresponds to

$$\begin{aligned} S_{i+1}^A &= F_{2i-1,4}(S_i^A) \oplus S_i^B, \\ S_{i+1}^B &= F_{2i,4}(S_i^D) \oplus S_i^C, \\ S_{i+1}^C &= S_i^D, \\ S_{i+1}^D &= S_i^A. \end{aligned}$$

The recommended number of rounds for Simpira-4 is 15, with output words $(S_{16}^B, S_{16}^C, S_{16}^D, S_{16}^A)$.

Permutation-based Hashing

Simpira's designers identify several application areas for the Simpira permutation, such as block ciphers via an Even-Mansour construction. One particular suggested application is permutation-based hashing for short inputs, where "short" means the state size of any Simpira variant. The proposal is to use a single-block, keyless Davies-Meyer-like construction with a feed-forward, and compute the hash $h(x)$ of x as

$$h(x) = \text{Simpira-}b(x) \oplus x.$$

This approach provides an efficient construction for hashing inputs of limited length, which is required by many applications, such as Lamport signatures [72].

2.5.3 Collision Attacks on Simpira-4 Hash

In this section, we show that the number of rounds recommended by the designers is not sufficient to obtain a secure permutation. In particular, we provide collisions for full-round Simpira-4 when used in the hash mode suggested by the designers. While our analysis is focused primarily on Simpira-4, the basic observations also apply to the larger Simpira variants with the same construction approach, that is, Simpira-b with $b \geq 4$, $b \neq 6, 8$.

Differential Trail with 40 Active S-Boxes over 15 Rounds

The analysis performed by Simpira's designers [52] relies on two basic bounds: full bit diffusion, and minimum number of active S-boxes. The recommended number of rounds for each variant is selected as 3 times the number of rounds necessary to prove full bit diffusion and a minimum number of 25 differentially or linearly active S-boxes. While the proofs for full bit diffusion are based on generic results on the underlying generalized Feistel construction by Yanagihara and Iwata [115], the bounds for active S-boxes were obtained with a Mixed-Integer Linear Programming (MILP) model [89]. For Simpira-4, both full bit diffusion and at least 25 active S-boxes are claimed to be provided by 5 rounds of the round function. For the full number of 15 rounds, this method would imply at least 75 active S-boxes.

The bound is derived under the assumption that all F-function inputs are processed independently. For example, if the F-functions were indeed independent, the 4-round differential trail illustrated in Figure 2.16 would contain 20 independently active S-boxes. Since the trail is iterative, and adds 5 active S-boxes per round, this trail also demonstrates the tightness of the 15-round bound.
Of course, in an unkeyed primitive like a permutation or a hash function, the S-boxes are not really independent, since there are no random, independent round keys. Nevertheless, it is usually a reasonable assumption that the differential probabilities behave as if the values were actually independently random. We thus count S-boxes as independently active when it can reasonably be expected that their multiplied differential probabilities give a good estimate for the overall differential probability of the trail.

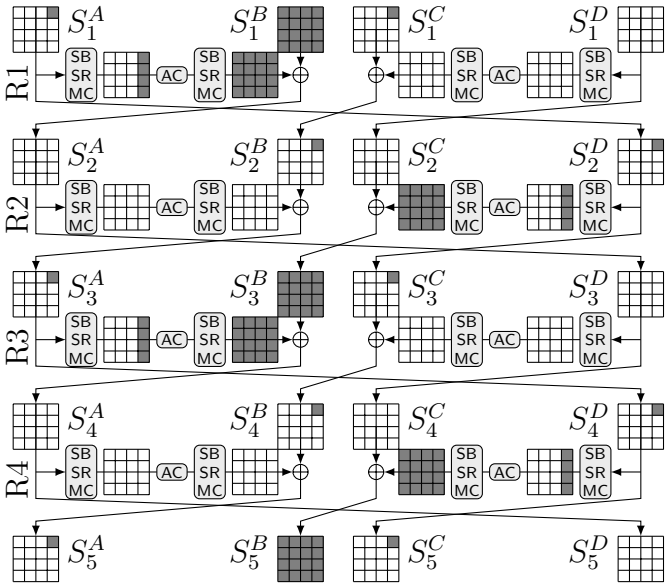


Figure 2.16: Iterative 4-round trail for Simpira-4 with 10 independently active S-boxes.

However, for all instances of Simpira-b with $b \geq 4$, $b \neq 6, 8$, this independence is violated by the generalized Feistel construction, and the particular definition of F. Consider, for example, the inputs to the active F-functions in rounds 1 and 2, S_1^A and S_2^D. The input values to the two F-functions are identical. Recall the definition of $F = F_{c,b}$, in our case $F_{1,4}$ and $F_{3,4}$. The only difference between $F_{1,4}$ and $F_{3,4}$ is the round-constant addition at the end of the first AES round. This means that the inputs and outputs of the S-boxes of the first AES round must be identical, i.e., $S_1^{A,\mathsf{MC1}} = S_2^{D,\mathsf{MC1}}$. The round constant only differs in state byte s_0, so this means the S-box transitions in the second AES round will also be identical except in s_0. In fact, the outputs $S_1^{A,\mathsf{MC2}}$ of $F_{1,4}$ an $S_2^{D,\mathsf{MC2}}$ of $F_{3,4}$ will have identical values except for the first column.
Considering the 4-round trail of Figure 2.16, this means that the entire output difference of $F_{3,4}$ will be identical to that of $F_{1,4}$ with probability 1, as illustrated in Figure 2.17. Note that s_0 is not active in the second AES round, and the differential behaviour of MixColumns is independent of the actual values of s_0. Consequently, if we fix all full-state differences to the same bitwise difference pattern, all single-byte differences to the same difference pattern, and all columnwise differences to the same difference pattern, the actual cost of the iterative trail of Figure 2.16 is equivalent to only 5 active S-boxes per 2 rounds, or 40 S-boxes overall for the recommended 15 rounds, which is about half as many as suggested by the MILP-based bound. In fact, the MILP model can be adapted to take this into account by counting only the activity of the left-hand F-functions, and only S-box s_0 for the right-hand F-functions, except in the first round. With

this modification, it is easy to prove that 40 active S-boxes is a tight bound for 25 rounds. The minimum number of rounds to achieve at least 25 active S-boxes is then 9, instead of 5.

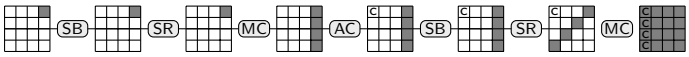


Figure 2.17: Trail for the F-function with 5 active S-boxes.

Collision Attack on 8 Rounds

We now want to use this iterative differential trail of Figure 2.16 to find collisions for the permutation-based hash construction suggested for Simpira permutations. Recall that in this short-input Davies-Meyer construction, the $b \cdot 128$-bit message is used as input to the Simpira permutation, and finally added as a feed-forward to the permutation output to produce the untruncated $b \cdot 128$-bit hash value. Our trail is incidentally very well suited to produce collisions for this feed-forward construction. Observe that if we fix all state differences to the same patterns as discussed in section 2.5.3, the feed-forward will cancel out the message difference with probability 1 for any number of rounds that is a multiple of $b = 4$.
To optimize the complexity of the collision attack, we need to fix the bitwise difference patterns suitably. Recall that the AES S-box has maximum differential probability $\frac{4}{256} = 2^{-6}$. For each nonzero input difference, there is exactly one output difference with this probability (and vice versa), while the other probabilities are either $\frac{2}{256} = 2^{-7}$ or 0. We can easily choose difference patterns so that all S-box transitions have this optimal probability, at least for uniformly random round constants. For example, if we fix the one-byte input difference to 75, the trail illustrated in Figure 2.18 satisfies our requirements. The probability of the differential for the F-function is then at least 2^{-30}. Overall, the probability of such an 8-round trail is at least $2^{-30 \cdot 4} = 2^{-120}$, and the resulting complexity for finding the 512-bit collision is at most 2^{120}.

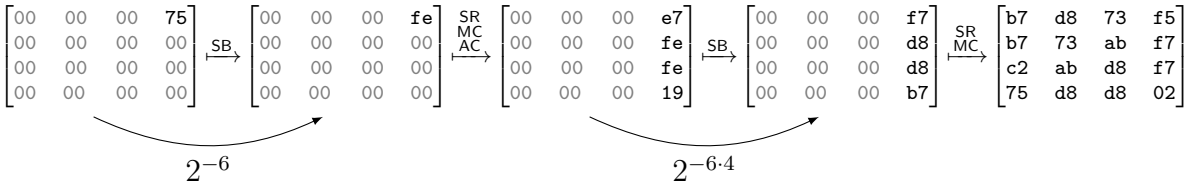


Figure 2.18: Trail for the F-function with probability 2^{-30}

Note that we are actually not interested in the probability of the trail within the F-function, but just in the input-output differential from the fixed 1-byte difference to the fixed 16-byte difference. The probability of this differential is typically higher than that of the trail, since several different trails can contribute to the same differential. In the case of 2-round AES, Keliher and Sui [63] proved that for a random round constant, the probability of the differential in Figure 2.18 is actually $2^{-30} + 74 \cdot 2^{-35} \approx 2^{-28.272}$.
If we consider additionally that the round constant is not random, but in our case fixed to $(00, 00, 00, 00)^\top$ for the relevant state bytes, the transition probabilities can increase even further. For example, the differential in Figure 2.19 is satisfied with probability $22 \cdot 2^{-32} \approx 2^{-27.54}$. With this differential, the probability of the 8-round trail is increased to $2^{4 \times 27.54} = 2^{-110.16}$.

Collision Attack on 16 Rounds

Since the permutation involves no round keys, we can try to satisfy the conditions for some active F-functions with message modification. We will try to find messages (or rather, initial structures

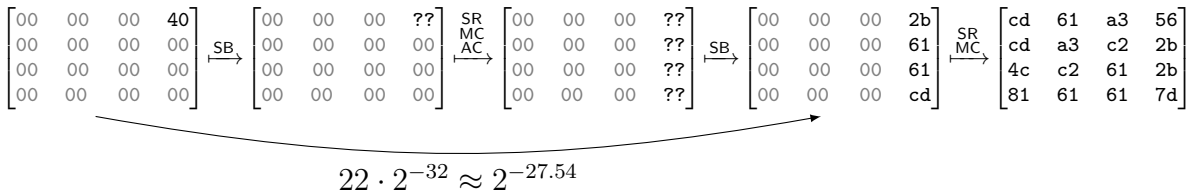


Figure 2.19: Differential for F-function with probability $2^{-27.54}$

for intermediate Simpira states) such that the conditions for several rounds are satisfied "for free" with probability 1, and append the 8-round trails - described in the previous paragraph - to be satisfied probabilistically. We first propose a simple initial structure covering 6 rounds, and then improve it to satisfy all conditions over 8 rounds, thus extending the previous 8-round trail to a 16-round trail with the same probability.

Initial structure for 6 rounds. It is sufficient to set the 4 bytes x_1, x_6, x_{11}, x_{12} of a state S_i^A to a suitable assignment in order to follow the trail for this F-function deterministically. We will refer to these 4 bytes as the diagonal in the following, and to a valid assignment as a valid diagonal. We can reuse one precomputed valid diagonal for all necessary diagonals.
We want to fix the values of the diagonals in S_1^A, S_3^A, and S_5^A to the valid diagonal. Observe that $S_1^A = S_3^C$, and $S_3^A = S_5^C$. Thus, by fixing the diagonals of S_5^A and S_5^C, we have already satisfied 2 F-trails. The remaining $12 + 16 + 12$ bytes of S_5^A, S_5^B, S_5^C can be filled arbitrarily, which will immediately determine the value of S_3^D and thus $S_3^{D,\mathsf{MC2}}$. If we now set the diagonal of S_3^C to the valid diagonal, and fill its remaining 12 bytes with arbitrary values, we completely determine S_5^D via S_4^B and S_4^A, and thus complete the state after 4 rounds. By varying the 52 arbitrary byte values, we can obtain the necessary $2^{110.16}$ candidates to satisfy the 8-round trail. The approach is illustrated in rounds 1–6 of Figure 2.20, where ▨ and ▧ mark the 52 arbitrary bytes.

Improved initial structure for 8 rounds by matching diagonals. With some additional effort, we can find initial structures that also satisfy the F-trail in round 7. We will again initialize the values of $S_5^A, S_5^B, S_5^C, S_3^C$ as in the previous 6-round initial structure. However, we can use the $12 + 12$ arbitrary bytes of S_5^A and S_5^C to obtain a valid diagonal in S_7^A. This will provide us with a 16-round collision attack with the same computational complexity as the 8-round trail discussed above.
Our goal is to obtain a match between the diagonals of $S_5^{D,\mathsf{MC2}}$ and $S_6^{A,\mathsf{MC2}}$, as illustrated in Figure 2.20. If these two diagonals sum to zero, the diagonal of S_7^A will take the exact same value as that of S_5^C, which is the valid diagonal. For this purpose, we want to initialize part of the initial structure to generate random values in $S_6^{A,\mathsf{MC2}}$, and independently a different part of the initial structure, to independently get random values in $S_5^{D,\mathsf{MC2}}$. Then, any match between the two corresponds to an initial structure that satisfies 4 F-trails.
Assume that S_3^C and S_5^B are already fixed to some arbitrary constants, with the valid diagonal in S_3^C. We first use the free bytes of S_5^A to randomize $S_6^{A,\mathsf{MC2}}$. Any complete assignment of S_5^A will directly determine $S_6^{A,\mathsf{MC2}}$ via $S_5^{A,\mathsf{MC2}}$ and S_6^A. We can assume the values are distributed reasonably close to uniformly random, since the values are processed by 4 AES rounds, and only 4 input bytes are fixed.
Independently, we can vary the 12 bytes of S_5^C to randomize the diagonal of $S_5^{D,\mathsf{MC2}}$. To see the independence of the values in S_5^A, consider the diagonal of $S_4^{A,\mathsf{MC2}}$. Its values will always be identical to that of $S_5^{D,\mathsf{MC2}}$, except for the first column, which is influenced by the round constant and will be considered separately in a moment. Since the diagonals of S_5^A and S_3^C

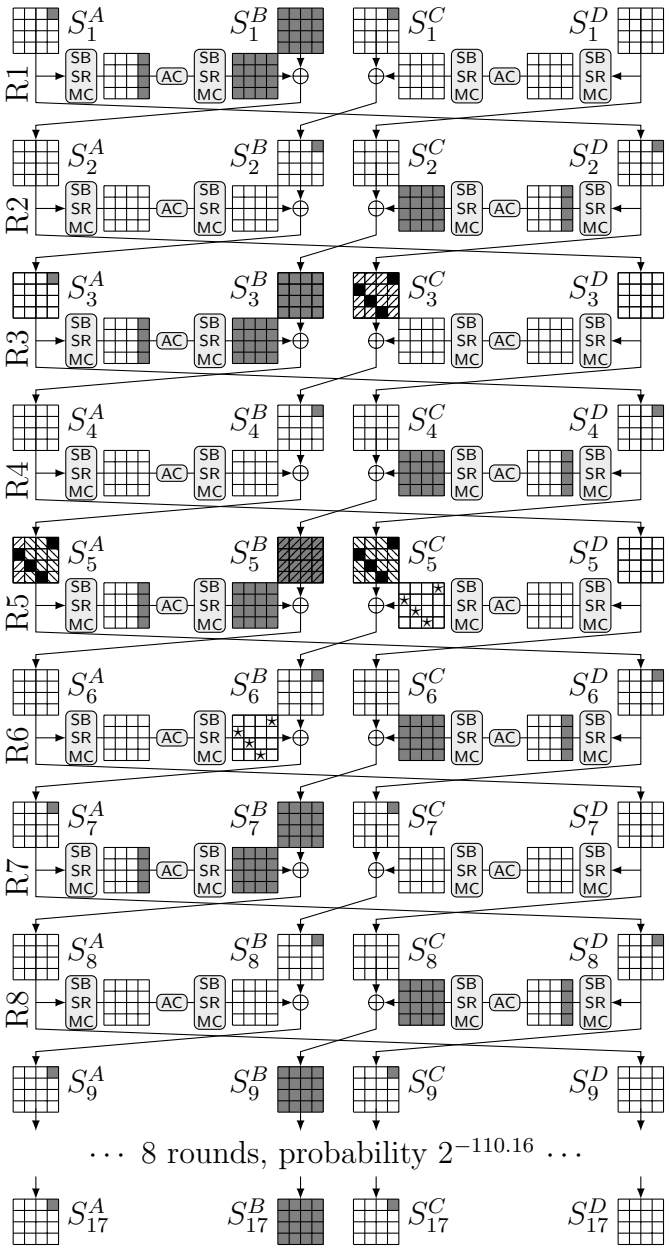


Figure 2.20: 16-round collision attacks on Simpira-4 hash using 8-round initial structure. ■ fixed difference, ■ valid diagonal, ▨ arbitrary bytes, ▧ matching inputs, ⊠ match

are fixed and predetermined, these values can further be traced back right to $S_3^{D,\mathsf{MC2}}$. Thus, knowing the diagonal of $S_3^{D,\mathsf{MC2}}$ is equivalent to knowing the target diagonal of $S_5^{D,\mathsf{MC2}}$, except for 1 byte in s_1. This equivalent diagonal is derived easily from S_5^C, again by 4 AES rounds via $S_4^D, S_4^{D,\mathsf{MC2}}, S_4^C$.

Evaluating the missing match byte s_1 of $S_5^{D,\mathsf{MC2}}$. Now we still need to account for the missing byte s_1. Fortunately, with some minor modifications of our guessing strategy, this value can also be computed directly from $S_3^{D,\mathsf{MC2}}$. Instead of varying all 12 arbitrary bytes of S_5^A to produce our matching candidates, we will keep the first column (bytes s_0, s_2, s_3) fixed. In fact, for simplicity, we will set them to the exact same values as the first column of S_3^C:

$$S_5^A[0,\ldots,3] = S_3^C[0,\ldots,3].$$

This implies that the values of the first column and diagonal (bytes $s_0,\ldots,s_3$, s_6, s_{11}, s_{12}) must be identical between $S_3^{D,\mathsf{MC2}}$ and $S_4^{A,\mathsf{MC2}}$. By partially inverting the last few steps of F, we can also easily verify that this means that

$$S_3^{D,\mathsf{AC}}[0] = S_4^{A,\mathsf{AC}}[0].$$

To determine our target value s_1 in $S_5^{D,\mathsf{MC2}}$, consider a differential view of the intermediate variables in the computations $F(S_4^A)$ and $F(S_5^D)$. The input values are identical, but a difference in s_0 is introduced by AddConstant. We are interested in how this difference ΔS^{AC} propagates to the target byte in ΔS^{MC2}. Since we only introduced a single-byte difference before the final MixColumns, we get

$$\begin{aligned}\Delta S^{\mathsf{MC2}}[1] &= \mathtt{01}\cdot\Delta S^{\mathsf{SB2}}[0]\\ &= \mathcal{S}\left(S_4^{A,\mathsf{AC}}[0]\right)\oplus\mathcal{S}\left(S_4^{A,\mathsf{AC}}[0]\oplus\Delta S^{\mathsf{AC}}[0]\right).\end{aligned}$$

By using the previously established identities between $F(S_4^A)$ and $F(S_3^D)$, and observing $\Delta S^{\mathsf{AC}}[0] = \mathtt{07}\oplus\mathtt{0A} = \mathtt{0D}$, we finally obtain all our target match bytes in $S_5^{D,\mathsf{MC2}}$ directly from $F(S_3^D)$:

$$\begin{aligned}S_5^{D,\mathsf{MC2}}[1] &= S_4^{A,\mathsf{MC2}}[1]\oplus\Delta S^{\mathsf{MC2}}[1]\\ &= S_4^{A,\mathsf{MC2}}[1]\oplus\mathcal{S}\left(S_4^{A,\mathsf{AC}}[0]\right)\oplus\mathcal{S}\left(S_4^{A,\mathsf{AC}}[0]\oplus\mathtt{0D}\right)\\ &= S_3^{D,\mathsf{MC2}}[1]\oplus\mathcal{S}\left(S_3^{D,\mathsf{AC}}[0]\right)\oplus\mathcal{S}\left(S_3^{D,\mathsf{AC}}[0]\oplus\mathtt{0D}\right),\\ S_5^{D,\mathsf{MC2}}[6] &= S_3^{D,\mathsf{MC2}}[6],\\ S_5^{D,\mathsf{MC2}}[11] &= S_3^{D,\mathsf{MC2}}[11],\\ S_5^{D,\mathsf{MC2}}[12] &= S_3^{D,\mathsf{MC2}}[12].\end{aligned}$$

Complexity of generating initial structures. Summarizing, we can now generate a large number of initial structures as follows. First, fix the diagonals in S_3^C and S_5^C to any valid diagonal. Fix all remaining bytes of S_3^C and S_5^B to arbitrary values. Copy the valid diagonal and first column of S_3^C to S_5^A. Vary the remaining 9 bytes of S_5^A, storing the resulting values of the diagonal of $S_6^{A,\mathsf{MC2}}$ in a list. Independently vary the 12 bytes of S_5^C, derive the diagonal of $S_5^{D,\mathsf{MC2}}$, and store it in a second list. Any match between the two lists gives a valid initial structure that follows the differential trail up to round 8.

If we only wanted one match on the 4 bytes of the diagonal, we could try 2^{16} values each for S_5^A and S_5^C, and would expect roughly $2^{2\cdot 16-32} = 1$ match due to the birthday effect. However, consider using 2^{32} values each instead. The expected number of 4-byte matches is roughly $2^{2\cdot 32-32} = 2^{32}$. Now we evaluate the complexity for generating these 2^{32} solutions. Computing the match bytes requires to evaluate $2 \cdot 2 \cdot 2^{32} = 2^{34}$ F-functions. Since 16-round Simpira-4 evaluates more than $16 = 2^4$ F-functions, this corresponds to a complexity of about $2^{32-4} = 2^{30}$ Simpira-4 evaluations. Thus, we were able to produce solutions with amortized complexity less than 1. With this initial structure, we obtain a 16-round collision with computational complexity about $2^{4\times 27.54} = 2^{110.16}$. The memory requirements are only about $2^{32} \cdot 2$ AES states.

2.6 Conclusion

The main research focus of this chapter are cryptanalytic techniques that capture the concept of "non-ideal" keys: related-key attacks, known-key attacks, and related-tweakey (differential) attacks. These techniques have been applied to Rijndael-160/160 and Rijndael-192/192, iFeed, MANTIS-5, and Simpira v1. The result of this security exercise is an improvement of the best-known cryptanalytic attacks on these ciphers. These techniques often exploit undesired mathematical relations that exist between sub/round keys. The lack of independent, random secret variables can affect the security strength of the cryptographic primitive at several stages, such as for example the number of active S-boxes, the necessary security margin in the number of rounds, etc.
Although each of these research results stands on its own, the main lesson that can be learned from this cryptanalysis is that the lack of independent keying material (e.g. subkeys, round keys, etc.) could strongly reduce the cryptographic strength of an algorithm. In practice, such dependencies in the keying material can be caused either externally by non-ideal randomness sources, or internally by cryptographic constructions using (parts of) the block cipher with interdependent or biased inputs. This shows that designers of cryptographic primitives should not limit their security evaluation to linear and differential cryptanalysis, but should also consider related-key attacks, known-key attacks and related-tweakey (differential) attacks when assessing the mathematical strength of their algorithms.

Chapter 3

Weak Cipher Model: cryptanalysis of hash functions

3.1 Introduction

Cryptographic hash functions are often constructed based on permutations or block ciphers, and security proofs are typically done in the ideal permutation or cipher model. In this model, the underlying ciphers are assumed to behave ideally. However, in practice this assumption does not necessarily hold, as for example demonstrated in Sect. 2.5. As a result, once these random primitives are instantiated in the hash functions, vulnerabilities of these instantiations may nullify the security. For example, Biryukov et al. [19] derived a related-key attack on AES and claimed that it invalidates the security of the Davies-Meyer compression function when the underlying primitive is instantiated with AES. A more recent approach in the design of cryptographic hash function is to base the compression function on a limited number of permutations [22, 83, 85, 100, 106]. These permutations could be designed from scratch, or obtained by fixing a small set of keys and using a block cipher for these keys only. Related- or chosen-key attacks on block ciphers do not help the adversary here, as the keys are fixed.
While in the classical security models for block ciphers the key is secret and randomly drawn and the adversary's target is to distinguish the instantiation of the cipher from a random permutation (also known as (strong) pseudorandom permutation security), this notion does not apply if the key is known to the adversary. At ASIACRYPT 2007, Knudsen and Rijmen [68] introduced known-key security of block ciphers. Here, the key is presumed known, and the adversary succeeds in distinguishing if it identifies a structural property of the cipher. Andreeva et al. [3] proposed a way to formalize the known-key security of block ciphers based on the underlying primitives. The model is derived from the indifferentiability framework [81] and hence all composition results carry over. Intuitively: suppose some cryptosystem F is proven to achieve a certain level of security in the ideal permutation model, and consider F' to be F with the permutations replaced by independent block cipher instantiations. Then, F' achieves the same level of security as F, up to the known-key indifferentiability bound of the underlying block ciphers.
In [3], several block cipher constructions are proven to be known-key indifferentiable, such as the multiple Even-Mansour cipher and 14 rounds of balanced Feistel with random functions (using a result of Holenstein et al. [56]). However, many constructions also have been demonstrated not to be known-key indifferentiable. Just to give one example, Knudsen and Rijmen demonstrated that the Feistel network on n bits with 7 rounds (called "Feistel$_7$") is *not* known-key indifferentiable [3, 68]: an adversary can generically find $2^{n/2}$ plaintext/ciphertext tuples (m, c) and (m', c')

satisfying $\mathsf{Ri}_{n/2}(m \oplus c \oplus m' \oplus c') = 0$ (where $\mathsf{Ri}_r(x)$ outputs the r rightmost bits of x). This result has lead to a wave of other known-key attacks on practical constructions, including generalized/extended variants of Feistel, reduced versions of AES or Rijndael, reduced variants of the block ciphers underlying SHA-2 and SHA-3 finalists BLAKE and Skein, and many more. In any of these cases, the traditional and commonly employed ideal cipher/permutation model falls short: results achieved in this model do not *necessarily* hold if the primitives are instantiated with Feistel_7, AES_8, or any other known-key distinguishable cipher.
This observation has led to the fundamental question whether known-key attacks invalidate the security of primitive-based hash functions and was the starting point of our research. The outcome is the **Weak Cipher Model (WCM)**: a model that goes beyond the traditional ideal cipher model as well as the principle of known-key attacks and that allows to generically analyze the impact of various weaknesses of block ciphers on various block cipher- and permutation-based cryptosystems. To our knowledge, we are the first to formally analyze the effect of a wide class of block cipher attacks on higher level cryptographic functions. Nonetheless, the weak cipher model is in essence still a model: we use an abstraction of the cryptanalytic known-key attacks in such a way that the ideal cipher model can be relaxed to cope with them. Furthermore, we apply our weak cipher model to various Block cipher-Based and Permutation-Based Hash Functions. The results of this chapter were published at ASIACRYPT 2015 [84].

3.2 The Weak Cipher Model

Notation

If X is a set, by $x \xleftarrow{\$} X$ we denote the uniformly random sampling of an element from X. By $X \xleftarrow{\cup} x$, we denote $X \leftarrow X \cup \{x\}$. For a bit string x, its bits are numbered $x = x_{|x|} \cdots x_2 x_1$. If $C \subseteq \{1, \ldots, |x|\}$, the function $\mathsf{Bits}_C(x)$ outputs a string consisting of all bits of x whose index is in C. Abusing notation, $\mathsf{Bits}_{\overline{C}}(x)$ always denotes the remaining bits (technically, $\overline{C} = \{1, \ldots, |x|\} \backslash C$). For $0 \leq r \leq |x|$, we consider $\mathsf{Ri}_r(x)$ that outputs the r rightmost bits of x. In other words, $\mathsf{Ri}_r(x) = \mathsf{Bits}_{\{1,\ldots,r\}}(x)$. For a function f, by $\mathsf{dom}(f)$ and $\mathsf{rng}(f)$ we denote its domain and range, respectively. For $\kappa \geq 0$ and $n \geq 1$, by $\text{BC}(\kappa, n)$ we denote the set of all block ciphers with κ-bit key operating on n bits. If $\kappa = 0$, $\text{BC}(n) := \text{BC}(0, n)$ denotes the set of all n-bit permutations. If Φ is a predicate, by $\text{BC}[\Phi](\kappa, n)$ we denote the subset of ciphers of $\text{BC}(\kappa, n)$ that satisfy predicate Φ. For $\pi \in \text{BC}[\Phi](\kappa, n)$, the input-output tuples are denoted (k, x, z), where $\pi(k, x) = \pi_k(x) = z$ and $\pi^{-1}(k, z) = \pi_k^{-1}(z) = x$. The key k is omitted in case $\kappa = 0$.

3.2.1 Security Model

Let $\mathsf{F} : \{0,1\}^s \to \{0,1\}^n$ be a compressing function instantiated with $\ell \geq 1$ primitives from $\text{BC}[\Phi](\kappa, n)$, for some predicate Φ. Throughout, we consider security of F in an idealized model: we consider an adversary $\mathcal{A}$ that is a probabilistic algorithm with oracle access to a randomly sampled primitive $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_\ell) \xleftarrow{\$} \text{BC}[\Phi](\kappa, n)^\ell$. $\mathcal{A}$ is information-theoretic and its complexity is only measured by the number of queries made to its oracles. The adversary can make forward and inverse queries to its oracles, and these queries are stored in a query history $\mathcal{Q}$.
A collision-finding adversary $\mathcal{A}$ for F aims at finding two distinct inputs to F that compress to the same range value. In more detail, we say that $\mathcal{A}$ succeeds if it finds two distinct inputs X, X' such that $\mathsf{F}(X) = \mathsf{F}(X')$ and $\mathcal{Q}$ contains all queries required for these evaluations of F.

We define by

$$\mathbf{Adv}^{\mathrm{col}}_{\mathsf{F}}(\mathcal{A}) = \mathbf{Pr}\left(\boldsymbol{\pi} \xleftarrow{\$} \mathrm{BC}[\Phi](\kappa,n)^{\ell},\ X, X' \leftarrow \mathcal{A}^{\boldsymbol{\pi}}\ :\ X \neq X' \wedge \mathsf{F}(X) = \mathsf{F}(X')\right)$$

the probability that $\mathcal{A}$ succeeds in this. By $\mathbf{Adv}^{\mathrm{col}}_{\mathsf{F}}(q)$ we define the maximum collision advantage taken over all adversaries making q queries.
For preimage resistance, we focus on everywhere preimage resistance [99], which captures preimage security for every point of $\{0,1\}^n$. Let $Z \in \{0,1\}^n$ be any range value. Then, we say that $\mathcal{A}$ succeeds in finding a preimage if it obtains an input X such that $\mathsf{F}(X) = Z$ and $\mathcal{Q}$ contains all queries required for this evaluation of F. We define by

$$\mathbf{Adv}^{\mathrm{epre}}_{\mathsf{F}}(\mathcal{A}) = \max_{Z\in\{0,1\}^n} \mathbf{Pr}\left(\boldsymbol{\pi} \xleftarrow{\$} \mathrm{BC}[\Phi](\kappa,n)^{\ell},\ X \leftarrow \mathcal{A}^{\boldsymbol{\pi}}(Z)\ :\ \mathsf{F}(X) = Z\right)$$

the probability that $\mathcal{A}$ succeeds, maximized over all possible choices for Z. By $\mathbf{Adv}^{\mathrm{epre}}_{\mathsf{F}}(q)$ we define the maximum (everywhere) preimage advantage taken over all adversaries making q queries.
If Φ is a trivial relation, we have $\mathrm{BC}[\Phi](\kappa,n) = \mathrm{BC}(\kappa,n)$, and the above definitions boil down to security in the ideal cipher model (ICM) if $\kappa > 0$ or the ideal permutation model (IPM) if $\kappa = 0$. On the other hand, if Φ is a non-trivial predicate, it strictly reduces the set $\mathrm{BC}(\kappa,n)$. In this case, we will refer to the model as the WCM for both $\kappa > 0$ and $\kappa = 0$. Very informally, this model still involves random ciphers/permutations, with the difference that an adversary may exploit certain additional properties. The modeling of a randomly drawn weak ciphers is much more delicate.

3.2.2 Random Weak Cipher

For a certain class of predicates, we discuss how to model a randomly drawn weak cipher π from $\mathrm{BC}[\Phi](\kappa,n)$. Let $A, B \in \mathbb{N}$. We will consider predicates that imply, *for every* $k \in \{0,1\}^{\kappa}$, the existence of A sets of B distinct queries $\{(x^1,z^1),\ldots,(x^B,z^B)\}$ that satisfy $\varphi_k\big(\{(x^1,z^1),\ldots,(x^B,z^B)\}\big)$ for some condition φ depending on key k. The predicate is denoted $\Phi(A,B,\varphi)$. A is merely a technical parameter, and throughout we assume it is larger than q, the number of oracle calls an adversary can make. This definition of $\Phi(A,B,\varphi)$ is fairly general. Particularly, predicate B-sets may overlap and the condition φ can represent any function on the inputs. We note that Φ can be easily generalized to tuples of different length and/or to multiple types of conditions at the same time.
Traditionally, an adversary has only forward $\pi_k(x)$ and inverse $\pi_k^{-1}(z)$ query access. In order for the adversary to be able to exploit the weakness present in π, we give it additional access to π via a "predicate query" $\pi_k^{\Phi}(y)$: on input of $y \in \{1,\ldots,A\}$, the adversary obtains a B-set $\{(x^1,z^1),\ldots,(x^B,z^B)\}$ that satisfies $\varphi_k\big(\{(x^1,z^1),\ldots,(x^B,z^B)\}\big)$.
A formal description of how to model $\pi \xleftarrow{\$} \mathrm{BC}[\Phi(A,B,\varphi)](\kappa,n)$ is given in Figure 3.1. Here, for every $k \in \{0,1\}^{\kappa}$, P_k is an initially empty list of π_k-evaluations, where a regular forward/inverse query adds one element (x,z) to P_k and a π_k^{Φ}-query may add up to B elements. Additionally, P_k^{Φ} is an initially empty list of queries to π_k^{Φ}. We denote by $\Sigma_k(P_k,P_k^{\Phi}) \subseteq (\{0,1\}^n \times \{0,1\}^n)^B$ the set of all tuples $\{(x^1,z^1),\ldots,(x^B,z^B)\}$ such that

1. $x^1,\ldots,x^B$ are pairwise distinct and $z^1,\ldots,z^B$ are pairwise distinct;
2. $\forall_{\ell=1}^{B}:\ x^{\ell} \in \mathsf{dom}(P_k) \Longrightarrow z^{\ell} = P_k(x^{\ell})$ and $z^{\ell} \in \mathsf{rng}(P_k) \Longrightarrow x^{\ell} = P_k^{-1}(z^{\ell})$;

procedure $\pi_k(x)$
if $P_k(x) = \perp$:
  $z \xleftarrow{\$} \{0,1\}^n \backslash \mathsf{rng}(P_k)$
  $P_k \xleftarrow{\cup} (x, z)$
end if
return $P_k(x)$

procedure $\pi_k^{-1}(z)$
if $P_k^{-1}(z) = \perp$:
  $x \xleftarrow{\$} \{0,1\}^n \backslash \mathsf{dom}(P_k)$
  $P_k \xleftarrow{\cup} (x, z)$
end if
return $P_k^{-1}(z)$

procedure $\pi_k^{\Phi}(y)$
if $P_k^{\Phi}(y) = \perp$:
  $\{(x^1, z^1), \ldots, (x^B, z^B)\} \xleftarrow{\$} \Sigma_k(P_k, P_k^{\Phi})$
  for $\ell = 1, \ldots, B$:
    if $(x^\ell, z^\ell) \notin P_k$:
      $P_k \xleftarrow{\cup} (x^\ell, z^\ell)$
    end if
  end for
  $P_k^{\Phi} \xleftarrow{\cup} (y, \{(x^1, z^1), \ldots, (x^B, z^B)\})$
end if
return $P_k^{\Phi}(y)$

Figure 3.1: Random weak cipher π. An adversary has access to π, π^{-1}, and π^{Φ}.

3. $\varphi_k\big(\{(x^1, z^1), \ldots, (x^B, z^B)\}\big)$ holds;
4. $\{(x^{p(1)}, z^{p(1)}), \ldots, (x^{p(B)}, z^{p(B)})\} \notin \mathsf{rng}(P_k^{\Phi})$ for any permutation p on $\{1, \ldots, B\}$.

For a new query $\pi_k^{\Phi}(y)$, the response is then randomly drawn from $\Sigma_k(P_k, P_k^{\Phi})$. Conditions (1-3) are fairly self-evident; note particularly that an existing $(x, z) \in P_k$ may appear in multiple predicate queries. Condition (4) assures that the drawing from $\Sigma_k(P_k, P_k^{\Phi})$ is not just an old predicate query or a reordering thereof. The usage of this set $\Sigma_k(P_k, P_k^{\Phi})$ allows for a uniform behavior of π_k^{Φ} for every k, and in general of $\pi \xleftarrow{\$} \mathrm{BC}[\Phi(A, B, \varphi)](\kappa, n)$, modulo the known existence of condition φ. This step is fundamental to our model and new compared with previous approaches of [26, 27, 75]. We remark that the model allows adversaries to make their queries at their own discretion, e.g., duplicate queries and regular queries after predicate queries are allowed.

3.2.3 Random Abortable Weak Cipher

Security analyses in the WCM are significantly more complex than in the ICM or IPM, which is in part because predicate queries may consist of older queries. This will particularly be an issue once collisions among queries are investigated. To suit the analysis for this case, we transform the WCM to an abortable weak cipher model (AWCM), which we denote as $\overline{\mathrm{BC}}[\Phi(A, B, \varphi)](\kappa, n)$. At a high-level, an abortable weak cipher responds to predicate queries with *new* query tuples only, and aborts once it turns out that an older query appears in a newer predicate query.
For any $k \in \{0,1\}^\kappa$ and partial P_k and P_k^{Φ}, define by $\bar{\Sigma}_k(P_k^{\Phi}) \subseteq (\{0,1\}^n \times \{0,1\}^n)^B$ the set of all tuples $\{(x^1, z^1), \ldots, (x^B, z^B)\}$ such that

3. $\varphi_k\big(\{(x^1, z^1), \ldots, (x^B, z^B)\}\big)$ holds;
4. $\{(x^{p(1)}, z^{p(1)}), \ldots, (x^{p(B)}, z^{p(B)})\} \notin \mathsf{rng}(P_k^{\Phi})$ for any permutation p on $\{1, \ldots, B\}$.

$\bar{\Sigma}_k(P_k^{\Phi})$ differs from $\Sigma(P_k, P_k^{\Phi})$ in that conditions (1) and (2) are omitted, and particularly: it is independent of P_k. A formal description of a random cipher $\bar{\pi} \xleftarrow{\$} \overline{\mathrm{BC}}[\Phi(A, B, \varphi)](\kappa, n)$ is given in Figure 3.2. It deviates from Figure 3.1 as follows: for every key k, $\bar{\pi}_k^{\Phi}$ responds randomly from $\bar{\Sigma}_k(P_k^{\Phi})$, and it aborts if the response violates one of the two skipped conditions of $\Sigma_k(P_k, P_k^{\Phi})$.

procedure $\bar{\pi}_k(x)$
if $P_k(x) = \perp$:
  $z \xleftarrow{\$} \{0,1\}^n \backslash \mathsf{rng}(P_k)$
  $P_k \xleftarrow{\cup} (x, z)$
end if
return $P_k(x)$

procedure $\bar{\pi}_k^{-1}(z)$
if $P_k^{-1}(z) = \perp$:
  $x \xleftarrow{\$} \{0,1\}^n \backslash \mathsf{dom}(P_k)$
  $P_k \xleftarrow{\cup} (x, z)$
end if
return $P_k^{-1}(z)$

procedure $\bar{\pi}_k^{\Phi}(y)$
if $P_k^{\Phi}(y) = \perp$:
  $\{(x^1, z^1), \ldots, (x^B, z^B)\} \xleftarrow{\$} \bar{\Sigma}_k(P_k^{\Phi})$
  for $\ell = 1, \ldots, B$:
    if $x^\ell \in \mathsf{dom}(P_k) \wedge z^\ell \neq P_k(x^\ell)$: abort
    if $z^\ell \in \mathsf{rng}(P_k) \wedge x^\ell \neq P_k^{-1}(z^\ell)$: abort
    if $(x^\ell, z^\ell) \in \{(x^1, z^1), \ldots, (x^{\ell-1}, z^{\ell-1})\}$: abort
    if $(x^\ell, z^\ell) \notin P_k$:
      $P_k \xleftarrow{\cup} (x^\ell, z^\ell)$
    end if
  end for
  $P_k^{\Phi} \xleftarrow{\cup} (y, \{(x^1, z^1), \ldots, (x^B, z^B)\})$
end if
return $P_k^{\Phi}(y)$

Figure 3.2: Random abortable weak cipher $\bar{\pi}$. An adversary has access to $\bar{\pi}, \bar{\pi}^{-1}$, and $\bar{\pi}^{\Phi}$.

The next lemma shows that the WCM and AWCM are indistinguishable as long as the abortable weak cipher does not abort, approximately up to the birthday bound. Here, we assume that $\bar{\Sigma}_k(P_k^{\Phi})$ is always large enough.

Lemma 1. *Let* $\bar{\pi} \xleftarrow{\$} \overline{\mathrm{BC}}[\Phi(A, B, \varphi^C)](\kappa, n)$. *Consider an adversary that makes* q *queries to* $\bar{\pi}$. *Then,*

$$\mathbf{Pr}\left(\bar{\pi} \textit{ sets } \mathsf{abort}\right) \leq \frac{B^2 q(q+1)}{2^n - \frac{B! q 2^n}{|\bar{\Sigma}_k(\varnothing)|}}.$$

Proof. Consider the i^{th} query, for $i \in \{1, \ldots, q\}$, and assume it is a predicate query $\bar{\pi}_k^{\Phi}(y)$. We will consider the probability that this query makes $\bar{\pi}$ abort, provided it has not aborted so far. Prior to this i^{th} query, $|P_k| \leq B(i-1)$ and $|P_k^{\Phi}| \leq i$. Basic combinatorics shows that

$$|\bar{\Sigma}_k(P_k^{\Phi})| = |\bar{\Sigma}_k(\varnothing)| - B! \cdot |P_k^{\Phi}|,$$

where we use that $\bar{\pi}$ has not aborted so far. This i^{th} query aborts only if for some $\ell \in \{1, \ldots, B\}$, the value x^ℓ equals an element in $\mathsf{dom}(P_k) \cup \{x^1, \ldots, x^{\ell-1}\}$ or the value z^ℓ equals an element in $\mathsf{rng}(P_k) \cup \{z^1, \ldots, z^{\ell-1}\}$.
Define by $\bar{\Sigma}_k^{\mathsf{abort}}(P_k^{\Phi})$ the set of all elements of $\bar{\Sigma}_k(P_k^{\Phi})$ that would lead to abort. We have $2B$ possible values to cause the abort (namely, $x^1, \ldots, z^B$), and it causes the abort if it equals an element in a set of size at most $|P_k| + B$. For any of these $2B(|P_k| + B)$ choices, the number of tuples in $\bar{\Sigma}_k(P_k^{\Phi})$ complying with this choice is at most $\frac{|\bar{\Sigma}_k(\varnothing)|}{2^n}$. Thus,

$$\mathbf{Pr}\left(\bar{\pi}^{\Phi}(y) \text{ sets } \mathsf{abort}\right) = \frac{|\bar{\Sigma}_k^{\mathsf{abort}}(P_k^{\Phi})|}{|\bar{\Sigma}_k(P_k^{\Phi})|} \leq \frac{2B(|P_k| + B) \cdot \frac{|\bar{\Sigma}_k(\varnothing)|}{2^n}}{|\bar{\Sigma}_k(\varnothing)| - B! \cdot |P_k^{\Phi}|} \leq \frac{2B^2 i}{2^n - \frac{B! q 2^n}{|\bar{\Sigma}_k(\varnothing)|}}.$$

The proof is completed by summation over $i = 1, \ldots, q$. □ □

3.3 Modeling Known-Key Attacks

We next apply the WCM to known-key attacks. For the sake of explanation, we first reconsider the Knudsen-Rijmen attack on Feistel_7 [68]. Let $n \in \mathbb{N}$, and let $\pi := \pi_k$ be an instance of Feistel_7 with fixed key k. Knudsen and Rijmen revealed four functions $f, f', g, g' : \{0,1\}^{n/2} \to \{0,1\}^n$ such that for all $y \in \{0,1\}^{n/2}$:

$$\begin{aligned} &g(y) = \pi(f(y)) \text{ and } g'(y) = \pi(f'(y))\,, \\ &\mathsf{Ri}_{n/2}\left(f(y) \oplus g(y)\right) = \mathsf{Ri}_{n/2}\left(f'(y) \oplus g'(y)\right). \end{aligned} \tag{3.1}$$

These four functions depend on the cryptographic primitive underlying Feistel_7 in a complicated way. Therefore, we can safely assume that these functions behave sufficiently random, besides this particular relation (3.1), and that they are unknown to the adversary. f, f', g, g' are all injective and satisfy $f(y) \neq f'(y)$ and $g(y) \neq g'(y)$ for all y. On the other hand, collisions of the form $f(y) = f'(y')$ and $g(y) = g'(y')$ may occur.
Generically, the attack demonstrates that for key k there exist $2^{n/2}$ possibly overlapping sets of distinct queries $\{(x^1, z^1), (x^2, z^2)\}$ that satisfy $\mathsf{Ri}_{n/2}\left(x^1 \oplus z^1 \oplus x^2 \oplus z^2\right) = 0$. In other words, Feistel_7 meets predicate $\Phi(2^{n/2}, 2, \varphi^{\text{Feistel}_7})$, where

$$\varphi_k^{\text{Feistel}_7}\left(\{(x^1, z^1), (x^2, z^2)\}\right) \;:\; \mathsf{Ri}_{n/2}\left(x^1 \oplus z^1 \oplus x^2 \oplus z^2\right) = 0\,.$$

Here, we remark that the Knudsen-Rijmen attack works for *any* fixed but known key k, and that condition $\varphi_k^{\text{Feistel}_7}$ is in fact independent of the key. In this work, we will consider a more general predicate $\Phi(A, B, \varphi^C)$ for $A, B \in \mathbb{N}$ and $C \subseteq \{1, \ldots, n\}$, where

$$\varphi_k^C\left(\{(x^1, z^1), \ldots, (x^B, z^B)\}\right) \;:\; \mathsf{Bits}_C\left(x^1 \oplus z^1 \oplus \cdots \oplus x^B \oplus z^B\right) = 0\,. \tag{3.2}$$

This generalized predicate considers the case of arbitrary but fixed and known keys, where the adversary can even choose the key every time it makes a predicate query. Note that also similar attacks on AES_8 and Threefish_{36} are covered, as they satisfy $\Phi(2^{n/8}, 2, \varphi^C)$ for certain C of size $10n/16$ and $\Phi(2^{n/8}, 4, \varphi^{\{1,\ldots,n\}})$, respectively. In general, all rebound- or boomerang-based known-key attack in literature are covered by predicate $\Phi(A, B, \varphi^C)$ for some A, B, C. Here, B is always a value independent of n (usually 2 or 4) and C is regularly a large subset (of size at least $n/4$). Throughout, we consider A to be sufficiently large.

Basic Computations for AWCM

For the specific condition φ^C of (3.2), we derive a simpler bound on the probability that a primitive $\bar{\pi} \xleftarrow{\$} \overline{\text{BC}}[\Phi(A, B, \varphi^C)](\kappa, n)$ aborts, along with some other elementary observations for $\bar{\pi}$. To this end, we define the notation "$[X]$", which equals 1 if X holds and 0 otherwise. For conciseness, we introduce the function $\delta_{B,C}[b]$ defined as

$$\delta_{B,C}[b] = 2^{|C|}[B = b] + [B > b] = \begin{cases} 2^{|C|} \text{ if } B = b\,, \\ 1 \text{ if } B > b\,, \\ 0 \text{ otherwise}\,. \end{cases} \tag{3.3}$$

Lemma 2. *Let $\bar{\pi} \xleftarrow{\$} \overline{\text{BC}}[\Phi(A, B, \varphi^C)](\kappa, n)$. Consider an adversary that makes $q \leq 2^{n-1}/B$ queries to $\bar{\pi}$. Then,*

$$\mathbf{Pr}\left(\bar{\pi} \textit{ sets } \mathsf{abort}\right) \leq \frac{B^2 q(q+1)}{2^n - Bq}\,. \tag{3.4}$$

Let $k \in \{0,1\}^\kappa$ and let $Z, Z', Z'' \in \{0,1\}^n$. Consider any new query $\bar{\pi}_k^\Phi(y)$ and assume it does not abort. Write the response as $\{(x^1, z^1), \ldots, (x^B, z^B)\}$. Then,

1. $\forall\, a \in \{1, \ldots, B\}$: $\mathbf{Pr}\left(x^a = Z\right), \mathbf{Pr}\left(z^a = Z\right) \leq \frac{1}{2^n - Bq}$;
2. $\forall\, a \in \{1, \ldots, B\}$: $\mathbf{Pr}\left(x^a \oplus z^a = Z\right) \leq \frac{\delta_{B,C}[1]}{2^n - Bq}$;
3. $\forall\, \{a, b\} \subseteq \{1, \ldots, B\}$: $\mathbf{Pr}\left(x^a \oplus z^a = Z \wedge x^b \oplus z^b = Z'\right) \leq \frac{\delta_{B,C}[2]}{2^{2n} - Bq}$;
4. $\forall\, \{a, b\} \subseteq \{1, \ldots, B\}$:
   $\mathbf{Pr}\left(x^a = Z \wedge x^b = Z' \wedge x^a \oplus z^a \oplus x^b \oplus z^b = Z''\right) \leq \frac{\delta_{B,C}[2]}{2^{3n} - Bq}$.

Proof. Recall from the proof of Lem. 1 that

$$|\bar{\Sigma}_k(P_k^\Phi)| = |\bar{\Sigma}_k(\varnothing)| - B!|P_k^\Phi|\,,$$

where $|P_k^\Phi| \leq q$. For the specific predicate analyzed in this lemma, $|\bar{\Sigma}_k(\varnothing)| = (2^n)^{2B-1}2^{n-|C|}$. In the remainder, we regularly bound $B! \leq B \cdot (2^n)^{2B-2}$ for $B \geq 1$ or $B! \leq B \cdot (2^n)^{2B-4}$ for $B \geq 2$.

Probability of abortion. The bound of (3.4) directly follows from Lem. 1, the above-mentioned size of $\bar{\Sigma}_k(\varnothing)$, and the bound on $B!$.
Part (i). Define by $\bar{\Sigma}_k^{(\mathrm{i})}(P_k^\Phi)$ the set of all elements of $\bar{\Sigma}_k(P_k^\Phi)$ that satisfy $x^a = Z$. Then, $|\bar{\Sigma}_k^{(\mathrm{i})}(P_k^\Phi)| \leq (2^n)^{2B-2}2^{n-|C|}$, and

$$\mathbf{Pr}\left(x^a = Z\right) = \frac{|\bar{\Sigma}_k^{(\mathrm{i})}(P_k^\Phi)|}{|\bar{\Sigma}_k(P_k^\Phi)|} \leq \frac{1}{2^n - Bq}\,.$$

A similar analysis applies to the case $z^a = Z$.
Part (ii). Define by $\bar{\Sigma}_k^{(\mathrm{ii})}(P_k^\Phi)$ the set of all elements of $\bar{\Sigma}_k(P_k^\Phi)$ that satisfy $x^a \oplus z^a = Z$. We make a distinction between $B = 1$ and $B > 1$. In case $B > 1$, a similar reasoning as in (i) applies, and we have $|\bar{\Sigma}_k^{(\mathrm{ii})}(P_k^\Phi)| \leq (2^n)^{2B-2}2^{n-|C|}$. On the other hand, if $B = 1$, we have $|\bar{\Sigma}_k^{(\mathrm{ii})}(P_k^\Phi)| = 0$ if $\mathsf{Bits}_C(Z) \neq 0$ and $|\bar{\Sigma}_k^{(\mathrm{ii})}(P_k^\Phi)| \leq 2^n$ if $\mathsf{Bits}_C(Z) = 0$. In any case,

$$|\bar{\Sigma}_k^{(\mathrm{ii})}(P_k^\Phi)| \leq (2^n)^{2B-2}2^{n-|C|}\delta_{B,C}[1]\,,$$

and

$$\mathbf{Pr}\left(x^a \oplus z^a = Z\right) = \frac{|\bar{\Sigma}_k^{(\mathrm{ii})}(P_k^\Phi)|}{|\bar{\Sigma}_k(P_k^\Phi)|} \leq \frac{\delta_{B,C}[1]}{2^n - Bq}\,.$$

Part (iii). This part only applies to $B > 1$; if $B = 1$ the probability equals 0 by construction. Define by $\bar{\Sigma}_k^{(\mathrm{iii})}(P_k^\Phi)$ the set of all elements of $\bar{\Sigma}_k(P_k^\Phi)$ that satisfy $x^a \oplus z^a = Z$ and $x^b \oplus z^b = Z'$. We make a distinction between $B = 2$ and $B > 2$. In case $B > 2$, a similar reasoning as in (i) and (ii) applies, and we have $|\bar{\Sigma}_k^{(\mathrm{iii})}(P_k^\Phi)| \leq (2^n)^{2B-3}2^{n-|C|}$. On the other hand, if $B = 2$, we have $|\bar{\Sigma}_k^{(\mathrm{iii})}(P_k^\Phi)| = 0$ if $\mathsf{Bits}_C(Z \oplus Z') \neq 0$ and $|\bar{\Sigma}_k^{(\mathrm{iii})}(P_k^\Phi)| \leq (2^n)^2$ if $\mathsf{Bits}_C(Z \oplus Z') = 0$. In any case,

$$|\bar{\Sigma}_k^{(\mathrm{iii})}(P_k^\Phi)| \leq (2^n)^{2B-3}2^{n-|C|}\delta_{B,C}[2]\,,$$

and

$$\mathbf{Pr}\left(x^a \oplus z^a = Z \wedge x^b \oplus z^b = Z'\right) = \frac{|\bar{\Sigma}_k^{(\mathrm{iii})}(P_k^\Phi)|}{|\bar{\Sigma}_k(P_k^\Phi)|} \leq \frac{\delta_{B,C}[2]}{2^{2n} - Bq}\,.$$

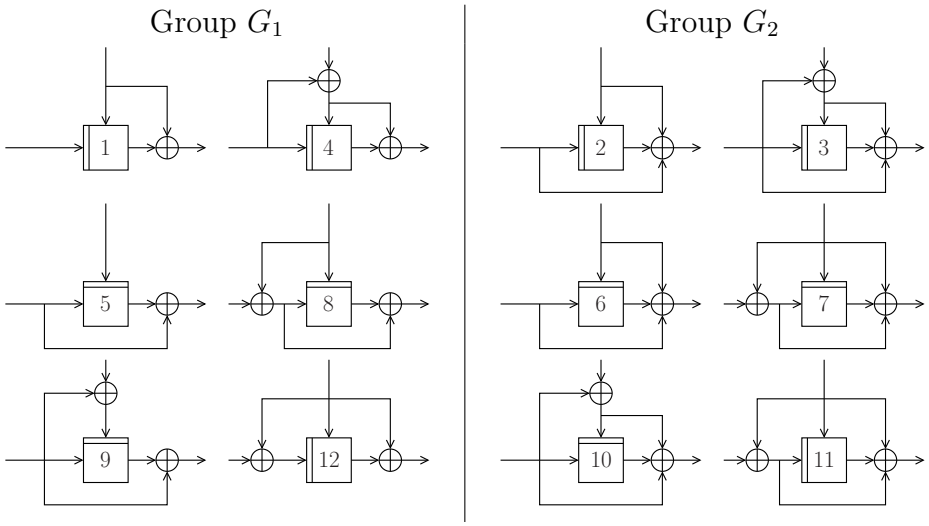


Figure 3.3: The 12 PGV compression functions. When in iteration mode, the message comes in at the top. The groups G_1 and G_2 refer to Lem. 3.

Part (iv). The approach is fairly similar to case (iii). If $B = 1$ the probability is 0 by construction. Define by $\bar{\Sigma}_k^{(\mathrm{iv})}(P_k^\Phi)$ the set of all elements of $\bar{\Sigma}_k(P_k^\Phi)$ that satisfy $x^a = Z$, $x^b = Z'$, and $x^a \oplus z^a \oplus x^b \oplus z^b = Z''$. In case $B > 2$, we have $|\bar{\Sigma}_k^{(\mathrm{iv})}(P_k^\Phi)| \leq (2^n)^{2B-4}2^{n-|C|}$. On the other hand, if $B = 2$, we have $|\bar{\Sigma}_k^{(\mathrm{iv})}(P_k^\Phi)| = 0$ if $\mathsf{Bits}_C(Z'') \neq 0$ and $|\bar{\Sigma}_k^{(\mathrm{iv})}(P_k^\Phi)| \leq 2^n$ if $\mathsf{Bits}_C(Z'') = 0$. In any case,

$$|\bar{\Sigma}_k^{(\mathrm{iv})}(P_k^\Phi)| \leq (2^n)^{2B-4}2^{n-|C|}\delta_{B,C}[2]\,,$$

and

$$\mathbf{Pr}\left(x^a = Z \wedge x^b = Z' \wedge x^a \oplus z^a \oplus x^b \oplus z^b = Z''\right) = \frac{|\bar{\Sigma}_k^{(\mathrm{iv})}(P_k^\Phi)|}{|\bar{\Sigma}_k(P_k^\Phi)|} \leq \frac{\delta_{B,C}[2]}{2^{3n} - Bq}\,. \qquad \square$$

$\square$

3.4 Application to PGV Compression Functions

We consider the 12 block cipher-based compression functions from Preneel, Govaerts, and Vandewalle (PGV) [97]. In the ICM these constructions achieve tight collision security up to about $2^{n/2}$ queries and preimage security up to about 2^n queries [23, 24, 39, 107]. The 12 constructions are depicted in Figure 3.3. Here, we follow the ordering of [24], where PGV1, PGV2, and PGV5 are better known as the Matyas-Meyer-Oseas [80], Miyaguchi-Preneel, and Davies-Meyer [88] compression functions.
Baecher et al. [4] analyzed the 12 PGV constructions under ideal cipher reducibility, which at a high level covers the idea of two constructions being equally secure for the same underlying idealized block cipher. They divide the PGV functions into two classes, in such a way that if some block cipher makes one of the constructions secure, it makes all functions in the corresponding class secure. Applied to our WCM, the results of Baecher et al. imply the following:

Lemma 3 (Ideal Cipher Reducibility of PGV [4], informal)**.** *Let* $\pi \xleftarrow{\$} \mathrm{BC}[\Phi](n,n)$ *for some predicate* Φ*. Let*

$$G_1 = \{1, 4, 5, 8, 9, 12\}\,, \text{ and } G_2 = \{2, 3, 6, 7, 10, 11\}\,.$$

For any $\alpha \in \{1,2\}$ and $i,j \in G_\alpha$, PGVi and PGVj achieve the same level of collision and preimage security once instantiated with π.

Baecher et al. also derive a reduction between the two classes, but this reduction requires a non-direct transformation on the ideal cipher π,[1] making it unsuitable for our purposes. Thanks to Lem. 3, it suffices to only analyze PGV1 and PGV2 in the WCM: the bounds carry over to the other 10 PGV constructions. In Sect. 3.4.1 we analyze the collision security of these functions in the WCM. The preimage security is considered in Sect. 3.4.2.

3.4.1 Collision Security

Theorem 1. *Let $n \in \mathbb{N}$. Let $\alpha \in \{1,2\}$ and consider* PGVα. *Suppose $\pi \xleftarrow{\$} \mathrm{BC}[\Phi(A,B,\varphi^C)](n,n)$. Then, for $q \le 2^{n-1}/B$,*

$$\mathbf{Adv}^{\mathrm{col}}_{\mathrm{PGV}\alpha}(q) \le \frac{B^2\delta_{B,C}[1]q^2}{2^n} + \binom{B}{2}\frac{2\delta_{B,C}[2]q}{2^n} + \frac{4B^2q^2}{2^n}\,.$$

Proof. We focus on PGV2. The analysis for PGV1 is a simplification due to the absence of the feed-forward of the key. We consider any adversary that has query access to $\pi \xleftarrow{\$} \mathrm{BC}[\Phi(A,B,\varphi^C)](n,n)$ and makes q queries. As a first step, we move from π to $\bar\pi \xleftarrow{\$} \overline{\mathrm{BC}}[\Phi(A,B,\varphi^C)](n,n)$. By Lem. 2, this costs us an additional term $\frac{B^2q(q+1)}{2^n-Bq}$.
A collision for PGV2 would imply the existence of two distinct query pairs (k,x,z), (k',x',z') such that $k \oplus x \oplus z = k' \oplus x' \oplus z'$. We consider the i^{th} query ($i \in \{1,\dots,q\}$) to be the first query to make this condition satisfied, and sum over $i = 1,\dots,q$ at the end. For regular (forward or inverse) queries, the analysis of [23,24,107] mostly carries over. The analysis of predicate queries is a bit more technical.
Query $\bar\pi_k(x)$ or $\bar\pi_k^{-1}(z)$. The cases are the same by symmetry, and we consider $\bar\pi_k(x)$ only. Denote the response by z. There are at most $B(i-1)$ possible (k',x',z'). As z is randomly drawn from a set of size at least $2^n - Bq$, it satisfies $z = k \oplus x \oplus k' \oplus x' \oplus z'$ with probability at most $\frac{B(i-1)}{2^n-Bq}$.
Query $\bar\pi_k^{\Phi}(y)$. Denote the query response by $\{(k,x^1,z^1),\dots,(k,x^B,z^B)\}$. In case the B-set contributes only to (k,x,z), the same reasoning as for regular queries applies with the difference that any query of the B-set may be successful and that the bound of Lem. 2 part (2) applies: $\frac{B^2\delta_{B,C}[1](i-1)}{2^n-Bq}$.
Now, consider the case the predicate query contributes to both (k,x,z) and (k,x',z'). There are $\binom{B}{2}$ ways for the predicate query to contribute (or 0 if $B=1$). By Lem. 2 part (3), which considers the success probability for any such combination, the predicate query results in a collision with probability at most $\binom{B}{2}\frac{\delta_{B,C}[2]2^n}{2^{2n}-Bq}$.
Conclusion. Taking the maximum of all success probabilities, the i^{th} query is successful with probability at most $\frac{B^2\delta_{B,C}[1](i-1)}{2^n-Bq} + \binom{B}{2}\frac{\delta_{B,C}[2]2^n}{2^{2n}-Bq}$. Summation over $i = 1,\dots,q$ gives

$$\mathbf{Adv}^{\mathrm{col}}_{\mathrm{PGV2}}(q) \le \frac{B^2\delta_{B,C}[1]q^2}{2(2^n-Bq)} + \binom{B}{2}\frac{\delta_{B,C}[2]q}{2^n-Bq} + \frac{B^2q(q+1)}{2^n-Bq}\,,$$

where the last part of the bound comes from the transition from WCM to AWCM. The proof is completed by using the fact that $2^n - Bq \ge 2^{n-1}$ for $Bq \le 2^{n-1}$, and that $q+1 \le 2q$ for $q \ge 1$. □ □

[1] If π makes the PGV constructions from group G_1 secure, there is a transformation τ such that τ^π makes the constructions from G_2 secure, and vice versa.

We note that the bound gets worse for increasing values of B. This has a technical cause: predicate queries are counted equally expensive as regular queries, but result in up to B new query tuples. This leads to several factors of B in the bound. As this work is mainly concerned with differential known-key attacks for which B is regularly small, these factors are of no major influence.
The implications of the bound of Thm. 1 become more visible when considering particular choices of B and C.

1. If $B = 1$, then $\mathbf{Adv}^{\mathrm{col}}_{\mathrm{PGV}\alpha}(q) \leq \frac{2^{|C|}q^2}{2^n} + \frac{4q^2}{2^n}$;
2. If $B = 2$, then $\mathbf{Adv}^{\mathrm{col}}_{\mathrm{PGV}\alpha}(q) \leq \frac{20q^2}{2^n} + \frac{4\cdot 2^{|C|}q}{2^n}$;
3. If $B \geq 3$ (independent of n), then $\mathbf{Adv}^{\mathrm{col}}_{\mathrm{PGV}\alpha}(q) \leq \frac{5B^2q^2}{2^n} + \frac{B^2q}{2^n}$.

In other words, for $B = 2$ and C with $|C| \leq n/2$, or for $B \geq 3$ constant and C arbitrary, the PGV functions achieve the same $2^{n/2}$ collision security level as in the ICM. On the other hand, if $B = 1$, collisions can be found in about $2^{(n-|C|)/2}$ queries, and if $B = 2$ with $|C| > n/2$, in about $2^{n-|C|} < 2^{n/2}$ queries.

Tightness

For the cases $B = 1$ and C arbitrary, and $B = 2$ and C arbitrary such that $|C| > n/2$, we derive generic attacks that demonstrate tightness of the bound of Thm. 1. Knudsen and Rijmen [68] and Sasaki et al. [102,103] already considered how to exploit a known-key pair for the underlying block cipher to find a collision for the Matyas-Meyer-Oseas (PGV1) and/or Miyaguchi-Preneel (PGV2) compression functions. Their attacks correspond to our $B = 2$ case.

Proposition 1 ($B = 1$). *Let $n \in \mathbb{N}$. Let $\alpha \in \{1, 2\}$ and consider* PGVα. *Suppose $\pi \xleftarrow{\$} \mathrm{BC}[\Phi(A, 1, \varphi^C)](n, n)$. Then, $\mathbf{Adv}^{\mathrm{col}}_{\mathrm{PGV}\alpha}(q) \geq \frac{q^2}{2^{n-|C|}}$.*

Proof. We construct a collision-finding adversary $\mathcal{A}$ for PGV2. It fixes key $k = 0$, and makes predicate queries to π_k^Φ on input of distinct values y to obtain q queries (k, x_y, z_y) satisfying $\mathsf{Bits}_C(x_y \oplus z_y) = 0$. Any two such queries collide on the entire state, $k \oplus x_y \oplus z_y = k \oplus x_{y'} \oplus z_{y'}$, with probability at least $\frac{q^2}{2^{n-|C|}}$. The attack for PGV1 is the same as we have taken $k = 0$. □

Proposition 2 ($B = 2$ and $|C| > n/2$). *Let $n \in \mathbb{N}$. Let $\alpha \in \{1, 2\}$ and consider* PGVα. *Suppose $\pi \xleftarrow{\$} \mathrm{BC}[\Phi(A, 2, \varphi^C)](n, n)$. Then, $\mathbf{Adv}^{\mathrm{col}}_{\mathrm{PGV}\alpha}(q) \geq \frac{q}{2^{n-|C|}}$.*

Proof. We construct a collision-finding adversary $\mathcal{A}$ for PGV2. It fixes key $k = 0$, and makes predicate queries to π_k^Φ on input of distinct values y to obtain q 2-sets $\{(k, x_y^1, z_y^1), (k, x_y^2, z_y^2)\}$ satisfying $\mathsf{Bits}_C\left(x_y^1 \oplus z_y^1\right) = \mathsf{Bits}_C\left(x_y^2 \oplus z_y^2\right)$. These two queries collide on the entire state, $k \oplus x_y^1 \oplus z_y^1 = k \oplus x_y^2 \oplus z_y^2$, with probability at least $\frac{1}{2^{n-|C|}}$. If the adversary makes q predicate queries, we directly obtain our bound. The attack for PGV1 is the same as we have taken $k = 0$. □

3.4.2 Preimage Security

Theorem 2. *Let $n \in \mathbb{N}$. Let $\alpha \in \{1, 2\}$ and consider* PGVα. *Suppose $\pi \xleftarrow{\$} \mathrm{BC}[\Phi(A, B, \varphi^C)](n, n)$. Then, for $q \leq 2^{n-2}/B$,*

$$\mathbf{Adv}^{\mathrm{epre}}_{\mathrm{PGV}\alpha}(q) \leq \left(\frac{2Bq}{2^n}\right)^B + \frac{2B^2\delta_{B,C}[1]q}{2^n}.$$

Entering various choices of B and C shows that in the PGV functions remain mostly unaffected in the WCM if $B \geq 2$, and the same security level as in the ICM is achieved [23, 24, 107]. A slight security degradation appears for $B = 1$ as preimages can be found in about $2^{n-|C|}$.

Tightness

For the case $B = 1$, we derive a generic attack that demonstrates the tightness of the bound of Thm. 2.

Proposition 3 ($B = 1$). *Let $n \in \mathbb{N}$. Let $\alpha \in \{1, 2\}$ and consider* PGVα. *Suppose $\pi \xleftarrow{\$} \mathrm{BC}[\Phi(A, 1, \varphi^C)](n, n)$. Then, $\mathbf{Adv}^{\mathrm{epre}}_{\mathrm{PGV}\alpha}(q) \geq \frac{q}{2^{n-|C|}}$.*

Proof. Let Z be any given range value with $\mathsf{Bits}_C(Z) = 0$ (note that epre guarantees security for *every* range point). A preimage-finding adversary $\mathcal{A}$ for PGV2 proceeds as follows. It fixes key $k = 0$, and makes predicate queries to π_k^{Φ} on input of distinct values y to obtain q queries (k, x_y, z_y) satisfying $\mathsf{Bits}_C(x_y \oplus z_y) = 0$. Any such query hits Z on the entire state, $k \oplus x_y \oplus z_y = Z$, with probability at least $\frac{q}{2^{n-|C|}}$. The attack for PGV1 is the same as we have taken $k = 0$. □

3.5 Application to Grøstl Compression Function

We consider the provable security of the compression function mode of operation of Grøstl [45] (see also Figure 3.4):

$$\mathsf{F}_{\mathrm{Grøstl}}(x_1, x_2) = x_2 \oplus \pi_1(x_1) \oplus \pi_2(x_1 \oplus x_2)\,. \tag{3.5}$$

The Grøstl compression function is in fact designed to operate in a wide-pipe mode, and in the IPM, the function is proven collision secure up to about $2^{n/4}$ queries and preimage secure up to $2^{n/2}$ queries [42]. We consider the security of $\mathsf{F}_{\mathrm{Grøstl}}$ in the WCM, where $(\pi_1, \pi_2) \xleftarrow{\$} \mathrm{BC}[\Phi(A, B, \varphi^C)](n)^2$. We remark that in this section we consider keyless primitives, hence $\kappa = 0$ and the k-input is dropped throughout. We furthermore note that finding collisions and preimages for $\mathsf{F}_{\mathrm{Grøstl}}$ is equivalent to finding them for

$$\mathsf{F}'_{\mathrm{Grøstl}}(x_1, x_2) = x_1 \oplus x_2 \oplus \pi_1(x_1) \oplus \pi_2(x_2)\,, \tag{3.6}$$

as $\mathsf{F}_{\mathrm{Grøstl}}(x_1, x_2) = \mathsf{F}'_{\mathrm{Grøstl}}(x_1, x_1 \oplus x_2)$, and we will consider $\mathsf{F}'_{\mathrm{Grøstl}}$ throughout.

3.5.1 Collision Security

Theorem 3. *Let $n \in \mathbb{N}$. Suppose $(\pi_1, \pi_2) \xleftarrow{\$} \mathrm{BC}[\Phi(A, B, \varphi^C)](n)^2$. Then, for $q \leq 2^{n-1}/B$,*

$$\mathbf{Adv}^{\mathrm{col}}_{\mathsf{F}'_{\mathrm{Grøstl}}}(q) \leq \frac{B^4 \delta_{B,C}[1] q^4}{2^n} + \binom{B}{2} \frac{2\delta_{B,C}[2](q^2 + 2^{n/2-|C|}q)}{2^n} + \frac{B^2 q^2}{2 \cdot 2^{n/2}} + \frac{4B^2 q^2}{2^n}\,.$$

If we enter particular choices of B and C into the bound, we find results comparable to the case of Sect. 3.4.1. In more detail, for $B = 2$ and C with $|C| \leq n/2$, or for $B \geq 3$ constant and C arbitrary, $\mathsf{F}_{\mathrm{Grøstl}}$ achieves the same $2^{n/4}$ collision security level as in the ICM [42]. If $B = 1$, the bound guarantees security up to about $2^{(n-|C|)/4}$, and if $B = 2$ with $|C| > n/2$, collisions can be found in about $2^{(n-|C|)/2}$ queries.

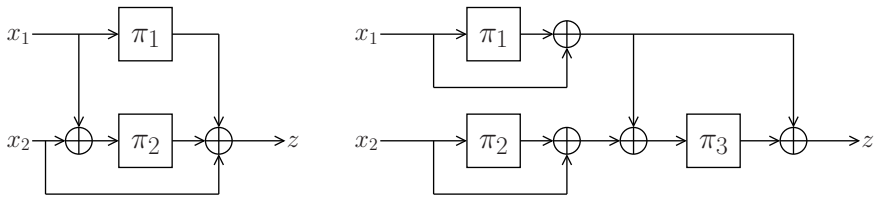


Figure 3.4: Grøstl compression function (left) and Shrimpton-Stam (right).

3.5.2 Preimage Security

Theorem 4. *Let $n \in \mathbb{N}$. Suppose $(\pi_1, \pi_2) \xleftarrow{\$} \mathrm{BC}[\Phi(A, B, \varphi^C)](n)^2$. Then, for $q \leq 2^{n-1}/B$,*

$$\mathbf{Adv}^{\mathrm{epre}}_{\mathsf{F}'_{\text{Grøstl}}}(q) \leq \frac{2B^2\delta_{B,C}[1](q^2 + 2^{n/2-|C|}q)}{2^n} + \frac{Bq}{2^{n/2}} + \frac{4B^2q^2}{2^n}\,.$$

3.6 Application to Shrimpton-Stam Compression Function

In this section, we consider the provable security of the Shrimpton-Stam compression function [106] (see also Figure 3.4):

$$\mathsf{F}_{\mathrm{SS}}(x_1, x_2) = x_1 \oplus \pi_1(x_1) \oplus \pi_3(x_1 \oplus \pi_1(x_1) \oplus x_2 \oplus \pi_2(x_2))\,. \tag{3.7}$$

This function is proven asymptotically optimally collision and preimage secure up to $2^{n/2}$ queries in the IPM [83, 100, 106]. We consider the security of F_{SS} in the WCM, where $(\pi_1, \pi_2, \pi_3) \xleftarrow{\$} \mathrm{BC}[\Phi(A, B, \varphi^C)](n)^3$. (As in Sect. 3.5 we consider keyless functions, hence $\kappa = 0$ and the key inputs are dropped throughout.) Our findings readily apply to the generalization of F_{SS} of [83]. The analysis of this construction is significantly more complex than the ones of Sect. 3.4 and Sect. 3.5.

3.6.1 Collision Security

Theorem 5. *Let $n \in \mathbb{N}$. Suppose $(\pi_1, \pi_2, \pi_3) \xleftarrow{\$} \mathrm{BC}[\Phi(A, B, \varphi^C)](n)^3$. Then,*

1. *If $B = 1$ and C arbitrary, $\mathbf{Adv}^{\mathrm{col}}_{\mathsf{F}_{\mathrm{SS}}}(2^{(n-|C|)/2-n\varepsilon}) \to 0$ for $n \to \infty$;*
2. *If $B = 2$ and C with $|C| \leq n/2$, $\mathbf{Adv}^{\mathrm{col}}_{\mathsf{F}_{\mathrm{SS}}}(2^{n/2-n\varepsilon}) \to 0$ for $n \to \infty$;*
3. *If $B = 2$ and C with $|C| > n/2$, $\mathbf{Adv}^{\mathrm{col}}_{\mathsf{F}_{\mathrm{SS}}}(2^{n-|C|-n\varepsilon}) \to 0$ for $n \to \infty$;*
4. *If $B \geq 3$ (independent of n) and C arbitrary, $\mathbf{Adv}^{\mathrm{col}}_{\mathsf{F}_{\mathrm{SS}}}(2^{n/2-n\varepsilon}) \to 0$ for $n \to \infty$.*

Due to the technicality of the proof, the results are expressed in asymptotic terms.

3.6.2 Preimage Security

Theorem 6. *Let $n \in \mathbb{N}$. Suppose $(\pi_1, \pi_2, \pi_3) \xleftarrow{\$} \mathrm{BC}[\Phi(A, B, \varphi^C)](n)^3$. Then,*

1. *If $B = 1$ and C with $|C| \leq n/2$, $\mathbf{Adv}^{\mathrm{epre}}_{\mathsf{F}_{\mathrm{SS}}}(2^{n/2-n\varepsilon}) \to 0$ for $n \to \infty$;*

2. *If* $B = 1$ *and* C *with* $|C| > n/2$, $\mathbf{Adv}^{\mathrm{epre}}_{\mathsf{F}_{\mathrm{SS}}}(2^{n-|C|-n\varepsilon}) \to 0$ *for* $n \to \infty$*;*

3. *If* $B \geq 2$ *(independent of* n*) and* C *arbitrary,* $\mathbf{Adv}^{\mathrm{epre}}_{\mathsf{F}_{\mathrm{SS}}}(2^{n/2-n\varepsilon}) \to 0$ *for* $n \to \infty$*.*

As for collision resistance, the results are expressed in asymptotic terms. The bounds match the ones in the IPM, except for the case of $B = 1$ and $|C| > n/2$. We leave it as an open problem to prove tightness of Thm. 6 part (2).

3.7 Conclusion

Within this chapter, the Weak Cipher Model (WCM) has been presented. This model allowed to assess to what extent known-key attacks on blockciphers influence the security of cryptographic functions that are based on these known-key blockciphers. The latter includes block cipher-based and permutation-based hash functions. By applying the WCM model, we could prove that the PGV compression functions, the Grøstl compression function, and the Shrimpton-Stam compression function remain mostly unaffected by the generalized weakness. Additionally, preimage security of the functions turned out to be significantly less susceptible to these types of weaknesses than collision security. The results can be readily generalized to other primitive-based functions, such as the permutation-based sponge mode.

Chapter 4

Lightweight cryptographic post-processing

4.1 Introduction

When designing security solutions for resource-constrained devices, there is a strong focus on efficient cryptographic primitives. Efficiency can be defined in various ways. For some applications, it could mean low area or low memory footprint. In other applications, the focus on efficiency could mean that one aims at minimizing the energy consumption or latency. This work has been addressed in task 3.3 of the HECTOR project, where research has been carried out on efficient implementations of cryptographic algorithms for constrained devices, with a particular focus on AEAD and the ongoing CAESAR competition. These results have been summarized in deliverable D3.1 [82]. However, the traditional approach in designing symmetric key primitives is to consider in first instance the robustness of these primitives in a mathematical context (i.e. perform cryptanalysis) and to study its performance requirements. During this analysis, one just assumes that the symmetric key primitives have a random key as input which is stored "somewhere". Where this key exactly comes from, is out of scope of the design and cryptanalysis. The only assumption that is made, is that the key is completely random and does not contain weaknesses, such as dependencies on other (sub)keys or a lack of entropy.[1]
Within this chapter, we now want to zoom in one level deeper and take into account the origin of the symmetric key. This key could be stored in secure NVM (non-volatile memory), but it could also originate from a PUF or a TRNG. Let us now focus on the latter two cases: i.e. the key from a PUF or TRNG is feeded directly into a lightweight cryptographic algorithm. Note that this excludes security schemes in which the PUF key is not used directly as the input of a cryptographic algorithm, but rather as a memory encryption key to securely store cryptographic material generated externally. Furthermore, for the sake of clarity, let us also assume that the cryptographic key is used as input for a **lightweight encryption function**. Note that in the discussion below, this encryption part can be replaced by any lightweight symmetric-key primitive, such as a MAC or an authenticated encryption scheme, in a straight forward way. A simplified high-level overview is depicted in Figures 4.1 and 4.2. As shown in these figures and studied in WP2, PUFs and TRNGs need to be combined with additional post-processing to guarantee their robustness and security. This post-processing is typically a combination of algorithmic post-processing (e.g. error-correcting codes (ECC), Von-Neumann's post-processing, etc.) and cryptographic post-processing (e.g. cryptographic hash function, encryption function, etc.). Although this post-processing is needed to ensure properties such as

[1] This has been extensively discussed in the two previous chapters of this deliverable.

unpredictability, it obviously comes with a cost. This leads to the observation that (part of) the efficiency that is gained by using lightweight cryptographic primitives could be lost due to the cost of this post-processing. Even more, only optimizing the efficiency of lightweight cryptographic primitives without considering this post-processing will never result in lightweight system. *Therefore, the research question we wanted to tackle in the HECTOR project, and which will be discussed further in this chapter, is how to optimize the efficiency of the overall system.*

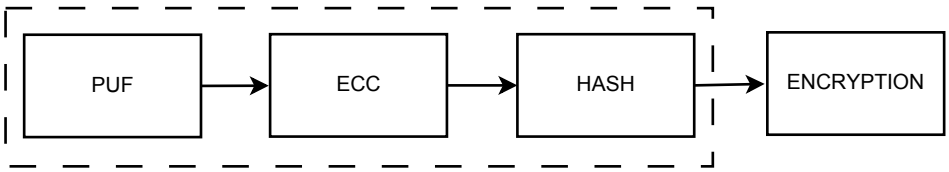


Figure 4.1: PUF is used as input key for cryptographic algorithm.

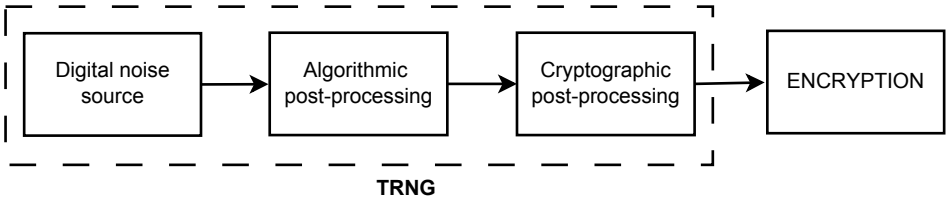


Figure 4.2: TRNG provides input key for cryptographic algorithm.

4.2 Optimization by reusing cryptographic primitives

The optimization we propose in the HECTOR project is to combine the cryptographic post-processing used in PUFs and TRNGs with the lightweight symmetric-key algorithm which makes use of the cryptographic key. This obviously results in an efficiency gain, more particularly to a lower footprint of the chip, as a single cryptographic primitives could be reused within the same system. The optimized system is depicted in Figure 4.3 and 4.4 for PUFs and TRNGs respectively. The next question that now needs to be solved, is which primitives could be reused when combining the cryptographic post-processing and the lightweight cryptographic algorithm (in our example the encryption function). Before we can answer this, we first need to discuss the cryptographic post-processing used in PUFs and TRNGs respectively.

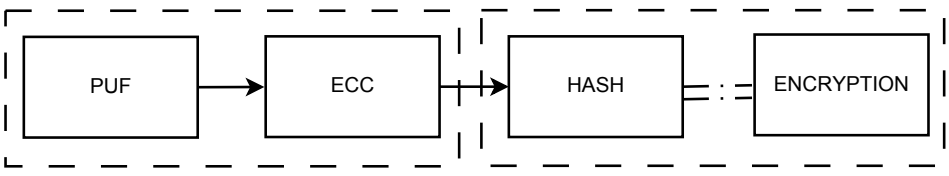


Figure 4.3: PUF with cryptographic post-processing and encryption combined.

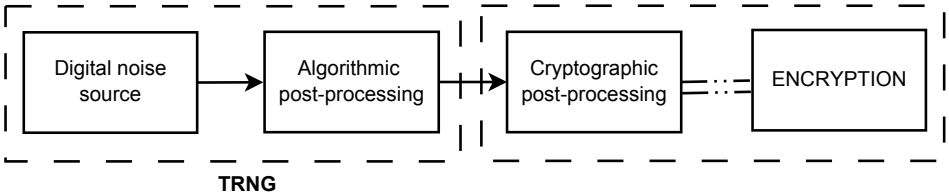


Figure 4.4: TRNG with cryptographic post-processing and encryption combined.

4.2.1 Cryptographic post-processing for PUFs

Typically, the outgoing bits of a PUF, after applying ECC, have non-maximum entropy. The entropy of a PUF response r is non-maximum because of two reasons: (1) the correlations and bias of the PUF and (2) additional entropy loss by the helper data algorithm, referred to as leakage. Indeed, helper data unavoidably leaks information about the PUF response. A compression step ensures the key k to be nearly uniform. This step is also referred to as privacy amplification. The total amount of entropy is preserved, but the bit-average increases by having more input than output bits (i.e. compress the input bits). The well-established solution is to apply a cryptographic hash function. One computes the key of the PUF as follows: $k = Hash(r)$.

4.2.2 Cryptographic post-processing for TRNGs

The AIS 31 methodology [65,66] adds mandatory cryptographic post-processing to the TRNG to ensure unpredictability of the generated numbers in forward and/or backward direction during a permanent or temporary failure of the entropy source for the highest security levels. Furthermore, in case of such a failure, the cryptographic post-processing of the TRNG can temporarily serve as a deterministic random number generator (DRNG). The cryptographic post-processing block should use an approved cryptographic algorithm depending on the required security level (direction of unpredictability) and according to the rules defined for cryptographically secure pseudo-random number generators.
One of the most popular and most robust postprocessing technique is to run the output of a TRNG design through a cryptographically strong hash function. When high security levels need to be achieved, then a hash function is not sufficient. If the TRNG must be AIS 20/31 Class PTG.3 compliant, for example a design requirement of demonstrator 1 in WP4, then the cryptographic post-processing must be DRG.3 compliant. One can always use a DRNG for this purpose [41], as for example shown in Figure 4.5. We refer to AIS 31 for the full requirements on having a DRG.3 compliant cryptographic post-processing, such as forward secrecy and enhanced backwards secrecy [66]. These requirements result in a more complex post-processing algorithm. However, it can be constructed using a cryptographically secure block cipher.

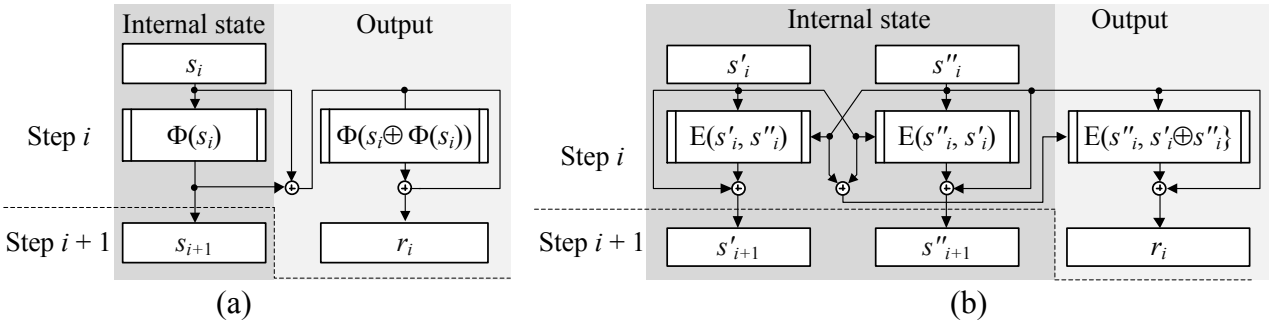


Figure 4.5: DRNG with forward and enhanced backward secrecy: (a) when using a one-way function; (b) when using a block cipher behaving as a one way function.

4.3 How to integrate cryptographic post-processing and lightweight symmetric-key crypto

A first optimization strategy would be to **reuse a lightweight block cipher**. Indeed, many designs for cryptographic hash functions and MAC functions are solely based on a block cipher (such as AES). Therefore, by only using these block cipher-based designs, it is perfectly feasible to design cryptographic post-processing for PUFs and TRNGs that relies exclusively on this block cipher. This block cipher is then obviously also reused in the encryption function that uses the input of the PUF/TRNG as the cryptographic key.
However, one can optimize the overall system even further by using a sponge construction. A sponge is a mode based on a fixed permutation as underlying primitive that can handle arbitrarily long input and output sizes. We propose to use a **sponge(duplex)-like construction**[2] where one can easily combine the functionality of the cryptographic post-processing and the symmetric-key primitive that needs to use the key (in our example the encryption function). More specifically, our solution makes use of the motorist construction, an improved mode of operation, proposed by the authors of Keyak [9], which extends the plain duplex construction of a sponge primitive. Below, we will briefly recap the motorist-layer construction (see D3.1 [82] for more details) and then explain the proposed sponge-construction which enables the combination of cryptographic post-processing and the symmetric-key primitive.

4.3.1 Motorist-layer construction

The Motorist mode has been introduced in [9] in order to specifically target AEAD built on top of sponge-based cryptographic primitives. The Motorist mode allows to encrypt and to guarantee the authenticity of sequences of messages in sessions (rather than only a single message). A message consists of a plaintext and possibly some associated data. The main advantages of the motorist construction is that it is a mode that can be built on top of any generic cryptographic sponge primitive and it improves the overall efficiency. The motorist construction is universal, as it can be applied to any fixed-length permutation with sufficient width. One of the main differences between the motorist construction and the original duplex construction proposed for sponges, is that the motorist construction uses the full size of its underlying permutation to process data, rather than only a part (denoted by the parameter bitrate r).

[2]The duplex mode [10] is a particular way of using the sponge construction. It allows the alternation of input and output blocks at the same rate as the sponge construction.

We refer to [9,82] for more details on the motorist construction. For the discussion in this chapter, it is important to note the motorist construction works in sessions. Within the motorist mode, one can compute a message authentication tag over the full sequence of messages sent/received since the start of the session. To start a session, the motorist construction takes as input a secret and unique string, denoted by *SUV*. The SUV plays the classical role of the secret key and initialization vector. Once this SUV is "absorbed", key stream bits (for encryption) and a message authentication tag (over the entire session) can be computed. Since the motorist construction is based on the original duplex construction, it inherits its interesting features. For example, the encryption/decryption process coincides with the absorbing operation of the sponge construction, so no buffer is needed and plaintexts/ciphertexts can be processed as they arrive.
It is interesting to note that one of the instances of a sponge construction is exactly a hash function. During the absorption phase, the input is "absorbed" into the sponge, block by block. When used as a cryptographic hash function, the output is generated during the squeezing phase. This is shown in Figure 4.6. Therefore, the absorption phase of the sponge can be used to hash the SUV, which is the input of our sponge function. But instead of just "squeezing" the output bits, as would happen in the hash function, we can compute key streams bits based on the hashed SUV, by applying the motorist layer construction. This corresponds to the use of a stream cipher with a key equal to the hashed SUV. This is exactly what we need for our cryptographic post-processing and lightweight encryption function.

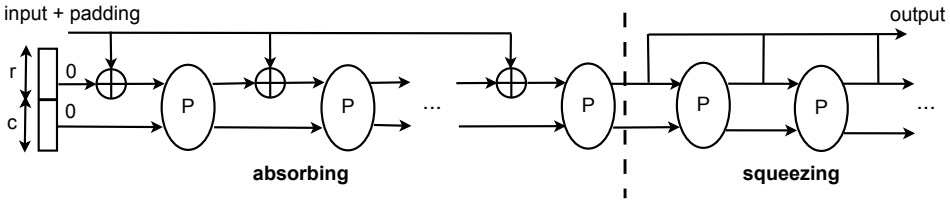


Figure 4.6: General sponge construction [9].

4.3.2 Duplex-sponge construction

The overall system view is depicted in Figure 4.7. On the left hand side, we have either a PUF combined with an ECC, or a TRNG combined with algorithmic post-processing. The output of this building block is denoted by x. We apply padding to this output and split it in blocks $x_0, x_1, \ldots, x_n$. The padding scheme and size of the blocks x_i depends on the choice of the permutation P chosen in the sponge construction (see below). These blocks are used as input for the sponge construction, together with the plaintext $P_0, P_1, \ldots, P_n$. The output of the sponge construction will be the ciphertexts $C_0, C_1, \ldots, C_n$. Each P_i and C_i has a length of 128 bits, besides the last blocks P_n and C_n.

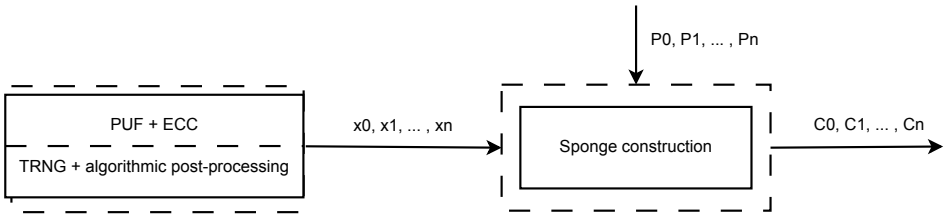


Figure 4.7: Overall system view.

The schematics of the proposed sponge construction is shown in Figure 4.8. It consists of two phases, respectively left and right of the dotted line in the figure. First, the initialization takes place, where the PUF/TRNG output x is absorbed in the sponge. This secret value x takes the role of the SUV in the motorist construction, as explained above. This initialization step is basically a hashing operation. Once the PUF/TRNG output is hashed, it is combined with a nonce, which is used as the initialization vector of the encryption algorithm. The encryption algorithm basically generates key stream bits needed for the encryption of the plaintexts. After each permutation P, a new block of keystream bits K_i is generated. The ciphertext C_i is computed by xoring P_i and K_i.
The proposed sponge construction can be applied using any secure fixed-length permutation with sufficient width. For example, one could use the Keccak, Keyak, ASCON or PRIMATEs permutation, each of them with 12 rounds[3]. However, since we are aiming for a lightweight construction, the Keccak permutation with a width of 400 bits might be an interesting option.

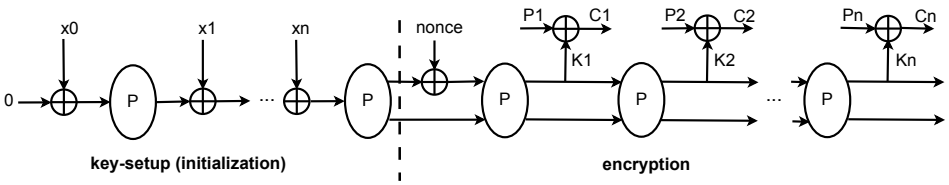


Figure 4.8: Sponge construction combining cryptographic post-processing and encryption.

It is important to note that the sponge construction described above is a deterministic process. When using the same input x and initialization vector, it will generate the same key stream bits K_i. Therefore, if two different devices need to perform cryptographic operations using the same key (e.g. respectively encrypting and decrypting data), then both devices need to store the secret value x. This value will be a non-ideal random number and/or will not have non-maximum entropy. However, both devices will apply the sponge construction to compute a hashed SUV, which will then act as an internal key to compute exactly the same key stream bits K_i in both devices. The same story holds for multiple readouts of the PUF. Due to the post-processing, the secret value x – having non-maximum entropy – will be reconstructed during each readout. This value x will then be initialized in the sponge function to an internal key, which is then used to generate key stream bits K_i. An additional advantage of this approach, is that the PUF key is never stored in memory during the entire encryption process. This key only exists as an internal value, i.e. as the output of one of the permutations P during the execution of the sponge function. This offers additional side channel protection.

[3]See D3.1 [82] for more details on each of these algorithms.

4.4 Conclusion and other use cases

In the discussion above, we have shown that one can improve the efficiency by making use of the *hash-then-encrypt / hash-then-authenticate / hash-then-authenticated encrypt* functionality which is intrinsically built in the duplex(motorist) construction of a sponge function. We applied this concept to optimize the cryptographic post-processing of PUFs and TRNGs in case of using lightweight cryptography. However, there are other examples where one needs both a cryptographic hash function and an (authenticated) encryption function. For example, key derivation functions are often based on cryptographic hash functions. When this derived key is then fed into an (authenticated) encryption function, one could apply the same optimization strategy. Another important example are key agreement protocols. Again, many of these protocols compute a shared secret value (e.g. by applying the Diffie-Hellman protocol [33]), and then use this shared secret to derive a shared cryptographic key, which then can be used to encrypt and/or authenticate data. Again, this can be optimized by directly applying the sponge construction, discussed above, on the shared secret value.
To demonstrate its feasibility, we applied the proposed optimization strategy in demonstrators 2 and 3 of the HECTOR project (see the deliverables in WP4 for more details). In both these demonstrators, the output of a PUF and a passphrase - entered by the user - need to be combined to generate a unique key. Instead of first applying cryptographic post-processing and then using the derived key in the authenticated encryption function, the output of the PUF (without hash function) and the passphrase are used directly as input (i.e. SUV) of the authenticated encryption.
It is also important to note that our approach is not compliant to the current version of the AIS 31 standard. This standard imposes the restriction that the same cipher should not be used for data enciphering and cryptographic post-processing. Because of this reason, our optimization strategy cannot yet be applied in commercial applications that need to be AIS 31 compliant, which hence could limit the immediate commercial adoption. Compliance to the existing standards is exactly the reason why we applied conventional cryptographic post-processing techniques for the TRNGs in our demonstrators, instead of this optimized solution based on a sponge construction. However, the research insights presented in this chapter could be useful inputs for future versions of the standard, which might accept this sponge-constructions for lightweight cryptography.

Chapter 5

Security degradation of side channel countermeasures

In this chapter we introduce our proposed framework to analyze the security degradation of side channel countermeasures in the presence of non-ideal random numbers. We begin by giving some background on how a typical side channel attack works and a brief overview of countermeasures that have been proposed in the literature. These countermeasures are the ones which will be evaluated in Chapter 6 (using side channel measurement simulations) and in Chapter 7 (using real measurements from implementations on the HECTOR boards). Next, we describe our testing approach proposal which uses as degradation metric the number of measurements required for an attack to succeed. We also provide here a description of the experimental setup used in Chapter 7. Lastly, we describe the different sets of non-ideal random numbers that are used through our study. We employ both synthetic sets, generated by custom scripts according to certain criteria, as well as real sets, obtained by altering the environmental conditions of a TRNG implementation running on the HECTOR board.

5.1 Background

Side channel attacks encompass any type of attack against cryptographic implementations that exploits information emanated through physical channels. Classical examples of exploitable side channel sources are the execution time [69], the power consumption [70], and the electromagnetic radiation [44]. All these measurable quantities leak information about the internal operations performed by an implementation. Side channel attacks exploit the relationship between leakage of internal operations and secret values processed by the implementations in order to extract secret cryptographic keys.

5.1.1 Differential side channel attacks

The main steps of a classic differential side channel attack as introduced in [70] are as follows. Consider an encryption algorithm $\mathbb{E}_K$ which on input a plaintext P returns a ciphertext $C = \mathbb{E}_K(P)$. The algorithm is implemented on an integrated circuit, e.g. an embedded processor, and parametrized by a secret cryptographic key K stored in the device. In order to mount an attack, the adversary collects a set of n side channel observations $(L_1, L_2, \ldots L_n)$ for different executions of the encryption algorithm when processing variable plaintexts $(P_1, P_2, \ldots P_n)$. In the following we assume the adversary monitors the instantaneous power consumption of the circuit by e.g. measuring the voltage drop over a shunt resistor connected to the device as

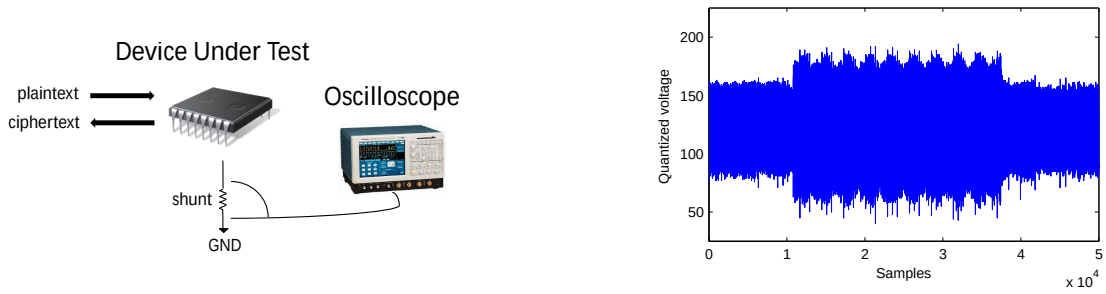


Figure 5.1: Experimental setup to measure power consumption of a device (left). Exemplary power measurement from an AES-128 execution on an embedded processor (right).

depicted on the left hand side of Figure 5.1. Using this setup, the adversary obtains a power measurement $L = (l_1, l_2, \ldots l_m)$ of m samples for each execution of the algorithm. For illustration purposes, an exemplary power measurement corresponding to a full execution of an AES-128 block cipher encryption on an embedded processor is shown in the right hand side of Figure 5.1. The 10 transformation rounds can be distinguished in the central repetitive patterns of the plot. This already indicates that the measurement carries information about the inner workings of the circuit. We note that similar side channel measurements can be obtained (and exploited) by monitoring the electromagnetic (EM) emanations of a circuit instead of its power consumption. This is, in fact, what will be done in Chapter 7.

For convenience, the set of collected side channel measurements can be seen as a matrix $\mathfrak{L}$ of size $n \times m$, where each row corresponds to a measurement L_i of length m samples. We denote as *sensitive* variables $v_{i,k}$ any intermediate computation which depends on known, variable data (e.g. a part p_i of the variable input plaintext P_i) and a guessable part k of the constant secret key K. The processing of these variables, which are the target of a side channel adversary, leak information through samples in the power measurement in the form $l = f(v_{i,k})$. Here f represents the leakage function of the device which depends, among other factors, on the value $v_{i,k}$ processed by the device at that particular time.

For the attack to be practical, it is necessary that the adversary chooses a target sensitive variable for which all possible values of k can be enumerated. This allows to compute a list of intermediates $\hat{v}_{i,\tilde{k}}$ for each possible key guess $\tilde{k} \in k$ and for each execution of the algorithm $1 \leq i \leq n$. The adversary uses next a *power model* g to characterize the leakage of the device. This allows to obtain a prediction leakage array of n samples for each $\tilde{k} \in k$, which we denote $\hat{l}_{\tilde{k}} = \hat{g}(\hat{v}_{i,\tilde{k}})$ for all $1 \leq i \leq n$. Once the leakage prediction arrays are computed, the last stage of the attack is to employ a *statistical distinguisher* to compare each $\hat{l}_{\tilde{k}}$ with the *columns* of the side channel measurements stored in the matrix $\mathfrak{L}$. The outcome of the comparison is a vector of m samples (one per column of $\mathfrak{L}$) for each key guess $\tilde{k} \in k$. For the correct key, it is expected that the resulting vector exhibits a maximum score in the sample(s) where $v_{i,k} == \hat{v}_{i,\tilde{k}}$. In contrast, the comparison should yield only low scores for the incorrect keys.

For illustration purposes, we show in Figure 5.2 the outcome of a classical single-bit Differential Power Analysis (SB-DPA) attack [70] against an AES-128 when selecting as sensitive variable the output of the first Sbox transformation in `SB`. The score vector corresponding to the correct key guess (left) yields clear peaks at the time samples where v_i is processed by the device, while this is not the case for an arbitrary wrong key guess (right).

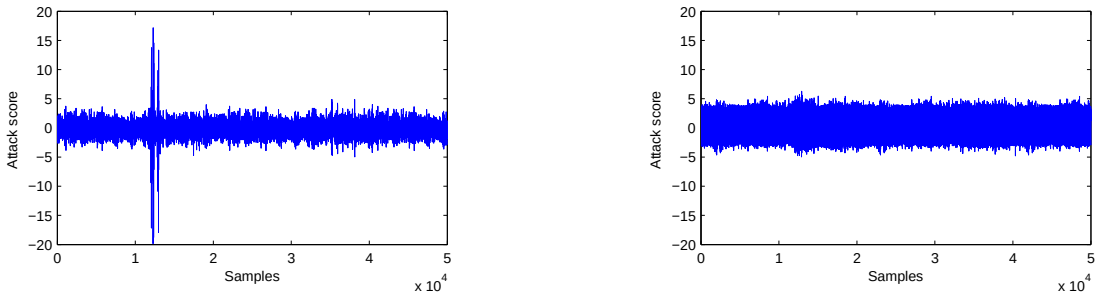


Figure 5.2: Result of classical SB-DPA attack against an unprotected AES Sbox implementation: scores for the correct key (left), scores for a representative wrong key (right).

5.1.2 Side channel countermeasures

Countermeasures against side channel attacks are often categorized into two groups [78]: *masking* and *hiding*. A description of each category follows.

Masking. Masking [31, 47] countermeasures aim to randomize the intermediate values processed by the implementation in order to make the leakage *independent* of sensitive variables. To achieve this all sensitive variables are randomly split into $d+1$ shares such that $v = m_1 \odot m_2 \odot \ldots \odot m_{d+1}$, where $\odot$ corresponds to a group operation (or a combination thereof) and d denotes the *order* of the masking protection. The security of masking implementations is achieved by operating on the so-called *shares*, i.e. $m_1 \ldots m_{d+1}$, instead than on the sensitive variable v.

Imagine the case of $1st$-order masking with $d = 1$, where sensitive variables are split as $v = m_1 \odot m_2$. The masked implementation will leak information about $l_{m_1} = f(m_1)$ and $l_{m_2} = f(m_2)$, but not about l_v. This characteristic inherently prevents mounting the previously illustrated SB-DPA attack: since m_2 is a fresh random share re-generated at each execution, the variable $m_1 = v \odot m_2$ leaks no information about v. Because of this property $1st$-order masking provides security against any type of *univariate* attack, i.e. attacks that perform a sample-wise analysis in the leakage measurements by applying the distinguisher directly to the columns of $\mathfrak{L}$.

However, it is possible for the adversary to improve the attack by combining leakage samples from the two shares, i.e. from two columns of $\mathfrak{L}$. This is referred to as a *bivariate* attack, and it requires a pre-processing stage in which, for each measurement $1 \leq i \leq n$, the leakages l_{m_1} and l_{m_2} are combined into a new value l' prior to mounting the attack. There are many proposals on how to combine leakage points, for instance, using the absolute difference function $l' = |l_{m_1} - l_{m_2}|$ as proposed in [87] or the normalized product combining function $l' = (l_{m_1} - E\{l_{m_1}\}) \times (l_{m_2} - E\{l_{m_2}\})$ proposed in [98], where $E\{\cdot\}$ is the sample mean across all n measurements.

Combining multiple leakage samples per measurement can therefore overcome the security provided by a masking scheme. In general, a d-order masking scheme can always be broken by a $d+1$-variate side channel attack that combines information of $d+1$ shares. Despite this, it has been shown that the complexity of mounting such attacks grows exponentially with the masking order d [31]. Moreover, finding the correct leakage samples to combine is not a trivial task and increases the computational complexity of the attack. These characteristics make masking a sound countermeasure and explains its popularity in the side channel literature.

Hiding. Hiding countermeasures [78] aim to remove data dependencies in side channel leakage by introducing *noise* in the measurements. This can be achieved by altering the leakage characteristics of the device (vertical noise) or by modifying the temporal occurrence of internal computations (horizontal noise).

Vertical noise can be incorporated by running operations in parallel to the cryptographic implementation (thus lowering the exploitable information signal available to the the adversary) or by using dedicated logic styles which make power consumption independent of the processed data (see e.g. SABL [108] or WDDL [109]). Horizontal noise can be inserted by randomizing the expected sequence of internal computations (e.g. by adding dummy operations or shuffling order of of operations) or by manipulating the clock signal (e.g. switch between multiple frequencies, increase jitter, skip clock cycles, etc.).

In general, the insertion of hiding countermeasures to an implementation will increase the complexity of mounting an attack in terms of number of measurements. However, and in contrast to masking, adding noise is not a sound countermeasure by itself as it does not fully prevent the presence of exploitable leakage in the measurements.

5.2 Testing Framework

The proposed testing approach to analyze the security degradation of side channel countermeasures in the presence of non-ideal random numbers is depicted in Figure 5.3. We assume an adversary that targets an implementation $\mathbb{E}_K$ equipped with countermeasures which consume randomness. For the purposes of our study, we model the random number sequences used by the countermeasures as an external input R to the cryptographic implementation. One of the goals of the work in task 3.2 is to obtain a metric that captures the security degradation of the countermeasures. To this end, we propose to use the *minimum number of measurements* necessary for the attack to succeed as metric to quantify the adversarial effort.

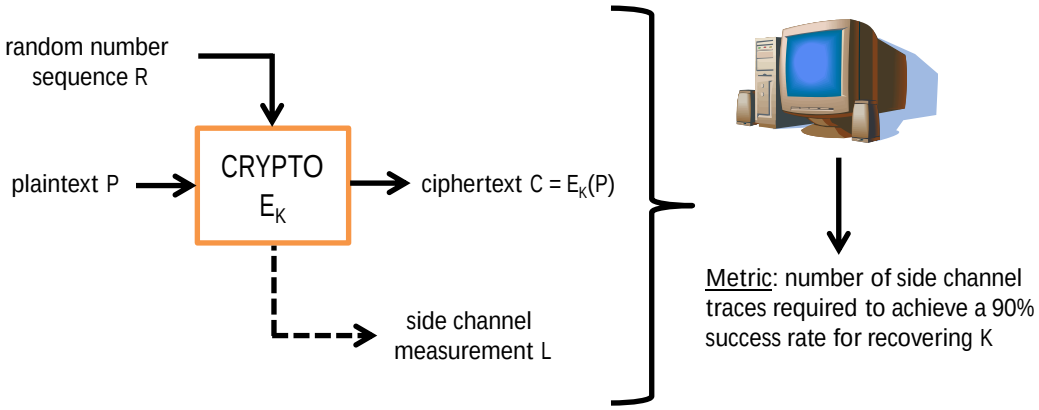


Figure 5.3: Our proposed testing approach.

The proposed framework allows to study several scenarios. First and most important, we can tune the randomness sets R used by the encryption engine to analyze multiple cases: the ideal situation (random numbers are drawn from a uniform distribution), the worst situation (random numbers are set to zero, i.e. equivalent to an unprotected implementation) and essentially all situations in between (random number sequences contain any type of non-ideality). Second, the

genericity of the framework allows to accommodate different type of attacks as well as to analyze any countermeasure that demands randomness. More specifically, we can instantiate any type of $d+1$-variate attack against any masking protection of order d, as well as any type of hiding-based countermeasures. And third, it can be used both with simulated measurements (using a mathematical model to characterize the leakage function f of a device) and real measurements obtained from an actual implementation (where f is determined by the implementation and/or platform under attack). The former approach will be used in Chapter 6, while the latter will be used in Chapter 7.

5.2.1 Experimental setup with HECTOR board

In what follows we describe the experimental setup that will be used in the evaluation of Chapter 7. The schematic representation of the setup is shown in Figure 5.4. We opt to monitor the side channel leakage of the target of evaluation (TOE) by measuring its electromagnetic (EM) emanations. To this end, the setup is placed inside a Faraday cage that minimizes the influence from external EM sources (e.g. GSM phone signals).

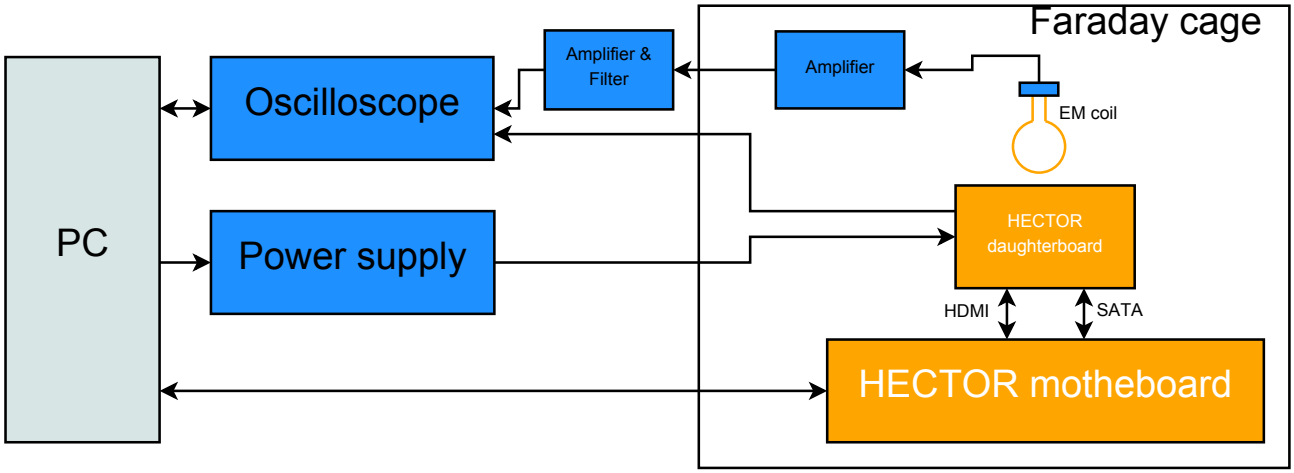


Figure 5.4: Side channel analysis measurement setup used for HECTOR experiments.

The TOE, which corresponds here to a crypto engine implemented in the HECTOR daughterboard, is connected to the HECTOR motherboard either directly through the SATA connector or indirectly using an HDMI cable. A pickup coil is used to measure the EM emanations from the TOE and is connected to an amplifier. A PC is used to control the measurement setup and uses an oscilloscope to digitize the signals obtained from the TOE. We additionally connect a low noise power supply to the input voltage of the TOE. The components used in the measurement setup are listed in Table 5.1.

Description	Manufacturer	Type
Power supply	Agilent	E3631A
Oscilloscope	Teledyne Lecroy	WaveRunner 620Zi
Faraday Cage	Tescom	TC-5970A
RF Amplifier	RF Bay	LNA-1800

Table 5.1: List of components used in the measurement setup.

Figure 5.5 shows a detailed view of the amplification and filtering circuitry used to condition the EM signal picked up by the coil. It is composed by two low noise amplifiers: the first one is placed as close as possible to the EM coil (inside the Faraday cage) in order to reduce the influence of the ambient noise. The second one is located outside the Faraday cage. The signal is then split in order to get a raw version of it and a filtered one. The filter used for this test was selected in order to get the best out of the EM signal picked up by the coil, while ensuring that the frequency of the clock used by the device lied in that frequency range. The selection of antenna and filtering was done after some preliminary experiments with different combinations (for instance using different antennas or different filters; or none) on an unprotected implementation of the Data Encryption Standard (DES) engine, which will be the algorithm implemented in Chapter 7. A close-up of the loop antenna used during the practical experiments is shown in Figure 5.6.

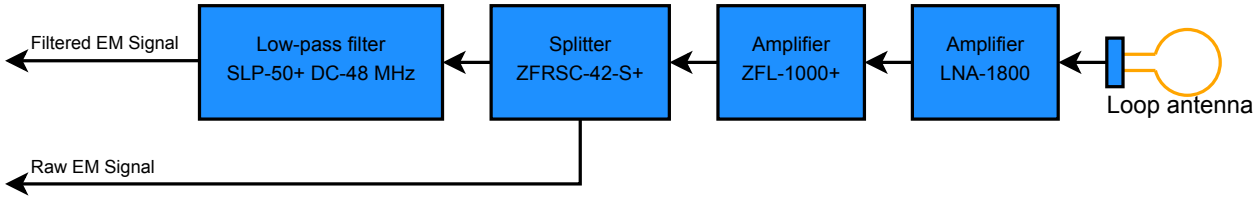


Figure 5.5: Close look at the amplification and filtering used during the practical measurements.

Figure 5.6: Picture of the loop antenna used during the practical measurements.

Figure 5.7 shows a high level representation of the communication interface between the PC and the HECTOR motherboard and between the HECTOR motherboard and the HECTOR daughterboard. This communication framework was built to give flexibility and ease to switch from one target to another target (a target being a different type of countermeasure for example).

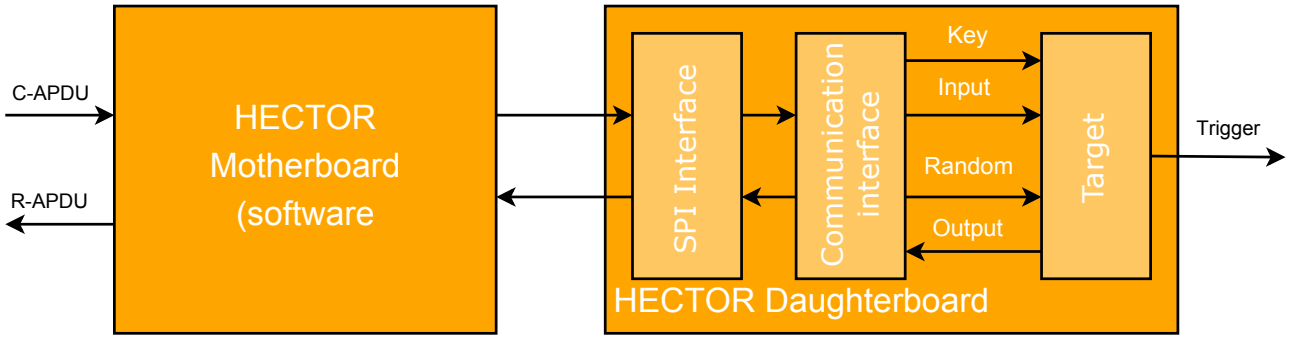


Figure 5.7: Communication framework used in the experiments.

The HECTOR motherboard implements an APDU (Application Protocol Data Unit - usually used for smartcard communication) parser programmed in software. The software then transforms these APDUs into commands recognizable by the communication interface implemented in the HECTOR daughterboard (read and write commands are available with different parameters) and transfers them through SPI to the daughterboard. The communication interface then, depending on the parameters provided with the commands, set or read the key, input and random used by the target or enable the target to perform an operation. Finally a trigger signal is available on the daughterboard that indicates the start and the end of operation of the target. This trigger signal is mainly used to speed up the experiment and get an efficient reference trigger for the oscilloscope to record the EM traces. The entire HECTOR daughterboard logic is implemented in VHDL and the targeted HECTOR daughterboard contains a Xilinx Spartan-6 FPGA.

5.3 Generating non-ideal random numbers

In this section we describe the sets of non-ideal random numbers that we use in our study. We employ two classes of sets: synthetic and real. Synthetic sets are generated by custom scripts which introduce non-idealities to a proper random number sequence. In contrast, real measurements are obtained from an actual RNG implementation where non-idealities are introduced through environmental modifications. A detailed description of each class, as well as a motivation of their suitability to the study in task 3.2, is given in the next subsections.

Note that, in an ideal situation, we would like to have a metric that allows us to quantify the quality of randomness in the same way that we use the number of measurements to capture the security degradation. The most commonly used metric in security applications is the min-entropy, which is directly linked to the success probability of an attacker using the optimal guessing strategy. However, for evaluating impact of imperfect randomness on the security of side channel countermeasures, this metric is too general. This is simply because two randomness sources with the same min-entropy level can have a vastly different impact on the success probability and the required adversarial effort. We therefore abstain from using min-entropy as metric in our study and use an alternative approach as described next.

5.3.1 Synthetic sets

Synthetic sets can be generated by inserting non-idealities to a proper random sequence. There are however many ways in which such non-idealities can be defined. Our approach is to link them to the statistical tests defined in AIS 31 [65,66] which can detect several type of weaknesses

in random data. To this end, we have created different scripts that aim at introducing one type of weakness and therefore fail a particular AIS 31 test.

We argue that this approach allows us to gain more insight into the security degradation of countermeasures than using arbitrary imperfections. This is because we can use the results of our study to determine which type of weaknesses are most severe when it comes to side channel countermeasures.

Monobit bias. The first family of synthetic sets contain the most straightforward type of defect that can appear in random number sequences, namely, *monobit bias*. For a truly random sequence one naturally expects the number of ones and zeros to be approximately the same. The verification of this property corresponds to the first statistical test in AIS 31 [65, 66], i.e. the monobit test (T1).

Biasing a sequence can simply be done by favouring the apparition of certain bits. We created a script that takes a proper random bit sequence and artificially biases it by performing a uniform replacement of ones by zeros (or viceversa). A nice property of this family of sets, is that their non-ideality can be nicely quantified by the *bias level*, which can range from 0% (no bias) to 100% (fully biased). For illustrative purposes, we provide in Figure 5.8 the histogram plots (grouped at byte level) of a representative selection of sets biased towards zero with levels spanning from 0% to 25%, in increments of 5%. The complementary sets, i.e. biased towards one, can be obtained using the same script.

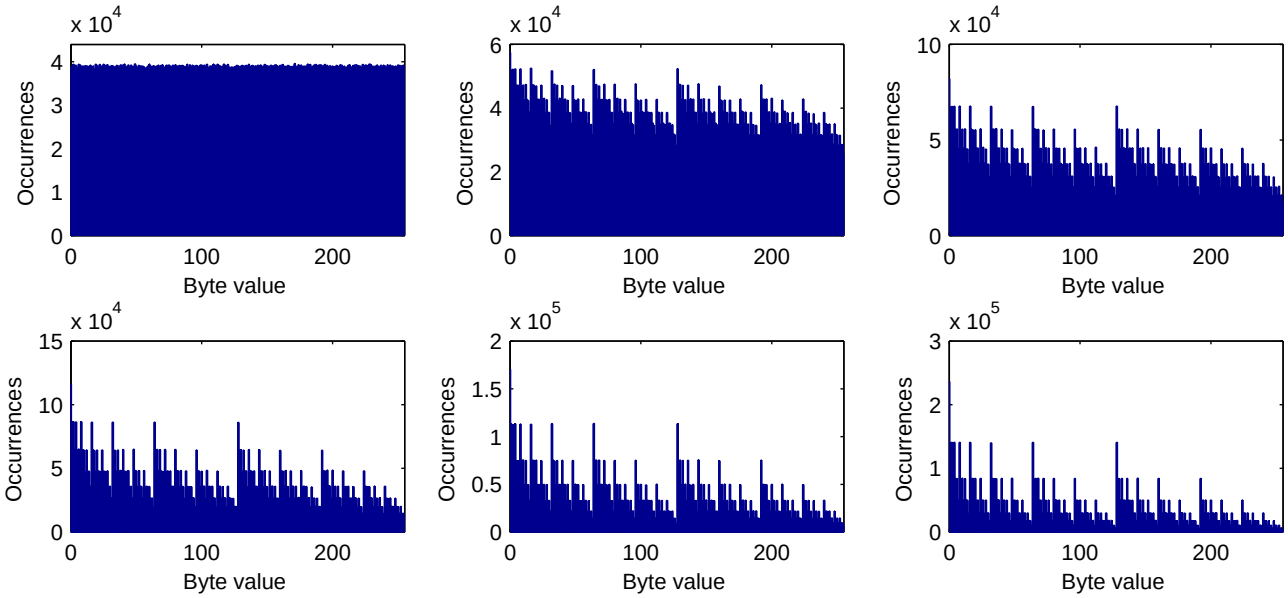


Figure 5.8: Histogram plots (byte grouping) of random sequences with different biases towards zero: 0% bias (top left), 5% bias (top middle), 10% bias (top left), 15% bias (bottom left), 20% bias(bottom middle), 25% bias (bottom right). Each sequence contains 10 million bytes. Note that plots are at different vertical scale.

Failure to other AIS 31 tests. In contrast to the monobit biased sets, in which the non-ideality of sequences can be easily captured with their bias level, quantifying the weaknesses from other statistical defects is more challenging. In order to overcome this, the non-ideality

level of the rest of synthetic sets is defined in a more generic manner. In particular, we define the following three levels: a sequence with *small* degradation will only fail the target statistical test, but pass the rest of tests; a sequence with *medium* degradation will fail the target statistical test most of the time (at least 90%), but other tests may also fail; lastly, a sequence with *large* degradation will always fail the target statistical test (100%), but naturally other test will also fail.

Our scripts take as input a proper random bit sequence (which would pass all the AIS 31 tests) and subsequently replace small parts of the sequence at random positions. For different types of weaknesses, alternative replacement methods are chosen. We concentrate on four different statistical tests from the AIS 31 suite: power test (T2), run test (T3), autocorrelation test (T5) and entropy test (T8). A description of each method follows.

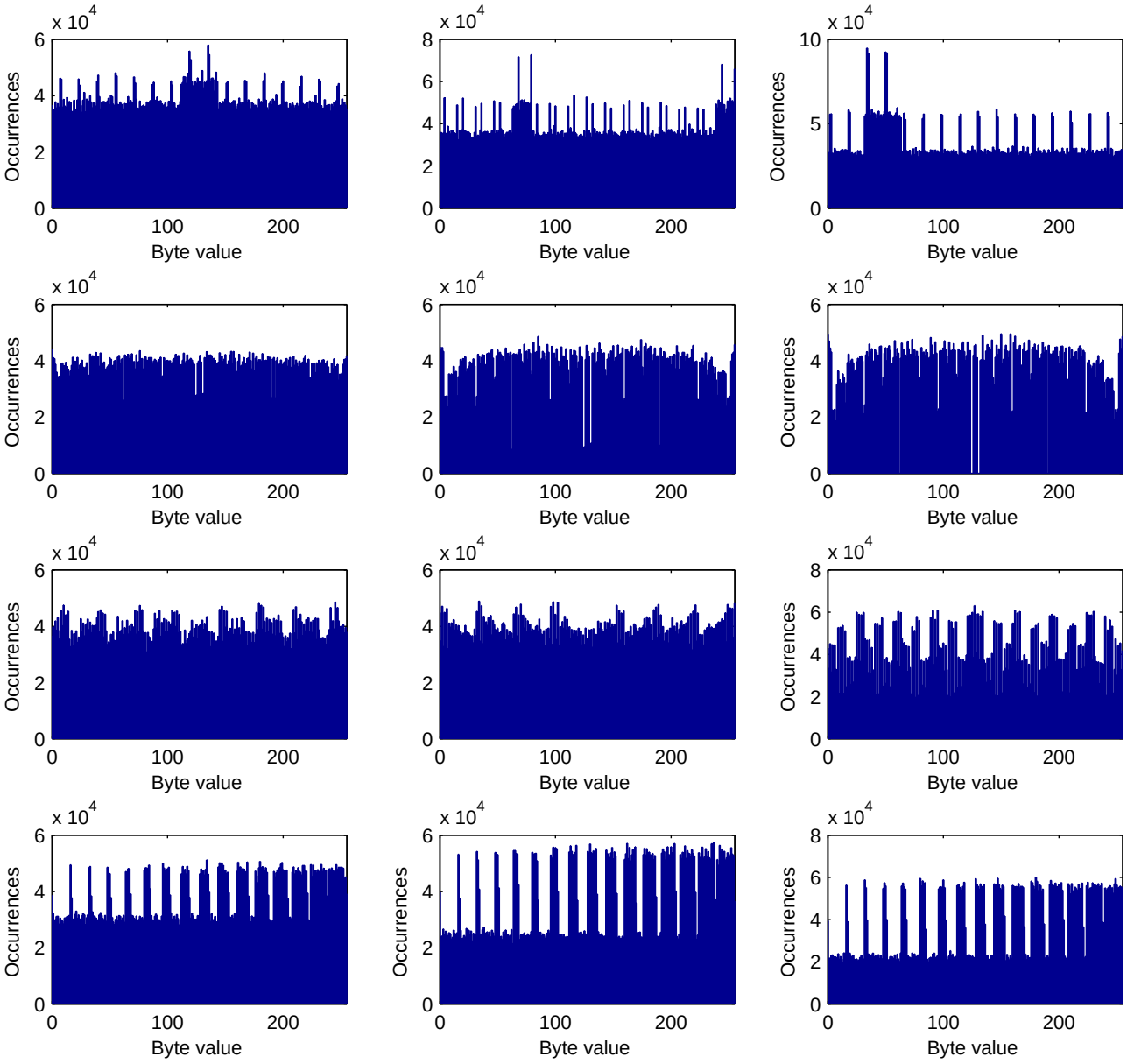


Figure 5.9: Histogram plots (byte grouping) of random sequences failing T2 (top), T3 (second from top), T4 (second from bottom) and T8 (bottom) for small degradation (left), average degradation (middle), high degradation (right). Each sequence contains 10 million bytes. Note that plots are at different vertical scale.

- Poker test degradation (T2):
  The method for degradation consists in the replacement of groups of four bits. The values to be inserted are chosen at random. Obviously the number of different values to be inserted should be (much) smaller than sixteen. The use of two different values turns out to work fine, as it reduces the impact on other tests. After some experiments, it turns out that it is beneficial if values zero and fifteen are avoided, as those will negatively influence the monobit test (T1) and the run test (T3). Histogram plots for of random sequences failing the poker test for different degradation levels are visible in Figure 5.9 (top).

- Run test degradation (T3):
  The method for degradation consists to remove or insert runs of a certain length. After some experimenting it turns out optimal to remove runs of length five as this does not harm the poker test results. Histogram plots for of random sequences failing the run test for different degradation levels are visible in Figure 5.9 (second from top).

- Autocorrelation test degradation (T5):
  An autocorrelation defect can be achieved by inserting a same sequence or inverted sequence later in the bit string. The latter works better to avoid influence on monobit and poker tests. Another improvement was achieved by skipping two out of four bits in the inserted sequence. Histogram plots for of random sequences failing the autocorrelation test for different degradation levels are visible in Figure 5.9 (second from bottom).

- Entropy test degradation (T8):
  The method used for this type of degradation consists in replacing the bytes 0xab in 0xba when $a \leq b$. Histogram plots for of random sequences failing the entropy test for different degradation levels are visible in Figure 5.9 (bottom).

5.3.2 Real sets.

The second class of sets used in our study are obtained by intentionally altering the environmental conditions of a TRNG implementation on an FPGA. More specifically, they correspond to the DC TRNG implementation on the HECTOR Spartan-6 daughter board described in Deliverable D2.2 and whose complete evaluation will be reported, together with the PLL TRNG, in Deliverable D2.4.

As part of the evaluation work within WP2, we have obtained 676 sets of 2 million bytes each with random data from the DC TRNG obtained at different combinations of temperature (from -40 degrees to 80 degrees, in steps of 10 degrees) and voltages (from 0.9 V to 1.4 V, in steps of 0.02 V). Half the sets correspond to the case where the TRNG implementation contains only the entropy source, while for the other half it contains also a post-processing module.

In order to keep the complexity of our study at a reasonable level, we have selected a representation of these sets to carry out our study. In particular, we use the data obtained at temperatures (-40,20,80) degrees and voltages (0.9,1,1.1,1.2,1.3,1.4) Volts. The histograms for these sets (once again grouped at byte level) are shown in Figure 5.10, Figure 5.11 and Figure 5.12, respectively, when the TRNG implementation contains only the entropy source. These sets are the ones that exhibit clear non-uniformity in their distributions, and therefore will be analyzed in the next chapter. We point out here that, as will be described in D2.4, the quality of the random numbers collected when altering the temperature/voltage conditions of the DC TRNG with

post-processing is rather good.

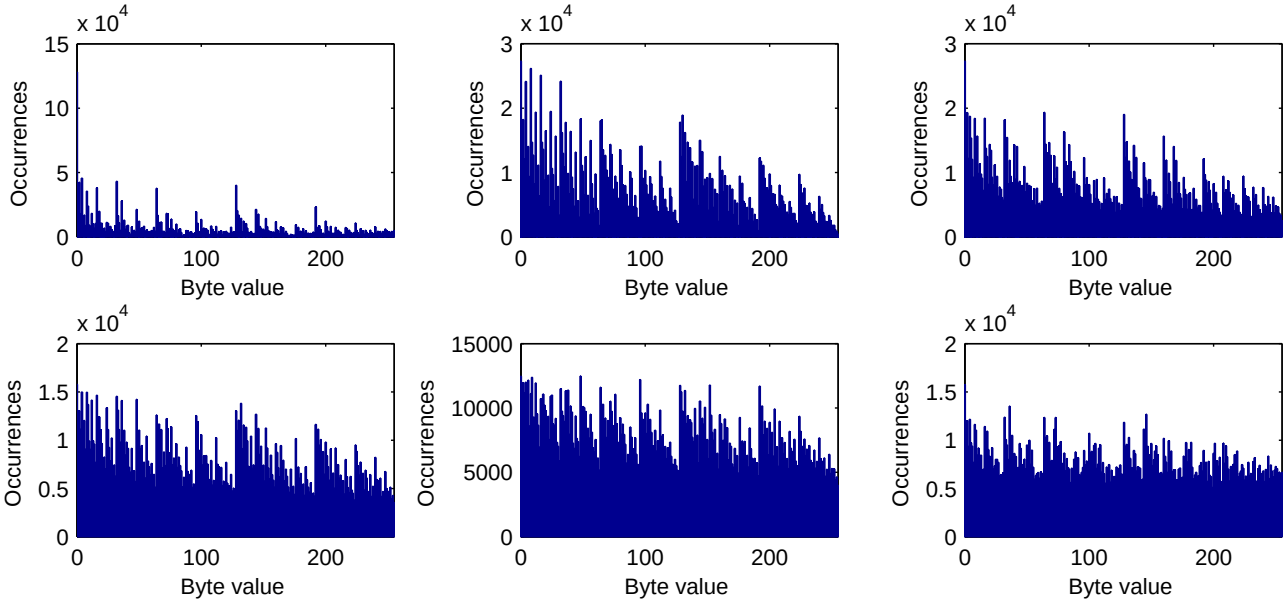


Figure 5.10: Histogram plots (byte grouping) of random sequences obtained at -40 degrees for different power supply voltages: 0.9 V (top left), 1.0 V (top middle), 1.1 V (top left), 1.2 V (bottom left), 1.3 V (bottom middle), 1.4 V (bottom right). Each sequence contains 2 million bytes. Note that plots are at different vertical scale.

5.4 Conclusions

In this chapter we have laid the foundations to carry out our study of the security degradation of side channel countermeasures. We have proposed and described a generic and flexible framework that allows to determine the effects of non-ideal random number sequences on different countermeasures. In addition, we have selected as metric to capture the security degradation the number of measurements necessary for the attack to succeed. Lastly, we have presented the different sets of non-ideal random number sets to use in the study and argued its suitability.

All these elements will be used in the next two chapters to perform the security degradation analysis. Chapter 6 will present a generic evaluation that uses side channel simulations in order to model the leakage of the device, while Chapter 7 will provide actual results when using side channel measurements obtained from protected cryptographic implementations running on the HECTOR board.

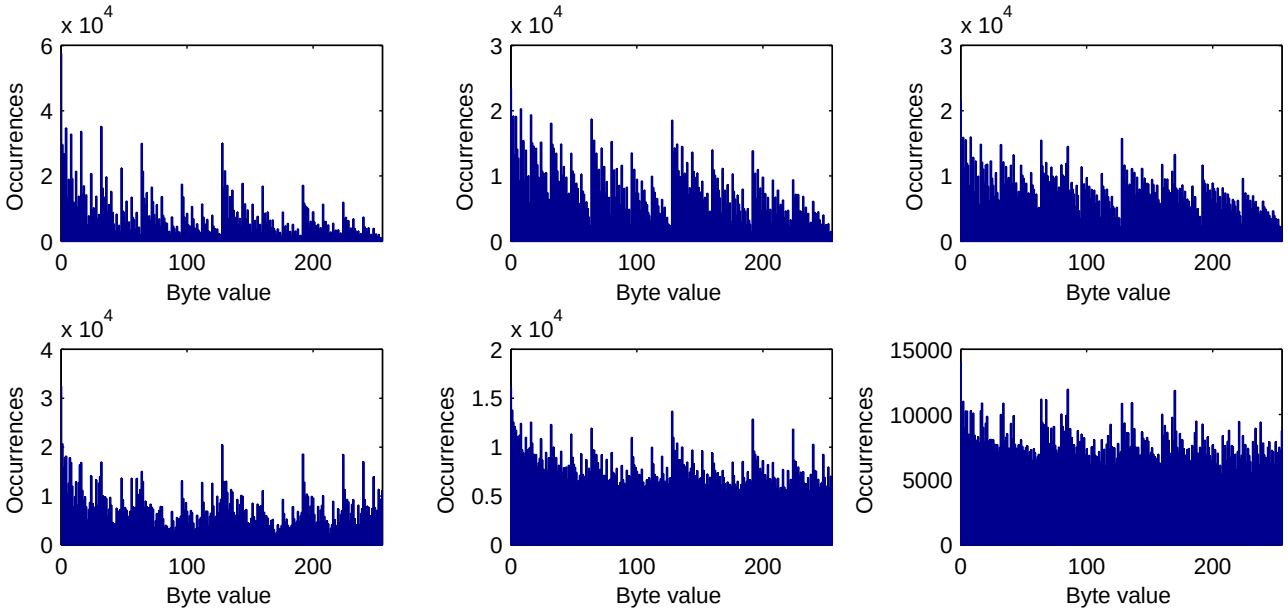


Figure 5.11: Histogram plots (byte grouping) of random sequences obtained at 20 degrees for different power supply voltages: 0.9 V (top left), 1.0 V (top middle), 1.1 V (top left), 1.2 V (bottom left), 1.3 V (bottom middle), 1.4 V (bottom right). Each sequence contains 2 million bytes. Note that plots are at different vertical scale.

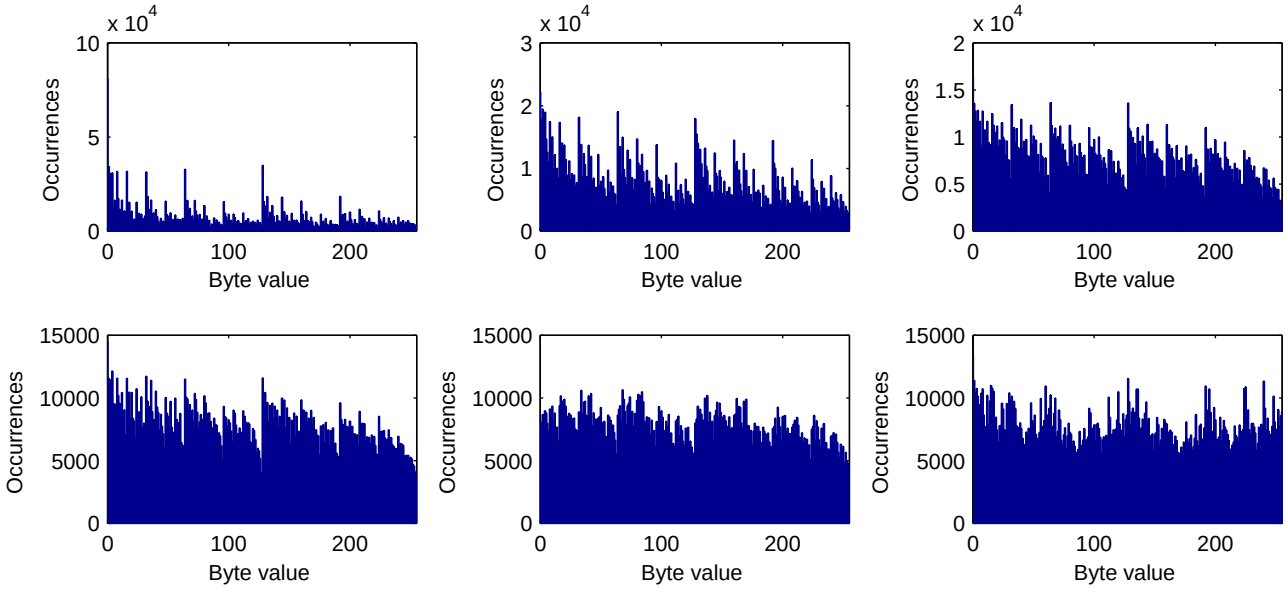


Figure 5.12: Histogram plots (byte grouping) of random sequences obtained at 80 degrees for different power supply voltages: 0.9 V (top left), 1.0 V (top middle), 1.1 V (top left), 1.2 V (bottom left), 1.3 V (bottom middle), 1.4 V (bottom right). Each sequence contains 2 million bytes. Note that plots are at different vertical scale.

Chapter 6

Simulation-based analysis of security degradation

In this chapter we provide the first part of the analysis of the security degradation of side channel countermeasures in the presence of non-ideal random numbers. Our testing approach follows the methodology laid out in Chapter 5, but we do not process yet real side channel measurements from an actual implementation. Instead, we use a mathematical model to characterize the leakage function f of an implementation and generate side channel simulated measurements. This allows us to provide a more generic analysis of the security of both masking and hiding countermeasures. In order to verify the soundness of our testing framework, we do however complement the simulation-based study in this chapter with experimental results obtained on a legacy embedded microcontroller platform. The results obtained using implementations on the HECTOR boards will be presented in Chapter 7.

6.1 Introduction

Our simulation-based analysis leverages on the framework and models put forward by Doget et al. [37] to compare univariate side channel attacks. Using the success rate of an attack as quality metric, the authors provide comparative results of various attacks against an unprotected implementation of the AES block cipher. In particular, the authors analyze the following attacks: single-bit DPA (SB-DPA) [70], all-or-nothing DPA (AON-DPA) [86], generalized DPA (G-DPA) [86], correlation power analysis (CPA) [28], partitioning power analysis (PPA) [73], absolute-sum DPA (AS-DPA) [1], and linear regression analysis (LRA) [37]. The analysis involves two different leakage models: the so-called perfect model (also known as Hamming weight model) and its generalization coined random linear leakage model.

Our study follows a similar methodology, but we consider instead the case of protected implementations. Rather than comparing attacks, our aim is to quantify the security loss of countermeasures in the presence of weak random number sequences. Motivated by this, we restrict our simulation-based study to the perfect leakage model and CPA attacks, which are the most efficient attack against this type of leakage. Following this, we denote the leakage of a sensitive variable by $l = HW(v_i) + B$, where $HW(v_i)$ is the deterministic part of the leakage corresponding to the Hamming weight of v_i, and B is independent Gaussian noise with mean zero and standard deviation σ. This definition allows to model the effects of vertical noise in the side channel measurements by varying the value of σ, i.e. $\sigma = 0$ will yields noiseless measurements (optimal from an attacker's perspective), while increasing σ will naturally make

the attack more difficult. Note that on a realistic setting the value σ is given by both the measurement setup of the adversary and the characteristics of the target device. This will be demonstrated in Section 6.5.

In line with the testing approach described in Chapter 5, we consider a side channel adversary with physical access to a realization of $\mathbb{E}_K$ storing a secret key K. For the purposes of the study we assume $\mathbb{E}_K$ is an implementation of the AES-128 block cipher [95]. We further assume the sensitive variable targeted by the adversary corresponds to one byte output of the `SubBytes` transformation. More formally, the sensitive variable is given by $v_{i,j} = Sbox(p_{i,j} \oplus k_j)$ where $1 \leq i \leq n$ corresponds to the algorithm's execution with plaintext P_i, the symbol $\oplus$ denotes bitwise XOR, and $Sbox$ is the non-linear transformation applied at byte level in `SubBytes`. The index $1 \leq j \leq 16$ identifies the plaintext and subkey bytes involved in the calculation of $v_{i,j}$. Since the efficacy of the attack is the same for any selection of j, we assume $j = 1$ in the following to simplify notation.

A high-level illustration of our test scenario is depicted in Figure 6.1. We assume an adversary mounts a CPA attack against an instance of the AES-128 `SubBytes` transformation protected with side channel countermeasures.

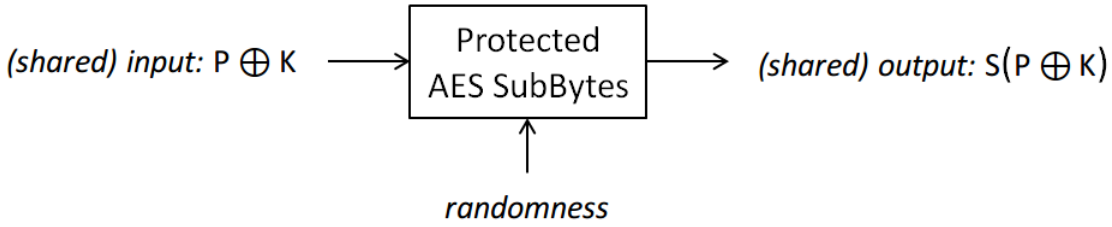


Figure 6.1: Test scenario: we evaluate the side channel resistance of an AES `SubBytes` transformation protected with countermeasures that consume randomness.

6.2 Analysis of unprotected implementation

Before diving into the details of our study, we show the outcome of the simulation framework assuming univariate attacks against an unprotected implementation. Note that this is the same setting as studied in [37], and it is equivalent to the particular case in Figure 6.1 in which randomness is set to zero and, consequently, countermeasures are effectively *disabled*.

The results of the simulation analysis can be summarized in a single plot, see Figure 6.2. The black curve indicates the number of measurements needed for the attack to achieve a 90% success rate in function of different noise levels σ. The success rate is calculated by running 50 independent instances of the attack, each with a different key k. Note that performing more experiments would lead to more stable results, as our study is naturally sampling-based. This, however, would significantly increase the computational costs of the analysis, specially when lots of measurements need to be processed. In order to keep the complexity of our study at a reasonable level, we opt to set the number of experiments at 50 and set upper bounds for $\sigma_{MAX} = 2^4$ (the maximum noise standard deviation of the setup) and $n_{MAX} = 10M$ (the maximum number of measurements available to an adversary). Both these values are standard

in these type of studies. Note that $n_{MAX} = 2M$ in the experiments involving real TRNG measurements in which, as this is the amount of data collected in the evaluation from WP2.

Let us focus on the results of a univariate CPA attack as shown in Figure 6.2 (left). For small values of noise, the number of measurements required for the attack to succeed is very low (less than 10), and grows up to 3k for the highest noise level tested ($\sigma = 2^4$). The same trend can be observed in Figure 6.2 (right), which shows the results of a SB-DPA attack. Note however that the number of measurements for the attack to succeed is in this case significantly larger. This is because the attack does not exploit all information leaked through the measurements, as it targets a single bit. This justifies our choice of using only CPA attacks through our study.

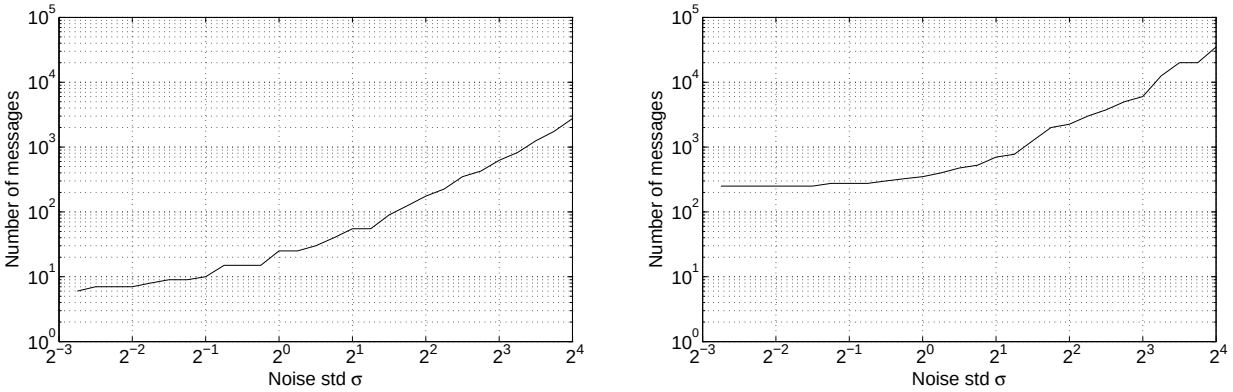


Figure 6.2: Univariate attacks against an unprotected AES Sbox: CPA (left), SB-DPA (right).

6.3 Analysis of Masking countermeasures

In this section we provide results for the security degradation of masking countermeasures. We focus on two types of masking schemes: *Boolean* masking [31, 47] and *Inner Product* (IP) masking [5]. Although both types of masking can be instantiated at any order d we restrict our study to the particular case of 1*st*-order masking which should prevent any type of univariate attack. The aim of the study is to determine whether the use of non-ideal random numbers breaks this property and, if so, to quantify the security degradation of the countermeasure.

6.3.1 Boolean Masking

We present here the outcome of the analysis of 1*st*-order Boolean masking. Recall that, in this situation, the output of the AES Sbox is given by $S(p \oplus k) = m_1 \oplus m_2$, where m_2 is a random value that changes for each execution and $m_1 = S(p \oplus k) \oplus m_2$ is the masked value which should not leak any exploitable information about $S(p \oplus k)$ when masks are drawn from a uniform distribution. In the following we assume that m_2 comes from one of the non-ideal sets presented in Chapter 5.

Monobit bias. The results for the case of biased random sequences are given in Figure 6.3. The plot on the left hand side shows results for sets biased towards 0, while the plot on the right hand side shows results for sets biased towards 1. Note that the security degradation is the same for both types of bias, i.e. the curves have very similar shapes. The slight differences in

the plots stem from our sampling-based approach and should vanish by increasing the number of independent experiments that determine the 90% success rate (recall this number is currently set at 50 to keep t he computational effort at a reasonable level).

The most interesting observation is that biasing the random numbers consumed by the countermeasure enables a univariate CPA attack. For low levels of noise, around 225, 400, 575, 1.5k and 6.2k measurements are needed to break the implementation for bias levels 25%, 20%, 15%, 10%, and 5%, respectively. The number of measurements naturally increases as the vertical noise grows, reaching 32k, 52k, 80k, 275k and 650k for the maximum value of σ tested in our experiments. Note also that the shape and slope of all curves is very similar. In fact, the only difference is a vertical shift that varies depending on the bias level of the random number sets.

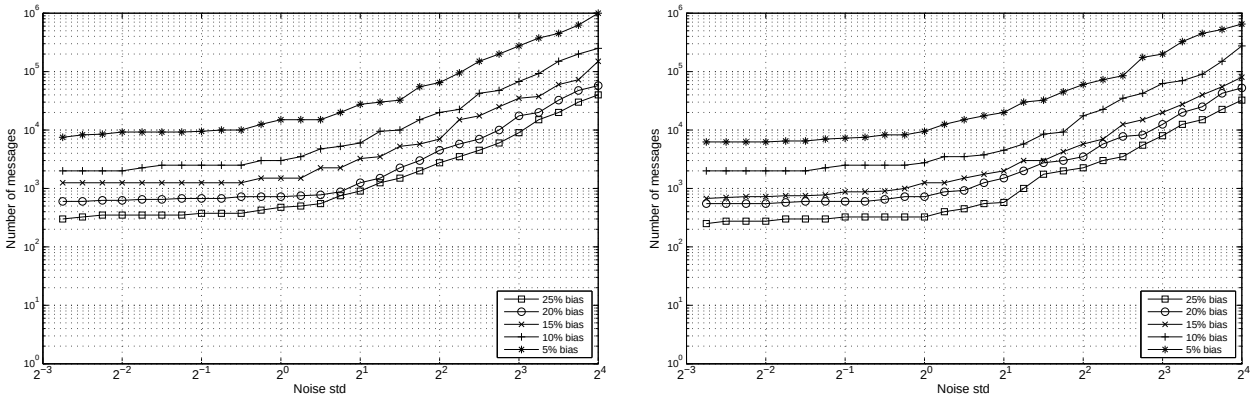


Figure 6.3: Univariate CPA attack against AES Sbox protected by 1*st*-order Boolean masking with random numbers biased towards 0 (left) and 1 (right).

Failure to other AIS 31 tests. Next, we provide in Figure 6.4 the results obtained when using random sequences that fail other statistical sets from the AIS 31 suite. The plot on the left hand side shows results for sets failing T2 (poker test), while the plot on the right hand side shows results for sets failing T3 (run test), T5 (autocorrelation test) and T8 (entropy test). In contrast to the previous analysis, we observe that the number of measurements required to enable a univariate attack is significantly larger. Moreover, for certain sets the attack can not be mounted even with our upper bound of 10M measurements and for low levels of noise.

Let us first focus on the left hand side plot. Here we observe that the attack is only enabled when using sets with *high* and *average* degradation. In this case, the attack complexity for low noise levels starts at 35k and 100k measurements, respectively, reaching 5M and 10M for the highest noise level tested. Using the set with *small* degradation yields no results, meaning the attack is never successful. The results from the right hand side plot are even more pronounced. In particular, only the *high* degradation sets corresponding to T3 and T5 can be attacked with a rather high complexity, i.e. starting from 6M measurements for low noise levels. Finally, no set corresponding to failures to T8 enabled univariate CPA attacks.

Real data from TRNG implementation. The last set of experiments for Boolean masking corresponds to the analysis with random sequences obtained from a TRNG implementation subject to temperature and voltage variations. The results are provided in Figure 6.5 when

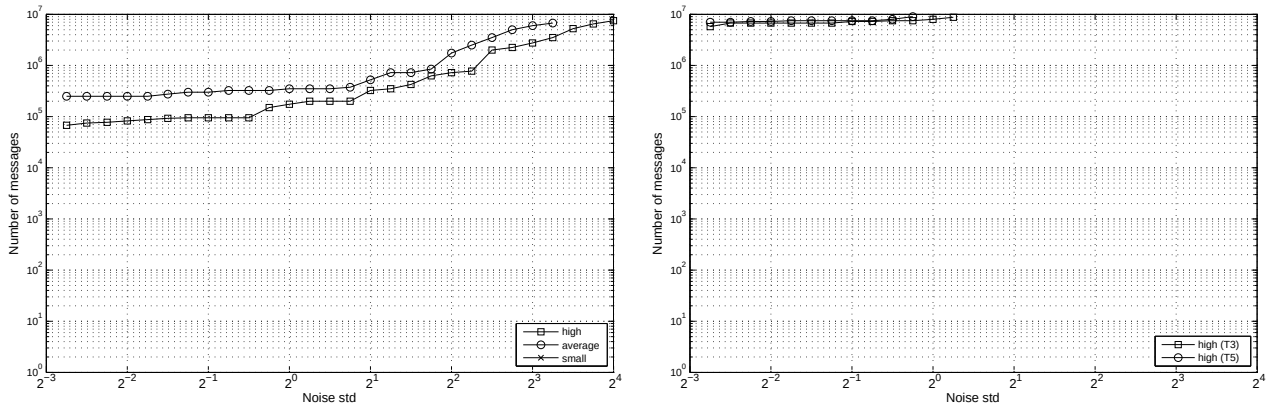


Figure 6.4: Univariate CPA attack against AES Sbox protected by 1*st*-order Boolean masking with random numbers failing test 2: poker test (left) and tests 3,5,8: run test, autocorrelation, entropy (right).

temperature is fixed at -40 degrees (top left), 20 degrees (top right) and 80 degrees (bottom). The first observation is that, similar to the monobit bias case, univariate CPA attacks are enabled when using any of the datasets. A second observation is that the vertical shift of the curves depends on the voltage level. More specifically, the *weaker* set always corresponds to the measurements at 0.9 V, while the *stronger* set is always obtained at 1.4 V.

A closer look at the plots further shows that, for low levels of noise, the most efficient attack at 0.9 V demands always less than 1k measurements independently of the value of temperature. At 1.4 V, this number is more sparse and reaches 7k, 15k, and 30k measurements at -40, 20, and 80 degrees, respectively. For high levels of noise, all random sets obtained at -40 degrees enable a univariate attack with an upper bound of 2M measurements. This is not the case for 20 and 80 degrees, for which the set obtained at 1.4 V demands already 2M measurements when the noise standard deviation is 2^3.

We stress here once again that the sets used in Figure 6.5 correspond to the case where the TRNG implementation contains only the entropy source. Repeating the same experiments with the sets obtained when the TRNG implementation contains a post-processing module yields completely different results. More precisely, the univariate CPA attack never succeeds for any temperature/voltage combination, even for low levels of noise. This is because, as mentioned in Chapter 5, the post-processing module limits the effects of temperate/voltage modifications on the quality of random number sequences.

6.3.2 Inner Product Masking

In this part we extend our analysis of masking countermeasures to other schemes other than Boolean. We consider in particular the 1*st*-order Inner Product (IP) masking scheme as proposed in [5]. Using this construction, the output of the AES Sbox is given by $S(p \oplus k) = m_1 \oplus (l_2 \otimes m_2)$. The variable m_2 corresponds to the random share, and l_2 is a parameter of the schemes which is fixed and assumed public information. In our experiments we set $l_2 = 5$ for consistency with the analysis provided in [5].

Note that the encoding function used in IP masking is more involved than in Boolean masking.

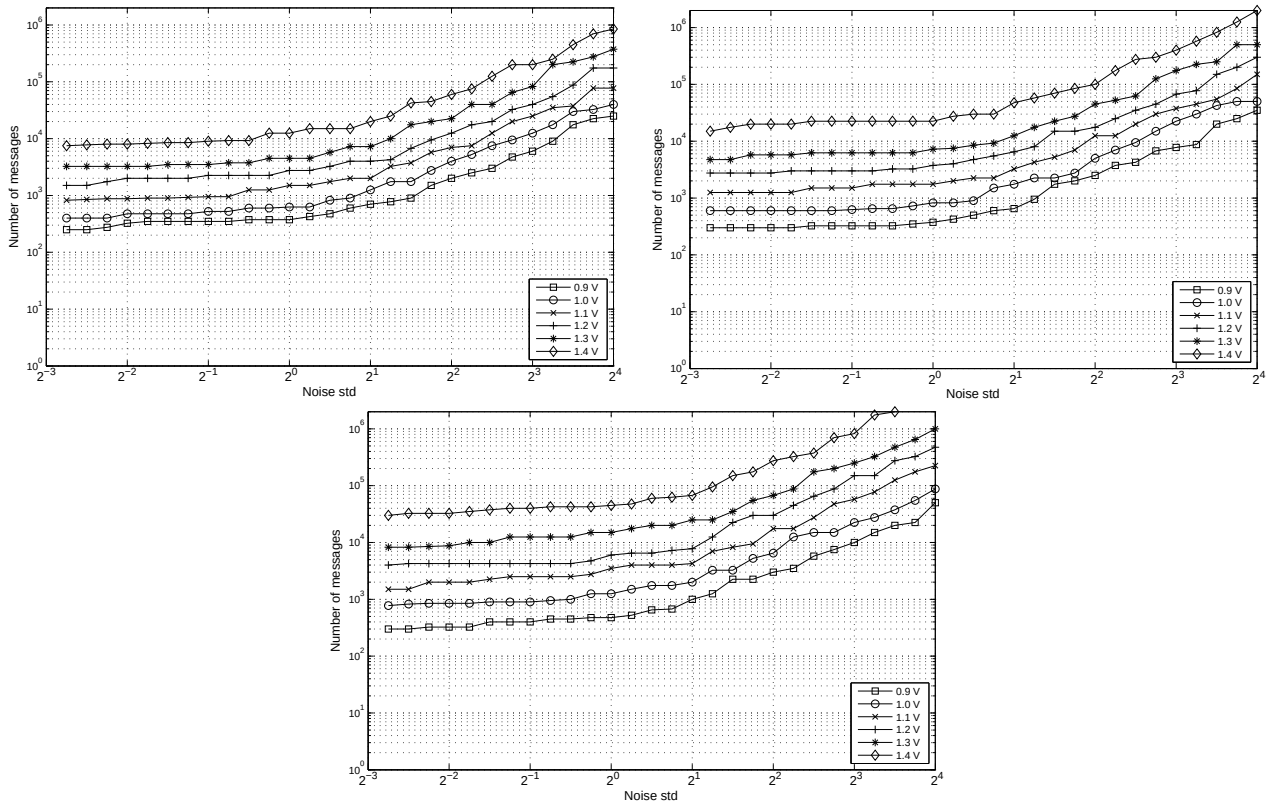


Figure 6.5: Univariate CPA attack against AES Sbox protected by 1*st*-order Boolean masking with random numbers obtained from a real TRNG when modifying its environmental conditions: -40 degrees (top left), 20 degrees (top right), 80 degrees (bottom).

This brings some security improvements at the cost of slightly more complex constructions to operate in the masked domain. For more details on the characteristics of IP masking and its comparison to IP masking, we refer the reader to [5].

Monobit bias. The results for the case of biased random sequences are given in Figure 6.6. The plot on the left hand side shows results for sets biased towards 0, while the plot on the right hand side shows results for sets biased towards 1. Note that, in contrast to Boolean masking, the security degradation differs in function of the type of bias. More specifically, a bias towards zero makes the attack more efficient than a bias towards one. The reason for this is the multiplicative structure of the inner product encoding, i.e. when all bits of m_2 tend to zero the product with l_2 is effectively disabled, while this is not the case when the bits tend to one.

Apart from this observation, the experiments corroborate the analysis in [5] highlighting the improved resilience of IP masking when compared with Boolean masking. For biases towards zero and low levels of noise, the minimum number of measurements for the attack to succeed is set to 22k, 55k, 250k, and 1.25M for bias levels 25%, 20%, 15% and 10%, respectively. For biases towards one, these values increase to 92k, 200k, 325k, and 2.5M, respectively. Independently of the type of bias, the results show that the sequences with bias level 5% do not enable a univariate CPA attack with an upper bound of 10M measurements.

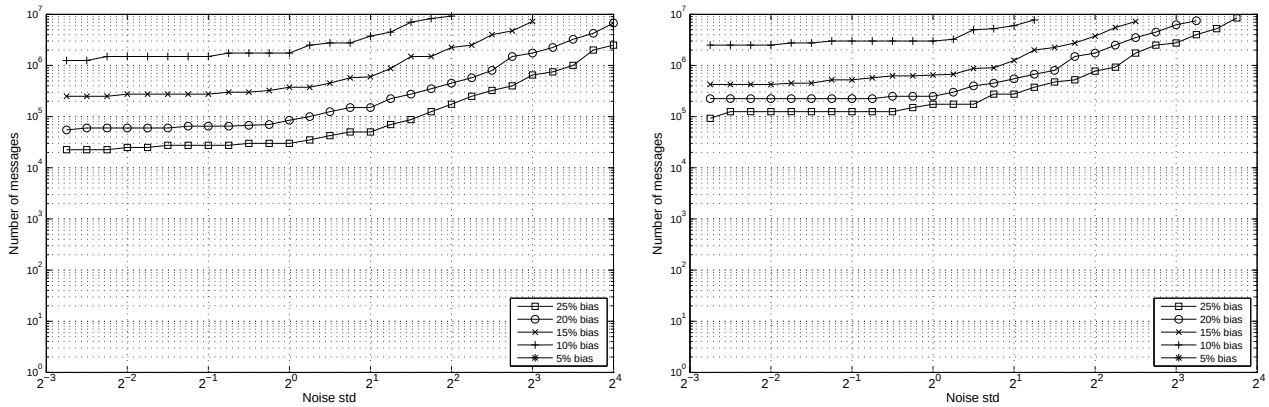


Figure 6.6: Univariate CPA attack against AES Sbox protected by 1*st*-order IP masking with random numbers biased towards 0 (left) and 1 (right).

Failure to other AIS 31 tests. The results of the analysis when using the sequences failing certain AIS 31 tests are provided in Figure 6.7. Note that univariate CPA attacks are only possible when using the datasets failing T2 (poker test), but not when using datasets corresponding to T3 (run test), T5 (autocorrelation test) and T8 (entropy test).

Focusing on the plot, we see once again that the results are somewhat similar to what obtained for Boolean masking, i.e. the attackable sets correspond to the *high* and *average* weakness levels. However, we observe once again a vertical shift of the curves that indicates more measurements are required to enable the attack. More specifically, low levels of noise demand 175k and 3M measurements, and the upper bound of 10M is reached at noise levels $\sigma = 2$ and $\sigma = 2^{3.5}$, respectively.

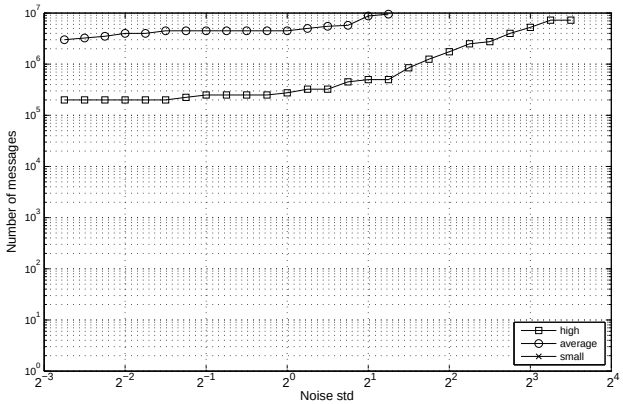


Figure 6.7: Univariate CPA attack against AES Sbox protected by 1*st*-order IP masking with random numbers failing test 2: poker test.

Real data from TRNG implementation. Lastly, we provide in Figure 6.8 the results of the analysis when using the random numbers obtained from a TRNG at different temperature/voltages. In line with previous results we observe that certain sets do enable univariate CPA attacks, albeit the minimum number of measurements is considerably larger than the case of Boolean masking. The worst results leading to most efficient attacks are always obtained at 0.9 V, but in this case we do not see a direct relation between voltage and complexity of an attack, e.g. for the case at -40 degrees the results at 1.1 V are worse than at 1.0 V. Also

noteworthy is the fact that at 80 degrees only the sets at 0.9 V and 1.0 V enable univariate CPA attacks with an upper bound of 2M measurements.

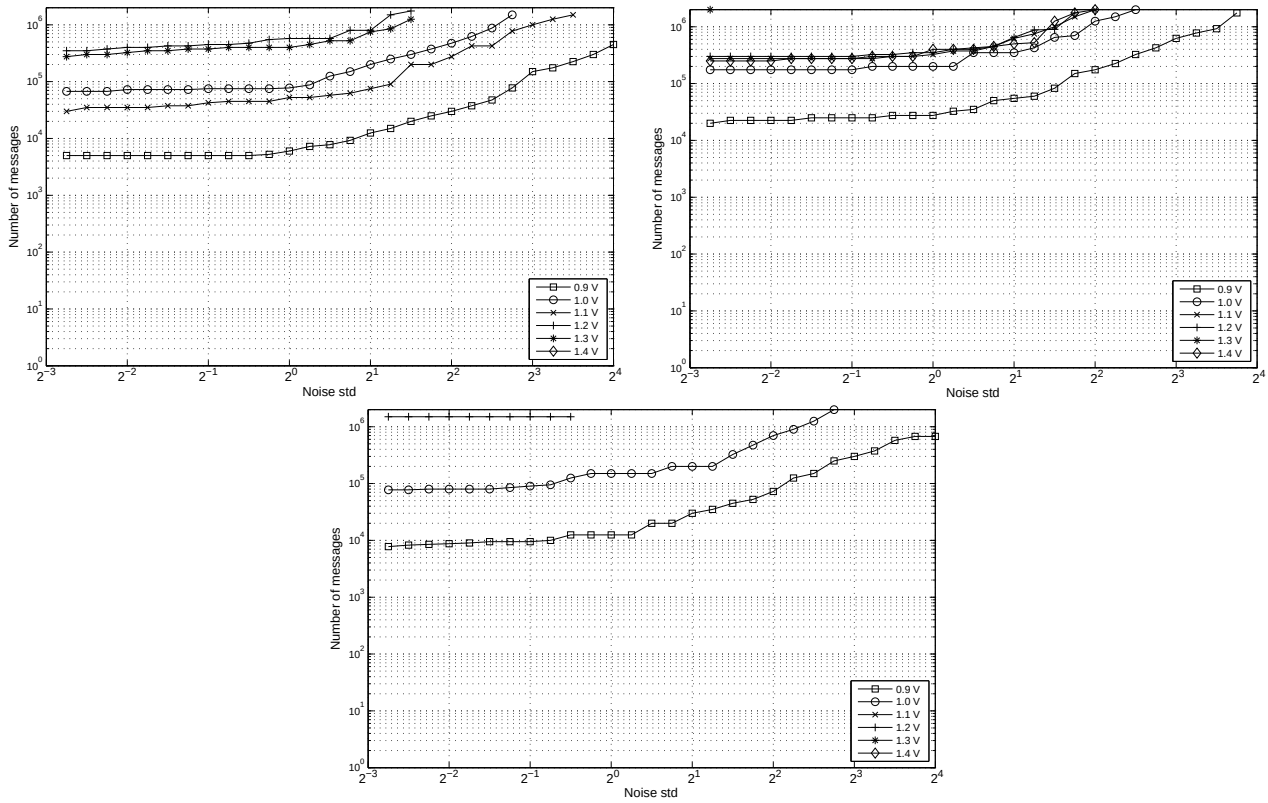


Figure 6.8: Univariate CPA attack against AES Sbox protected by 1*st*-order IP masking with random numbers obtained from a real TRNG when modifying its environmental conditions.

6.4 Analysis of Hiding countermeasures

In this section we provide results for the security degradation of hiding countermeasures. We focus in particular on techniques which aim to insert horizontal noise in the measurements by randomizing the expected execution order of inner computations. Recall that side channel measurements need to be correctly aligned in the time domain in order for the attack to succeed. If this is not the case, then wrong samples are mixed in the analysis and this increases the complexity of the attack.

We opt to model the insertion of horizontal noise with a parameter $\hat{p}$ which captures the probability of a correctly aligned measurement to occur. Imagine the case, for instance, that temporal randomization is achieved by *shuffling* the order in which the Sbox lookups are performed within the `SubBytes` transformation. In that case $\hat{p} = 1/16$ is the probability that the target Sbox is processed in the expected time position, i.e. on average, only 1 measurement out of 16 will carry exploitable information for the attack. The insertion of *dummy rounds* can be modeled similarly, i.e. if 16 dummy Sbox lookups are interleaved with an already shuffled implementation, then the probability of obtaining a measurement with exploitable information decreases to $\hat{p} = 1/32$. Other temporal randomization techniques such as the insertion of random delays could also be

modeled by tuning the variable $\hat{p}$.

It is known from earlier studies on side channel countermeasures that the probability $\hat{p}$ linearly reduces the correlation coefficient obtained in a CPA attack. This, in turn, quadratically increases the number of required measurements for the attack to succeed [78]. We naturally expect this effect to be visible in our results in the form of a vertical shift in the curves. Note also that there exist techniques to overcome or minimize the noise introduced by hiding countermeasures, i.e. by using pattern matching to identify and remove temporal misalignments or by using integration or windowing techniques to reduce their effect [78]. Such techniques are however highly implementation dependent, and therefore we do not consider them in our study.

In an attempt to abstract from implementation details, we assume that the countermeasure consumes n bits to determine whether the target Sbox lookup is processed at the expected time instant. More specifically, the measurement is correct if $n = 0$ and incorrect otherwise. In an ideal situation (random bits come from a uniform distribution) this ensures that $\hat{p} = 1/2^n$. If random numbers are biased towards 0, then $\hat{p}$ will not be uniform and the pool of measurements will contain, on average, more than $1/\hat{p}$ exploitable measurements. Based on this observation, we select for this part of the analysis the random number sets biased towards 0.

Figure 6.9 shows the results obtained. For completeness, we include the case when random bits have no bias, i.e. equivalent to drawn from a uniform distribution. This is the upper curve in the plots. As expected, the complexity of the attack decreases in function of the bias level, i.e. the lower curve corresponds to set with bias 25%. In this case, the adversary would need approximately 10 times less curves in order to mount the attack. The same trend can be observed in all cases explored: $\hat{p} = 1/16$, $\hat{p} = 1/32$ and $\hat{p} = 1/64$. The only difference is a vertical shift in the curves, which depends on the factor $\hat{p}$ used to model the countermeasure.

6.5 Validation of simulation results

In this last section we validate the soundness of the results obtained using our simulation framework by comparing them with real CPA attacks. In particular, we analyze an AES-128 software implementation protected by 1*st*-order Boolean masking. Our TOE is an 8-bit AVR Atmega163 micro-controller. We select this platform as it has a similar behavior to what we assumed in our simulation model. More specifically, it exhibits a leakage function very close to the Hamming weight model. The target implementation is furthermore written in software, therefore ensuring that shares are no manipulated in parallel (as could be the case for hardware implementations).

Our 1*st*-order implementation is based on the design put forward by Herbst et al. in [55]. The `SubBytes` operation is computed by means of a masked Sbox table S' generated at the start of each encryption as $S'(x \oplus m) = S(x) \oplus m'$, where m and m' are the input and output masks, respectively. Similar to the simulation study, we analyze the cases where m' is set to zero (unprotected implementation), m' comes from a uniform distribution (ideal situation), and m' comes from a biased distribution. For each of these cases, we have collected a set of power measurements by monitoring the voltage drop over a 50 Ohm resistor. The TOE runs at a fixed clock of 3.58 MHz and we set the sampling rate of the oscilloscope at 250 MS/s.

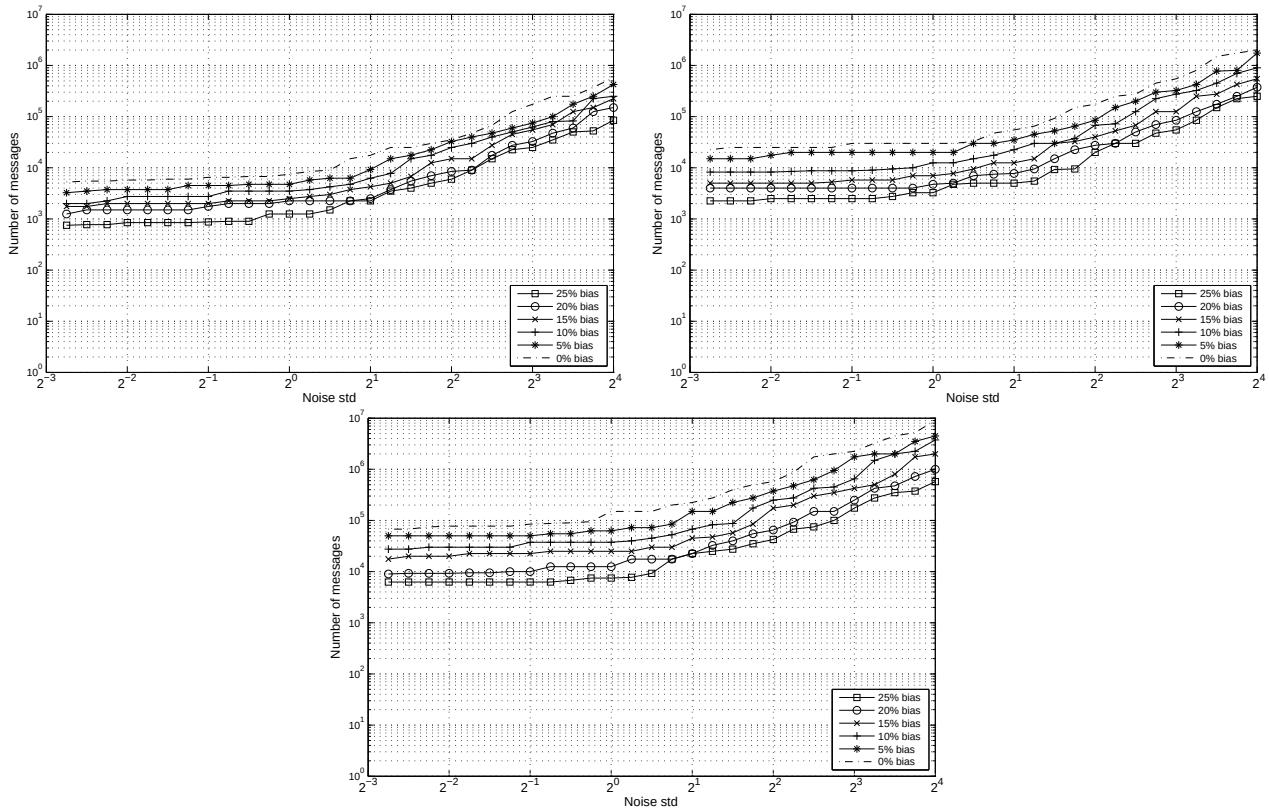


Figure 6.9: Univariate CPA attack against AES Sbox protected by generic hiding countermeasures with biased random numbers: $\hat{p} = 1/16$ (top left), $\hat{p} = 1/32$ (top left),$\hat{p} = 1/64$ (bottom).

The results of attacking the implementation for different selections of m' are shown in Figure 6.10. Each plot shows the outcome of the CPA attack (correlation coefficients) for all possible key byte hypotheses in function of the number of measurements. The results for the incorrect keys are plotted in grey, while the result of the correct key is plotted in black. The minimum number of measurements necessary for the attack to succeed corresponds to the sample where the black line is distinguishable from the grey lines.

The top left plot corresponds to the unprotected implementation, which can be broken starting from as few as 35 measurements. The next plots show the outcome of the attack for the cases where m' is biased. The number of measurements increases to approximately 250, 500, 700, 2 500 and 20 000 for bias levels 25%, 20%, 15%, 10% and 5%, respectively. Finally, the bottom plot shows the analysis of the implementation protected with masks drawn from a uniform distribution. In this case, the attack fails even when analyzing up to 200k measurements [1].

These results are consistent with the outcome of the simulation study. In particular, note that the number of measurements is very similar to what obtained in our analysis when the noise standard deviation is around 0.25. This is illustrated in Figure 6.11 recapping the simulation results and indicating the *slice* corresponding to our platform (c.f. red vertical line).

[1]Note that we do not claim this implementation to be side channel secure, but simply point out that a straightforward univariate CPA attack targeting the HW of the Sbox output fails to succeed with 200k traces. It could well be possible that alternative attacks targeting other variables or using more complex leakage models would succeed in breaking the implementation.

6.6 Conclusions

In this chapter we have presented the first part of the results of our study on the security degradation of countermeasures. In order to perform a generic analysis, we have opted to use simulated side channel measurements obtained by applying the well-known Hamming weight model with Gaussian noise. A benefit of this approach is that it enables to model the effects of vertical noise in the measurements through the parameter σ.

The results of our analysis using synthetic random number sequences indicate that monobit bias is the most damaging defect when it comes to side channel security. In fact, of the rest of non-idealities that we have tested, only the sets generated to fail the poker test (T2) enable univariate attacks with an affordable complexity. And even in this case, the security degradation is orders of magnitude smaller than for monobit bias. The results obtained using the random sequences from the TRNG implementation (without post-processing) show also a clear security degradation. We believe the reason for this is that the dominant defect in the TRNG data sets is actually monobit bias. In fact, we have verified that the bias levels for the different tested temperature/voltage combinations range from 2.5% to 29%.

Based on these results, we conclude that monobit bias is the most critical non-ideality when randomness is used for side channel countermeasures. Consequently, if a designer uses an TRNG implementation for this purpose, it should be preferable to prioritize on-line tests checking the bias of the generated sequences over other, less critical, types of non-idealities.

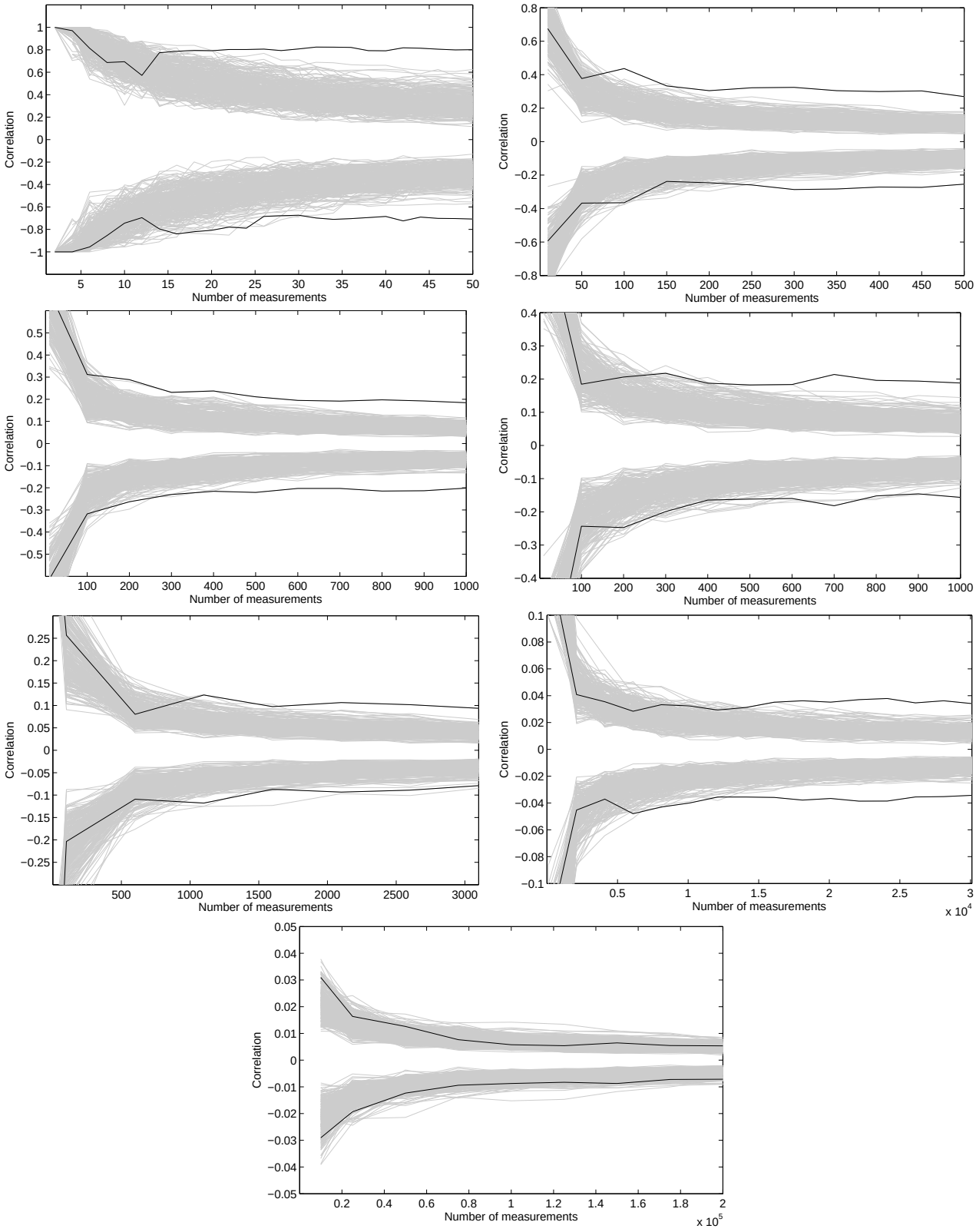


Figure 6.10: Univariate CPA attack against AES Sbox implementation protected by 1*st*-order Boolean masking running on an AVR 8-bit controller. Results with masks off (top left); masks with 5% bias (top right); masks with 5% bias (second from top, left); masks with 5% bias (second from top, right); masks with 5% bias (second from bottom, left); masks with 5% bias (second from bottom, right); uniform masks (bottom).

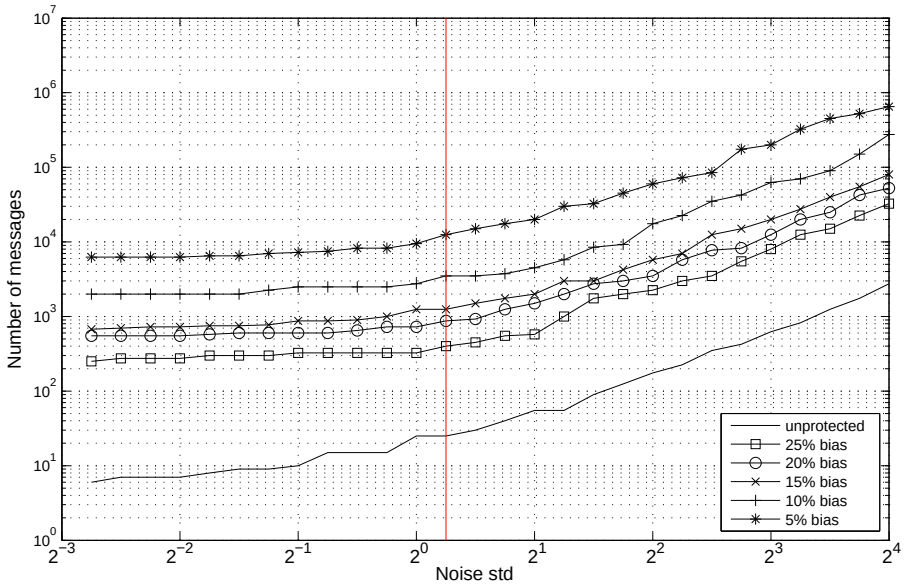


Figure 6.11: 1*st*-order CPA attack against AES Sbox implementation protected by 1*st*-order Boolean masking with random numbers biased towards 0. Red line indicates the noise level of our AVR platform determined by the results of our experiments.

Chapter 7

Experimental-based analysis of security degradation

In this chapter, we present the second part of the study of the security degradation of countermeasures. In contrast to Chapter 6, we use side channel measurements obtained from real FPGA implementations in the HECTOR evaluation board using the experimental described in Chapter 5. In order to keep the computational and storage complexity of the experiments to a reasonable level, we limit the study to the sets with monobit bias which, as shown in Chapter 6, are the ones leading to the most security degradation. In our experiments we also set the maximum number of collected measurements to 1M, which is a common upper bound for side channel evaluations. The target cryptographic algorithm corresponds to the Data Encryption Standard (DES) block cipher. In particular, we consider the following protected implementations:

- DES with Boolean masking
- DES dummy round (horizontal noise)
- DES and vertical noise
- DES dummy round and vertical noise

For each of these implementations, we will first briefly zoom in on the countermeasure principle that has been implemented, and then discuss the measurement results on the HECTOR board.

7.1 Boolean masking

7.1.1 Countermeasure principle

Recall that the principle behind Boolean masking is to manipulate secret data indirectly by adding to them a random value. This type of protection works very effectively in the case of linear operations (for instance addition) where it is possible to remove the mask by adding it in the same way. The targeted implementation of the Boolean masking in our DES implementation is represented in Figure 7.1. The figure shows only a bit representation of the Boolean masking, but byte wise data are the target. In addition, a total of 16 of these structures were present in the TOE (and therefore 16 key bytes were target) and operations were performed one after each other (byte per byte). For each of these operations, two clock cycles are used; one to release the input, key and random data, and one to store the corresponding output. It is important to note

that the interest of this experiment lies only into understanding the effect of the random on the single order resistance.

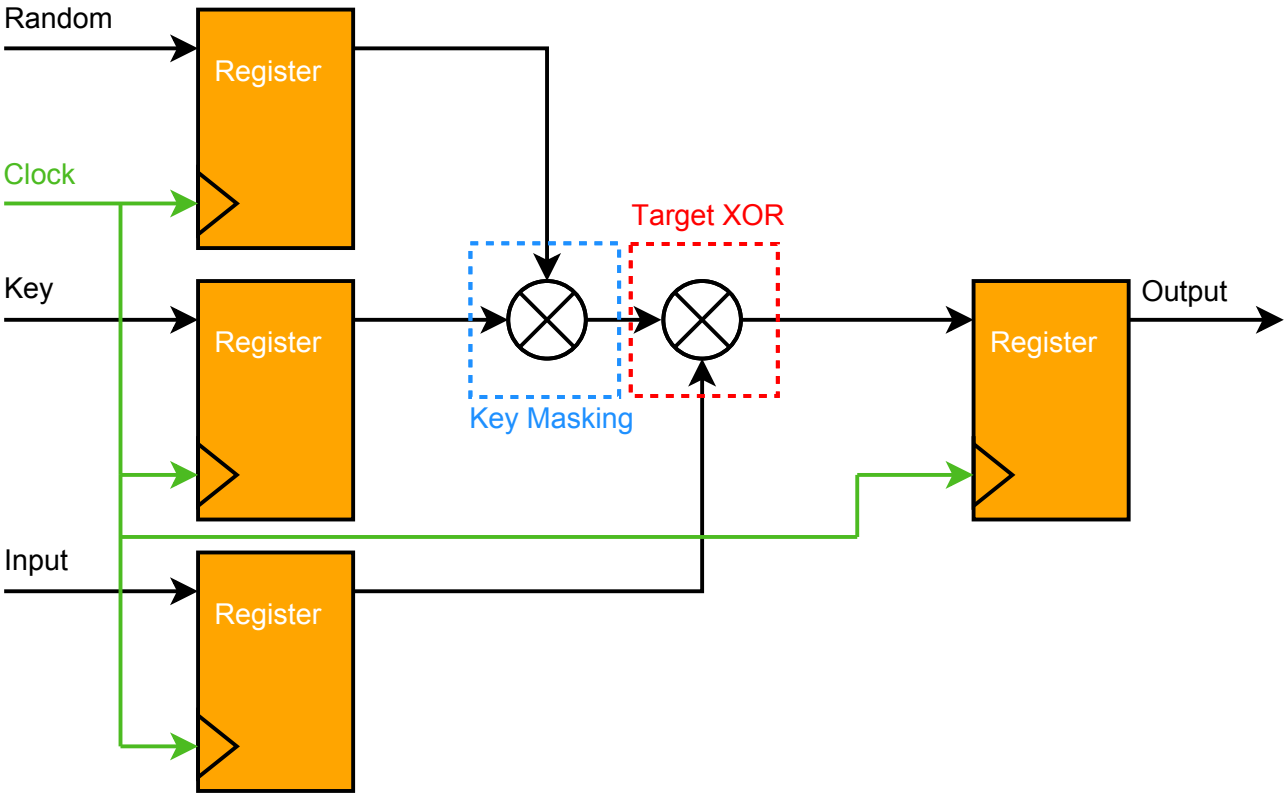


Figure 7.1: Boolean XOR masking base circuitry.

7.1.2 Measurement results

As mentioned before, sixteen XOR operations of individual bytes, which correspond to the 16-byte input being XORed with the masked key, occur within the interval shown in the top graph of Figure 7.2. A univariate CPA attack was performed in order to find the XOR operations within this interval. A set of 100 000 traces was measured and a null mask was used (equivalent to countermeasure disabled). It is often the case in practice that alignment of traces needs to be performed before the attack. In this case, however, there is no countermeasure which causes horizontal variations in the traces. Thus, skipping the alignment, we directly compute the correlation between the measured EM signals and the input and output signals, respectively, as indicated in Figure 7.1.

The attack results are shown in the middle and bottom graphs of Figure 7.2 for the input and output, respectively. When comparing the results it should be noted that the input correlation of byte one occurs before the output correlation of byte one, which in turn occurs before the input correlation of byte two, and so on (this is as expected according to the implementation details). The interval(s) used for the attack should concur with the output correlation. It is important to exclude the intervals in which the input correlation occurs, as it can interfere with the attack calculations. Due to the size of the FPGA and the spreading of the XOR operations on the chip, the leakage of the bytes were not equally measured by the coil. That is, the coil could not measure all the EM emissions equally of all the 16 XOR operations. To determine these bytes, several attacks were executed on the set measured for the input/output correlation

analysis using all of the output bytes intervals. By executing multiple attacks and each time decreasing the number of traces used, the leakage of the different bytes could be determined. In the end four bytes had similar leakage, namely 3, 5, 7, and 8. The intervals of these four bytes are shown in the top graph of Figure 7.3. In that situation (no masking), the minimum number of traces needed to identify the four bytes is 56,000.

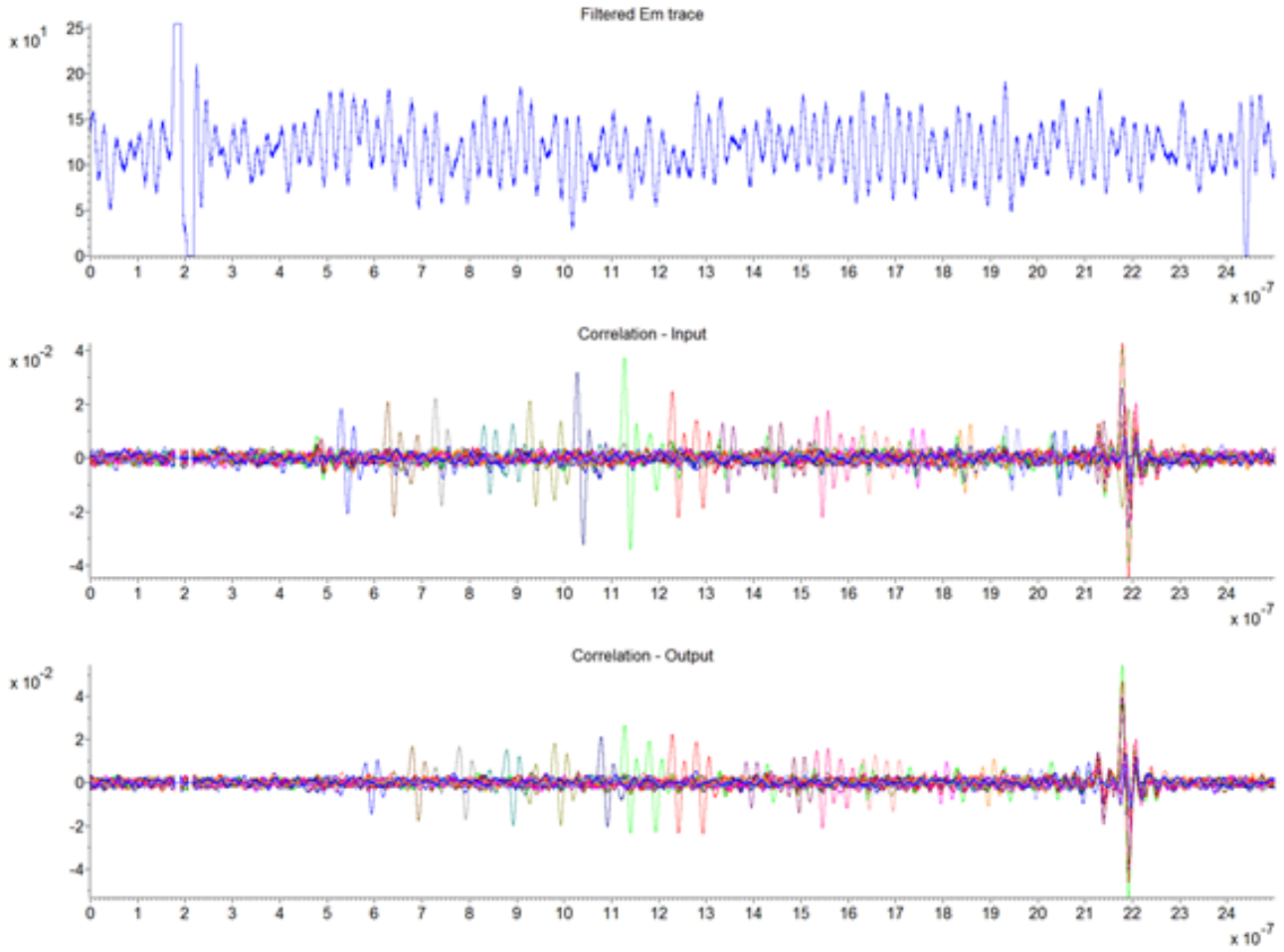


Figure 7.2: One of the traces measured during the targeted XOR operation (top). Correlation with the 16 input bytes (middle). Correlation with the 16 output bytes (bottom).

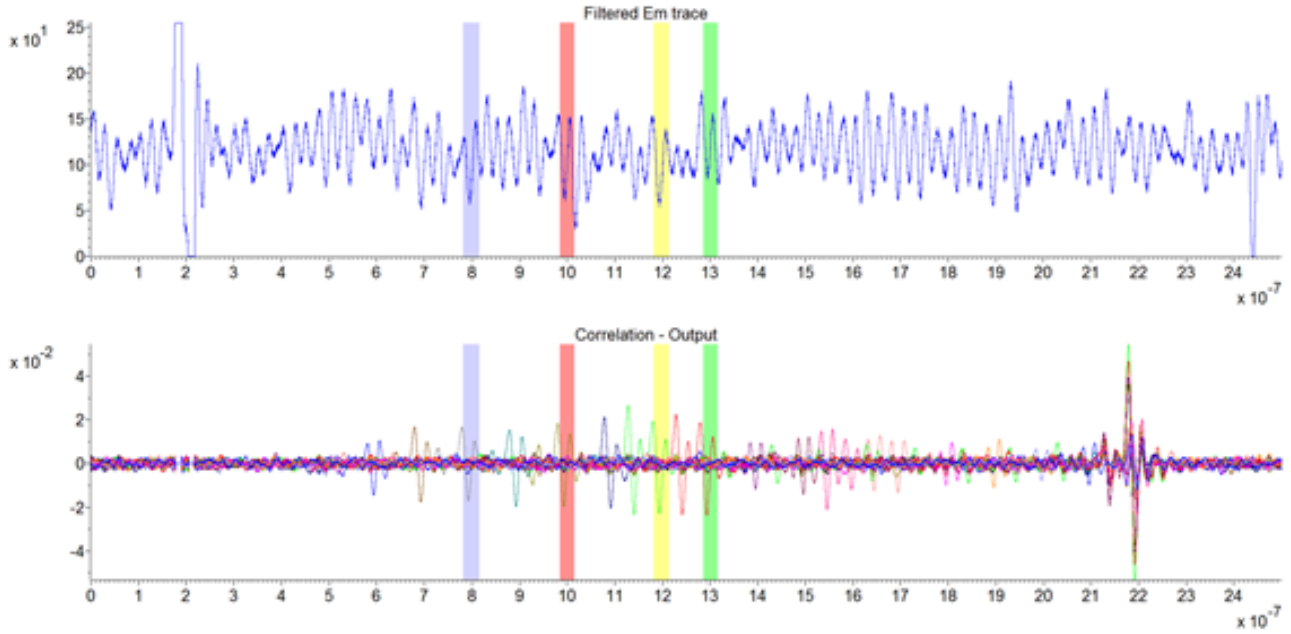


Figure 7.3: Intervals selected for the test (top), with the corresponding output correlation traces (bottom).

Using the selected intervals, six sets of measurements were made with each using a different mask coming from our biased random sets. In particular, we used the sets with bias levels 75%, 50%, 25%, 20%, 15% and 5%. For each set, a maximum of 1M traces were measured at a sampling frequency of 10 GS/s.

The number of traces needed to correctly identify all four bytes for different bias levels are shown in Table 7.1 and can be visualized as a graph in Figure 7.4. For a 20% bias level, 1M traces were not enough to identify the four bytes. Because the number of traces was not sufficient to reveal the selected key bytes for the 20% bias set, it was decided not to investigate the sets with bias levels 15% and 5%, as it would only lead to an even bigger number of traces needed.

Bias level	Number of traces (rounded to the closest 1,000)
No Mask	56,000
75%	90,000
50%	450,000
25%	972,000
20%	$>$ 1,000,000

Table 7.1: Results summary of the analysis on the impact of the Boolean masking on the number of traces needed to retrieve fully all S-Box sub-key candidates.

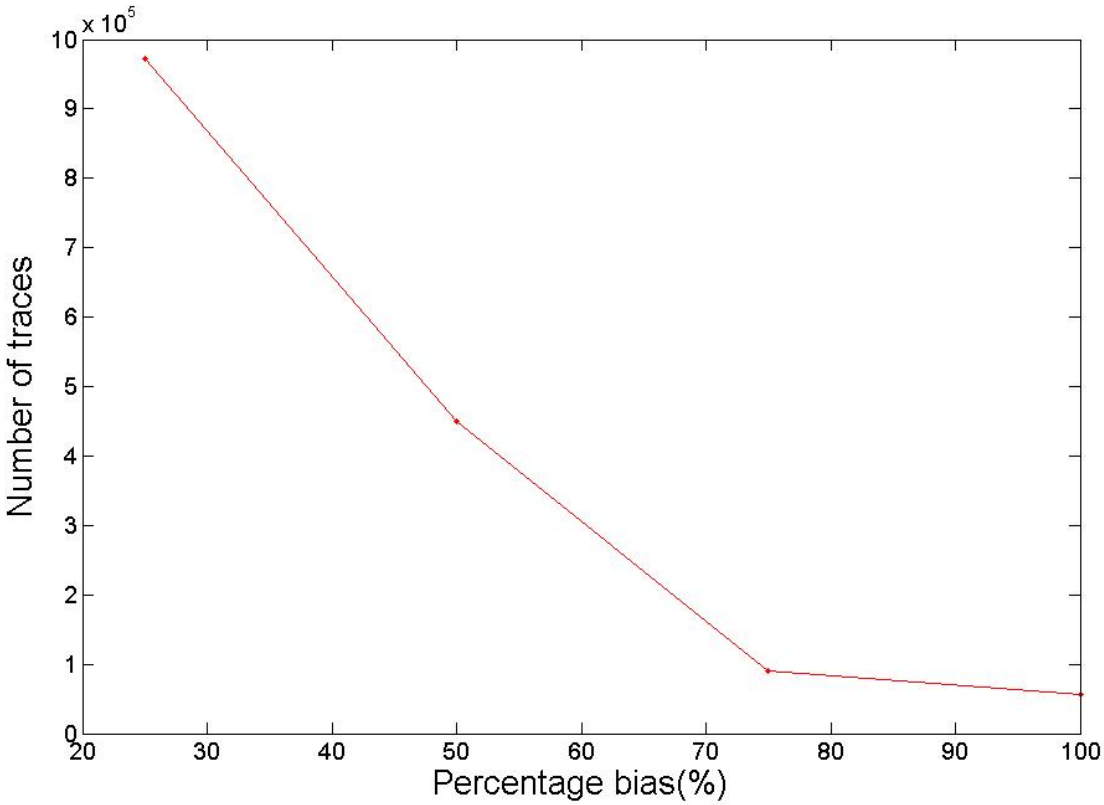


Figure 7.4: Graphical summary of the results with the Boolean masking countermeasure.

7.2 DES dummy round

7.2.1 Countermeasure principle

Figure 7.5 describes the second targeted countermeasure during our practical tests. This countermeasure consists in the insertion of dummy rounds during an actual DES computation. The truth table of the countermeasure logic as defined in the Figure 7.5 is shown in Table 7.2. This corresponds to an inverted XOR operation of the four input random bits. For every new incoming clock, the random input is regenerated (a new fresh bit is actually shifted in). When the signal clock enable is set to '1' the circuitry corresponding the actual DES is clocked (meaning that one actual round is achieved). In the other case, a dummy DES round is performed. The two circuitry are similar, the only difference is that the data (input and key) used by the dummy DES circuitry is random. In other word, it is not possible to distinguish visually which circuitry is active at a time (as they use exactly the same hardware). When the number of real DES round executed reach 16, the DES operation is finished; before that, an unlimited number of dummy DES rounds can be inserted (the goal was not to reach performance in this case). Figure 7.6 shows two traces recorded during the simultaneous execution of the DES computation when this countermeasure was enable. As it can be seen from that figure, the time for one DES execution is not constant from one execution to the next one as the number of inserted dummy round is dependent on the random input.

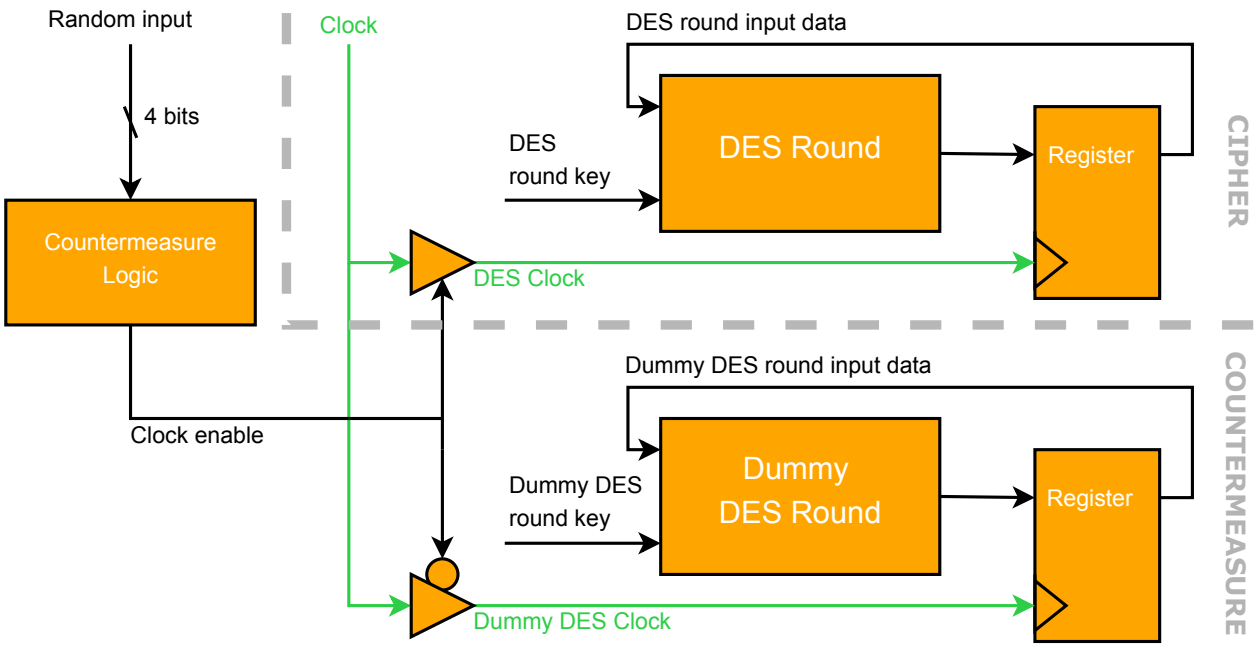


Figure 7.5: DES Dummy round countermeasures principle.

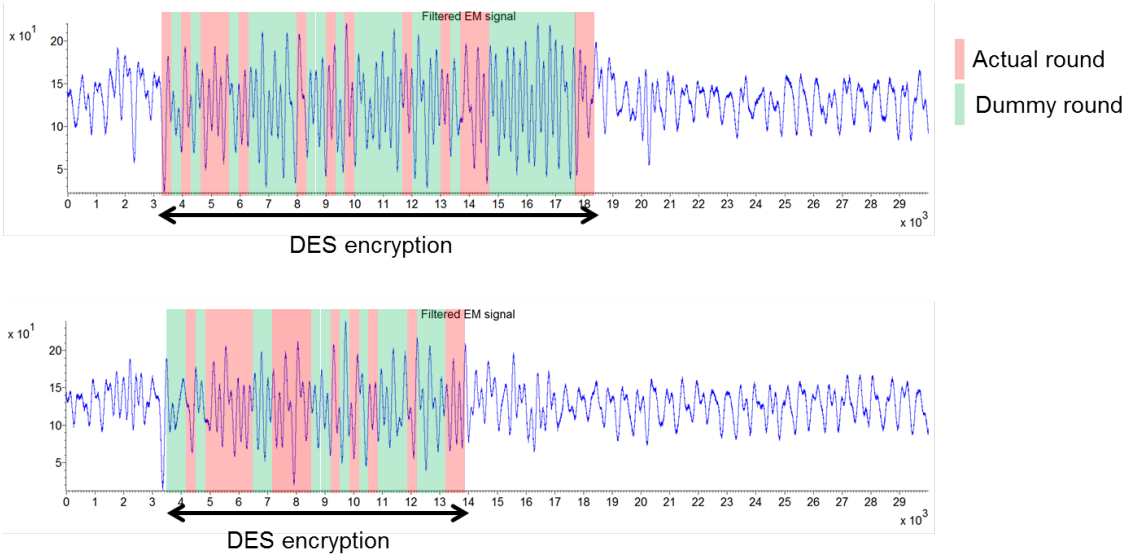


Figure 7.6: Example of two EM traces recorded during a DES execution when the countermeasures is on.

Random[3]	Random[2]	Random[1]	Random[0]	Out
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Table 7.2: Truth table of the logic driving the dummy round selection.

7.2.2 Measurement results

In this experiments, we collected measurements with random sets with bias levels 0% (i.e. drawn from uniform distribution), 25%, 50%, 75%, and 100% (i.e. countermeasure disabled). The first measured set was performed with the countermeasure disabled. Both the raw and filtered EM signals were measured and are shown in the first two graphs of Figure 7.7, respectively. To decide the interval to carry out the attack, the correlation is calculated between the power consumption and the S-box output of round one. First the correlation with the raw EM signal is calculated and the result is shown in the third graph of Figure 7.7 in the form of eight superimposed correlation graphs. No peaks stand out in this correlation result. Calculating the same for the filtered EM signal gives the result seen in the last graph of Figure 7.7. Here, there are clear correlation peaks visible.
The interval used for the CPA attack is highlighted in blue in Figure 7.7. 100,000 traces were not enough for a successful attack when using the raw EM signals. The number of minimum required traces when using the filtered EM signals was 4,800 (as a comparison 100,000 traces were not enough for a successful attack when using the raw EM signal). From this result, it was determined that only the filtered EM signal would be used for the rest of the analysis.
For bias levels 75%, 50%, 25%, and 0%, alignments were again made on the pattern that looks like a DES round and is the first one following the trigger signal. The correlation between the power consumption and the S-box output of round one was calculated again and the intervals, in which correlation was found, were used for the attacks. The results of these tests are summarized in Table 7.3 and can be visualized as a graph in Figure 7.8.

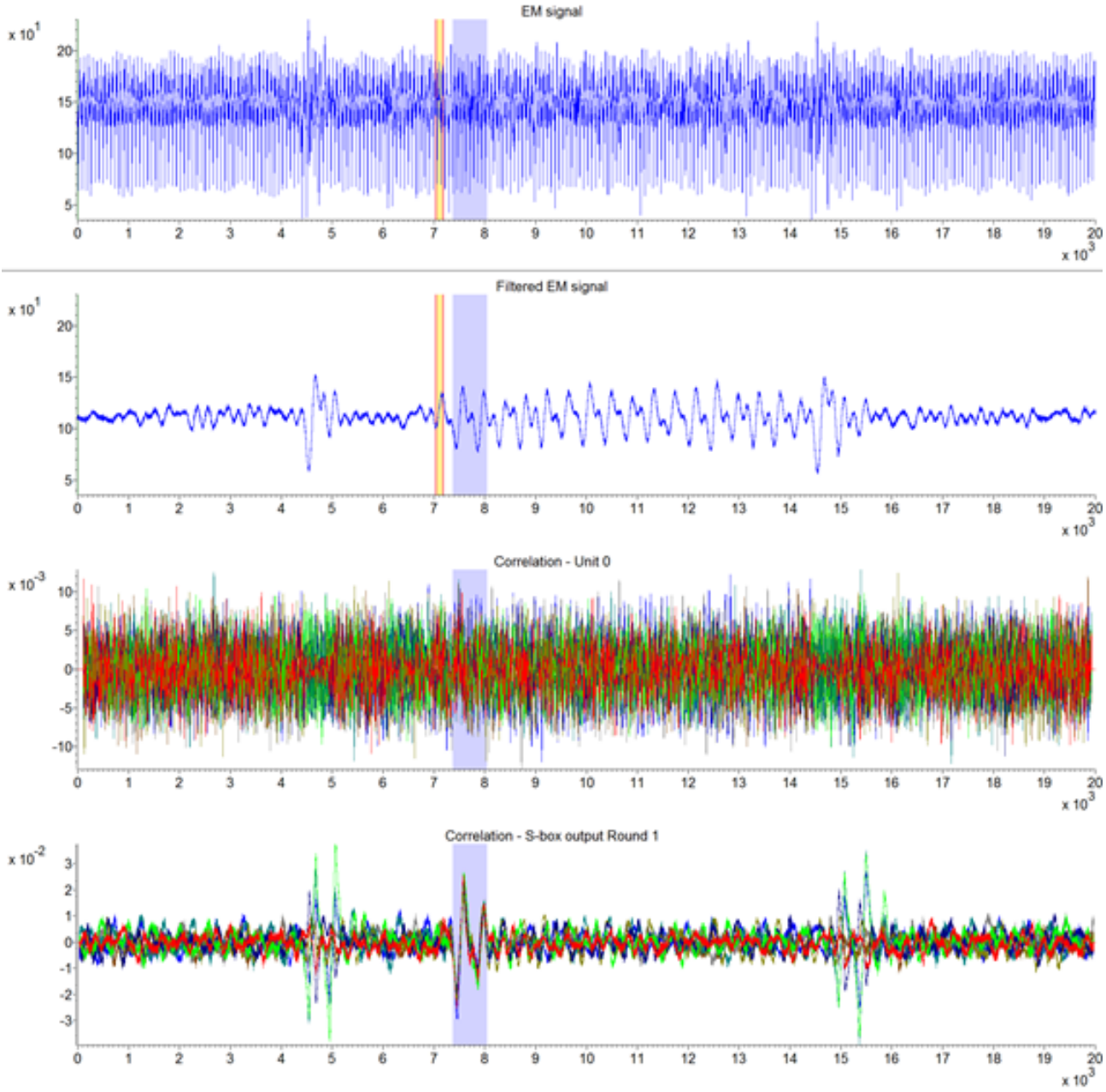


Figure 7.7: From top to bottom: Raw and filtered EM traces measured during the test of the dummy round countermeasure respectively, eight superimposed correlation traces computed with the raw EM traces and eight superimposed correlation traces computed with the filtered EM traces.

Bias level	Number of traces (rounded to the closest 200)
Countermeasure disabled	4,800
75%	15,200
50%	27,000
25%	107,600
0%	248,200

Table 7.3: Results summary of the analysis on the impact of the dummy round countermeasure on the number of traces needed to retrieve fully all S-Box sub-key candidates.

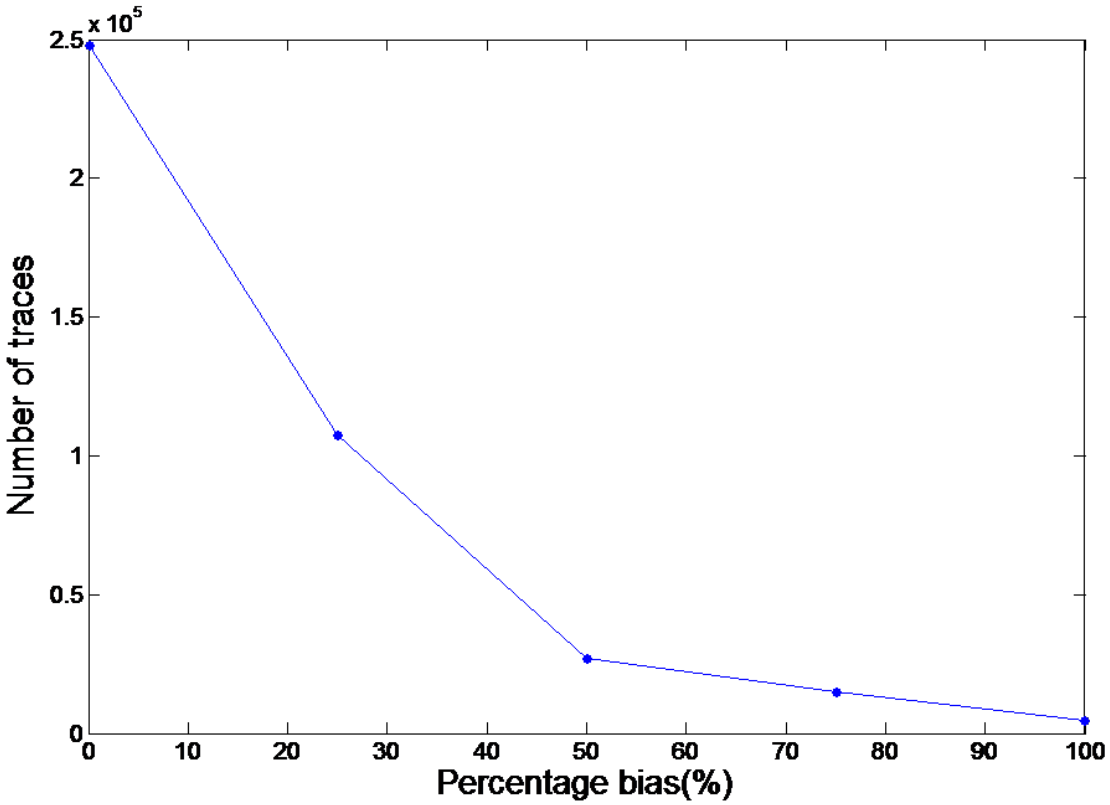


Figure 7.8: Graphical summary of the results with the countermeasure which adds vertical noise.

7.3 Vertical noise addition

7.3.1 Countermeasure principle

Figure 7.9 represents the base circuitry of the vertical noise addition countermeasure. It is composed of 16 registers all connected to the same clock source. For each register, the inverted output of is connected to the input, creating an oscillation effect on the output. Figure 7.10 illustrates the principle of this countermeasures. Every time the clock is gated, 16 bits are flipped at the same time (alternatively from '0' to '1' or '1' to '0') which will create a current spike (which will be visible in the measured EM field over the TOE). The clock source used to drive these registers is the same as the one used by the DES engine but is gated (if a clock enable is set, the clock is going through and the register can flip, if not, nothing happens). The clock enable signal is connected to the output of a shift register preloaded, prior to the DES execution, with a 16 bit random number (one bit for each round of the DES). Finally, these registers are always reset (default output value is '0') prior to every DES execution. In Figure 7.10 the current spikes are alternatively different to account from the difference in current consumption when a register is switching from '0' to '1' and from '1' to '0'.
In total, eight structures like the one shown in Figure 7.9 are implemented in the FPGA alongside the DES engine (in total 128 registers are used to create vertical noise). All these structures use different random inputs. Half of these structures are also using a clock gating circuitry where the clock enable signal is inverted (in order to improve the performance of the circuitry in case of a bias of the random sequence towards '0').

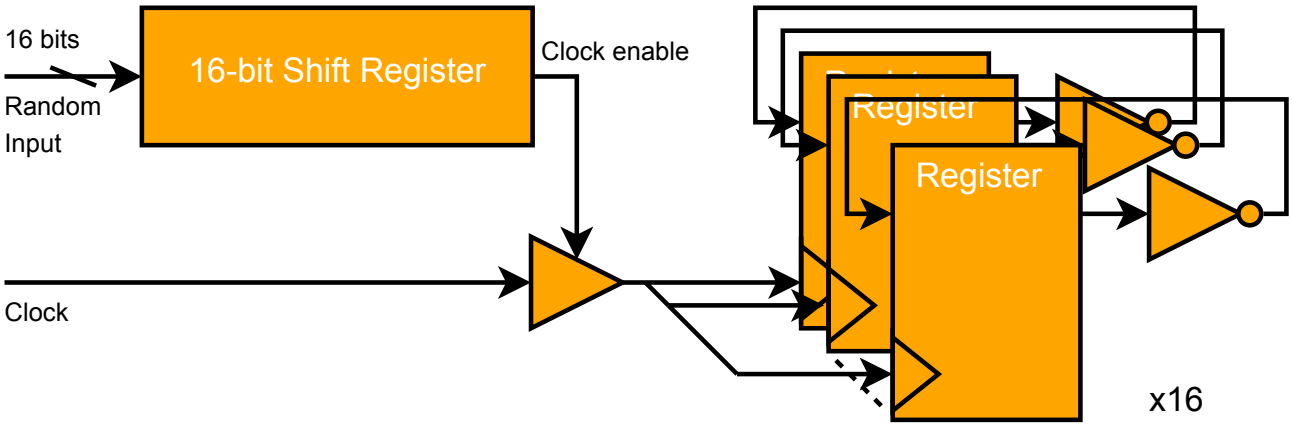


Figure 7.9: Vertical noise addition base circuitry.

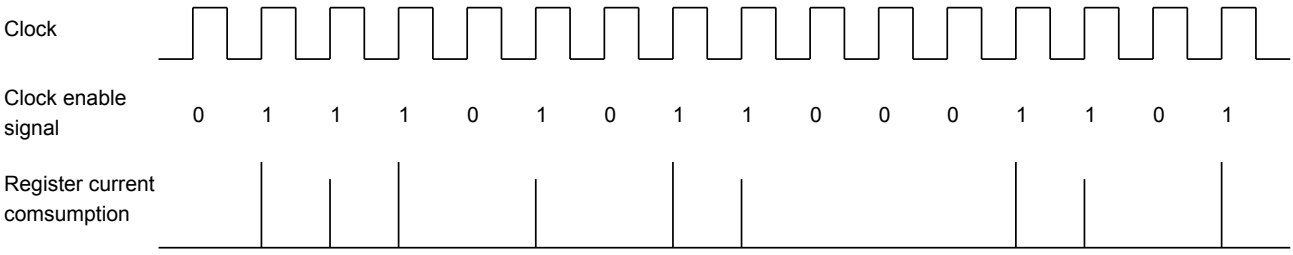


Figure 7.10: Vertical noise addition base circuitry.

7.3.2 Measurement results

In line with previous experiments, we collected measurements for random sets with bias levels 5%, 15%, 20%, 25%, 50%, 75%, and 100% (countermeasure disabled).

The first tested set was obtained for bias level 75%. One of the measured traces is shown in the top part of Figure 7.11. Correlation with the input and the S-box output of Round 1 is calculated in order to identify the interval for the attack. No correlation was found with the S-box output of Round 1, however, correlation with the input was found, as shown in the bottom graph of Figure 7.11. Thus, the start of the DES is identified, but not the end of the first round which means that an interval for the attack must be selected in another way. Thus, multiple intervals were selected and attacks were executed using each of these intervals. The interval which yield the best result (the least amount of traces to identify all S-Box sub-key candidates) is highlighted in red in the top graph of Figure 7.11. For that particular random input set (75%), 109,000 traces were needed to identify all the correct S-box sub-keys.
The same analysis was performed for the remaining random sets using the same red interval as identified in Figure 7.11. The results of these tests are summarized in Table 7.4 and can be visualized as a graph in Figure 7.12.
There are two important things to note in the results. First, the number of traces to find all the correct S-Box subkey candidates when the countermeasure is disabled is not the same as for the first experiment (dummy rounds), even though they both use the same DES engine and target FPGA. The reason for this difference can be due to several different things:

- Non optimal antenna location
- Different place and route inducing less leakage VHDL files were modified in between the two measurements to add the new countermeasures)

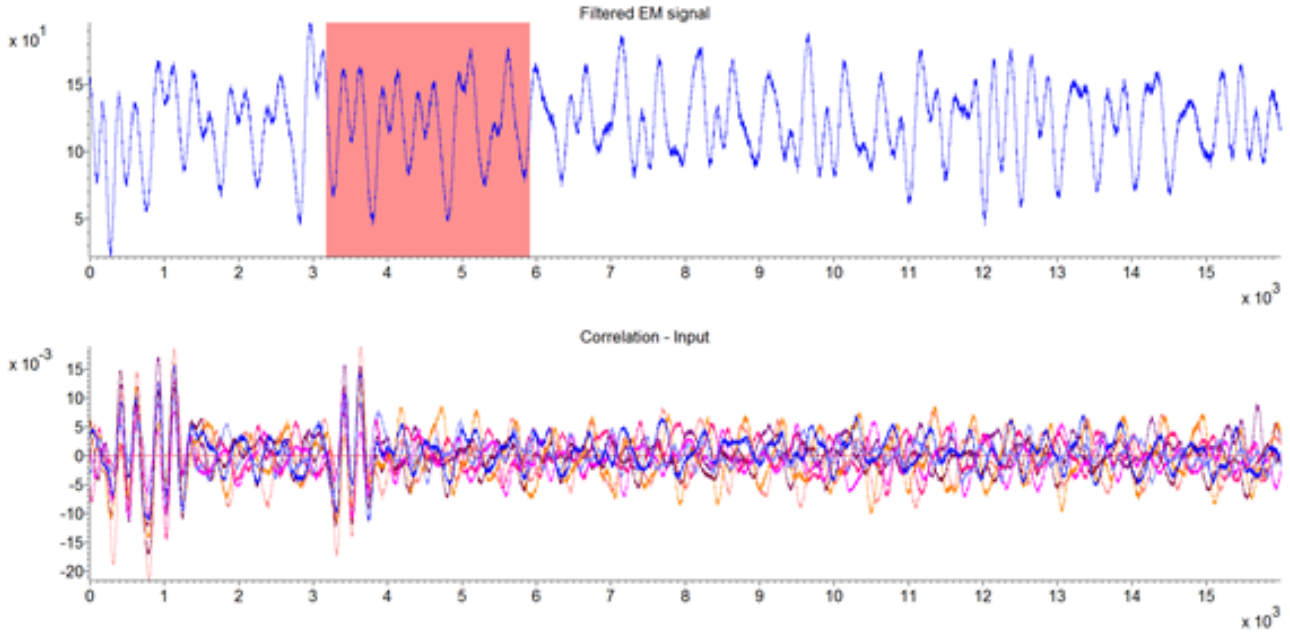


Figure 7.11: From top to bottom: Filtered EM trace measured during the test of the vertical noise countermeasure with 75% bias and eight superimposed correlation traces showing the input leakage.

However, the interest here is not to compare countermeasures between each other but to see the impact of non-ideal random numbers on one particular countermeasure. We stress that during the measurements for one particular countermeasures, neither the antenna location nor the FPGA configuration were modified.
The second observation relates to the fact that the maximum number of measurements is not obtained for unbiased random sequences, which may be expected, but rather when the bias level is around 20%. The reason for this stems from the design of the countermeasure, which was build to account for potential biases in the random numbers. In particular, it is expected that the maximal switching activity of the registers will occur for a bias of roughly 25%, which is very close to the results obtained in the experiments.

Bias level	Number of traces (rounded to the closest 500)
Countermeasure disabled	102,000
75%	109,000
50%	144,000
25%	190,000
20%	231,000
15%	217,000
5%	164,500

Table 7.4: Results summary of the analysis on the impact of the vertical noise countermeasure on the number of traces needed to retrieve fully all S-Box sub-key candidates.

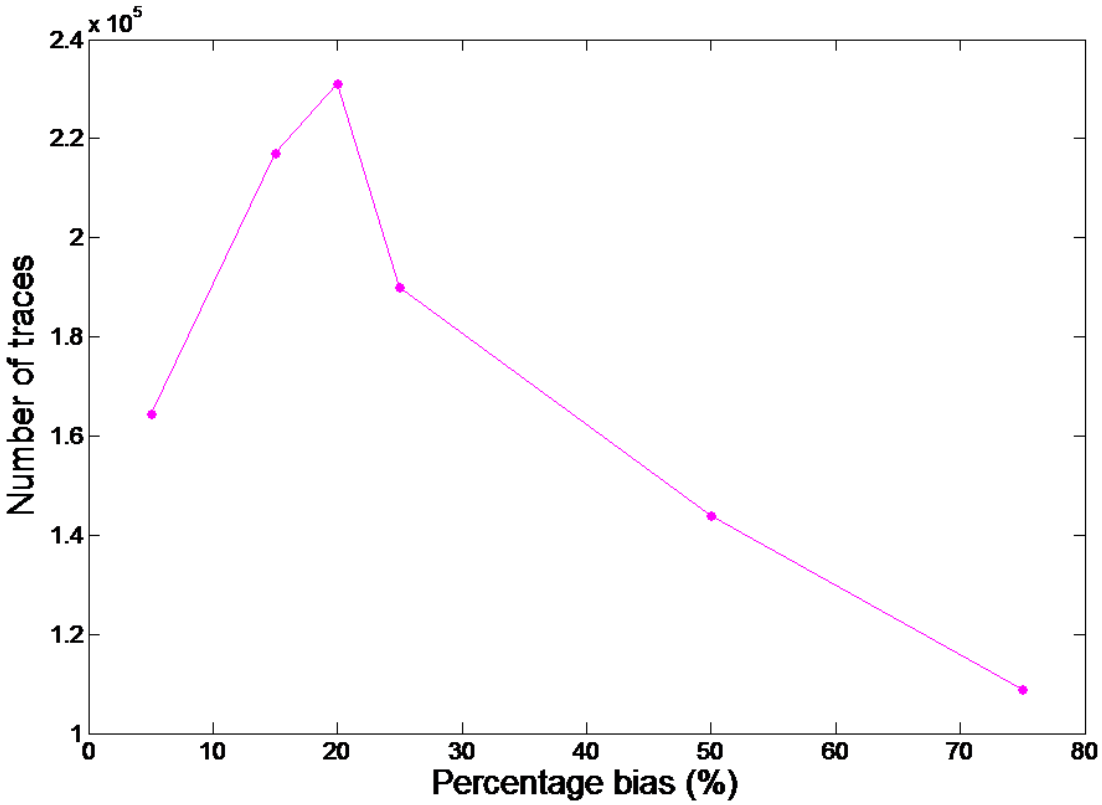


Figure 7.12: Graphical summary of the results with the countermeasure which adds vertical noise.

7.4 Dummy rounds and vertical noise

7.4.1 Countermeasure principle

This countermeasure corresponds simply to the combination of the two previously presented countermeasures. Both countermeasure uses different random inputs.

7.4.2 Measurement results

The first measurements used the set with 75% bias level. One of the measured traces is shown in the top part of Figure 7.13. Correlation with the input and the S-box output of Round 1 is calculated in order to identify the interval for the attack. No correlation was found with the S-box output of Round 1, however, correlation with the input was found, as shown in the bottom graph of Figure 7.13. Thus, the start of the DES is identified, but not the end of the first round which means that an interval for the attacks must be selected in another way. From the previous experiments on the dummy round insertion countermeasure, the interval highlighted in red in the top graph of Figure 7.13 was selected as interval for the attack. For that particular input biased random set (75%), 119,000 traces were needed to identify all the correct S-box sub-keys. The same analysis was performed for the other biased sets and the same red interval as identified in Figure 7.13. The results of these tests are summarized in Table 7.5 and can be visualized as a graph in Figure 7.14.

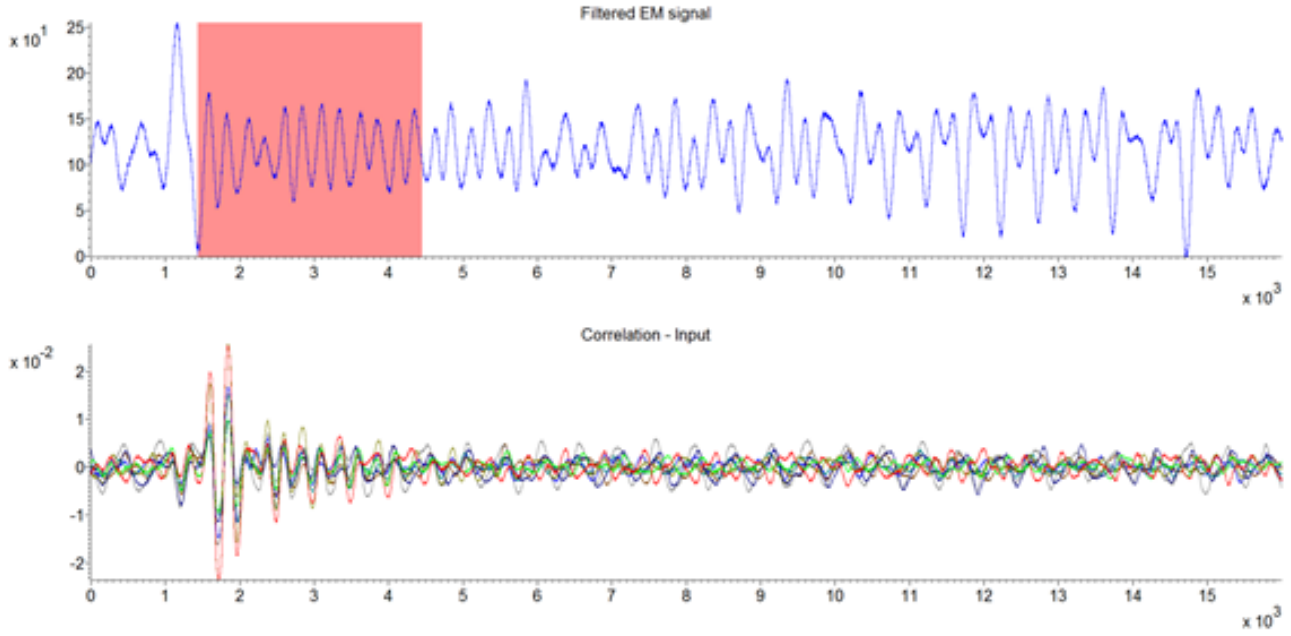


Figure 7.13: From top to bottom: Filtered EM trace measured during the test of the combination of the vertical noise and dummy round countermeasures with 75% bias level and eight superimposed correlation traces showing the input leakage.

Bias level	Number of traces (rounded to the closest 200)
Countermeasure disabled	102,000
75%	119,000
50%	322,200
25%	413,000
15%	518,000
5%	662,000

Table 7.5: Results summary of the analysis on the impact of the combination of the vertical noise and dummy round countermeasures on the number of traces needed to retrieve fully all S-Box sub-key candidates.

7.5 Conclusions

In this chapter we have presented the second part of the results on our study on the security degradation of side channel countermeasures. In particular, we have analyzed the effects of monobit bias sets on protected implementations of the DES cipher running on the HECTOR board. The results of our experiments, covering a selection of different masking and hiding techniques, validate the observations made in the previous chapter. In particular, they highlight the strong impact that biased sequences can have on the security of countermeasures. Note that, for most cases investigated, attacks become completely feasible when bias levels reach 25%.

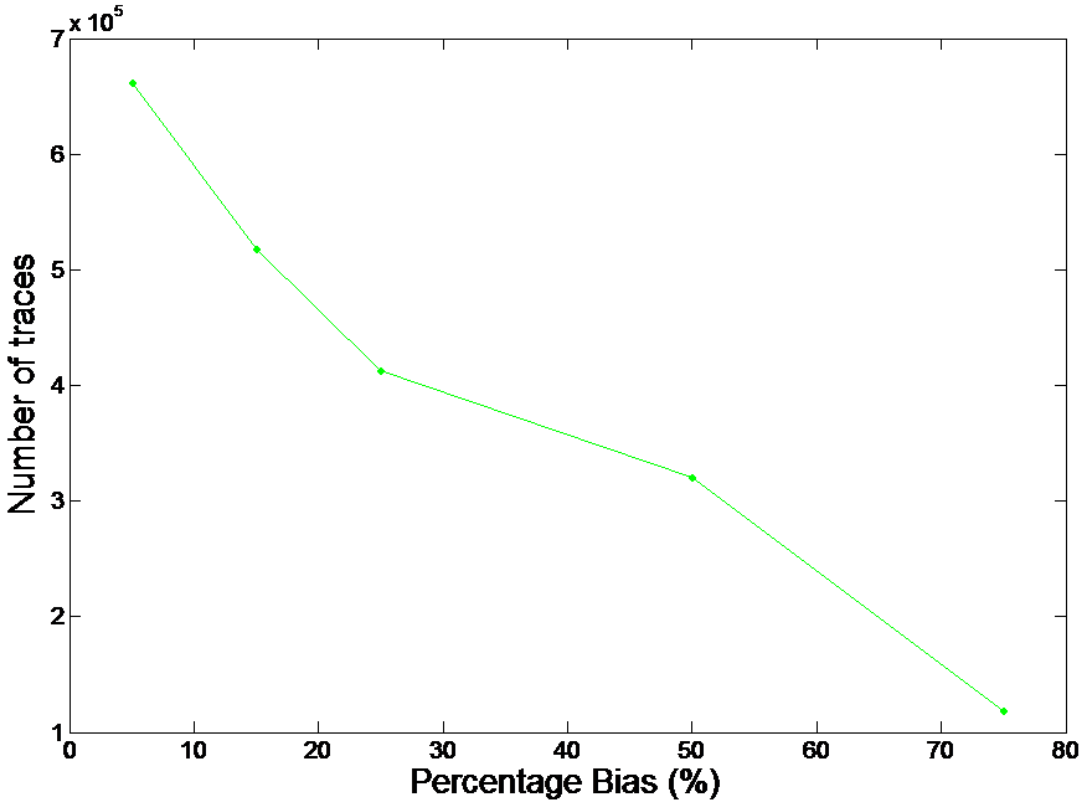


Figure 7.14: Graphical summary of the results with the combination of the vertical noise and dummy round countermeasures.

Chapter 8

Progress efficient crypto and countermeasures

In this chapter, we report several results obtained in the context of HECTOR Task T3.4, but not yet reported in deliverables D3.1 and D3.2. Some of our results bridge several tasks of WP3 and are thus reported here, in the final concluding deliverable of WP3.

8.1 Unified Masking Approach: Application to Ascon

In this section, we perform a practical evaluation of the new Unified Masking Approach (UMA) published at CHES 2017 by Gross and Mangard [49]. The results and methods reported in this section originally result from our efforts in Task 3.4: Efficient Countermeasures. To validate the improvement, we provide an implemention of the method for authenticated cipher Ascon and confirm both the efficiency of UMA and the design decisions made in Task 3.3: Efficient Crypto. The implementations are publicly available online at https://github.com/hgrosz/ascon_dom.

8.1.1 Introduction to the Unified Masking Approach

Masking is used to protect software implementations as well as hardware implementations. However, since it was shown that software based masking schemes (that lack resistance to glitches) are in general not readily suitable to protect hardware implementations [79], the research has split into masking for software implementations and masking for hardware implementations. The implementation costs of every masking scheme is thereby highly influenced by two factors. At first, the number of shares (or masks) that are required to achieve d^{th}-order security, and second the randomness costs for the evaluation of nonlinear functions. For the first one, there exists a natural lower bound of $d+1$ shares in which every critical information needs to be split in order to achieve d^{th}-order security.
For the evaluation of nonlinear functions, the required number of fresh random bits have a huge influence on the implementation costs of the masking because the generation of fresh randomness requires additional chip area, power and energy, and also limits the maximum throughput. Recently proposed software based masking schemes require (with an asymptotic bound of $d(d+1)/4$) almost half the randomness of current hardware based masking schemes. In this work we combine the most recent masking approaches from both software and hardware in a unified masking approach (UMA). The basis of the generic UMA algorithm is the algorithm of Barthe et al. which we combine with Domain Oriented Masking (DOM) [50]. The randomness requirements of UMA are in all cases less or equal to generic software masking approaches. As a

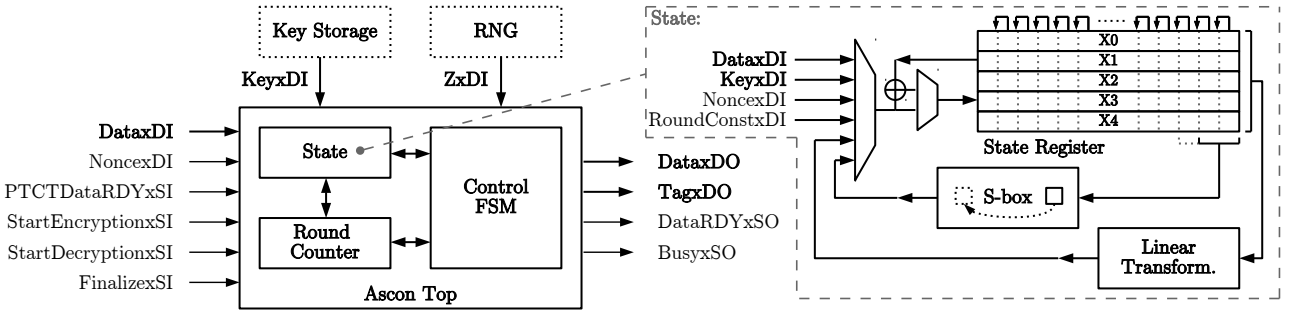


Figure 8.1: Overview of the Ascon core (left) and the state module of the Ascon design (right)

non-generic optimization, for the second protection order, we also take the solution of Belaïd et al. into account.
We show how the UMA algorithm can be efficiently ported to hardware and thereby reduce the asymptotic randomness costs from $d(d+1)/2$ to $d(d+1)/4$. Therefore, we analyze the parts of the algorithm that are susceptible to glitches and split the algorithm into smaller independent hardware modules that can be calculated in parallel. As a result, the delay in hardware is at most five cycles. We refer to the CHES 2017 paper [49] for full details of the UMA algorithm.
In the following, we compare the implementation costs and randomness requirements of UMA to the costs of DOM in a practical and scalable case study for protection orders up to 15, and analyze the SCA resistance of the UMA design with a t-test based approach.

8.1.2 Practical Evaluation on Ascon

To show the suitability of the UMA approach and to study the implications on a practical design, we decide on implementing the CAESAR candidate Ascon [36] one time with DOM and one time with the UMA approach. We decided on Ascon over the AES for example, because of its relatively compact S-box construction which allows to compare DOM versus UMA for a small percentage of non-linear functionality, but also for a high percentage of non-linear functionality if the S-box is instantiated multiple times in parallel. The design is for both DOM and UMA generic in terms of protection order and allows some further adjustments. Besides the different configuration parameters for the algorithm itself, like block sizes and round numbers, the design also allows to set the number of parallel S-boxes and how the affine transformation in the S-box is handled, for example.

Proposed Hardware Design

An overview of the top module of our hardware design is given in Figure 8.1 (left). It consists of a simple data interface to transfer associated data, plaintext or ciphertext data with ready and busy signaling which allows for simple connection with, e.g., AXI4 streaming masters. Since the nonce input and the tag output have a width of 128 bit, they are transferred via a separate port. The assumptions taken on the key storage and the random number generator (RNG) are also depicted. We assume a secure key storage that directly transfers the key to the cipher core in shared form, and an RNG that has the capability to deliver as many fresh random bits as required by the selected configuration of the core.
The core itself consists of the *control FSM* and the *round counter* that form the control path, and the *state* module that forms the data path and is responsible for all state transformations. Figure 8.1 (right) shows a simplistic schematic of the state module. The state module has a

separate FSM and performs the round transformation in four substeps: (1) during *IDLE*, the initialization of the state with the configuration constants, the key, and the nonce is ensured.
(2) in the *ADD_ROUND_CONST* state the round constant is added, and optionally other required data is either written or added to the state registers like input data or the key. Furthermore, it is possible to perform the linear parts of the S-box transformation already in this state to save pipeline registers during the S-box transformation and to save one delay cycle. This option, however, is only used for the configuration of Ascon where all 64 possible S-box instances are instantiated.
(3) the *SBOX_LAYER* state provides flexible handling of the S-box calculation with a configurable number of parallel S-box instances. Since the S-box is the only non-linear part of the transformation, its size grows quadratically with the protection order and not linearly as the other data path parts of the design. The configurable number of S-boxes thus allows to choose a trade-off between throughput and chip area, power consumption, et cetera. During the S-box calculation the state registers are shifted and the S-box module is fed with the configured number of state slices with five bits each slice. The result of the S-box calculation is written back during the state shifting. Since the minimum delay of the S-box changes with the protection order and whether the DOM or UMA approach is used, the S-box calculation takes one to 70 cycles.
(4) in the *LINEAR_LAYER* state the whole linear part of the round transformation is calculated in a single clock cycle. The linear transformation simply adds two rotated copies of one state row with itself. It would be possible to breakdown this step into smaller chunks to save area. However, the performance overhead and the additional registers required to do so, would relativize the chip area savings especially for higher orders.

S-box construction. Ascons's S-box is affine equivalent to the Keccak S-box and takes five (shared) bits as an input (see Figure 8.2). The figure shows where the pipeline registers are placed in our S-box design (green dotted lines). The first pipeline stage (Stage 0, grey) is optionally already calculated in the *ADD_ROUND_CONST* stage. The registers after the XOR gate in State 0 are important for the glitch resistance and therefore for the security of the design. Without this registers, the second masked AND gate from the top (red paths), for example, could temporarily be sourced two times by the shares of x_1 for both inputs of the masked AND gate. Because the masked AND gate mixes shares from different domains, a timing dependent violation (glitch) of the d-probing resistance could occur. Note that the XOR gates at the output do not require an additional register stage because they are fed into one of the state registers. As long as no share domains are crossed during the linear parts of the transformation the probing security is thus given. We assure this by associating each share and each part of the circuit with one specific share domain (or index) and keeping this for the entire circuit. The other pipelining registers are required because of the delay of the masked AND gates which is one cycle for the DOM gate, and up to five cycles for the UMA AND gate [49].

Implementation Results

All results stated in this section are post-synthesis results for a 90 nm Low-K UMC process with 1 V supply voltage and a 20 MHz clock. The designs were synthesized with the Cadence Encounter RTL compiler v14.20-s064-1. Figure 8.3 compares the area requirements of the UMA approach with DOM for the pipelined Ascon implementation with a single S-box instance. The figure on the left shows the comparison of single masked AND gates inside the Ascon design, while figure right compares the whole implementations of the design. Comparing this results with our predictions [49] reveals that the expected gate counts for DOM quite nicely

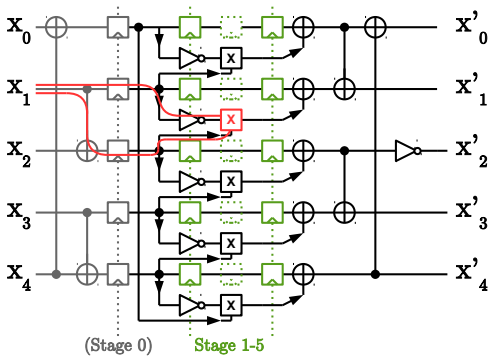


Figure 8.2: Ascon's S-box module with optional affine transformation at input (grey) and variable number of pipeline registers (green)

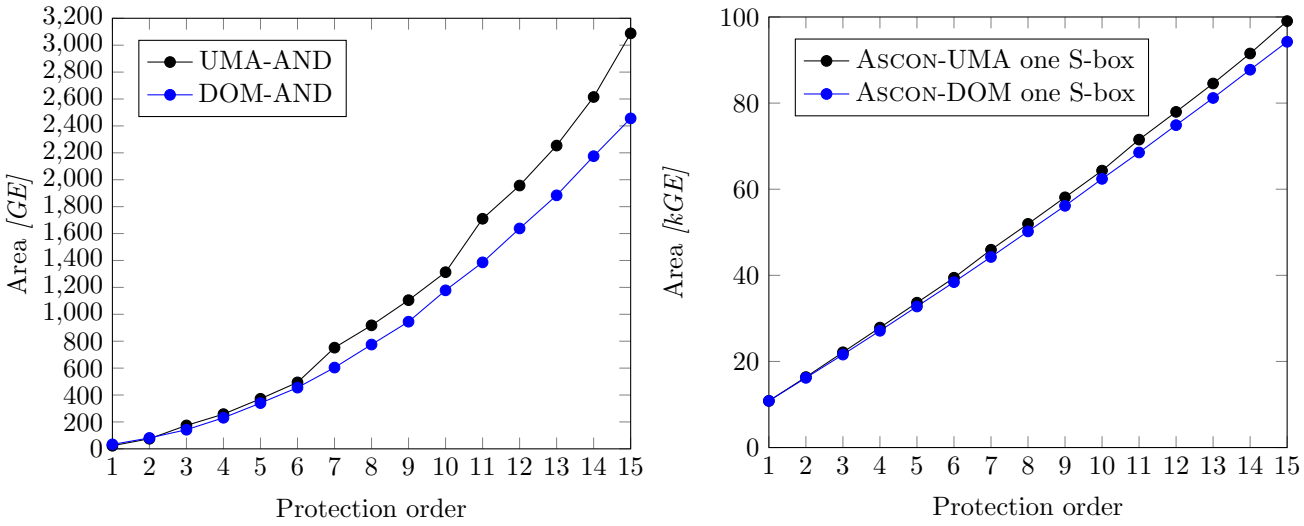


Figure 8.3: UMA versus DOM area requirements for different protection orders. Left figure compares masked AND gates, right figure compares full Ascon implementations

match the practical results. For the UMA approach, on the other hand, the practical results are always lower than the stated numbers. The reduction results from the fact that the amount of required pipelining registers for the operands is reduced because the pipelining register are shared among the masked AND gates. This does not affect the DOM implementation because the multiplication results are always calculated within only one delay cycle.
The right figure shows that the difference for the single S-box Ascon implementation is relatively low especially for low protection orders, and seems to grow only linearly within the synthesized range for d between 1 and 15. For the first order implementation both designs require about 10.8 kGE. For the second order implementation the difference is still only about 200 GE (16.2 kGE for DOM versus 16.4 kGE). The difference grows with the protection order and is about 4.8 kGE for $d = 15$ which is a size difference of about 5 %. The seemingly linear growth in area requirements for both approaches is observed because the S-box is only a relatively small part with 3-20 % of the design which grows quadratically, while the state registers that grow linearly dominate the area requirements with 96-80 %.
We also synthesized the design for 64 parallel S-boxes which makes the implementation much faster in terms of throughput but also has a huge impact on the area requirements (see Figure 8.4). The characteristics for UMA and DOM look pretty similar to the comparison of the masked AND gates in Figure 8.3 (left) and shows a quadratic increase with the protection order. The

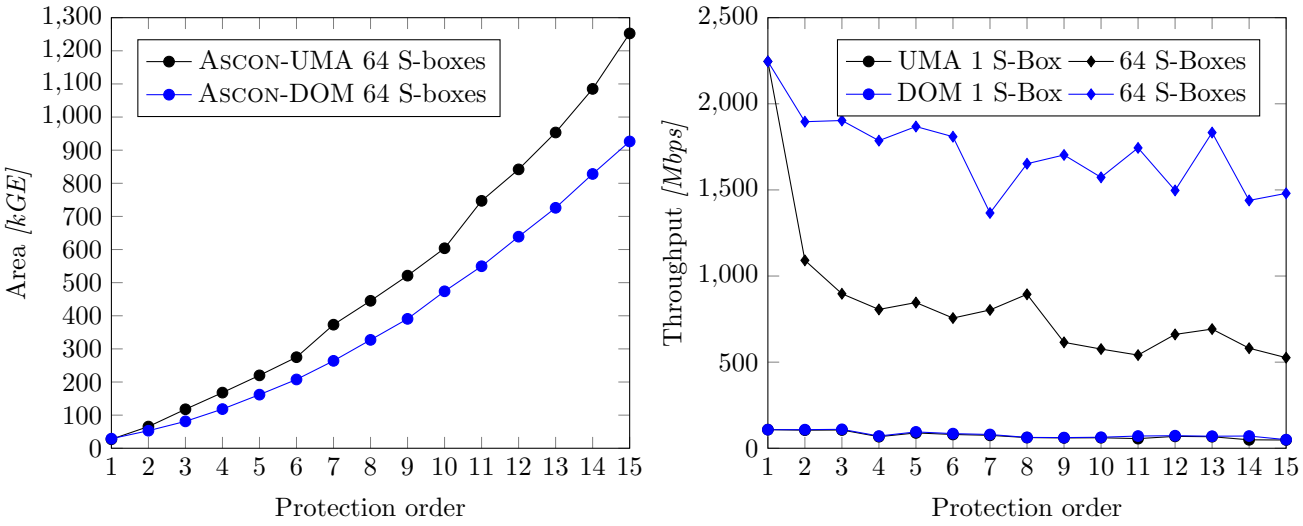


Figure 8.4: UMA versus DOM area requirements for different protection orders and 64 parallel S-boxes (left) and throughput comparison in the right figure

chip area is now between 28 kGE ($d = 1$) and 1,250 kGE ($d = 15$) for UMA and 926 kGE for DOM. The S-box requires between 55 % and 92 % of the whole chip area.

Throughput. To compare the maximum throughput achieved by our designs we calculated the maximum clock frequency for which our design is expected to work for typical operating conditions (1 V supply, and 25 °C) over the timing slack for the longest delay path. This frequency is then multiplied with the block size for our encryption (64 bits) divided by the required cycles for absorbing the data in the state of Ascon (for six consecutive round transformations).
The results are shown in Figure 8.4. The throughput of both masking approaches with only one S-box instance is quite similar which can be explained with the high number of cycles required for calculating one round transformation (402-426 cycles for UMA versus 402 cycles for DOM). The UMA approach achieves a throughput between 48 Mbps and 108 Mbps, and the DOM design between 50 Mbps and 108 Mbps for the single S-box variants.
For 64 parallel S-boxes the gap between DOM and UMA increases because DOM requires only 18 cycles to absorb one block of data while UMA requires between 18 and 42 cycles which is a overhead of more than 130 %. Therefore, also the throughput is in average more than halved for the UMA implementation. The UMA design achieves between 0.5 Gbps and 2.3 Gbps, and DOM Ascon between 1.5 Gbps and 2.3 Gbps.

Randomness. The amount of randomness required for the UMA and DOM designs can be calculated from [49] by multiplying the stated number by five (for the five S-box bits), and additionally with 64 in case of the 64 parallel S-box version. For the single S-box design, the (maximum) amount of randomness required per cycle for the UMA design is thus between 5 bits for $d = 1$ and 320 bits for $d = 15$, and for DOM between 5 bits and 600 bits. For the 64 parallel S-boxes design, the first-order designs already require 320 bits per cycle, and for the 15^{th}-order designs the randomness requirements grow to 20 kbits and 37.5 kbits per cycle, respectively.

8.1.3 Side Channel Evaluation

In order to analyze the correctness and the resistance of our implementations, we performed a statistical t-test according to Goodwill et al. [46] on leakage traces of the S-box designs of the UMA variants. We note that t-tests are unfeasible to prove any general statements on the security of a design (for all possible conditions and signal timings) as it would be required for a complete security verification. However, to the best of our knowledge there exist no tools which are suitable to prove the security of higher-order masked circuits in the presence of glitches in a formal way. T-tests only allow statements for the tested devices and under the limitations of the measurement setup. Many works test masked circuits on an FPGA and perform the t-test on the traces gathered from power measurements. This approach has the drawback that due to the relatively high noise levels the evaluation is usually limited to first and second-order multivariate t-tests. We use the recorded signal traces from the post-synthesis simulations of the netlists, which are noise-free and allows us to evaluate the designs up to the third-order. Because of the simplified signal delay model this evaluation covers only glitches resulting from cascaded logic gates and no glitches caused by different signal propagation times resulting from other circuit effects. We emphasize that we use this t-test based evaluation merely to increase the trust in the correctness and security of our implementation, and keep a formal verification open for future work.
The intuition of the t-test follows the idea that an DPA attacker can only make use of differences in leakage traces. To test that a device shows no exploitable differences, two sets of traces are collected per t-test: (1) a set with randomly picked inputs, (2) a set with fixed inputs and the according t-value is calculated. Then the t-value is calculated according to Equation 8.1 where X denotes the mean of the respective trace set, S^2 is the variance, and N is the size of the set, respectively.

$$t = \frac{X_1 - X_2}{\sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}}} \tag{8.1}$$

The null-hypothesis is that the means of both trace sets are equal, which is accepted if the calculated t-value is below the border of ± 4.5. If the t-value exceeds this border then the null-hypothesis is rejected with a confidence greater than 99.999% for large enough trace sets. A so-called centered product pre-processing step with trace points inside a six cycle time window is performed for higher-order t-tests. Beyond this time frame, the intermediates are ensured to be unrelated to the inputs. We thus combine multiple tracepoints by first normalizing the means of the trace points and then multiplying the resulting values with other normalized points inside the time window.

Results. Figure 8.5 shows the results of the t-tests for the time offsets which achieved the highest t-values for the UMA S-box implementations of Ascon. From top to bottom the figures show the results for different protection orders from $d = 0$ to $d = 3$, and from left to right we performed different orders of t-tests starting from first order up to third order. Above $d = 3$ and third-order t-tests the evaluation of the t-tests becomes too time intensive for our setup.
On the y-axis of the figures the t-values are drawn, and the y-axis denotes the used number of traces at a fraction of a million. The horizontal lines (green, inside the figures) indicate the ± 4.5 confidence border. The protection border between the figures (the red lines) separates the t-tests for which the protection order of the design is below the performed t-test (left) from the t-tests for which the test order is above (right).

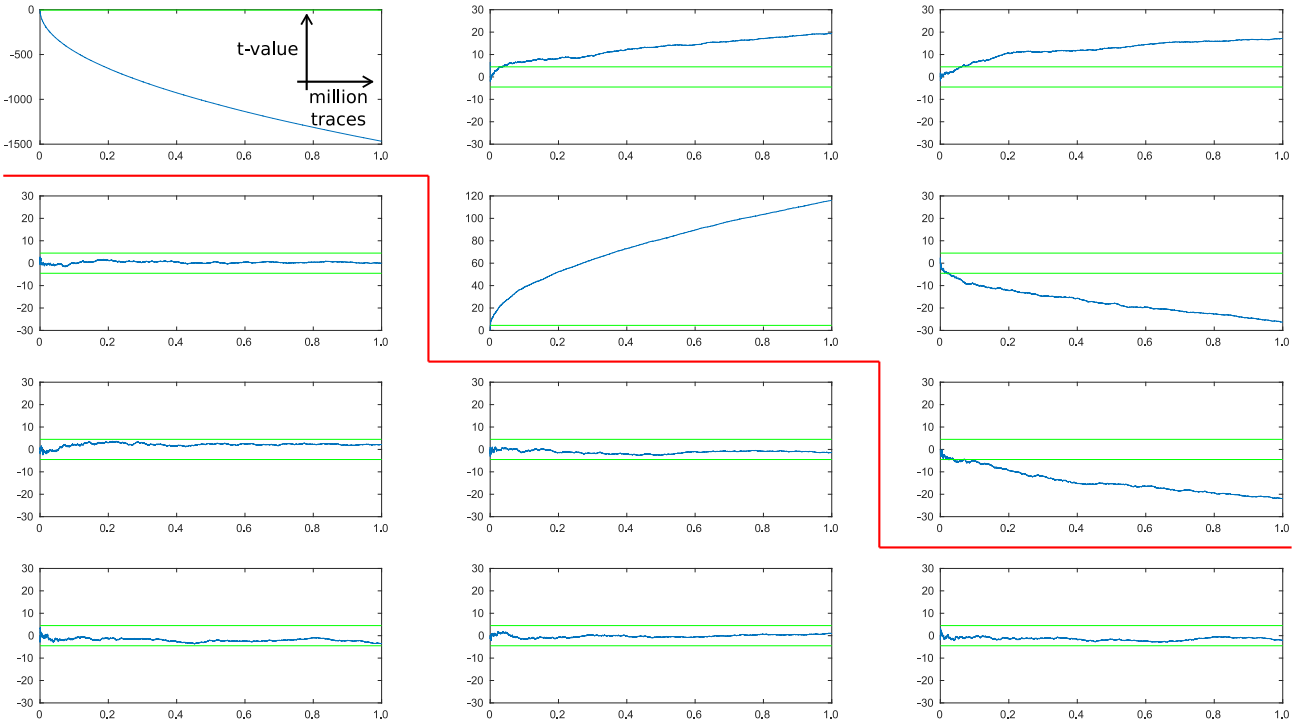


Figure 8.5: T-test evaluation for different protection orders $d = 0 \ldots 3$ (from top to bottom) and for different t-test orders (first to third, from left to right)

As intended, the t-values for the masked implementations below the protection border do not show any significant differences even after one million noise-free traces. For the unprotected implementation (top, left figure), for example, the first-order t-test fails with great confidence even after only a couple of traces, and so do the second and third-order t-tests on the right. The first-order t-test below of the first-order protected S-box does not show leakages anymore but the higher-order t-tests fail again as expected. The third-order implementation does not show any leakages anymore for the performed t-tests. We thus conclude that our implementations seem to be secure under the stated limitations.

8.1.4 Discussion on the Randomness Costs and Conclusions

In this work, we combined software and hardware based masking approaches into a unified masking approach (UMA) in order to save randomness and the cost involved. In practice, the generation of fresh randomness with high entropy is a difficult and costly task. It is, however, also difficult to put precise numbers on the cost of randomness generation because there exist many possible realizations. The following comparison should thus not be seen as statement of implementation results but reflects only one possible realization which serves as basis for the discussion.

A common way to generate many random numbers is the usage of PRNGs based on symmetric primitives, like Ascon for example. A single cipher design thus provides a fixed number of random bits, e.g., 64 bits in the case of Ascon, every few cycles. In the following comparison, we assume a one-round unrolled Ascon implementation resulting in six delay cycles and 7.1 kGE of chip area [51]. If more random bits are required, additional PRNGs are inserted, which increase the area overhead accordingly.

Figure 8.6 (left) shows the area results from Section 8.1.2 including the overhead cost for the required PRNGs. Starting with $d = 2$ for DOM, $d = 3$ for UMA for the single S-box variants,

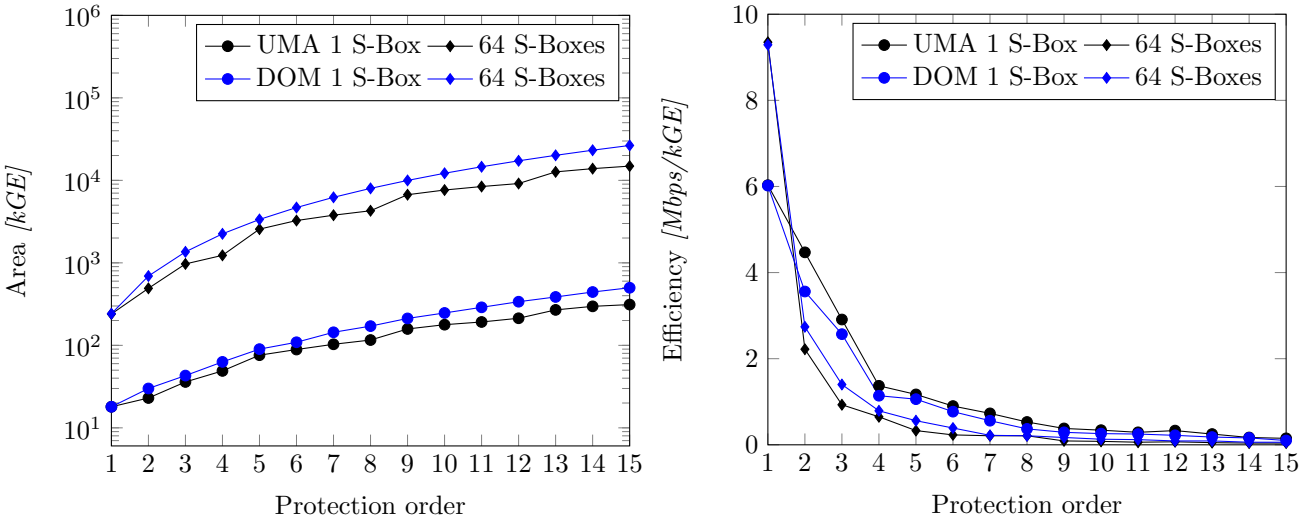


Figure 8.6: UMA versus DOM area requirements including an area estimation for the randomness generation in the left figure, and an efficiency evaluation (throughput per chip area) on the right

and for all of the 64 parallel S-box variants, one PRNG is no longer sufficient to reach the maximum possible throughput the designs offer. The randomness generation thus becomes the bottleneck of the design and additional PRNGs are required, which result in the chip area differences compared to Figures 8.3 and 8.4, respectively. As depicted, both UMA variants require less chip area than their DOM pendants. However, this comparison does not take the throughput of the designs into account (see Figure 8.4).
Figure 8.6 (right) compares the efficiency, calculated as throughput (in Mbps) over the chip area (in kGE). By using this metric, it shows that UMA is the more efficient scheme when considering the single S-box variants, while DOM is the more efficient solution for the 64 S-box variants. However, the practicality of the 64 S-box implementations with up to a few millions of GE and between 30 and 3,600 additional PRNGs is very questionable.
In practice, the most suitable approach for generating random bits and the constraints vary from application to application. While UMA seems to be the more suitable approach for low-area applications, DOM introduces less delay cycles which is a relevant constraint for performance oriented applications. To make our results comparable for future designs and under varying constraints, we make our hardware implementations available online [48].

8.2 Symbolic analysis of higher-order side channel countermeasures

In Deliverable 3.1 [82] we showed that specifying cryptographic primitives with strong security properties from the mathematical point of view is a fundamental first step, but it is not enough for achieving robust devices in practice. Designing, manufacturing and validating devices that are robust against side channel attacks once in the field is a challenging task. This statement is even more true when high-order attacks are within the scope, because of the increased size of data needed by the analysis and the increased complexity of the statistical tools.
This motivates the need for establishing a *side channel-aware design methodology*. In that context we introduced a methodology based on the functional specifications of cryptographic circuits, that is the definition of cryptographic hardware implementations in term of high-level functional

languages, such as Haskell. This creates an automated path from the initial specification down to the Register-Transfer Level (RTL) of the cryptographic primitive under scrutiny. There is a great deal of recent work targeting the formal verification of countermeasures against side channel attacks. The major concern is to provide ways to assess whether an existing specification does not leak sensitive data.
By construction, functional languages are extremely effective in modelling mathematical concepts, which allows to reason at a high level of abstractions without ambiguity.
In Deliverable 3.1 [82] we listed the three main goals that we aim at achieving by using functional languages:

1. **High assurance verification of the countermeasure**. The methodology must guarantee that masking schemes devised in the specification are effectively implemented correctly in the RTL.

2. **Statistical verification of the countermeasure**. The methodology should allow verifying that the statistical properties of a particular primitive do not present side channel information leakage, even when the implementation followed correctly the specification. This is a somewhat more important goal that can drive the exploration of new countermeasures.

3. **Specification supported by formal tools**. The methodology should allow to specify an algorithm and have automatic tools to prove that the algorithm is safe from side channel attacks. The goal is to decide, at development time, whether some formal property holds.

Once such a methodology funded on functional languages has been demonstrated viable in Deliverable 3.1 [82], we focused on how to take advantage of it, specifically for the third point listed above.
Our goal is to define some high-level symbolic properties that guarantees the robustness of a countermeasure in the contest of side channel attacks. When applied in the presented methodology, this brings two main benefits, compared to the classical workflow based on simulations:

1. The properties can be directly evaluated on the Haskell specification, without needing to generate the actual hardware design, synthesize and simulate it.

2. The verification is based on symbolic analysis rather than on statistical analysis over a large dataset coming from simulations.

Finding such general properties is a challenging task and then we start by limiting the scope and focusing first on the most used countermeasure, where sensitive data are combined in a Boolean additive way with random masks (e.g., Boolean masking or *threshold implementations* [16]).
We introduce a mathematical tool to assess the higher order vulnerability of a hardware cryptographic circuit. The method empowers the circuit designer to detect if the chosen countermeasure is effective up to the desired order. Our overarching goal is to promote the implied statistical reasoning behind the countermeasure into a symbolical one, eventually extending ordinary computer aided design of integrated circuits.
We summarize here the core results that are detailed in [21].

8.2.1 Notation and definitions

Before entering into the description of the properties, we define here the notation used, following some common standards [54]. We will also recall some concepts that were already previously introduced in Chapter 5. We use calligraphic letters for sets, e.g. $\mathcal{X}$, and capital letters, e.g. X, for random variables. A generic but deterministic value in $\mathcal{X}$ is denoted by lowercase x. A vector in $\mathcal{X}^m$ is denoted by $\mathbf{x} = (x_i)_{i=1}^m$, where $x_i \in \mathcal{X}$ for all $i = 1, \ldots, m$. We will denote by $H(\boldsymbol{x})$ the Hamming weight of a binary string $\boldsymbol{x}$.
If A and B are events, the notations $\mathbb{P}(A)$ and $\mathbb{P}(A \mid B)$ refer to the probability of A and to the conditional probability of A given B. Similarly, if X and Y are random variables, $\mathbb{E}[X]$ and $\mathbb{E}[X \mid Y]$ refer to the expectation of X and to the conditional expectation of X given Y respectively. Finally, the expectation of X given the event that Y takes the deterministic value y is denoted by $\mathbb{E}[X \mid Y = y]$.
Now we can formally provide the definition of univariate vulnerability in the context of side channel attacks.
Cryptographic primitives may expose, through a side channel, one or many intermediate values, which we call *visible variables* and denote by the letter V because they are actually processed by the hardware. On the other hand, we call *sensitive variables* the values that are deterministic functions of both the master key K and the public input P; we denote them by $S = S(K, P)$ [77] [7]. We note that visible variables are not always sensitive themselves.
Information about a visible value V, hence possibly about some sensitive values, can be derived from observations of a (data dependent) leakage [54]:

$$L = \psi(V) + N\,.$$

Here ψ is a pseudo-Boolean function that represents the way the binary values processed by the device translate to real side channel measurements (e.g. power consumption). N is a Gaussian random variable which is commonly used to account for measurement noise.
Under the assumption that V is actually a sensitive value S, one might consider a *prediction function* $\hat{L}$

$$\hat{L}(k, P) = \hat{\psi}(S(k, P))\,,$$

where k is any possible value of the key and P is a random plain text. A side channel attack can thus be mounted by using a *distinguisher function*

$$D(L(K, \cdot), \hat{L}(k, \cdot))$$

monotonically related to the statistical dependency between the actual leakage $L(K, P)$ measured with several random plain texts P, and the leakage model value $\hat{L}(k, P)$ computed using the same plain texts [54]. The attack consists in the optimization problem that aims to find the key guess k_g maximizing the distinguisher:

$$\text{argmax}_k D(L(K, \cdot), \hat{L}(k, \cdot))\,.$$

The previous statements lead to the definition of vulnerability to side channel attacks: a leakage L is *vulnerable* (to a side channel attack) if it is statistically dependent on a sensitive value, i.e.

$$\mathbb{P}(L = l \mid S = s_1) \neq \mathbb{P}(L = l \mid S = s_2)$$

for some $s_1, s_2 \in \mathcal{S}$ and $l \in \mathcal{L}$. If such a vulnerability exists, then a distinguisher D may be used to mount an attack.

To safeguard against the vulnerability, an usual solution is to prevent a *sensitive value* S to become *visible*, by splitting S into d shares $V_1, \ldots, V_d$, which are actually processed instead. This means that

$$S = V_1 \star V_2 \star \cdots \star V_d \,,$$

where $\star$ is a group operation (usually the bitwise XOR), $V_2, \ldots, V_d$ are random uniformly distributed values called *masks*, and V_1 is the *masked variable* defined in such a way that the previous equation holds. Thus, each of the shares turns out to be statistically independent of S and cannot be used alone to mount an attack.
However, this procedure does not remove any possible vulnerability: if the leakage L involves more than one share, it might still depend on S. This dependence is often revealed by a combining function $C(L)$ of the leakage: if the expected value of $C(L)$ given S is not constant, then an attack can still be mounted. See [98] for an in-depth analysis of available combining functions. In most cases, C is a polynomial of order greater than one, so that the term *higher order attack* is commonly adopted: L is *vulnerable to an n-th order attack* if one of its n-th conditional moments given S is not constant, i.e. there exists an n-th degree polynomial $C(L)$ such that

$$\mathbb{E}[C(L) \mid S = s_1] \neq \mathbb{E}[C(L) \mid S = s_2]$$

for some $s_1, s_2 \in \mathcal{S}$.
The concept of vulnerability can be directly extended to the case where the visible variables are spread over several leakages $\mathbf{L} = (L_i)_{i=1}^{l}$. If an attack exploits the information given by a vector of l leakages, we say that such an attack is *l-variate*. The definition of high order vulnerability can be generalized to the case of multivariate leakages, by considering an n-th degree polynomial $C(L)\colon \mathbb{R}^l \to \mathbb{R}$ *in l variables* as a combining function, so that the two sides of the inequality are *mixed* conditional moments.
Assuming that the leakage model is of the form:

$$L_i = \psi_i(\boldsymbol{V}) + N_i \,,$$

where $\psi_1, \ldots, \psi_l$ are pseudo-Boolean functions and $N_1, \ldots, N_l$ are Gaussian random variables independent of each other and independent of any other variable, it is possible to demonstrate the following theorem: a leakage vector is vulnerable if and only if it is vulnerable to an n-th order attack for some n (see [21] for the demonstration).
It is worth noting that some different attack approach that is more general (such as MIA [7]), or more efficient, in practical scenarios may exist. Yet, the previous theorem ensures that if a vulnerability is present, there must exist a corresponding polynomial combining function. Accordingly, the analysis performed on such polynomial combining functions is sufficient to assess the presence or lack of any vulnerability.

8.2.2 A method for detecting higher order vulnerability

We introduce now a method to detect higher-order leakage vulnerability by directly analyzing the relationship between visible variables and sensitive ones. This could be extremely useful when the designer of a (hardware) countermeasure wants to make sure that the device does not leak any information up to a certain attack order without performing a full-blown statistical analysis at design time. Provided the availability of a high level description of the implementation, as described in Deliverable D3.1 [82], the following method allows to check for vulnerabilities by performing a symbolic analysis.
For the sake of exposition, we consider all the sensitive, mask and visible bits in column vectors $\boldsymbol{S} = (S_i)_{i=1}^{s} \in \mathbb{F}_2^s$, $\boldsymbol{M} = (M_i)_{i=1}^{m} \in \mathbb{F}_2^m$ and $\boldsymbol{V} = (V_i)_{i=1}^{v} \in \mathbb{F}_2^v$.

We consider components of the leakage vector $\boldsymbol{L} = (L_i)_{i=1}^{l}$ of the form

$$L_i = \sum_{j=1}^{v} c_{i,j} V_j + N_i \qquad \forall i = 1, \ldots, l\,,$$

where $c_{i,j}$'s are real coefficients, and $N_1, \ldots, N_l$ are Gaussian noises independent of each other and of any other variable.
Given that $c_{i,j}$ can assume arbitrary values for each bit, this model covers:

1. a one to one mapping between each value of the visible variables and each value of the leakages (e.g. the identity function) — this is the most desirable case for an attacker —;
2. the Hamming weight of some binary string when $c_{i,j}$ is 1 iff the j-th visible bit leaks through the i-th leakage (and 0 otherwise).

We say that $\boldsymbol{V}$ is *vulnerable* if it is not independent of $\boldsymbol{S}$ and *n-vulnerable* if the *minimal* vulnerable subset of visible bits in $\boldsymbol{V}$ is of size n. It is worth highlighting that the property of n-vulnerability for a visible vector can be checked without resorting to statistical experiments. If the vector of visible variables is n-vulnerable, then any leakage of the form previously considered is secure against attacks of order lower than n. It becomes then possible to cast the problem of detecting an n-th order leakage vulnerability to the n-vulnerability of the corresponding visible variables.
In the case of Boolean-additive masking where visible variables are $\mathbb{F}_2$-sums of sensitive variables and masks, it is possible to devise a symbolic method to detect such a vulnerability. More formally, we assume that visible variables are related to masks and sensitive variables by the following matrix expression in $\mathbb{F}_2$:

$$\boldsymbol{V} = \begin{bmatrix} B & A \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{M} \\ \boldsymbol{S} \end{bmatrix} = B\boldsymbol{M} \oplus A\boldsymbol{S}\,,$$

where $A = (a_{i,j})$ and $B = (b_{i,j})$ are matrices with entries in $\mathbb{F}_2$, of size $v \times s$ and $v \times m$ respectively. We will call $C = \begin{bmatrix} B & A \end{bmatrix}$ the *visible variables' matrix*. Thus, V_i turns out to be the $\mathbb{F}_2$-sum of all M_h's such that $b_{i,h} = 1$ and all S_j's such that $a_{i,j} = 1$:

$$V_i = \bigoplus_{h=1}^{m} b_{i,h} M_h \oplus \bigoplus_{j=1}^{s} a_{i,j} S_j \qquad \forall i = 1, \ldots, v\,.$$

We say that a visible vector $\boldsymbol{V} \in \mathbb{F}_2^v$ satisfies the XOR-*condition* if there exists a constant row vector $\boldsymbol{\epsilon} = (\epsilon_i)_{i=1}^{v} \in \mathbb{F}_2^v$ such that the product

$$\boldsymbol{\epsilon V} = \bigoplus_{i=1}^{v} \epsilon_i V_i$$

cancels out any mask contribution (i.e. $\boldsymbol{\epsilon} B = \boldsymbol{0}$) and is a non-constant random variable.
Roughly speaking, the above condition (XOR-condition) holds when there is a combination of visible bits that: i) does not depend on masks and ii) does depend on some sensitive value. Its importance is highlighted by the following statement, which permits establishing if $\boldsymbol{V}$ is vulnerable:
Let $\boldsymbol{S} \in \mathbb{F}_2^s$, $\boldsymbol{M} \in \mathbb{F}_2^m$ and $\boldsymbol{V} = B\boldsymbol{M} \oplus A\boldsymbol{S} \in \mathbb{F}_2^v$ be the sensitive, mask and visible vectors respectively, being A and B deterministic $\mathbb{F}_2$-matrices.

1. The XOR-condition implies the vulnerability of $\boldsymbol{V}$.
2. Assuming that $\boldsymbol{S}$ and $\boldsymbol{M}$ are independent and $\boldsymbol{M}$ is uniformly distributed in $\mathbb{F}_2^m$, the vulnerability of $\boldsymbol{V}$ implies the XOR-condition.

Assuming now that all sensitive bits are independent, any $\mathbb{F}_2$-sum of them is non-constant, unless some of them cancel out. In such a case, define an easy-to-verify property of the visible variables' matrix. Assume that all the variables $S_1, \ldots, S_s$, $M_1, \ldots, M_m$ are independent of each other and $\boldsymbol{M}$ is uniformly distributed in $\mathbb{F}_2^m$. Then, $\boldsymbol{V}$ is n-vulnerable iff

$$n = \min\{H(\boldsymbol{\epsilon}) \colon \boldsymbol{\epsilon} \in \mathbb{F}_2^v, \boldsymbol{\epsilon}B = \mathbf{0},\ \boldsymbol{\epsilon}A \neq \mathbf{0}\}\,.$$

In other words, one needs to find the row vectors $\boldsymbol{\epsilon}$ that satisfy the (typically under-determined) $\mathbb{F}_2$-linear system $\boldsymbol{\epsilon}B = \mathbf{0}$ but do *not* satisfy $\boldsymbol{\epsilon}A = \mathbf{0}$: these correspond to all and only the vulnerable combinations.
The theoretical foundations we described here allow detecting early in the design cycle if a countermeasure holds up to the required protection order. When paired with the *side channel-aware design methodology* introduced in Deliverable 3.1 [82], it allows to quickly validate the soundness of a specific implementation with side channel protection.

8.3 Towards Side Channel Analysis at Design Time

Following on the motivation to establish sound side channel aware design methodologies, we present in this section our initial efforts towards building an automated side channel evaluation framework that can be integrated into Electronic Design Automation (EDA) flows. Complementary to the the approaches described in D3.1 and extended in Section 8.2 which build on high-level functional languages, our aim is to employ instantaneous power *estimations* obtained at different layers of the hardware design flow in order to provide design-time assessment of the security of an implementation.

8.3.1 Introduction

A typical silicon evaluation of a cryptographic implementation on an integrated circuit typically starts with the collection up to millions of real side channel measurements (power consumption or EM) and continues by running a battery of known attacks, e.g. SB-DPA [70] or CPA [28], as used for instance in Chapter 6. This is a manual and time consuming process that, in addition, demands a high degree of expertise. In this context, power estimations rise as an attractive alternative to assess the security of cryptographic implementations at design time. They have the potential to capture information leakage through the instantaneous power consumption already at pre-layout stages. Estimation techniques for typical hardware design constraints are long-studied and well integrated into EDA tools. This is the case for instance for *area* and *delay*, which are analyzed at various stages in the hardware design flow and optimized according to the application requirements. Already at high level, e.g. at RTL, hardware designers have access to rather accurate delay or area models. These are embedded in standard cell libraries, and they are used both for simulation purposes and by synthesis tools. Models to estimate power consumption are less accurate but still exist. They are however targeted towards low-power/low-energy design, a particular constraint for e.g. battery-operated devices.

In the remaining of this section, we document our initial efforts towards building a side channel aware design flow that can be integrated in standard EDA contexts. Although side channel analyses at design time have been partially tackled in earlier works, we aim to address the problem in a wholesome and methodical manner. Our efforts are placed along two major lines.

1. We devise a toolchain compatible with modern EDA tool chains for semi-custom ASIC digital design. Note that it is not our goal to create new synthesis tools, nor circuit simulators. Instead, we use a set of commercially available tools, enriched with several custom parsers and utility tools that allow us to perform SCA evaluation using at different abstraction levels.

2. We verify the soundness of our toolchain by evaluating the SCA security of target hardware circuits protected by first-order masking. In particular, we analyze sets of estimated measurements, acquired using several models power consumption, to check for the presence of leakages.

8.3.2 Side Channel Aware Design Flow

Semi-custom ASIC design flow is otherwise known as standard-cell design flow for it is based on collections, i.e. libraries of CMOS gates. All evaluations in this flow are based on characterized library models. These are embedded in the different "views" of standard cells, that are used for synthesis, simulation and optimization at various design stages, as depicted in Figure 8.7. Quite naturally, these models become more accurate as the flow approaches the actual layout, i.e. as more details on the circuit are available. If constraints are not met, further design iterations have to be performed. This is known as *design closure*. The models used for timing analysis have been perfected over previous decades. Already at higher abstraction layers, they provide remarkably good estimations of the final result. In contrast, models for security are completely lacking. It may be possible to use existing models (for timing or power), but it is unclear whether they will give upper/lower bounds or indicate trends on security.

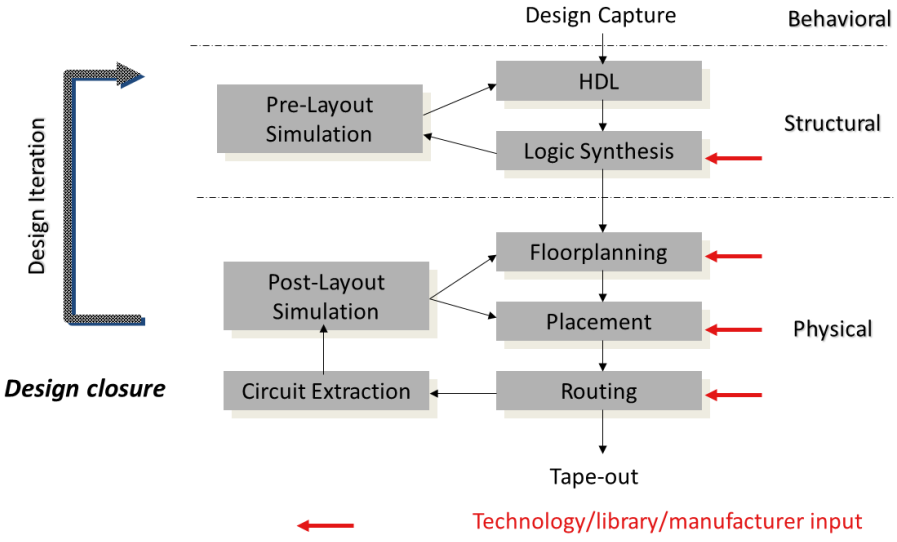


Figure 8.7: Traditional, semi-custom ASIC hardware design flow.

Starting with the well-studied semi-custom approach we propose a side channel aware design flow as depicted in Figure 8.8. Gray rectangles represent design steps, gray ellipses represent

simulation stages, and gray rounded rectangles represent different power models used for the side channel evaluation.

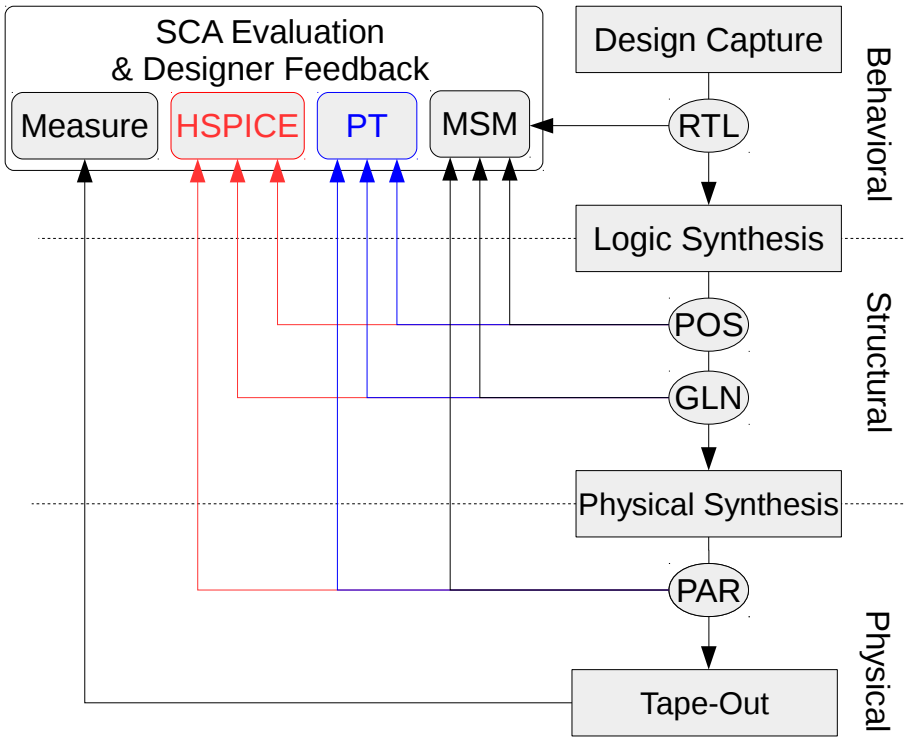


Figure 8.8: Proposed SCA aware design flow.

The main idea is to start trying to estimate the instantaneous power consumption of a design as early as possible in the design flow, in a manner that can be automated and seamlessly integrated in the existing EDA tool chains. Therefore, we rely on commercially available tools in all stages of our flow. Nevertheless, we often use tools in unorthodox manners, so we have to implement custom parsers and utility tools. Since we base our flow on standard-cell methodology, our flow and the corresponding toolchain can be used to evaluate any logic style based on CMOS standard-cell libraries. We perform all simulations and experiments using a free, publicly available 45nm library from NanGate. An additional advantage is that it provides models for SPICE simulation, commonly kept as closely guarded secrets by library vendors. Since it can not be manufactured, this library does not provide us with transistor models. We overcome this by using Predictive Technology Models (PTM) for the chosen 45nm process node.

For completeness, we list commercial tools we use: Mentor Graphics ModelSim (functional simulation), Synopsys Design Compiler (logic synthesis), Cadence SOC Encounter (physical synthesis), Synopsys PrimeTime with PX plugin (STA, and gate-level power simulation), Synopsys HSPICE (transistor-level simulations).

Models for Power Estimation

The models we investigate provide 3 different performance-precision trade-offs and can be applied independently, or consecutively, to provide different levels of assurance to the designer.

On the one end is power estimation using HSPICE, analogue simulation inherently fit to handle instantaneous power consumption, at the cost of immense computational complexity. To allow

simulation of complex digital designs, standard-cell libraries are characterized for power consumption. Therefore, event-driven instantaneous power consumption estimation using gate-level simulators such as PrimeTime (PT) results in waveforms formed by a series of superposed rectangles. In particular, existence of a rectangle at a certain moment in time is presupposed by a switching event of the corresponding cell in the functional simulation. For each switching event "height" and "width" of a rectangle depend on the characterized power and timing parameters of that cell, respectively.

On the other end, we investigate the use of simple Hamming distance models as often used in the side channel community, for both hardware and software targets. It is a very crude model, that works under the assumption that CMOS devices consume majority of power while changing states. Therefore, each toggle in the circuit results in a unitary contribution in form of a Dirac pulse at the time when the toggle occurs. In other words, Hamming distance of a transition is equated with the power consumed for this transition. We refer to it as marching-sticks model (MSM) for its graphical interpretation and to emphasize the difference from its theoretical use. For example, the design of side channel countermeasures often relies on making the number of toggles independent of the processed data, by making arithmetic assumptions about the behavior of the circuit. Consequently, they use Hamming distance to describe this behavior. Our approach is not based on theoretical considerations, but on outputs of functional simulations of implemented designs. In other words, MSM is a depiction of the design under test.

A set of normalized power consumption waveforms estimated using these different models are presented in Figure 8.9. The figure clearly depicts the switching-event-driven nature of PT power estimations, as waveforms are perfectly aligned. In detail, every "falling edge" of a blue rectangle coincides with a single toggle in the circuit. Unfortunately it is more difficult to align SPICE trace with the remaining two. For exact alignment we would need to reverse engineering all the analog parameters used in the characterization process. Instead, it is important to notice that PT estimation follows the trend of the HSPICE one to a great extent, while the number of MSM toggles only relates to the number of local extremes. Overall, Figure 8.9 depicts the qualitative loss of information on the power consumption waveform over time once we move away from HSPICE models.

Estimation Methods

For each of the traditional design stages we introduce one or more side channel evaluation stage. While these stages may be technically alike to the functional simulations for timing closure (e.g., use same tools), the rationale behind them is completely different. In the traditional design flow designers care about the values in the steady state, i.e. after all transitions have settled. On the contrary, we focus on the transitional period between states, for that is when the majority of power is consumed. As a consequence, instead of having to observe 2^n states of an n-input circuit, in order to capture all features of power consumption we need to observe 2^{2n} transitions. For example, it requires 256 observations to exhaust all logical values of an AES S-Box. On the other hand, 65 536 transitions have to be captured to fully specify circuits behavior in matters of power consumption over time. We call this type of estimations exhaustive dynamic power capturing (EDPC). Due to the exponential trends present, fully verifying behavior of larger circuits (e.g., AES round, or an entire CPU) in a straightforward manner is computationally unfeasible. Instead, elaborate verification methodologies have been developed (e.g. UVM), to provide reliable assurances using only a small subset of input vectors. Instead, for now we use sets of non-repeating random transition for circuits which exceed our computational capabilities.

Estimation steps

We start with an RTL-level estimation. At this point no physical information is available whatsoever. Therefore only MSM estimation can be performed. Within the structural level we distinguish post-synthesis (POS) and gate-level netlist (GLN) simulations. POS simulation is performed based on VITAL delays of cells, therefore timing parametrization is limited to choosing different process corner from the library VITAL models. GLN simulation are performed based on SDF back-annotation that contains information on characterized wire-load models and driving power of inputs. Therefore, GLN offers more accuracy, while POS is simpler and can be performed with less complex tools. Since a design is synthesized, mapped to a library, all three power models can be used from this point on. Simulating placed and routed (PAR) designs introduces effects of the layout in form of extracted parasitic elements. Unlike statistical approach of GLN, PAR simulations entail actual length of interconnect wires.

Automation

We aim to leverage decades of know-how of EDA tool development, therefore we abstain from creating custom synthesis tools, or simulators. On the other hand, since we often use tools in an unorthodox fashion, such as the use of SPICE simulations for digital design, we implement several utilities and parsers.

We start from a Verilog description of a design, input mappings for a test bench, and a configuration file containing: resource locations (e.g. compiled standard-cell library), as well as design and simulation parameters (e.g. critical path and process corner). From here we are able to automatically generate scripts for both logical and physical synthesis. These scripts are run manually to avoid having to automate error and warning handling, for these tools are quite complex and many things can go wrong in practice.
On the other hand, we are able to automatically generate Verilog test benches, and SPICE netlists for measuring power consumption. We have developed algorithms for efficient generations of EDPC transitions and non-repeating random transitions. Furthermore, we can automatically run

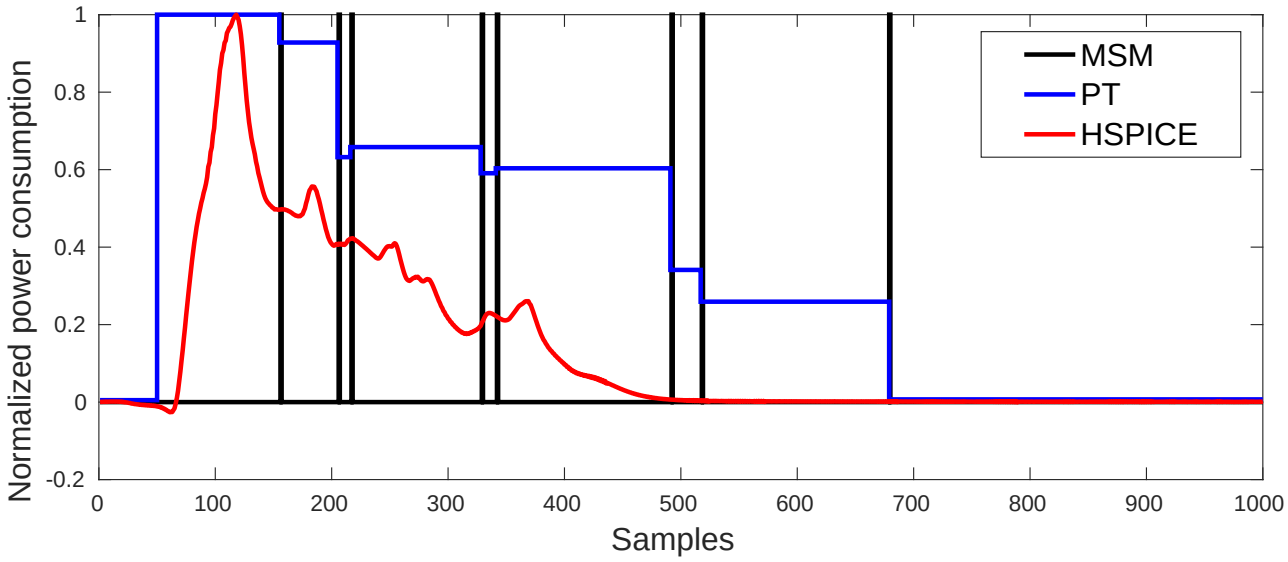


Figure 8.9: Normalized power consumption of a CMOS circuit estimated using MSM, PT, and HSPICE models.

parametrized simulations. Most importantly, we implement parsers for outputs from ModelSim, PrimeTime, and HSPICE. They allow us to handle simulated power traces in an efficient and uniform manner, compatible with the way we handle laboratory measurements.

8.3.3 Experimental Results

In this section we present results obtained from the analysis of target circuits representative of protected cryptographic implementations in hardware. We focus specifically on designs providing security against univariate attacks. The reason for this choice is twofold. First, there exists extensive literature on both the design and security analysis of 1*st*-order masked constructions that we can use to validate our results. And second, the number of inputs in most designs is sufficiently small as to analyze all possible transitions with moderate computational capabilities. Our experiments focus on a simple yet critical element in secure hardware implementations: an AND gate protected by masking. Early proposals of masked AND gates include the designs of Trichina [110] and of Ishai, Sahai and Wagner [57]. Mapping of Trichina and ISW designs to real-world implementations is however not straightforward, as both works make implicit assumptions about the behaviour of logic gates and/or signal propagations. As pointed out by Mangard and Schramm in [79], the existence of spurious glitches in hardware circuits leads to leakages that may completely undermine the security level of the countermeasure. To overcome this issue Nikova et al. introduce Threshold Implementations (TI) [94], an alternative masking construction inherently secure against glitches that works by decomposing component functions. In a more recent work Gross et al. [50] introduce an alternative masked AND design in which glitches are prevented by the insertion of a register stage.

Trichina AND gate. Our first set of experiments focuses on the masked AND gate design due to Trichina [110] shown in Figure 8.10 (left). Input and output variables are split into 2 shares such that $a = a_1 \oplus a_2$, $b = b_1 \oplus b_2$ and $c = c_1 \oplus c_2$. The design consumes one bit of randomness z. We have generated simulations for all possible input transitions, leading to $2^5 \times 2^5 = 1\,024$ measurements. Figure 8.10 (right) shows a selection of measurements obtained with different tools and models (vertically shifted and scaled for illustrative purposes).

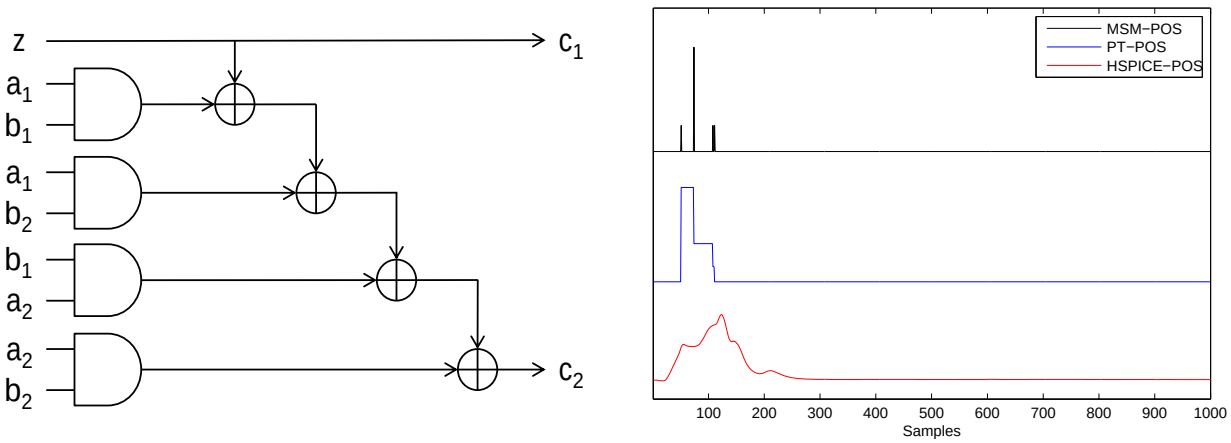


Figure 8.10: Trichina AND gate design (left). Overview of exemplary measurements from our toolchain (right).

Analyzing whether the implementation is secure against first-order attacks can be done by computing the difference of means of measurement sets partitioned according to the value of sensitive variables. In what follows the output c determines the splitting into sets. If the

implementation is secure, e.g. it does not exhibit first-order leakage, then one expects the differential trace to contain only zeros. This is shown in Figure 8.11 (left), which corresponds to the analysis at behavioural level. In contrast, the presence of first-order leakage is visualized by the presence of peaks in the differential trace. This case is shown in Figure 8.11 (right), corresponding to the analysis at structural level. Significant differences can be observed along the samples where signals propagate through the circuit. The result is consistent with the observations made in [79] related to the security of masked gates to hardware glitches, and serves to validate that our tools can correctly capture information leakage through IPC estimations at early design stages.

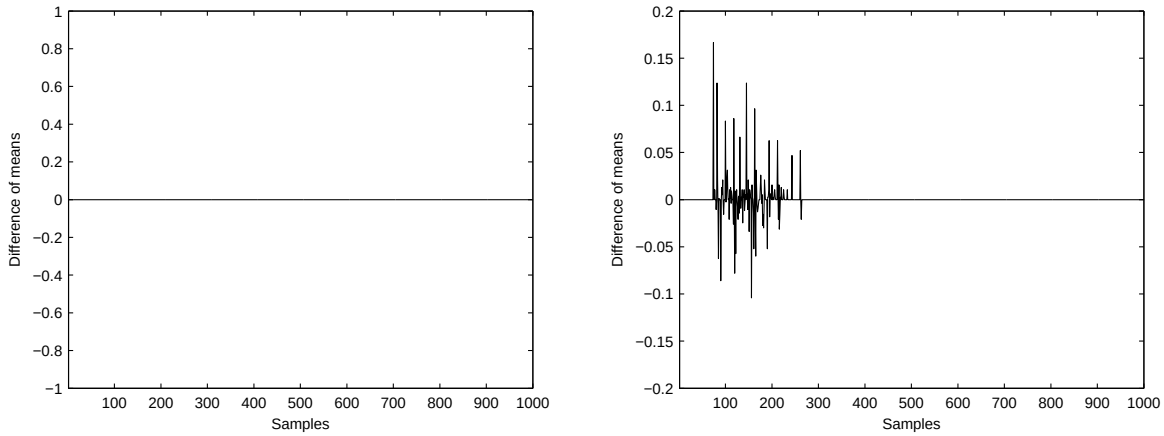


Figure 8.11: Differential trace using RTL simulations (left). Differential trace using MSM-POS simulations (right).

DOM-*indep* AND gate. Our next set of experiments focuses on the first-order DOM-*indep* multiplier from [50] depicted in Figure 8.12 (left). The design has some similarities to the Trichina AND gate: it uses two shares for the input/output variables, it consumes one bit of randomness, it demands 4 AND and 4 XOR logic gates, and guarantees first-order security. The main difference between both designs is the register stage inserted between logic gates in order to prevent leakage of sensitive information due to glitches. We show in Figure 8.12 (right) a set of representative simulations, which in this case stretch over two clock cycles.

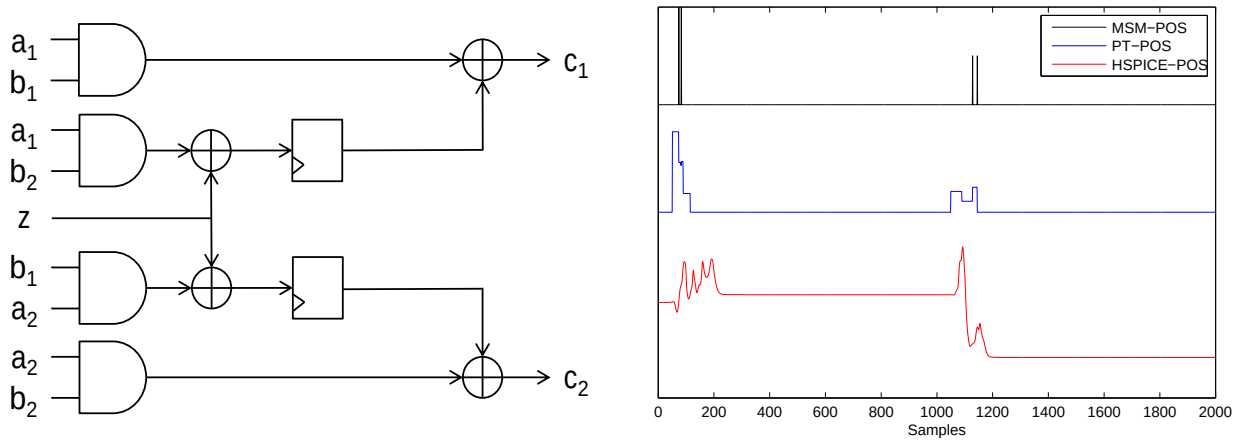


Figure 8.12: DOM-*indep* AND gate design (left). Overview of exemplary measurements from our toolchain (right).

The differential traces obtained when analyzing all 1 024 measurements (once again partitioned according to c) are zero both at behavioural and structural levels, thus confirming that the design is inherently resilient to glitches. For the purposes of validation, we evaluate the design for cases that violate certain design conditions and, consequently, are expected to exhibit first-order leakage. Figure 8.13 (left) shows the differential trace when we analyze only a subset of 512 measurements conditioned to $z = 0$. Removing the mask has no effects on the correctness of the computation, but it leads to the appearance of peaks in the differential trace during the second clock cycle. A second example is shown in Figure 8.13 (right). Here we analyze a subset of 256 measurements corresponding to the case where inputs are dependent, i.e. constrained to $a_1 = b_1$ and $a_2 = b_2$. This selection breaks one of the requirements of the DOM-*indep* multiplier, namely, to ensure that input shares are independent of each other (an alternative DOM-*dep* multiplier should be used in this case, see [50]). The peaks in the first cycle of the differential trace are due to the computations $AND(a_1, b_2)$ and $AND(a_2, b_1)$, which combine information from both shares and reveal information about the sensitive values.

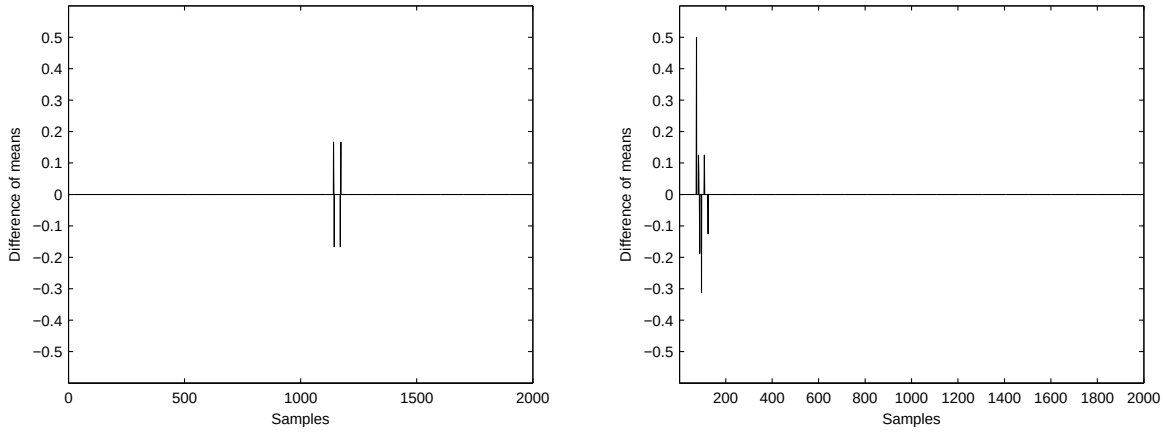


Figure 8.13: Differential trace using MSM-POS simulations with mask off (left). Differential trace using MSM-POS simulations with dependent inputs (right).

TI AND gate. Our last experiments targeting an AND gate focus on the first-order secure TI design due to Bilgin et al. [17] and shown in Figure 8.14. The design requires three shares per variable and consumes one random bit z, which acts internally as a virtual share. Security against glitches is guaranteed by TI schemes through the non-completeness rule, i.e. by ensuring that each component function is independent of at least one input share per sensitive variable. The total number of measurements for the TI AND gate increases with respect to previous designs, and demands the generation of $2^7 \times 2^7 = 16\,384$ measurements.
Again as expected, the analyses at behavioural and structural level give no indications of first-order leakage in the design. Additionally, and in contrast to the DOM-*indep* AND gate, no signs of leakage appear even when inputs are dependent. Focusing once again on a corner case, we plot in Figure 8.15 (left) the differential trace obtained when using the TI design with 2 instead of 3 inputs shares, i.e. by setting $a_3 = b_3 = 0$, which naturally exhibits first-order leakage. In Figure 8.15 (right) we plot the differential trace when checking for second-order leakage. This test requires a pre-processing step in which *expanded* traces are generated by combining all samples, i.e. each original measurement x of n samples leads to a new measurement x' of size $\binom{n}{2} + n$ samples. To reduce the computational complexity of the test, we reduce the size of the original traces from 1 000 to 250 samples, i.e. covering only the time window with circuit activity, leading to extended traces of 31 375 samples. Results shown in the plot are when combining

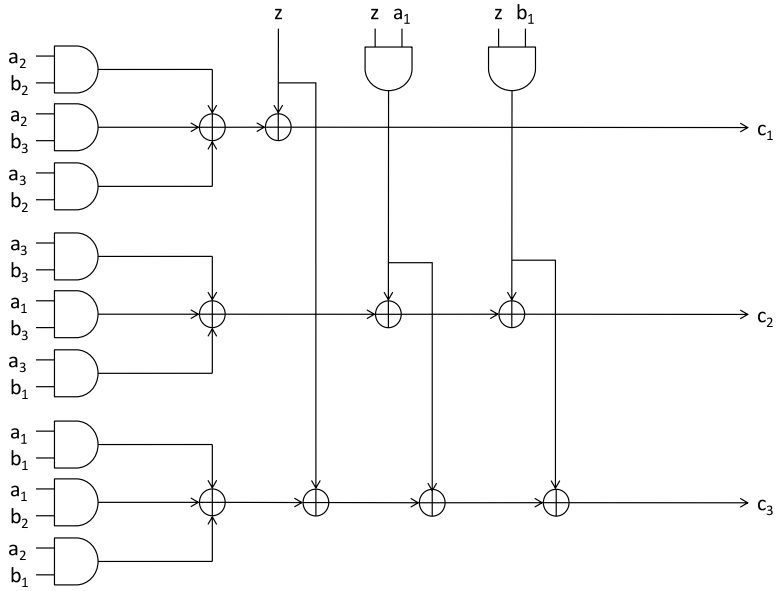


Figure 8.14: TI 3-share AND gate design.

samples using the product function [31], but similar results are obtained when using the absolute difference function [87].

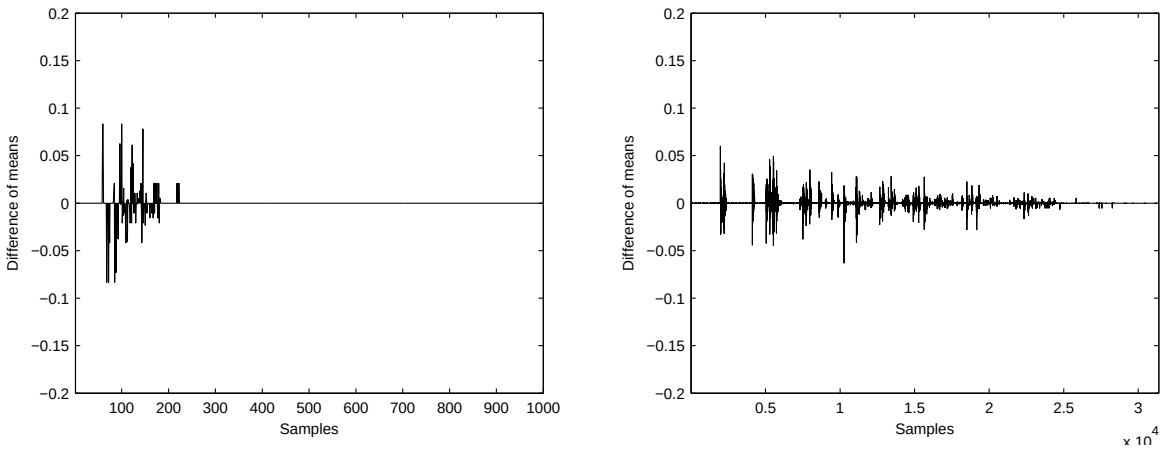


Figure 8.15: Differential trace using MSM-POS simulations without third share (left). Differential trace using pre-processed MSM-POS simulations to test for second-order leakage (right).

8.3.4 Conclusions

As illustrated by our experiments, the detection of side channel leakages in the early, pre-silicon design stages is possible. We have shown that this can even be achieved by leveraging on existing tools and using very simple and efficient estimation models such as MSM. From here, we plan to optimize our toolchain and start running experiments on the larger scale. In a second stage, we plan to further investigate the influence of different simulation and extraction parameters, and their correspondence to side channel security of manufactured chips.

Chapter 9

Conclusions

Robust cryptographic implementations and countermeasures against physical attacks both rely on randomization, and hence need random numbers. Ideally, these are "perfectly" random, having maximum entropy. Unfortunately, this security assumption does not necessarily hold in practice. Therefore, one of the main research questions investigated in the HECTOR project is to study the security degradation of cryptographic primitives and countermeasures against side channel attacks in the presence of non-ideal random numbers. This deliverable presented the scientific outcomes of tasks T3.1 and T3.2 in WP3.

The first part of this deliverable focused on the cryptanalytic techniques that capture the mathematical strength of a cryptographic primitive with respect to its keying material: related-key attacks, differential cryptanalysis based on related tweakeys, and known-key attacks. These techniques have been applied to Rijndael-160/160 and Rijndael-192/192, iFeed, MANTIS-5, and Simpira v1. These results improved the best-known cryptanalytic attacks on these ciphers and/or invalidated security claims made by the designers. Although each of these research results stands on its own, it shows that the lack of independent keying material (e.g. subkeys, round keys, etc.) could strongly reduce the cryptographic strength of an algorithm. Therefore, designers of cryptographic primitives should not limit the security evaluation to linear and differential cryptanalysis, but also consider the related and known-key setting to fully assess the mathematical strength of their algorithms. In the same line of work, the Weak Cipher Model (WCM) has been presented. This new security model allowed us to formally analyze the effect of known-key attacks on block cipher-based and permutation-based hash functions. We have applied this model to the PGV compression functions, as well as to the Grøstl (based on two permutations) and Shrimpton-Stam (based on three permutations) compression functions, and have shown that these designs do not seriously succumb to any differential known-key attack known to date.

Given the importance of high-quality random numbers for cryptographic primitives and countermeasures against physical attacks, algorithmic and cryptographic post-processing is needed to process the raw output of TRNGs and PUFs. We have explained that this design approach might reduce or even nullify the efficiency gain obtained by applying (ultra-) lightweight cryptography. Therefore, we have explored the research idea of reducing the complexity by integrating the cryptographic post-processing of TRNGs and PUFs with lightweight cryptography. As an illustration of this concept, we have presented a sponge(duplex)-like construction, based on the Motorist-layer construction, that allows to combine cryptographic hashing and encryption/authenticated-encryption within a single primitive. This sponge-construction can then directly handle the non-ideal, raw input originating from TRNGs or PUFs.

The third part of this deliverable has investigated the effects of non-ideal random numbers on

the efficacy of side channel countermeasures. Here, we have proposed to use the number of measurements required for an attack to succeed as metric to quantify the degradation of different mitigation strategies. Our analysis based on simulations showed that monobit biases, can significantly lower the security of countermeasures. In contrast, other type of non-idealities such as failure to tests T3, T5 and T8 from AIS 31, appear to have less impact on the security of the implementation. A natural interpretation of these results is that, when it comes to side channel protection, certain on-the-fly tests should be prioritized over others. In this respect, it is also worth stressing that our analyses with data from the TRNG under attack with post-processing module included, yield no apparent security degradation, due to the generated random bits being more "ideal".

The last part of this deliverable reported on the novel research results of task T3.4 of the HECTOR project, which have not yet been described in deliverable D3.1. Three main contributions have been presented: (1) a practical evaluation of the new Unified Masking Approach (UMA) by implementing this method for authenticated cipher Ascon, (2) a methodology based on the functional specifications of cryptographic circuits in high-level functional languages for symbolic analysis of higher-order side channel countermeasure, and (3) the first results towards building a side-channel aware design flow that can be integrated in standard EDA contexts.

The research results presented in this deliverable should be considered as far-out exploration, targeting a Technology Readiness Level (TRL) of 2-3. Therefore, compliance to existing standards and short-term commercial exploitation were not the driving factors of these research activities. Nevertheless, the research results presented here provide useful feedback for the other activities in the HECTOR project, in particular work packages WP2 and WP4. This also has led to the use of a single sponge construction that combines several cryptographic operations - including post-processing - into demonstrators 2 and 3 of WP4.

Chapter 10

List of Abbreviations

AEAD	Authenticated Encryption
AES	Advanced Encryption Standard
AK	AddRoundKey (AES/Rijndael transformation)
ASIC	Application-Specific Integrated Circuit
AWCM	Abortable Weak Cipher Model
BC	Block Cipher
CAESAR	Competition for Authenticated Encryption
CMOS	Complementary Metal Oxide Semiconductor
CPA	Correlation Power Analysis
CPU	Central Processing Unit
DEMA	Differential Electro-Magnetic Analysis
DES	Data Encryption Standard
DOM	Domain Oriented Masking
DPA	Differential Power Analysis
DRNG	Deterministic random number generator
EC	European Commission
ECC	Error Correcting Code
EDA	Electronic Design Automation
EDPC	Exhaustive Dynamic Power capturing
FPGA	Field Programmable Gate Array
GF	Galois Field
GLN	Gate-Level Netlist
ICM	Ideal Cipher Model
IPM	Ideal Permutation Model
MAC	Message Authentication Code
MC	MixColumns (AES/Rijndael transformation)
MSM	Marching-Sticks Model
PAR	Placed-and-Routed

PGV	Preneel, Govaerts and Vandewalle compression functions
PMAC	Parallelizable MAC
PMN	Public Message Number
POS	Post-Synthesis
PT	PrimeTime
PUF	Physically Uncloneable Function
RNG	Random Number Generator
RTL	Register Transfer Level
SB	SubBytes (AES/Rijndael transformation)
SB-DPA	Single-bit DPA
SR	ShiftRows (AES/Rijndael transformation)
TBC	Tweakable Block Cipher
TOE	Target of Evaluation
TRL	Technology Readiness Level
TRNG	True Random Number Generator
UMA	Unified Masking Approach
XOR	Exclusive-OR
WCM	Weak Cipher Model
WP	Work Package

Bibliography

[1] Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi. Multi-channel attacks. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2003.

[2] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014.

[3] Elena Andreeva, Andrey Bogdanov, and Bart Mennink. Towards understanding the known-key security of block ciphers. In *Fast Software Encryption 2013*, volume 8424 of *LNCS*, pages 348–366. Springer, Heidelberg, 2013.

[4] Paul Baecher, Pooya Farshim, Marc Fischlin, and Martijn Stam. Ideal-cipher (ir)reducibility for blockcipher-based hash functions. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 426–443. Springer, Heidelberg, 2013.

[5] Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. Inner product masking revisited. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 486–510. Springer, 2015.

[6] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015.

[7] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *Journal of Cryptology*, 24(2):269–291, 2010.

[8] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.

[9] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Caesar submission: Keyak v2. http://keyak.noekeon.org/, September 2016.

[10] Guido Bertoni, Joan Daemen, Michal Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *In Selected Areas in Cryptography*, pages 320–337.

[11] Eli Biham and Orr Dunkelman. The SHAvite-3 hash function. Submission to NIST, 2009.

[12] Eli Biham, Orr Dunkelman, and Nathan Keller. The Rectangle Attack - Rectangling the Serpent. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2001.

[13] Eli Biham, Orr Dunkelman, and Nathan Keller. A Related-Key Rectangle Attack on the Full KASUMI. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2005.

[14] Eli Biham, Orr Dunkelman, and Nathan Keller. New Combined Attacks on Block Ciphers. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *LNCS*, pages 126–144. Springer, 2005.

[15] Eli Biham, Orr Dunkelman, and Nathan Keller. Related-Key Boomerang and Rectangle Attacks. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 507–525. Springer, 2005.

[16] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(7):1188–1200, 2015.

[17] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold Implementations of All 3 x 3 and 4 x 4 S-Boxes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2012.

[18] Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[19] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and related-key attack on the full AES-256. In *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *LNCS*, pages 231–249. Springer, Heidelberg, 2009.

[20] Alex Biryukov and Ivica Nikolic. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 322–344. Springer, 2010.

[21] Elia Bisi, Filippo Melzani, and Vittorio Zaccaria. Symbolic analysis of higher-order side channel countermeasures. *IEEE Trans. Computers*, 66(6):1099–1105, 2017.

[22] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 526–541. Springer, Heidelberg, 2005.

[23] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, Heidelberg, 2002.

[24] John Black, Phillip Rogaway, Thomas Shrimpton, and Martijn Stam. An analysis of the blockcipher-based hash functions from PGV. *Journal of Cryptology*, 23(4):519–545, 2010.

[25] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE – A low-latency block cipher for pervasive computing applications. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012.

[26] Charles Bouillaguet, Pierre-Alain Fouque, and Gaëtan Leurent. Security analysis of SIMD. In *Selected Areas in Cryptography 2010*, volume 6544 of *LNCS*, pages 351–368. Springer, Heidelberg, 2011.

[27] Emmanuel Bresson, Anne Canteaut, Benoît Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, Marìa Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, and Marion Videau. Indifferentiability with distinguishers: Why Shabal does not require ideal ciphers. Cryptology ePrint Archive, Report 2009/199, 2009.

[28] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[29] Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

[30] Avik Chakraborti, Nilanjan Datta, Kazuhiko Minematsu, and Sourav Sen Gupta. Forgery on iFeed[AES] in RUP and Nonce-Misuse Settings, 2015. CAESAR mailing list.

[31] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

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

[33] W. Diffie and M. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, pages 644–654. IEEE, 1976.

[34] Christoph Dobraunig, Maria Eichlseder, Daniel Kales, and Florian Mendel. Practical key-recovery attack on MANTIS5. *IACR Transactions on Symmetric Cryptology*, 2016(2):248–260, 2016.

[35] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Cryptanalysis of Simpira v1. In Roberto Avanzi and Howard Heys, editors, *Selected Areas in Cryptography – SAC 2016*, LNCS. Springer, 2016. To appear.

[36] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to the CAESAR competition: http://competitions.cr.yp.to/round3/asconv12.pdf, 2016.

[37] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptographic Engineering*, 1(2):123–144, 2011.

[38] Orr Dunkelman, Nathan Keller, and Adi Shamir. A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2010.

[39] Lei Duo and Chao Li. Improved collision and preimage resistance bounds on PGV schemes. Cryptology ePrint Archive, Report 2006/462, 2006.

[40] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved Cryptanalysis of Rijndael. In Schneier [104], pages 213–230.

[41] Viktor Fischer and Patrick Haddad. *Circuits and Systems for Security and Privacy*, chapter 7 – Random Number Generators for Cryptography, pages 245–286. CRC Press, 2016.

[42] Pierre-Alain Fouque, Jacques Stern, and Sébastien Zimmer. Cryptanalysis of tweaked versions of SMASH and reparation. In *Selected Areas in Cryptography 2008*, volume 5381 of *LNCS*, pages 136–150. Springer, Heidelberg, 2009.

[43] Samuel Galice and Marine Minier. Improving Integral Attacks Against Rijndael-256 Up to 9 Rounds. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[44] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[45] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren Thomsen. Grøstl – a SHA-3 candidate, 2011. Submission to NIST's SHA-3 competition.

[46] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A Testing Methodology for Side-Channel Resistance Validation. In *NIST Non-Invasive Attack Testing Workshop*, 2011.

[47] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis (The "Duplication" Method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[48] Hannes Gross. DOM and UMA Masked Hardware Implementations of Ascon. https://github.com/hgrosz/ascon_dom, 2017.

[49] Hannes Gross and Stefan Mangard. Reconciling d+1 masking in hardware and software. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, 2017. To appear.

[50] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS '16, pages 3–3, New York, NY, USA, 2016. ACM.

[51] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhofer. Suit up! - Made-to-Measure Hardware Implementations of ASCON. In *DSD 2015, Madeira, Portugal, August 26-28, 2015*, pages 645–652, 2015.

[52] Shay Gueron and Nicky Mouha. Simpira: A family of efficient permutations using the AES round function. Cryptology ePrint Archive, Report 2016/122, 2016.

[53] Shay Gueron and Nicky Mouha. Simpira v2: A family of efficient permutations using the AES round function. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, volume 10031 of *LNCS*. Springer, 2016. To appear.

[54] Suvadeep Hajra and Debdeep Mukhopadhyay. Reaching the limit of nonprofiling DPA. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(6):915–927, 2015.

[55] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security - ACNS 2006*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 2006.

[56] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In *Proc. ACM Symposium on Theory of Computing 2011*, pages 89–98, New York, 2011. ACM.

[57] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[58] Jérémy Jean. Cryptanalysis of Haraka. Cryptology ePrint Archive, Report 2016/396, 2016.

[59] Jérémy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In Thomas Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *LNCS*, pages 334–353. Springer, 2016.

[60] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014.

[61] Jérémy Jean, Ivica Nikolić, Yu Sasaki, and Lei Wang. Practical cryptanalysis of PAES. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography – SAC 2014*, volume 8781 of *LNCS*, pages 228–242. Springer, 2014.

[62] Jérémy Jean, Ivica Nikolić, Yu Sasaki, and Lei Wang. Practical forgeries and distinguishers against PAES. *IEICE Transactions*, 99-A(1):39–48, 2016.

[63] Liam Keliher and Jiayuan Sui. Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. *IET IFS*, 1(2):53–57, 2007.

[64] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In Schneier [104], pages 75–93.

[65] W. Killmann and W. Schindler. AIS 31: Functionality classes and evaluation methodology for true (physical) random number generators (version 3.1). BSI, Germany. [online] Available from https://www.bsi.bund.de, 2001.

[66] W. Killmann and W. Schindler. A proposal for: Functionality classes for random number generators, Version 2.0. BSI, Germany. [online] Available from https://www.bsi.bund.de, 2011.

[67] Jongsung Kim, Seokhie Hong, Bart Preneel, Eli Biham, Orr Dunkelman, and Nathan Keller. Related-Key Boomerang and Rectangle Attacks: Theory and Experimental Analysis. *IEEE Transactions on Information Theory*, 58(7):4948–4966, 2012.

[68] Lars Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In *Advances in Cryptology - ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 315–324. Springer, Heidelberg, 2007.

[69] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[70] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[71] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka – efficient short-input hashing for post-quantum applications. Cryptology ePrint Archive, Report 2016/098, 2016.

[72] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.

[73] Thanh-Ha Le, Jessy Clédière, Cécile Canovas, Bruno Robisson, Christine Servière, and Jean-Louis Lacoume. A proposition for correlation power analysis enhancement. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2006.

[74] Yanjun Li and Wenling Wu. Improved Integral Attacks on Rijndael. *Journal of Information Science and Engineering*, 27(6):2031–2045, 2011.

[75] Moses Liskov. Constructing an ideal hash function from weak ideal compression functions. In *Selected Areas in Cryptography 2006*, volume 4356 of *LNCS*, pages 358–375. Springer, Heidelberg, 2007.

[76] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer, 2002.

[77] Victor Lomné, Emmanuel Prouff, Matthieu Rivain, Thomas Roche, and Adrian Thillard. How to estimate the success rate of higher-order side-channel attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2014*, pages 35–54. Springer, 2014.

[78] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.

[79] Stefan Mangard and Kai Schramm. Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2006.

[80] Stephen Matyas, Carl Meyer, and Jonathan Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Techn. Disclosure Bull.*, 27(10A):5658–5659, 1985.

[81] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Theory of Cryptography Conference 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, 2004.

[82] Filippo Melzani, Josep Balasch, Danilo Šijačić, Guido Bertoni, Ruggero Susella, Maria Eichelseder, and Thomas Korak. Hector deliverable 3.1 – report on the efficient implementations of crypto algorithms and building blocks and on cost and benefits of countermeasures against physical attacks. https://hector-project.eu/publications-deliverables/deliverables, 2016.

[83] Bart Mennink and Bart Preneel. Hash functions based on three permutations: A generic security analysis. In *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *LNCS*, pages 330–347. Springer, Heidelberg, 2012.

[84] Bart Mennink and Bart Preneel. On the impact of known-key attacks on hash functions. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 59–84. Springer, 2015.

[85] Bart Mennink and Bart Preneel. Efficient parallelizable hashing using small non-compressing primitives. *International Journal of Information Security*, 15(3):285–300, 2016.

[86] Thomas S. Messerges. *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms*. PhD thesis, University of Illinois, 2000.

[87] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.

[88] Shoji Miyaguchi, Kazuo Ohta, and Masahiko Iwata. Confirmation that some hash functions are not collision free. In *Advances in Cryptology - EUROCRYPT '90*, volume 473 of *LNCS*, pages 326–343. Springer, Heidelberg, 1990.

[89] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Security and Cryptology – Inscrypt 2011*, volume 7537 of *LNCS*, pages 57–76. Springer, 2011.

[90] Sean Murphy. The Return of the Cryptographic Boomerang. *IEEE Transactions on Information Theory*, 57(4):2517–2521, 2011.

[91] Jorge Nakahara, Daniel Santana de Freitas, and Raphael Chung-Wei Phan. New Multiset Attacks on Rijndael with Large Blocks. In Ed Dawson and Serge Vaudenay, editors, *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 2005.

[92] Jorge Nakahara and Ivan Carlos Pavão. Impossible-Differential Attacks on Large-Block Rijndael. In Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta, editors, *ISC*, volume 4779 of *Lecture Notes in Computer Science*, pages 104–117. Springer, 2007.

[93] Ivica Nikolić. Tiaoxin v2. Submission to the CAESAR competition, 2015.

[94] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of non-linear functions in the presence of glitches. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.

[95] NIST. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards (FIPS) Publication 197, 2001.

[96] Thomas Peyrin. Chosen-salt, chosen-counter, pseudo-collision for the compression function of SHAvite-3. NIST mailing list, 2009.

[97] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Advances in Cryptology - CRYPTO '93*, volume 773 of *LNCS*, pages 368–378. Springer, Heidelberg, 1993.

[98] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Transactions on Computers*, 58(6):799–811, 2009.

[99] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption 2004*, volume 3017 of *LNCS*, pages 371–388. Springer, Heidelberg, 2004.

[100] Phillip Rogaway and John Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *LNCS*, pages 433–450. Springer, Heidelberg, 2008.

[101] Sondre Rønjom. Invariant subspaces in Simpira. Cryptology ePrint Archive, Report 2016/248, 2016.

[102] Yu Sasaki, Sareh Emami, Deukjo Hong, and Ashish Kumar. Improved known-key distinguishers on Feistel-SP ciphers and application to Camellia. In *Australasian Conference on Information Security and Privacy - ACISP 2012*, volume 7372 of *LNCS*, pages 87–100. Springer, Heidelberg, 2012.

[103] Yu Sasaki and Kan Yasuda. Known-key distinguishers on 11-round Feistel and collision attacks on its hashing modes. In *Fast Software Encryption 2011*, volume 6733 of *LNCS*, pages 397–415. Springer, Heidelberg, 2011.

[104] Bruce Schneier, editor. *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*. Springer, 2001.

[105] Willem Schroé, Bart Mennink, Elena Andreeva, and Bart Preneel. Forgery and subkey recovery on CAESAR candidate ifeed. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 197–204. Springer, 2015.

[106] Thomas Shrimpton and Martijn Stam. Building a collision-resistant compression function from non-compressing primitives. In *International Colloquium on Automata, Languages and Programming - ICALP (2) 2008*, volume 5126 of *LNCS*, pages 643–654. Springer, Heidelberg, 2008.

[107] Martijn Stam. Blockcipher-based hashing revisited. In *Fast Software Encryption 2009*, volume 5665 of *LNCS*, pages 67–83. Springer, Heidelberg, 2009.

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

[109] Kris Tiri and Ingrid Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *Design, Automation and Test in Europe - DATE 2004*, pages 246–251. IEEE Computer Society, 2004.

[110] Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. Cryptology ePrint Archive, Report 2003/236, 2003. http://eprint.iacr.org/2003/236.

[111] David Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *LNCS*, pages 156–170. Springer, 1999.

[112] Qingju Wang, Dawu Gu, Vincent Rijmen, Ya Liu, Jiazhe Chen, and Andrey Bogdanov. Improved Impossible Differential Attacks on Large-Block Rijndael. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *ICISC*, volume 7839 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2012.

[113] Qingju Wang, Zhiqiang Liu, Deniz Toz, Kerem Varici, and Dawu Gu. Related-key rectangle cryptanalysis of rijndael-160 and rijndael-192. *IET Information Security*, 9(5):266–276, 2015.

[114] Hongjun Wu and Bart Preneel. AEGIS v1. Submission to the CAESAR competition, 2014.

[115] Shingo Yanagihara and Tetsu Iwata. Type 1.x generalized Feistel structures. *IEICE Transactions*, 97-A(4):952–963, 2014.

[116] Dingfeng Ye, Peng Wang, Lei Hu, Liping Wang, Yonghong Xie, Siwei Sun, and Ping Wang. PAES v1. Submission to the CAESAR competition, 2014.

[117] Lei Zhang, Wenling Wu, Je Hong Park, Bonwook Koo, and Yongjin Yeom. Improved Impossible Differential Attacks on Large-Block Rijndael. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC*, volume 5222 of *Lecture Notes in Computer Science*, pages 298–315. Springer, 2008.

[118] Liting Zhang, Wenling Wu, Han Sui, and Peng Wang. iFeed[AES] v1, 2014. Submission to CAESAR competition.