KU LEUVEN

Higher Order Glitch Free Implementation of 4-bit Optimal S-boxes

Thomas De Cnudde

Thesis submitted for the degree of
Master of Science in Artificial Intelligence

Thesis supervisor:
Prof. dr. Vincent Rijmen

Assessors:
Prof. dr. Liesbet Van der Perre
dr. Svetla Nikova

Mentors:
Begül Bilgin
Oscar Repáraz

Academic year 2013 – 2014

Preface

I would like to thank Begül for her endless source of patience and for the motivation she gave me during the last weeks.

Thomas De Cnudde

Contents

Abstract

The increasing deployment of small-scale wireless embedded systems calls for proper security measures. While cyptographic algorithms provide the necessary security, practical cryptographic implementations have shown to leak information that is supposed to be secret. This information is leaked through side-channels and can be observed using side-channel analysis. The order of the analysis is defined by the amount of observations one uses to compromise the system. Masking is one popular way to thwart attacks that exploit the side-channels. A masking scheme gives directions on how to divide a secret variable into shares and how to perform operations on these shares. Non-linear operations pose the difficulty in these schemes. Therefore, we focus on the non-linear S-boxes of block ciphers. A new side-channel occuring from CMOS transistor switching during computations, has shown to deteriorate the effect of masking. Therefore, extra restrictions need to be applied to a masking scheme to provide glitch resistance. One such a glitches free masking scheme has been proposed in the literature[26]. This scheme is provably-secure and can be implemented to thwart any order of side-channel analysis. The sharing in the proposed masking scheme is based on Shamir's secret sharing scheme. In addition, it uses a secure multi-party computation protocol introduced by Ben-Or *et al.* for the secure operations on these shares. The glitch resistance in this scheme is achieved by temporal or spatial separation. Temporal separation means that no operations on different shares that mask the same hidden variable, may be done at the same time. Spatial separation requires the implementation of each multi-party player to be on a different platform. This thesis considers a 1^{st}-order glitch resistant hardware implementations for the PRESENT S-box and the mCrypton S_0 S-box. Two implementations are proposed, one is based on the guidelines in [20], the other is an adaptation of this design in order to reduce the area. One of these designs is implemented for both S-boxes and their costs are compared *w.r.t.* the area, the speed and the amount of randomness.

List of Figures and Tables

List of Figures

List of Tables

Glossary

3-DES Triple Data Encryption Standard. 1

AES Advanced Encryption Standard. 1, 4, 33, 34

CMOS Complementary Metal Oxide Semiconductor. 2, 4

DPA Differential Power Analysis. 2

DUT Device Under Test. 16, 33

FPGA Field Programmable Gate Array. iv, 8, 33–35

HO-SCA Higher-order Side-channel Analysis. 2, 3

HOGF Higher-order Glitch Free. 3, 4, 11

RFID Radio-frequency Identification. 1

S-box Substitution-box. 2

SCA Side-channel Analysis. 2–5, 15, 36

SMC Secure Multi-party Computation. 3, 7–9

SP-Network Substitution-Permutation Network. 2, 11, 12

SPA Simple Power Analysis. 2

Chapter 1

Introduction

With the ever increasing presence of computing devices in our environment, the notion of ubiquitous computing is steadily taking shape. This paradigm introduces new challenges to the design of embedded systems. In addition to power consumption, speed, size and cost, the inherent security risk requires extra attention from the system designer.

Radio-frequency Identification (RFID) systems, wireless sensor networks, smart cards and other compact mobile applications have become prevalent in everyday life. Their deployment in applications ranging from supply chains to intelligent homes and even electronic body implants, has made a pressing issue of their security and privacy. Using common security measures like the Advanced Encryption Standard (AES) or the Triple Data Encryption Standard (3-DES) result in large implementation overheads with respect to *a.o.* silicon area. Therefore, lightweight, *i.e.* compact and power efficient, ciphers have emerged to reduce this overhead. Security in these ciphers is relaxed to meet other design goals, making them favourable for usage in real-world embedded systems.
One might argue that with the implications of Moore's law on miniaturisation and with enough time, the application of AES and 3-DES can become acceptable in constrained environments. However, as the computation capacity also increases, longer keylengths are required to keep exhaustive key searches unrealisable. This again increases the footprint of and implementation and therefore, the use of lightweight ciphers can not be omitted.

Encryption strategies can be divided in two categories: asymmetric and symmetric cryptography. In asymmetric cryptography, the keys to encrypt and decrypt are different. Symmetric cryptography on the other hand, uses only one secret key. In hardware environments, it makes sense to rely on symmetric cryptography as the keys can be *burned* in the devices and the area overhead for implementing both encryption and decryption is smaller. Symmetric ciphers can further be divided into stream ciphers and block ciphers. Stream ciphers are faster and more compact but their design is less understood than that of block ciphers [4]. Therefore, it is safer to

consider lightweight block ciphers for securing hardware.
A block cipher translates a plaintext into a ciphertext using a secret key. The goal for the ciphertext is to appear as a keyed pseudo-random sequence. The algorithm divides plaintext into equal blocks of bits, *e.g.* 64-bits or 128-bits. The blocks are then processed independently by iteratively repeating *round* operations. Before each round and after the last one, a *round key* is added to the input of the round, making the output key-dependent. This round key is derived from a key schedule, an algorithm that relies on the key for this derivation. When a round consists of a non-linear substitution followed by a permutation, it is referred to as a Substitution-Permutation Network (SP-Network). The permutation layer can be as simple as a scrambling of the bits. The substitution is performed by a Substitution-box (S-box) that maps an amount of input bits m to a number of output bits n, where m and n are not necessarily equal. One way to perform this substitution is by means of a lookup table stored in the memory. As this is expensive, lightweight block ciphers consider very small S-boxes. A common class of bijective S-boxes is obtained when m and n are set to four bits. In that case, they are referred to as 4-bit S-boxes.
Two 4-bit S-boxes are considered for hardware implementation. The candidates are the PRESENT S-box and the *mCrypton* S_0 S-box. These are designed with area and power constraints in mind [4, 16] and are therefore ideal for the evaluation of their practical, real-world security.

In practice, embedded systems leak information when performing cryptographic operations. These leakages, as pointed out by Kocher *et al.* [13], are known as side-channels. Values that are assumed to be secret on the algorithmic level, can be revealed or reconstructed by observing the side-channels of a cryptographic implementation. This Side-channel Analysis (SCA) can be done *e.g.* by measuring the electro-magnetic emission or the power consumption. In addition, [13] introduces Simple Power Analysis (SPA) and Differential Power Analysis (DPA), two methods to succesfully attack the side-channels by measuring the instantaneous power consumption of a device. Higher-order Side-channel Analysis (HO-SCA) generalizes such attacks by mixing several observations from different leakages. The amount of these one exploits, defines the order of the attack. To assure that the security of a device is not compromised too easily, proper countermeasures have to be taken to protect cipher implementations against SCA attacks. This problem is relevant as the required equipment to perform such attacks is available at reasonable costs and with a certain level of skill and experience, secret information can easily be extracted.

In [17], a new side-channel in the form hardware glitches was pointed out. The information leaked by the switching behaviour of Complementary Metal Oxide Semiconductor (CMOS) transistors have shown to be relevant in several papers[19, 18]. These *glitch attacks* can be performed successfully on devices that consider classical higher-order SCA countermeasures. To ensure the protection of *sensitive variables*, *i.e.* key and plaintext related information, it is important to also counteract these glitch attacks.

SCA attacks can be thwarted in numerous ways. Secure cell libraries that balance the power consumption on different values can be used. Another way to achieve resistance is to introduce noise in the form of random delays, random execution orders or even by inserting dummy operations. A specific way to cope with the effect from glitches is to balance the signal propagation, but this is expensive and requires expertise. All these methods increases the cost and size of a design significantly. A more popular countermeasure able to thwart higher-order SCA and in in some cases the effect of glitches is *masking*. This is a way to conceal sensitive intermediate values of computations by inserting randomness on the sensitive variables. It is characterized by both the amount of randomness n needed and the highest order of HO-SCA security d it can provide. Such a d^{th}-order masking scheme can theoretically always be broken by a $(d+1)^{th}$-order SCA attack. The complexity to mount a HO-SCA attack grows exponentially with the order d [6], this way, a desired level of security can be reached by choosing the right order d. Masking schemes operate on *(n,d)-sharings* of variables instead of on sensitive variables. Two schemes that have shown resilience against higher-order SCA in the presence of glitches are the *polynomial masking scheme* [26] and *threshold implementations* [21].

The general principle of masking is to split every sensitive variable x of a computation into $d+1$ shares $x_0, ..., x_d$. Shares $x_1, ..., x_d$ (called the masks) are picked randomly while the first share x_0 (called the masked variable) is chosen to satisfy an equality of the form:

$$x = x_0 \perp x_1 \perp ... \perp x_d$$

This masking scheme is referred to as *additive boolean masking* when $\perp$ denotes the addition over a field $\mathbb{F}_{2^n}$ with characteristic 2, where n is some positive integer. Operations on these shares must be executed such that each tuple of d shares is independent of any sensitive variable while still preserving the correctness of the computations. In addition, each sharing of a variable needs to be independent of all other sharings. In [22], it is shown that additive boolean masking in threshold implementations achieve d^{th}-order glitch free security at the first order only. Even though this thesis only considers first-order glitch free implementations, the polynomial masking scheme is chosen with higher-order implementations in mind. As masking schemes with greater algebraic complexity are more resistant against SCA attacks, the boolean masking scheme shows weaker provable security than the more complex polynomial masking[2]. This further motivates the choice to apply polynomial masking to secure the 4-bit S-box implementations against d^{th}-order SCA attacks.

Polynomial masking is a provably secure, Higher-order Glitch Free (HOGF) SCA countermeasure. Its sharing construction is based on Shamir's sharing scheme [27] and the shares are processed using the Secure Multi-party Computation (SMC) protocol introduced by Ben'Or *et al.* in [3], but can be built from any SMC protocol. To thwart the leakage from glitches, either a spatial or a temporal separation of the multi-party players is necessary [26]. A spatial separation, where all players

are implemented on different platforms, introduces an unacceptable area and cost overhead. While the temporal separation of players is slower, it is more feasible on CMOS platforms and will therefore be chosen.

1.1 Related Work

An algorithmic description of a d-glitches free AES implementation using polynomial masking is given in [26]. In [20], this masking scheme is used to implement AES on a hardware platform with temporally separated players. The PRESENT and mCrypton S-boxes have not yet been implemented using polynomial masking. The PRESENT S-box has been implemented in a first-order glitch free way using threshold implementations by *Kutzner et al.* [14] and *Poschmann et al.* [24] resulting in 2105 GE and 2282 GE respectively. An encryption only mCrypton block cipher with a 96-bit keylength has been implemented using multiplicative masking by *Karpinskyy et al.* [12] resulting in 7446 GE . No numbers are given for the S-boxes alone, but the key schedule, which includes the S-box, amounts to 2677 GE.

1.2 Contribution

In this thesis, the HOGF method is applied on the 4-bit S-box from the PRESENT block cipher and the 4-bit S_0 S-box from the mCrypton block cipher and the result is implemented in hardware. The polynomial masking scheme is used to acquire first-order glitch freeness. For both S-boxes, two designs are proposed. The first one, the *regular* design, is based on the guidelines for a first-order glitch free AES implementation as suggested in [20]. With a further reduction of the area in mind, a second design is proposed. A *folded* design results in a more compact implementation at the cost of computation time.
A validation of the security of the designs is considered by applying *Welsh's t-test*.

1.3 Thesis Overview

In Chapter 2, a formal overview of the necessary background theory is given *w.r.t.* the polynomial masking scheme, the PRESENT and mCrypton S-boxes and the SCA attacks. Afterwards, in Chapter 3, the four glitch free designs are described and their costs are evaluated. The results from the SCA attacks are assessed in Chapter 4. A conclusion is drawn in Chapter 5 and in addition, some open questions are presented.

Chapter 2

Preliminaries

In this chapter, an overview of the background information is given. In the first Section, the polynomial masking scheme is discussed in more detail. To better understand the working principles of this scheme, formal definitions *w.r.t.* the SCA attacks and the countermeasures are given. The operations on masked variables are discussed and special attention is given to the problem of shared multiplication. Furthermore, the complexity of this masking scheme is examined. In Section 2, the S-boxes of the PRESENT and mCrypton are described. Their setting and design objectives are explained. Using finite field arithmetics, the S-boxes are translated to polynomials and their computational costs are compared. Section 3 presents the details of the SCA evaluation method: the Welch's t-test.

2.1 Masking

The overview given here is similar to the original paper where the polynomial masking scheme is introduced [26].

2.1.1 Formal Definitions

The formal definitions of the attacks and the general principles to thwart these are given in what follows.

The system under SCA attack will be referred to as a *circuit* and is defined as:

Definition 1 *(**Circuit** C_f). A circuit C_f that implements a function f using operations from a set $\mathcal{O}$ is an oriented graph. Each node c_i defines an operation and each edge holds an intermediate variable V_{ij}. This variable serves as the output of operation c_i and as the input of operation c_j. The set of all edges of a circuit is denoted by I.*

This definition can be applied to both hardware and software implementations. In the former case, $\mathcal{O}$ contains logical binary operations. In the latter case, the set $\mathcal{O}$

contains field operations $\oplus$ and $\otimes$ in $\mathbb{F}_{2^m}$, where an m-bit architecture is assumed.

A popular model for a practical SCA attacker against a circuit C_f is the *probing adversary model* and is defined hereafter.

Definition 2 *(d^{th}**-order Probing Adversary Model**). A d^{th}-order Probing Adversary against a circuit C_f is an adversary that can choose a subset J of edges I in C_f with $\#J = d$. This Adversary can observe the variable $(L_{ij}(V_{ij})_{(i,j)\in J})$, where $(L_{ij}(.))_{ij}$ is a d-tuple of noisy leakage functions from a set of noisy leakage functions $\mathcal{L}$.*

In hardware implementations, this set of leakage functions $\mathcal{L}$ becomes the identity function. In the software context, $\mathcal{L}$ can be defined as the set of functions $L(.) = HW(.) + \mathcal{B}(\mu, \sigma)$, where HW denotes the Hamming weigth function and $\mathcal{B}$ represents gaussian noise with mean μ and standard deviation σ.

To assess the effect of glitches, the circuit definition has to be transformed from a static to a dynamic one. This can be done by making the edges time dependent. The edges of the dynamic circuit are then denoted as $V_{ij}(t)$. These can change over time, even when the circuit input is fixed.

In what follows, $C_f(t')$ refers to an internal state transition at time t' of a circuit C_f. This encompasses all non-zero transitions of the value $(V_{ij}(t))_{(i,j)\in I}$ at time $t = t'$. The d^{th}-*order Glitches Adversary Model* can then be defined as:

Definition 3 *(d^{th}**-order Glitches Adversary Model**). A d^{th}-order Glitches Adversary against a circuit C_f can choose d times $t_1, t_2, ..., t_d$ and can observe the internal state transition at these d selected times $(L_i(C_f(t_i)))_{1\le i\le d}$, where L_i is an element in the set of leakage functions $\mathcal{L}$.*

Together with these adversary models comes the notion of security. The following defines the security *w.r.t.* the d^{th}-order probing adversaries.

Definition 4 *(d**-probing Security**). A circuit C_f is d-probing secure if and only if no family of at most d elements in the set of edges is sensitive.*

The most common approach to achieve d-probing security is to split the sensitive variable into several *shares* and perform the operations on these *shares* instead. This technique of *(n, d)-sharing* is defined as:

Definition 5 *((n, d)**-sharing**). With n and d both positive integers satisfying $n > d$, an (n, d)-sharing of a variable $X \in \mathbb{F}_{2^m}$ is a family of n variables $(X_i)_{1\le i\le n}$ that satisfy the following conditions:*

1. *there exists a deterministic function $\mathcal{F} : \mathbb{F}_{2^m}^n \to \mathbb{F}_{2^m}$ such that:*

$$\mathcal{F}(X_1, ..., X_n) = X$$

2. *for every subset* $I \subset \{1, ..., n\}$ *with cardinality lower than or equal to* d*:*

$$Pr(X|(X_i)_{i\in I}) = Pr(X)$$

Together with the concept of (n, d)-sharing comes the definition of independent sharings:

Definition 6 ***(Independence of*** (n, d)***-sharings).*** *With* n *and* d *both positive integers such that* $n > d$*, two* (n, d)*-sharings* $(X_i)_{1\leq i\leq n}$ *and* $(Y_i)_{1\leq i\leq n}$ *of two arbitrary variables* X *and* Y *are independent if for every pair of subsets* $(I_x \subset \{1, ..., n\}, I_y \subset \{1, ..., n\})$*, each of cardinality lower than or equal to* d*, we have:*

$$Pr((X_i)_{i\in I_x}|(Y_i)_{i\in I_y}) = Pr((X_i)_{i\in I_x})$$

It is important to keep different sharings independent when performing operations. Remasking is therefore needed when two operands originally come from the same masking material. As pointed out in [7], it is important to perform this remasking correctly to prevent leakages.

In additive masking, d-probing security can be achieved for any order d. To be secure in the glitches adversary model however, extra countermeasures need to be taken. In [26], the authors propose a hermetical separation of computation by splitting a circuit C_f into n sub-circuits $C_{f_{1\leq i\leq n}}$. When each sub-circuit operates on one share only and if their computations leaks independently, the observation of d or less sub-circuits can give no information on the original input of the circuit C_f. When the function f can not be split into n independent computations, the different sub-circuits need to exchange shares. A SMC protocol introduces the ability to exchange intermediate results between sub-circuits. To keep the d-probing security intact, the sent information needs to be shared itself between all n sub-circuits. Each sub-circuit can then only be given a single share of a (n, d)-sharing of the intermediate result from the sub-circuit. Moreover, all shares accessed by a sub-circuit need to come from distinct and independent (n, d)-sharings. By introducing $n - 1$ communication channels $(\$_{ij})_{i\neq j}$ between all sub-circuits C_{f_i} and C_{f_j}, $i, j \in \{1, ...n\} \wedge i \neq j$, that are not accessible by another sub-circuit $C_{f_{k\neq i\vee j}}$, extended sub-circuits $(C_{f_i}, (\$_{ij})_{i\neq j})_i$ are created and the result is called an (n, d)*-multi-party circuit.*
It can be formally defined as follows:

Definition 7 ***(*** (n, d)***-multi-party circuit).*** *Let* $(X_i)_{1\leq i\leq n}$ *be an* (n, d)*-sharing of an input* X *to a circuit* C_f *that is composed of* n *extended sub-circuits* $((C_{f_1}, (\$_{1j})_{j\neq 1}), \ ... \ , (C_{f_n}, (\$_{nj})_{j\neq n}))$*.* C_f *is an* (n, d)*-multiparty circuit if and only if:*

1. *every sub-circuit* C_{f_i} *is input with share* X_i *only*
2. *for* $i \neq j$*, each sub-circuit* C_{f_i} *can access a share of an* (n, d)*-shared intermediate result from sub-circuit* C_{f_j} *through* $\$_{ij}$
3. *all shares received by a sub-circuit* C_{f_i} *relate to mutually independent* (n, d)*-sharings*

4. *the outputs of all sub-circuits* $(C_{f_i})_{1\leq i\leq n}$ *form an* (n, d)*-sharing of* $f(X)$

5. *for* $i \neq j$*, the leakage from* C_{f_i} *and* C_{f_j} *are independent*

Two possible ways to achieve independent leakage in the different sub-circuits are spatial and temporal separation. Spatial separation suggests the execution of different sub-circuits on different platforms, *e.g.* all sub-circuits are implemented on different FPGAs. In temporal separation, independent leakage is obtained by only allowing execution on one sub-circuit at a given time.

Note that d-glitches freeness implies d-probing security while the reverse is false.

Proof of Security

The (n, d)-multi-party circuit can be proven d-glitches free by following reasoning: when a secret variable X is (n, d)-shared and each share $X_{1\leq i\leq n}$ is confined to one sub-circuit only, the adversary needs to observe at least $d + 1$ sub-circuits to gain information on X, considering that the most powerful d^{th}-order glitches attack can uncover the input of a circuit. As the variables sent through the channels $\$_{ij}$ are shares independent of $X_{1\leq i\leq n}$, the adversary can only recover information about the input of the sender sub-circuit when observing these. When the circuit is designed to make the sub-circuits leak independently, the adversary has to perform at least a $(d + 1)^{th}$-order glitches attack to break the scheme.

2.1.2 Polynomial Masking

The polynomial masking scheme as introduced in [26], is built upon the conditions for d-glitches freeness. Its sharing is based on Shamir's sharing scheme [27] to mask the sensitive variables. The computations on these shared variables are then secured by applying the SMC protocol introduced by Ben Or *et al.* in [3], which is referred to as *BWG's protocol*.

In Shamir's scheme, a secret $Z \in K \equiv \mathbb{F}_{2^m})$ is shared among $n < 2^m$ players such that d players are needed to reconstruct Z. A *dealer* generates a degree-d polynomial $P_Z(X) \in K[X]$ with a constant term Z and secret coefficients a_i:

$$P_Z(X) = Z + \sum_{i=1}^{d} a_i X^i$$

This polynomial is then evaluated in n distinct, non-zero elements $\alpha_1, ..., \alpha_n \in K$, which are called the *public coefficients*. These coefficients are made publicly available and the value $Z_i = P_Z(\alpha_i)$ is distributed to each player. The secret Z can be reconstructed from the private values Z_i by polynomial interpolation and evaluation of $P_Z(X)$ in zero, as $Z = P_Z(0)$. Note that while in [26], the condition that coefficient a_d must be different from zero was found to introduce first order leakage. After correspondance with the authors, this condition is confirmed to be a syntax error.

This masking scheme is characterized by both the number n of random shares and the smallest number $d+1$ of them required to reconstruct the shared variable. Using this secret sharing scheme, a d-private SMC protocol is constructed where the number of players n satisfies $n > 2d$.

Note that there is an upper boundary on the order d of security that can be provided, namely $2^m - 1$. So technicaly, d-glitches freeness can not be provided for any order. Fortunately, in presence of noise the complexity of an attack grows exponentially with d [6] and there is no need (yet) to resort to countermeasures at those high orders.

Operations

An SMC protocol can be characterized by the maximum number of players d an adversary is allowed to corrupt before the security is compromised. Such a protocol with threshold d is then called a *d-private protocol*. BWG's protocol is said to be d-private when $2d < n$ [26]. In that case, it satisfies the conditions in definition 7. The protocol defines secure operations that are used in the polynomial masking scheme. The classes of operations in $K \equiv \mathbb{F}_{2^m}$ that can be distinguished are now discussed. Hereafter, the polynomials P_X and P_Y are defined over $K[z]$.

A univariate affine operation O over the field K, applied on a shared variable $X_{1\leq i\leq n}$ can be computed independently by all players. As O is affine, $O(X_{1\leq i\leq n})$ is an (n,d)-sharing of X and the polynomial $O(P_X(z))$ is a degree-d polynomial $P_{O(X)}(z)$. Both $P_{O(X)}(0) = O(X)$ and $P_{O(X)}(\alpha_{1\leq i\leq n}) = O(X_{1\leq i\leq n})$ are satisfied. Operations in this class are multiplication and addition with constants.

The bivariate addition operation $O \equiv \oplus$ over K can also be processed by all players independently. Adding two independent shares $X_{1\leq i\leq n}$ and $Y_{1\leq i\leq n}$ in each player results in $O(X_{1\leq i\leq n}, Y_{1\leq i\leq n}) = P_X(\alpha_{1\leq i\leq n}) \oplus P_Y(\alpha_{1\leq i\leq n})$ which is a polynomial with degree of at most d. Furthermore, correctness is guaranteed as $(P_X(0) \oplus P_Y(0)) = X \oplus Y$.

The bivariate multiplication operation $O \equiv \otimes$ over K is the tricky part in BWG's protocol. Players can not process this operation independently and as a result, communication is required between the players. The secure shared multiplication as presented in [26] is given in algorithm 1. First, a $2d$-degree polynomial $P_X(z) \otimes P_Y(z)$ is obtained. As the shares $O(X_i, Y_i)_i$ are not an (n,d)-sharing, a degree reduction and a re-randomization is needed. Therefore, an (n,d)-sharing $(Q_i(\alpha_j))_j$ of $C(X_i, Y_i)$ is sent to all other players. Finally, each player computes $\mathbf{Q}(\alpha_i) = \sum_{j=1}^{n} \lambda_j Q_j(\alpha_i)$. This (n,d)-sharing corresponds to the evaluation of polynomial $\mathbf{Q}(z)$ in the public coefficients α_i. Furthermore, this polynomial is of degree-d and has constant term $O(X,Y)$. When evaluations of the form $O \equiv b_k X^{2^k}$ are needed, the shared multiplication can be omitted when following conditions are imposed:

Algorithm 1: Secure Shared Multiplication in a Sub-circuit (C_{f_i}, $(\$_{ij})_{j\neq i}$)

`INPUT`: the i^{th} element $P_x(\alpha_i)$ of an (n,d)-sharing of x, the i^{th} element $P_y(\alpha_i)$ of an (n,d)-sharing of y and a set of channels $(\$_{ij})_{j\neq i}$.
`OUTPUT`:the i^{th} element $z_{x\otimes y}(\alpha_i)$ of an (n,d)-sharing of $x \otimes y$.
`PUBLIC`: the n public coefficients α_i, the first row $(\lambda_1, ..., \lambda_n)$ of the inverse Vandermonde matrix $(\alpha_i^j)_{1\leq i,j\leq n}$.

---

do $t_i \longleftarrow P_x(\alpha_i) \otimes P_y(\alpha_i)$
`// Generate a` d`-tuple of random numbers` a_j `in` $\mathbb{F}_{2^m}$
for $j = 1$ **to** d **do**
  $a_i \longleftarrow$**rand**(GF(2^m));
end
`// Compute an (`n`,`d`)-sharing` $(q_i(\alpha_1), ..., q_i(\alpha_n))$ `of` t_i
for $j = 1$ **to** n **do**
  $q_i(\alpha_j) \longleftarrow t_i \oplus \bigoplus_{k=1}^{d} a_k \alpha_i^k$;
end
`// Send the shares of` $t_i(\alpha_i)$ `to the other sub-circuits` C_{f_j} `using the channels` $\$_{ij}$
for $j = 1$ **to** n, $j \neq i$ **do**
  write($q_i(\alpha_j)$, $\$_{ij}$);
end
`// Receive share` $q_j(\alpha_i)$ `on` $\$_{ji}$ `from all sub-circuits` C_{f_j}
for $j = 1$ **to** n, $j \neq i$ **do**
  read($q_i(\alpha_j)$, $\$_{ij}$);
end
`// Compute the output share` $z_{x\otimes y}(\alpha_i)$
do $z_{x\otimes y}(\alpha_i) \longleftarrow \bigoplus_{j=1}^{n} \lambda_j q_j(\alpha_i)$;

- The public coefficients α_i are distinct and non-zero
- The public coefficients α_i are stable over the frobenius automorphism: for every α_i, there exists an α_j such that $\alpha_j = \alpha_i^2$

Each player can then separately perform $O(X) = b_k(P_X(\alpha_i))^{2^k} = P_{O(X)}(z)$. As the sets $\{\alpha_i^{2^k}\}_i$ and $\{\alpha_i\}_i$ are equal and as a result $\{P_{O(X)}(\alpha_i^{2^k})\}_{i\leq n} = \{P_{O(X)}(\alpha_i)\}_{i\leq n}$, this results in an (n, d)-sharing of $O(X)$. A reordering of the shares will in some cases be needed as α_j is not necessarily equal to α_i when the second condition is fulfilled.

Masking Complexity

The complexity of the masked addition, shared multiplication and squaring operations for the polynomial masking scheme in terms of the number of random values d is

given in table 2.1. Here, $n = 2d + 1$ is chosen as this is the minimal amount op players needed for d-glitches freeness. To cope with glitches, each addition and squaring operation in a function needs to be replaced with n versions of that operation. The shared multiplication on the other hand, requires $n^2(d + 1) + n$ multiplications, $n^2(d + 1) - n$ additions and 2(n-1) **read**/**write** operations. It is clear that, theoretically, it is beneficial to avoid the use of shared multiplications and consider the squared operation whenever possible.

Table 2.1: Polynomial masking complexity in terms of the number of random values d [26]

Polynomial Masked Operation	Operations in		
	$\oplus$	$\otimes$	Rand
ADDITION	$2d + 1$	-	-
MULTIPLICATION	$4d^3 + 8d^2 + 7d + 2$	$4d^3 + 8d^2 + 3d$	$2d^2 + d$
SQUARE	-	$2d + 1$	-

2.2 S-Boxes

A 4-bit S-box can be seen as a non-linear function $f : \mathbb{F}_{2^4} \to \mathbb{F}_{2^4}$. Its purpose is to increase the complexity of the relation between the key and the ciphertext in a block cipher, this is the property known as *confusion*. In this section, the PRESENT and *mCrypton* S_0 S-boxes considered for HOGF implementation are introduced.

2.2.1 PRESENT

The PRESENT block cipher performs 31 rounds, each consisting of an SP-Network. The block length equals 64-bits and key lengths of 80-bits and 128-bits are supported. The 80-bit version is recommended for lightweight applications. The substitution layer uses 16 identical 4-bit S-boxes to cover the whole block length. The S-boxes are followed by a bit oriented permutation layer that can easily be implemented in hardware using wiring. As bit-oriented permutations are not software-friendly, the target platform of the PRESENT is a hardware one.

The PRESENT S-box has been used in other lightweight crypto algorithms including the LED block cipher[11], the GOST *revisited* block cipher[23] and even the PHOTON lightweight hash function[10]. This block cipher is designed with constraints on performance, space and timing requirements in mind. Further details can be found in [4]. As of 2012, it has been standardized and it is part of the ISO/IEC 29192-2.

The proposed S-box fulfills following 4-bit to 4-bit substitution [4]:

Table 2.2: 4-bit to 4-bit substitution of the PRESENT S-box [4]

z	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[z]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

2.2.2 mCrypton

The mCrypton block cipher is derived from the CRYPTON block cipher, which was a candidate for the AES competition [15]. It operates on 64-bit block lenghts and three key length options are supported: 64-bits, 96-bits and 128-bits. The transformation from plain- to ciphertext is done by applying 12 consecutive rounds consisting of an SP-Network. A round consists of a substitution layer followed by a bit permutation within each column and a transposition of the state matrix. The state matrix is a 4×4 matrix of nibbles. Four different 4-bit S-boxes are used on this state matrix in the substitution layer. These S-boxes S_0, S_1, S_2 and S_3 are based on the inverse x^{-1} in $\mathbb{F}_{2^4}$) and are related as: $S_2 = S_0^{-1}$ and $S_3 = S_1^{-1}$.

The four mCrypton S-boxes are given in the table 2.3. In [16], further details can be found.

Table 2.3: The four mCrypton S-boxes [16]

z	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1[z]$	4	F	3	8	D	A	C	0	B	5	7	E	2	6	1	9
$S_2[z]$	1	C	7	A	6	13	5	3	F	B	2	0	8	4	9	E
$S_3[z]$	7	E	C	2	0	9	D	A	3	F	5	8	6	4	B	1
$S_4[z]$	B	0	A	7	D	6	4	2	C	E	3	9	1	5	F	8

2.2.3 Polynomial Normal Forms

The function $f : \mathbb{F}_{2^4} \to \mathbb{F}_{2^4}$ of a 4-bit S-box can be represented by a unique univariate polynomial over $\mathbb{F}_{2^4}$ with a degree of at most $2^4 - 1 = 15$. This polynomial can be obtained by expanding the following expression [8]:

$$S(x) = \sum_{z \in \mathbb{F}_{2^4}} S(z)(1 + (x + z)^{15})$$

Once expanded, the polynomial has the form:

$$S(x) = \bigoplus_{i=0}^{15} c_i x^i$$

Using the Mattson-Solomon polynomial, the coefficients of $S(x)$ can be directly computed by:

$$c_i = \begin{cases} S(0), & \text{if } i = 0 \\ \sum_{k=0}^{2^4-2} S(\alpha^k)\alpha^{-ki}, & \text{if } i \leq i \leq 2^4 - 2 \\ S(1) + \sum_{i=0}^{2^4-2} c_i, & \text{if } i = 2^4 - 1 \end{cases}$$

where α is a primitive element in $\mathbb{F}_{2^4}$.
The coefficients of the polynomial normal forms of the S-boxes are listed in table 2.4 when $r(x) = x^4 + x + 1$ is used as irreducible polynomial for the construction of $\mathbb{F}_{2^4}$.

Table 2.4: Polynomial normal form coefficients of the S-boxes

S-Box	c_{15}	c_{14}	c_{13}	c_{12}	c_{11}	c_{10}	c_9	c_8	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
PRESENT $S(x)$	0	D	D	C	E	9	9	7	4	C	A	E	7	7	0	C
mCrypton $S_0(x)$	0	D	6	D	8	9	A	7	8	6	5	9	6	6	3	4

In addition, as the mCrypton S-boxes are based on the multiplicative inverse, an affine transformation of the form $S = A(I(B(x))$ can be found that satisfies:

$$\begin{aligned} B(x) &= b_8x^8 \oplus b_4x^4 \oplus b_2x^2 \oplus b_1x \oplus b^0 \\ I(x) &= x^{-1} = x^{14} \\ A(x) &= a_8x^8 \oplus a_4x^4 \oplus a_2x^2 \oplus a_1x \oplus a_0 \end{aligned}$$

Coefficients b_8, b_4, b_2, b_1, b_0 and a_8, a_4, a_2, a_1, a_0 can be found by exhaustive search. An example of a valid transformation is given below.

$$\begin{aligned} B(x) &= 9x^8 \oplus 3x^4 \oplus 13x^2 \oplus 10x \\ I(x) &= x^{-1} = x^{14} \\ A(x) &= 2x^8 \oplus 6x^4 \oplus 12x^2 \oplus 12x \oplus 4 \end{aligned}$$

2.2.4 Comparison

The masked evaluation of the polynomial representation of a 4-bit S-box is composed of $\mathbb{F}_2^4$-affine functions and of non-linear multiplications. The masking complexity of an S-box is defined in [5] as the minimal amount of non-linear multiplications required to evaluate its polynomial. Based on this number, the complexity of the S-box evaluations can be compared.

When calculating a power x^α from another power x^β, non-linear multiplication can be omitted if and only if α and β lie in the same *cyclotomic class*. A cyclotomic class is define as:

Definition 8 *(**Cyclotomic Class** C_α). The cyclotomic class C_α of $\alpha \in [0; 2^n - 2]$ is the set define by:*

$$C_\alpha = \{\alpha \cdot 2^i \ mod\, 2^n - 1, \quad i \in [0; n-1]\} \tag{2.1}$$

An important result is that squaring is linear within a cyclotomic class. Consequently, the S-box complexity is related to the amount of these classes. For the field $\mathbb{F}_{2^4}$ with $r(x) = x^4 + x + 1$ as irreducible polynomial, following cyclotomic classes can be found:

$$C_0 = \{0\}, C_1 = \{1, 2, 4, 8\}, C_3 = \{3, 6, C, 9\}, C_5 = \{5, A\}, C_7 = \{7, E, D, B\}$$

PRESENT

The evaluation of the PRESENT S-box polynomial requires three non-linear multiplications:

$$\begin{aligned} x^3 &\leftarrow x \otimes x^2 \\ x^5 &\leftarrow x \otimes x^4 \\ x^7 &\leftarrow x \otimes x^2 \otimes x^4 \end{aligned}$$

In the last expression, x^3 or x^5 is assumed to be obtained from a previous multiplication.

mCrypton

To evaluate the mCrypton polynomial, the amount of traversed cyclotomic classes required is the same as for the PRESENT. When evaluating this way, the masking complexity is also three.

However, mCrypton is based on the inverse and the masking complexity is invariant *w.r.t.* $\mathbb{F}_2^4$-affine bijections in input or output so all S-boxes equivalent to the inverse have the same masking complexity [5].
The evaluation of the inverse power function requires only two nonlinear multiplications:

$$x^7 \leftarrow x \otimes x^2 \otimes x^4$$

From x^7, the inverse $x^{-1} = x^{14}$ is only a linear square away.

While the whole mCrypton block cipher is larger than the PRESENT one [4], the mCrypton S-box results in a less complex evaluation. It is therefore interesting to consider its higher-order glitch free implementation.

2.3 Side-channel Analysis Evaluation

One way to perform SCA on a cryptographic system is proposed by Goodwill *et al.* [9]. This standardized method allows developers to test for potential side-channel leakage. The test is designed to be both easy to perform and sensitive enough to cover a wide range of potential problems. It needs to be said though, that no single test can be designed to guarantee complete resistance against all attacks. In the first place, the test is an indication that sufficient care regarding SCA countermeasures was taken during the design of the device. The proposed test is based on the *Welsh's t-test*, a generalization of the Student's t-test that allows samples to have unequal variances [28]. It is similar to a difference of means test, but takes the sample variances and the sample sizes into account. The t-test statistic is calculated as:

$$t = \frac{\overline{T_a} - \overline{T_b}}{\sqrt{\frac{s_a^2}{N_a} + \frac{s_b^2}{N_b}}} \tag{2.2}$$

where $\overline{T_i}$, s_i^2, N_i are the sample means, sample variance and sample size of the set $T_{i \in a,b}$.

The null hypothesis of the test is that two sets of power traces have equal means, indicating that it is not influence by the processed sensitive data. The alternate hypothesis is a difference between the means of the sets, meaning that processing different intermediates influences the power consumption in a statistically significant way, making the device vulnerable to SCA.
The evaluator selects an intermediate for investigation and divides the power traces based on the values at that place. Different sets are acquired and the t-statistic trace is computed on pairs of these sets. When no trace exceeds a confidence threshold $\pm C$, the null hypothesis holds. If the threshold is crossed, another t-test is performed on different traces. When the t-statistic trace exceeds $\pm C$ at the same place as in the first test, the null hypothesis can be rejected with a percentage of certainty related to C.

To summerize, the t-test is performed as follows[9]:

1. The set of power traces is divided in two disjoint sets, Group 1 and Group 2. Two independent Welch's t-tests are executed on these two sets.

2. First t-test on Group 1:

   a) Partition the Group 1 in two subsets A and B based on some intermediate value.

   b) Compute the means $\overline{T_a}$, $\overline{T_b}$ and the standard deviations s_a^2, s_b^2 of subsets A and B for each point in time. The results have the same sample size N_a and N_b as the traces.

   c) Compute the t-statistic trace over each time instant using formula 2.2.

d) Note the time instants in the t-statistic trace where the value crosses the confidence threshold $\pm C$.

3. Second t-test on Group 2: The same steps as in the Group 1 test are performed. The traces and its two subsets will be different. The time instants where the t-statistic trace crosses the confidence threshold $\pm C$ are noted.

If there is any point in time for which the t-test statistic exceeds the confidence threshold in both Group 1 and Group 2, the Device Under Test (DUT) fails. Otherwise it passes.

Chapter 3

Implementation

In this chapter, the implementations are describe in detail. First, all operations of the polynomial masking scheme are listed for the case of 1^{st}-order glitch resistance. A secret is then shared between three players and the polynomial for hiding the secret input is a linear function. Afterwards, the multiplier in $\mathbb{F}_{2^4}$ is shortly discussed, as its implementation is the same for all designs. The designs for the PRESENT S-box and the mCrypton S_0 S-box are then described. This chapter is concluded with a comparison of all designs. The chapter follows the same pattern as in [20], but it is applied on the PRESENT and mCrypton S_0 S-boxes instead of the AES block cipher.

3.1 Polynomial Masking Scheme with (3,1)-sharing

This section lists the equations for the construction of, reconstruction from and the operations on the shares. In what follows, the multiplication in $\mathbb{F}_{2^4}$ is denoted as $\otimes$ and the finite-field addition by $\oplus$.

For all these operations, first, three distinct non-zero elements in $\mathbb{F}_{2^4}$ need to be chosen. These are referred to as the public coefficients α_1, α_2 and α_3. Together with these points, the first row of the inverse Vandermonde matrix $(\alpha_i^j)_{1\leq i,j\leq 3}$, $(\lambda_1, \lambda_2, \lambda_3)$ is needed. These values can be calculated as follows:

$$\begin{aligned}
\lambda_1 &= \alpha_2 \otimes \alpha_3 \otimes (\alpha_1 + \alpha_2)^{-1} \otimes (\alpha_1 \oplus \alpha_3)^{-1} \\
\lambda_2 &= \alpha_1 \otimes \alpha_3 \otimes (\alpha_1 + \alpha_2)^{-1} \otimes (\alpha_2 \oplus \alpha_3)^{-1} \\
\lambda_3 &= \alpha_1 \otimes \alpha_2 \otimes (\alpha_1 + \alpha_3)^{-1} \otimes (\alpha_2 \oplus \alpha_3)^{-1}
\end{aligned}$$

where $.^{-1}$ denotes the multiplicative inverse. Elements $\alpha_1, \alpha_2, \alpha_3, \lambda_1, \lambda_2$ and λ_3 are publicly available to all three Players.

Sharing a value $x \in \mathbb{F}_{2^4}$ requires a random secret coefficient a and the public

coefficients. The resulting shares x_1, x_2, x_3 are calculated as:

$$x_1 = x \oplus (a \otimes \alpha_1)$$
$$x_2 = x \oplus (a \otimes \alpha_2)$$
$$x_3 = x \oplus (a \otimes \alpha_3)$$

Each Player i has access to only one share x_i.

Reconstruction of the hidden secret relies on the values $\lambda_1, \lambda_2, \lambda_3$:

$$x = (x_1 \otimes \lambda_1) \oplus (x_2 \otimes \lambda_2) \oplus (x_3 \otimes \lambda_3)$$

Addition with a constant c can be done by each Player independently as:

$$z_1 = x_1 \oplus c = x \oplus (a \otimes \alpha_1) \oplus c = (x \oplus c) \oplus (a \otimes \alpha_1)$$
$$z_2 = x_2 \oplus c = x \oplus (a \otimes \alpha_2) \oplus c = (x \oplus c) \oplus (a \otimes \alpha_2)$$
$$z_3 = x_3 \oplus c = x \oplus (a \otimes \alpha_3) \oplus c = (x \oplus c) \oplus (a \otimes \alpha_3)$$

The resulting shares of the addition represents the correct secret $z = c \oplus x$, where the random secret coefficient is left unchanged.

Multiplication with a constant c is performed in a similar way and can also be processed by each Player independently as:

$$z_1 = x_1 \otimes c = (x \oplus (a \otimes \alpha_1)) \otimes c = (x \otimes c) \oplus (a \otimes c \otimes \alpha_1)$$
$$z_2 = x_2 \otimes c = (x \oplus (a \otimes \alpha_2)) \otimes c = (x \otimes c) \oplus (a \otimes c \otimes \alpha_2)$$
$$z_3 = x_3 \otimes c = (x \oplus (a \otimes \alpha_3)) \otimes c = (x \otimes c) \oplus (a \otimes c \otimes \alpha_3)$$

Considering $a \otimes c$ as the new secret coefficient, z_1, z_2 and z_3 result in the desired secret output. Reconstruction of the masked secret does not depend on the coefficient a but depends on λ_1, λ_2 and λ_3, which only depend on the public coefficients. The reconstructed value will correspond to the desired result.

Similar to the (3,1)-sharing of x, *i.e.* x_1, x_2, x_3, with secret coefficient a, $y \in \mathbb{F}_{2^4}$ is represented with the sharing y_1, y_2, y_3, constructed with an independent random secret coefficient b.

Addition of two shared secrets is executed in following way:

$$z_1 = x_1 \oplus y_1 = x \oplus (a \otimes \alpha_1) \oplus y \oplus (b \oplus \alpha_1) = (x \oplus y) \oplus (a \oplus b) \otimes \alpha_1)$$
$$z_2 = x_2 \oplus y_2 = x \oplus (a \otimes \alpha_2) \oplus y \oplus (b \oplus \alpha_2) = (x \oplus y) \oplus (a \oplus b) \otimes \alpha_2)$$
$$z_3 = x_3 \oplus y_3 = x \oplus (a \otimes \alpha_3) \oplus y \oplus (b \oplus \alpha_3) = (x \oplus y) \oplus (a \oplus b) \otimes \alpha_3)$$

Seeing $a \oplus b$ as the secret coefficient, the resulting shares account for the desired hidden variable, $z = x \oplus y$.

Multiplication of two shared secrets is composed of three steps:

1. The Players i first compute t_i

$$t_1 = x_1 \otimes y_1 = (x \otimes y) \oplus (((a \otimes y) \oplus (b \otimes x)) \otimes \alpha_1) \oplus (a \otimes b \otimes \alpha_1^2)$$
$$t_2 = x_2 \otimes y_2 = (x \otimes y) \oplus (((a \otimes y) \oplus (b \otimes x)) \otimes \alpha_2) \oplus (a \otimes b \otimes \alpha_2^2)$$
$$t_3 = x_3 \otimes y_3 = (x \otimes y) \oplus (((a \otimes y) \oplus (b \otimes x)) \otimes \alpha_3) \oplus (a \otimes b \otimes \alpha_3^2)$$

2. Each Player then randomly selects a coefficient a_i and remasks t_i as

$$q_{i,1} = t_i \oplus (a_i \otimes \alpha_1)$$
$$q_{i,2} = t_i \oplus (a_i \otimes \alpha_2)$$
$$q_{i,3} = t_i \oplus (a_i \otimes \alpha_3)$$

   Each $q_{i,\forall j \neq i}$ is subsequently send to the corresponding Player j.

3. The outputs $q_{1,i}$, $q_{2,i}$, $q_{3,i}$ of each Player i are then distributed and reconstructed as

$$z_i = (q_{1,i} \otimes \lambda_1) \oplus (q_{2,i} \otimes \lambda_2) \oplus (q_{3,i} \otimes \lambda_3)$$

When considering $(a_1 \otimes \lambda_1) \oplus (a_2 \otimes \lambda_2) \oplus (a_3 \otimes \lambda_3)$ as secret coefficient, z_1, z_2 and z_3 provide the correct and desired shared representation of secret z.

Square of a shared secret can only be computed in the straightforward way, i.e., as $z = x^2$ or

$$z_1 = x_1^2 = x^2 \oplus (a^2 \otimes \alpha_1^2)$$
$$z_2 = x_2^2 = x^2 \oplus (a^2 \otimes \alpha_2^2)$$
$$z_3 = x_3^2 = x^2 \oplus (a^2 \otimes \alpha_3^2)$$

when $\alpha_1, \alpha_2, \alpha_3$ satisfy the conditions for frobenious stability. In our case, this results in the condition of $\alpha_1 = 1$, $(\alpha_2)^2 = \alpha_3$ and $(\alpha_3)^2 = \alpha_2$. This shows that, a reordering between the second and the third Player is needed when squaring is of the form $.^{2^i}$ and is i uneven. When this reordering is not performed, two public coefficients have changed value and the result can not be reconstructed in the same way as the other sharings. The benefits of squaring over a shared multiplier are its smaller size, a lower complexity and that it does not require randomness.

3.2 Finite-field Multiplier

The finite-field multiplier in $\mathbb{F}_{2^4}$ used throughout this chapter is combinatorially implemented in VHDL using its algebraic normal form. The 4-bit inputs $A = (a_3, a_2, a_1, a_0)$ and $B = (b_3, b_2, b_1, b_0)$ result in output $C = (c_3, c_2, c_1, c_0)$ by following bitwise operations:

$$\begin{aligned}
c_0 &= a_0b_0 \oplus a_1b_3 \oplus a_2b_2 \oplus a_3b_1 \\
c_1 &= a_0b_1 \oplus a_1b_0 \oplus a_1b_3 \oplus a_2b_2 \oplus a_2b_3 \oplus a_3b_1 \oplus a_3b_2 \\
c_2 &= a_0b_2 \oplus a_1b_1 \oplus a_2b_0 \oplus a_2b_3 \oplus a_3b_2 \oplus a_3b_3 \\
c_3 &= a_0b_3 \oplus a_1b_2 \oplus a_2b_1 \oplus a_3b_0 \oplus a_3b_3
\end{aligned}$$

where A, B and C are in little-endian notiation.

3.3 Designs

In this section, the implementations of the S-box designs are discussed when considering 1^{st}-order glitch resistance. First, the PRESENT S-box is covered. The implementation of a design based on [20] is discussed, this will be called the *regular* design. Afterwards, a new design is proposed which will be referred to as the *folded* design. The same is done for the mCrypton S_0 S-box. First, its *regular* design is discussed, followed by the description of the *folded* design.

In all designs considered, we choose a temporal separation to thwart 1^{st}-order glitch attacks. This results in a much smaller implementation at the cost of more clock cycles. The implications of temporal separation are that at no point in time, operations can be done on more than one share. After each operation on a share, the intermediate result will be stored and left unaltered while operations are done on other shares. This has a drawback for the shared multiplication. The algorithm for this operation can be split in two levels: a remasking and a reconstruction. For the reconstruction part, the intermediate remasked results from all the other shares are needed. This gives rise to the need for intermediate storage registers and extra clock cycles. A last general issue arises from the control of the multiplexers [20]. Multiplexers are needed to assign different inputs to the shared multiplier. Their control signals must be hazardless, otherwise side-channel leakage between two shares can occur. As a result, extra registers are needed to synchronize these signals.

3.3.1 PRESENT

As seen in the previous chapter, the PRESENT S-box can be described with a polynomial. When using $r(x) = x^4 + x + 1$ as irreducible polynomial for the construction of $\mathbb{F}_{2^4}$, the S-box representation is:

$$\begin{aligned}
S(x) = {} & Dx^{14} + Dx^{13} + Cx^{12} + Ex^{11} + 9x^{10} + 9x^9 + 7x^8 \\
& + 4x^7 + Cx^6 + Ax^5 + Ex^4 + 7x^3 + 7x^2 + C
\end{aligned}$$

The order of the evaluation of this polynomial is based on the proposal by Carlet *et al.* in [5]. The proposed execution order is adapted to reduce the memory usage. Starting from the input x, squaring is consecutively carried out until all elements in a cyclotomic class are covered. A multiplication with the last element of that class is then done with a previously calculated element. This way, another cyclotomic class is accessed, where squaring can again be performed. This scheme is applied until all powers of x in the polynomial are evaluated. Starting with an input x, all but one cyclotomic classes can be visited by just squaring and multiplication with x. As a result an extra power of x needs to be stored during the evaluation.

The block diagram of the shared PRESENT S-box evaluation is depicted in Figure 3.1. The gray multiplier denotes a multiplication with a constant while the black multiplier represents shared multiplication.

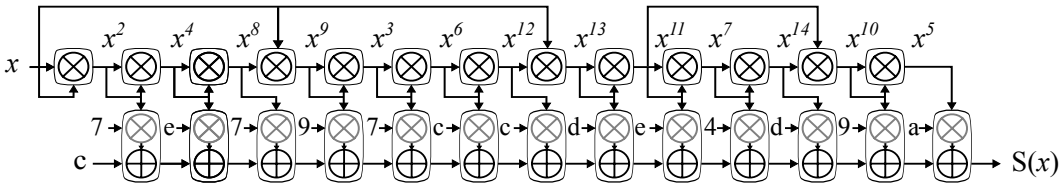

Figure 3.1: Evaluation of the PRESENT S-box.

By calculating the polynomial this way, the designs can easily be extended with a squaring module and benefit from the advantages it brings along.

Regular PRESENT Design

The shared multiplication is discussed first. The working principles are described in *series* of clock cycles. Such a serie consists of six clock cycles and is related to the control signals $em_{1\leq i\leq 6}$, which can be seen in both the architecture in Figure 3.2 and in the finite state machine in Figure 3.4.

- The first clock cycle of a series, enables signal em_1. The two necessary inputs for the shared multiplier are selected by $selm_1$, of which the resulting outputs are listed in Table 3.1. At the same time, a new random number a_1 is fed to the first multiplier element MULT_EL1_1. Together with this random number, the fixed public coefficients α_1, α_2 and α_3 are used to remask the multiplied result t_1.

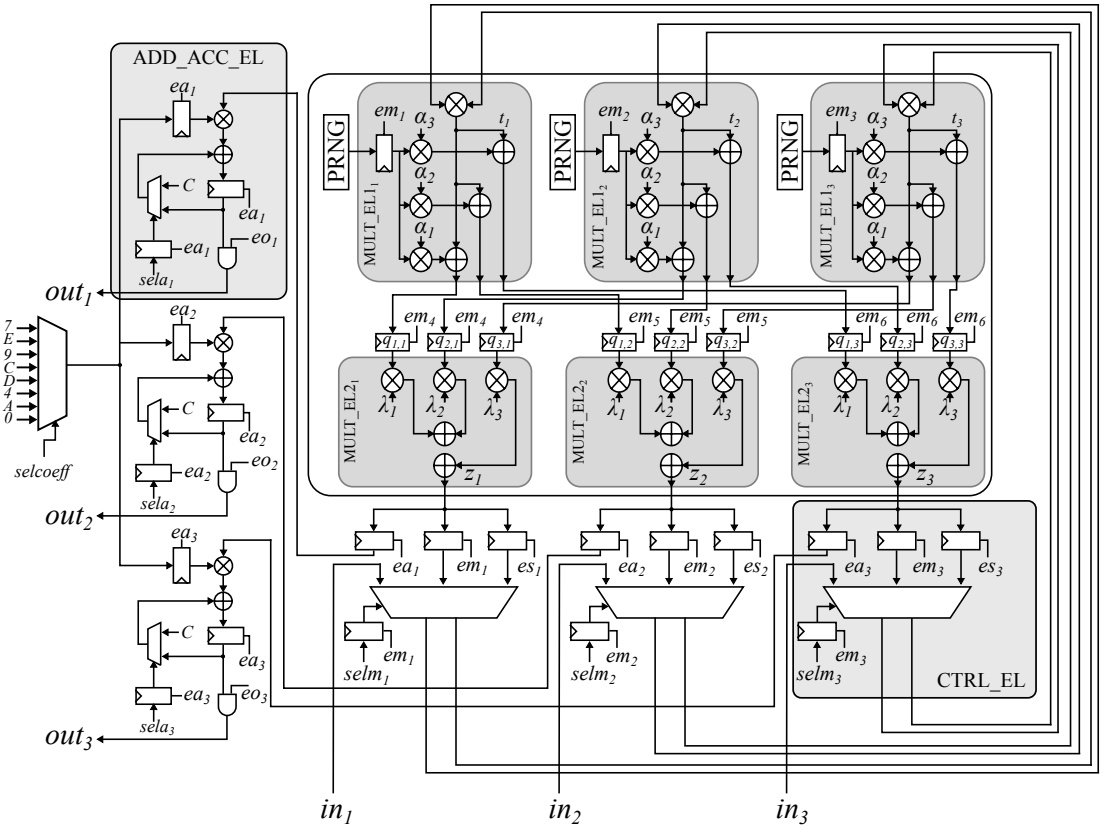

Figure 3.2: Architecture diagram for the regular PRESENT implementation

- The previous described procedure is repeated in the second clock cycle using signal em_2 and on the third clock cycle using em_3 for the second (in MULT_EL1_2) and third (in MULT_EL1_3) share respectively.
- In the fourth clock cycle, by activating signal em_4, the results from the previous three clock cycles with respect to the public coefficient α_1 are stored in the registers $q_{1,1}$, $q_{2,1}$, $q_{3,1}$. The combinatorial logic in MULT_EL2_2 then performs the unmasking using λ_1, λ_2 and λ_3. This outputs the first share of the shared multiplication. The result is not saved in this clock cycle, but will be done at the start of the next series, with the activation of the select signal em_1.
- In the fifth and sixth clock cycle, the same principles as in the fourth clock cycle are apply. The enable signals em_5 handles Player 2 (in MULT_EL2_2) and em_6 serves Player 3 (in MULT_EL2_2).

As can be seen in the block diagram of the PRESENT S-box in Figure 3.1, we need to store one extra intermediate value x^{11}. When this value is output at the MULT_EL2_i blocks, the es_1, es_2, es_3 signals follow the levels of the em_1, em_2, em_3 signals to store the shares of x^{11} in the registers controlled by $es_{1\leq i\leq 3}$.
To calculate the polynomial, the powers of x need to be multiplied with a constant

$selm_i$	Output 1	Output 2
00	in_i	in_i
01	em_i	em_i
10	em_i	in_i
11	em_i	es_i

Table 3.1: Selected multiplexer output signals based on $selm_i$ in the regular PRESENT design.

and accumulated with the previously obtained results. This is handled by the ADD_ACC$_{i\in\{1,2,3\}}$ blocks. When a desired power of a share x_i is obtained from the shared multiplier, the enable signal ea_i activates with the corresponding em_i. With this signal, a new coefficient (chosen by $selcoef$) is fed to an input of the ADD_ACC$_i$ multiplier, resulting in the right multiplication of a constant and its corresponding power of x. Using the $sela_i$ select signal, in the first series of clock cycles the constant value of the polynomial is added to an empty (zero) register. In the consecutive series, the register output is chosen for to accumulate the results of the other series. The eo_i signal enables the output share i of the S-box after the registers holds the right value.

To show that the temporal separation of calculations on independent shares is achieved, the relevant signal changes are shown in the Figure 3.3. The three ADD_BUF_O signals in the timing diagram, belong to the ouputs of the registers of the three different ADD_ACC_ELs. From top to bottom, they belong to the first, second and third share.

In the timing diagram, the clock signal is not included, as it is related to the em_i signals. In addition, the es_i, ea_i and eo_i signals are also omitted as they can be deduced from the described working principles and the finite state machine from Figure 3.4. The outputs of the shared multiplier and the outputs of the accumulation registers show the desired separation of share manipulation.

Folded PRESENT Design

Based on the regular design, a folded design is proposed with an area reduction in mind. In this design, MULT_EL1 and MULT_EL2 are used by all Players, instead of dedicating these two multiplier elements for all Players. Especially for higher-orders, this design is expected to have a significant impact on the area. The drawback of this design is an increase of clock cycles needed for an S-box evaluation. The architecture for the folded PRESENT S-box is shown in Figure 3.5.

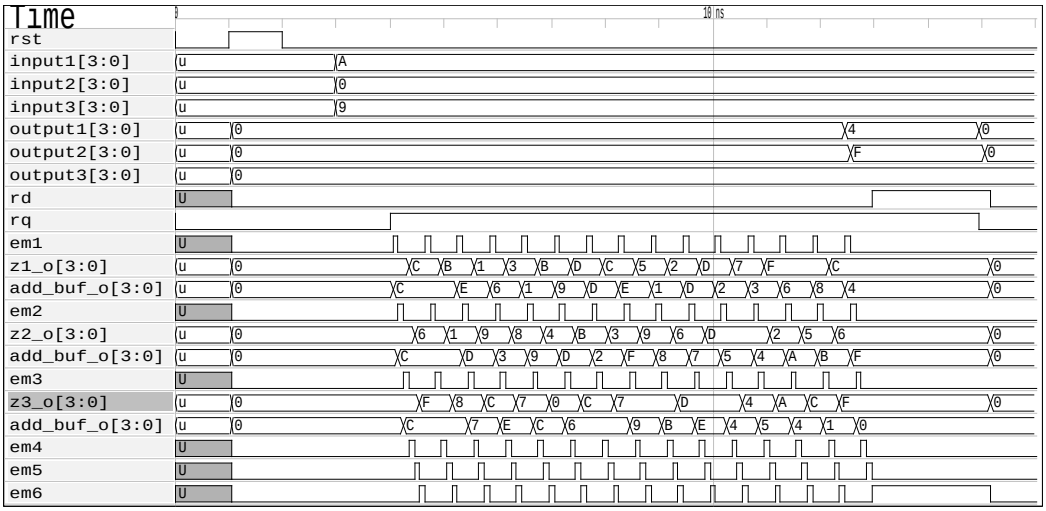

Figure 3.3: Timing diagram for the regular PRESENT implementation

Care has to be taken to make sure the design does not leak from the effect of glitches. As, except for the manipulation of the control and select signals, the only change involved is the shared multiplier structure. These changes are now discussed from top to bottom. To guide the right signals to MULT_EL1 one at a time, a multiplexer with select signal $sel0_1$ is used. To avoid influence of adjacent shares, the circuits must be initialized between all remaskings of intermediate result. For this purpose, the multiplexer can output zeros to the multiplier. In addition, the pseudo random number can be turned off by the multiplexer with control signal sel_rng, which then outputs zeros to MULT_EL1. The outputs of the first multiplier element are each wired to their three corresponding registers. The values in these registers are updated using the temporally separated em_4, em_5 and em_6 control signals. To get the right values to MULT_EL2 for reconstruction, a multiplexer with control signal $sel0_2$ is used. Again, the circuit must be initialized between every remasking. Therefore, the multiplexer can output zeros to the MULT_EL2 inputs. The outputs of this second multiplier element are wired to all registers, which are clocked by the temporally separated em_1, em_2 and em_3 control signals.

To process the outputs of the shared multiplier safely, following scheme is followed.

- While the remasking part is performed in MULT_EL1, zeros are input to MULT_EL2. In the first clock cycle of a series, the first pair of shares are input for multiplication and remasking. The results are saved in the second clock cycle using em_4. The MULT_EL1 circuit is initialized in the third cycle, by applying zeros to its inputs. In the fourth clock cycle, the second pair of shares is applied to the first multiplier element. In the following clock cycle, the result is stored using em_5. Afterwards, the circuit is initialized again. This is

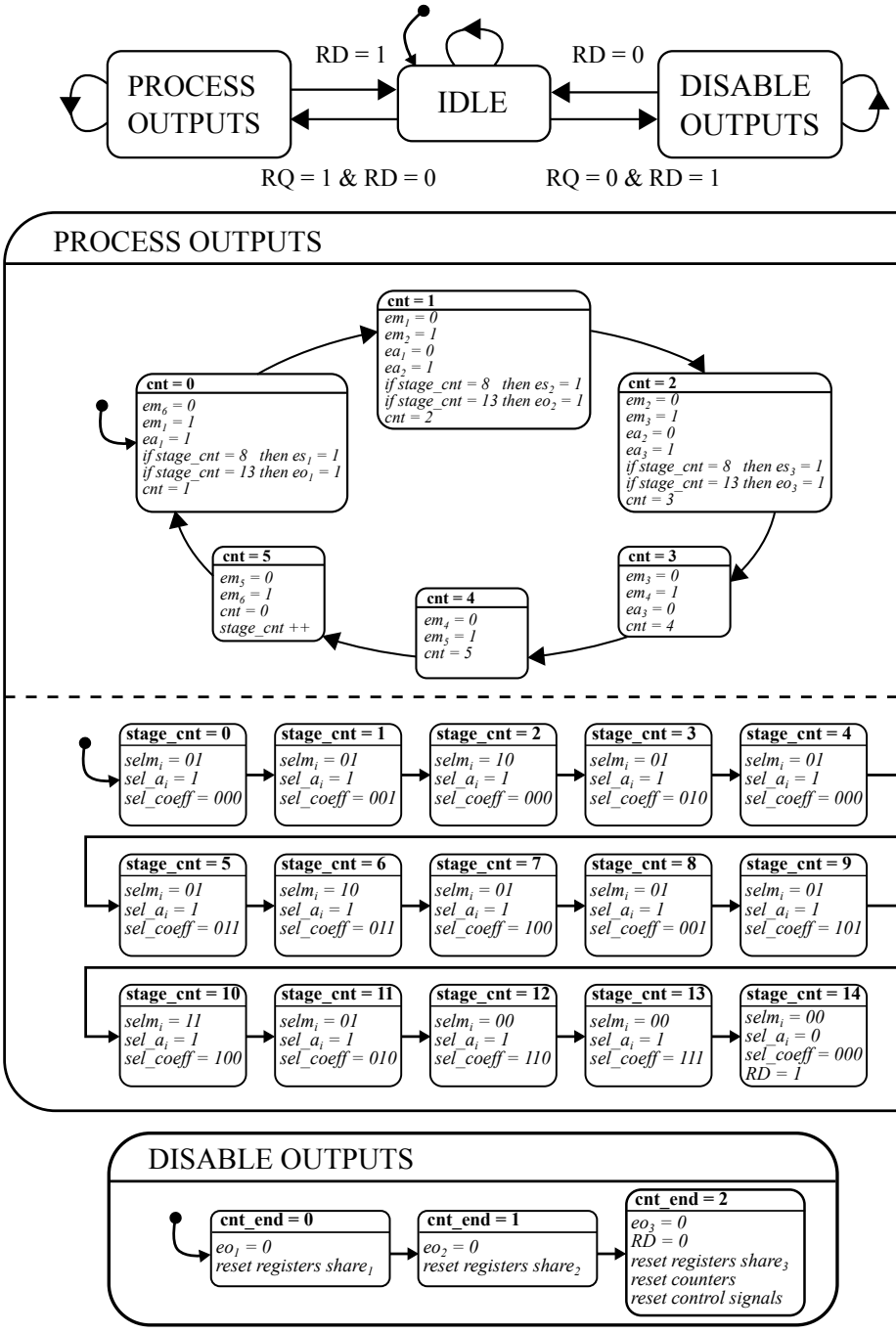

Figure 3.4: Finite state machines for the regular PRESENT implementation

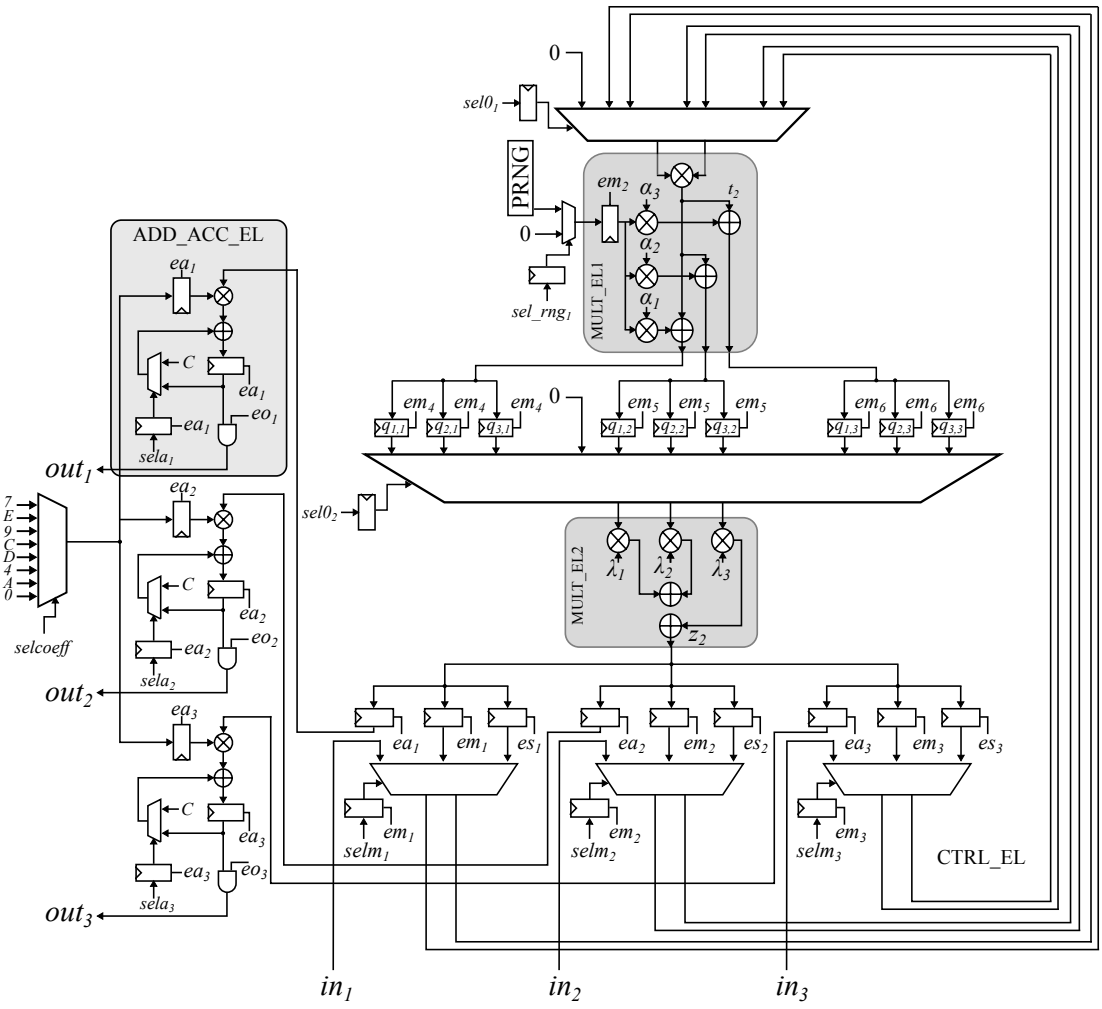

Figure 3.5: Architecture diagram for the folded PRESENT implementation

repeated for the third pair of shares. After the last circuit initialization, the inputs are held to zero while the reconstruction is performed in MULT_EL2.

- With the inputs of MULT_EL1 tied to zero, the right register values are chosen as input for MULT_EL2 using the lower multiplexer. The same *select-store-initialize* pattern is followed here to output the right shares to the three Players.

This version of the masked S-box has not yet been implemented. Another folded design architecture was implemented, but the multiplier circuits were not initialized in between shared operations. We suspect that this design leaks and therefore, this new design is proposed eventhough it is not yet implemented.

3.3.2 mCrypton

To calculate the mCrypton S_0 S-box, we choose the path that leads to the least amount of non-linear multiplications. As was seen in the previous chapter, the S-box can be calculated in that way using two affine transforms A and B and an inversion:

$$S(x) = A(I(B(x)))$$

with

$$\begin{aligned} B(x) &= 9x^8 \oplus 3x^4 \oplus 13x^2 \oplus 10x \\ I(x) &= x^{-1} = x^{14} \\ A(x) &= 2x^8 \oplus 6x^4 \oplus 12x^2 \oplus 12x \oplus 4 \end{aligned}$$

Similarly, the coefficients come from the construction of $\mathbb{F}_{2^4}$ with $r(x) = x^4 + x + 1$ as irreducible polynomial.

The same evaluation method as with the PRESENT S-box is used here. The affine transformations all lie in the same cyclotomic class. Only squaring is needed to evaluate these powers. For the inverse, elements from three cyclotomic classes need to be traversed. Its evaluation is chosen to only require squaring or multiplications with x. This way, no extra memory is needed. The order of calculations for the powers of x is then shown in the diagram of Figure 3.6. Again, the black and grey multiplication elements mean shared and constant multiplication respectively.

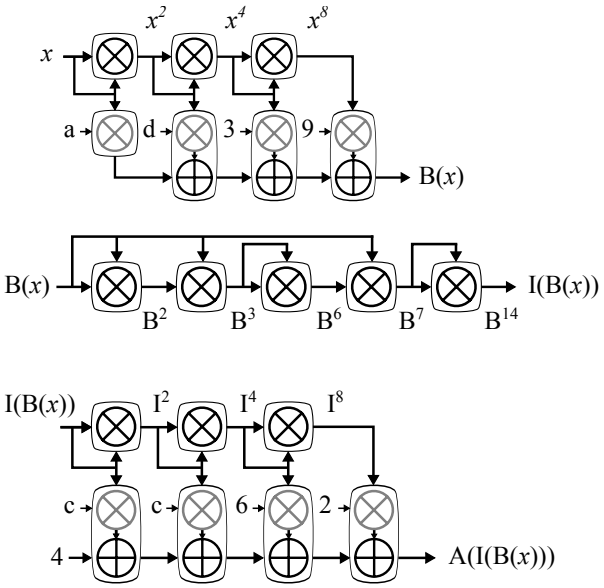

Figure 3.6: Evaluation of the mCrypton S-box.

As with the PRESENT S-box evaluation, the designs can also here be easily extended with a squaring module and benefit from its advantages.

Regular mCrypton Design

The working principles of the masked mCrypton S-box are now described for the regular design. The architecture diagram is shown in Figure 3.7 and the corresponding finite state machine is depicted in Figure 3.9. The outputs of the CTRL_EL multiplexers the are listed in Table 3.2.

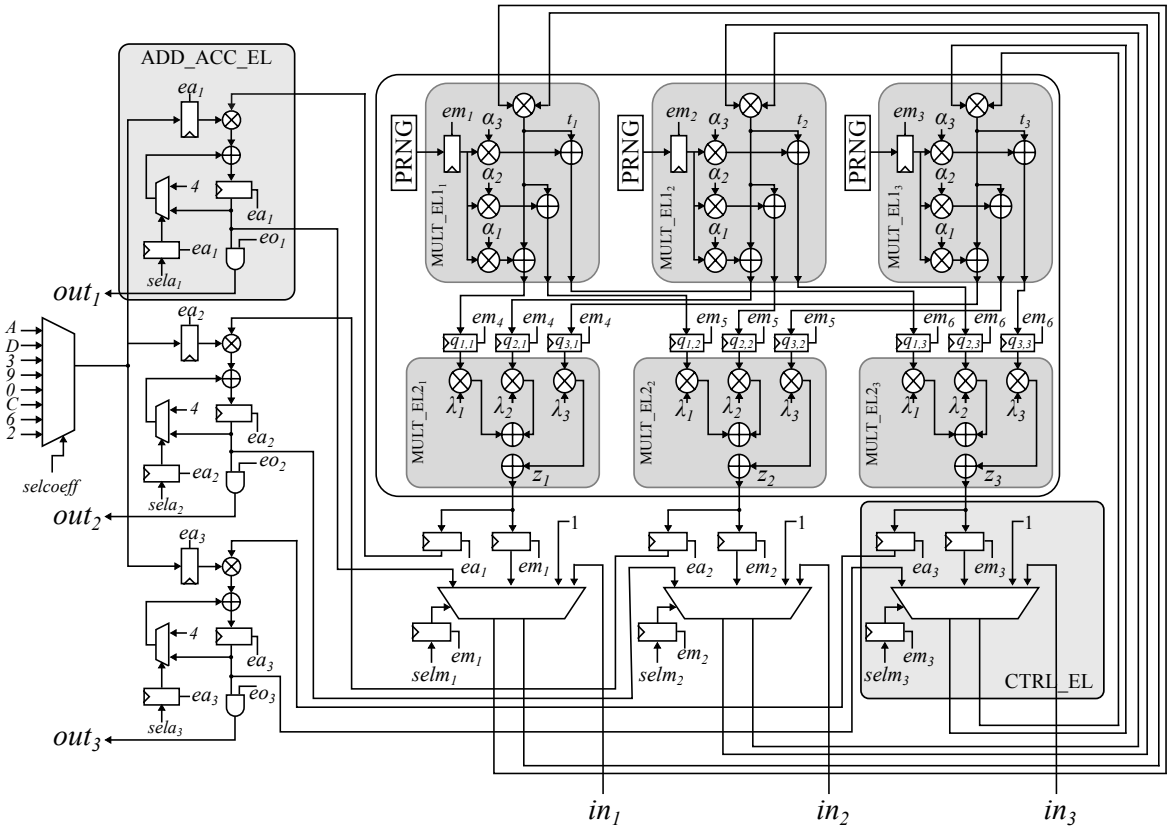

Figure 3.7: Architecture diagram for the regular PRESENT implementation

In general, the same working principles as with the regular PRESENT S-box apply regarding the shared multiplier and the accumulation block. Therefore, we will focus on the changes made to incorporate the affine transformation. The outputs of the accumulation registers are now fed back to the multiplexers in the CTRL_ELs. This is needed to perform the inversion from the result of the first affine.
The execution of the functions $B(x)$, $I(x)$ and $A(x)$ are now descibed.

- At the start of the first affine transformation, the input x is multiplied with unity in the shared multiplier. Afterwards, it is multiplied with its constant and accumulated in the register. This consumes one extra serie of clock cycles but saves us a multiplexer per ADD_ACC_EL$_i$ block. The multiplexer for this would select between either the input share in_i or the output of the ea_i register

$selm_i$	Output 1	Output 2
00	in_i	1
01	em_i	em_i
10	aff_i	aff_i
11	em_i	aff_i

Table 3.2: Selected multiplexer output signals based on $selm_i$ in the regular mCrypton design.

in the CNTRL_EL$_i$. From there on, the remainder of the affine polynomial can be evaluated using the standard working principles.

- The inverse operation starts when the resulting shares of the first affine are stored in the accumulation registers. These are input to the multiplexer in the CNTRL_EL$_i$ blocks. During the whole calculation of the inverse, the result of the first affine is kept in the registers.

- The second affine transformation can be initiated when the resulting shares of the inverse are stored in the ea_i registers from the CNTRL_EL$_i$ blocks. The registers ADD_ACC_EL$_i$ are set to the constant coefficient $B(x)$. This is done by selecting the zero coefficient in the multiplexer controlled by $selcoeff$ and by setting the constant input of the multiplexer controlled by $sela_i$ as output. This way, the value after the multiplier of the accumulator is zero, which is added to the right constant of $B(x)$. This can also be performed using a dedicated reset for each accumulator register. This results in an increased area, but saves a serie of clock cycles for the evaluation. From there on, the standard evaluation procedure can be followed to reach the required output shares.

As was noted before and in contrast with the PRESENT S-box, all cyclotomic classes can be traversed using squaring and multiplication with an input x. As a result, the registers controlled by es_i can be omitted.

To show the temporal separation, the relevant signal changes are shown in the Figure 3.8. Again, the clock signal and the es_i, ea_i, eo_i signals are omitted. Also here, the outputs of the shared multiplier and the outputs of the accumulation registers show the desired separation of share manipulation. The same remark about the ADD_BUF_Os as in the PRESENT S-box timing diagram applies.

Folded mCrypton Design

A folded design can also be applied to the mCrypton S-box. The architecture for the folded mCrypton S-box is shown in Figure 3.10. The same working principles as with the folded PRESENT S-box apply to the shared multiplier. The control and accumulator structure remain unchanged $w.r.t$ the regular mCrypton S-box. This

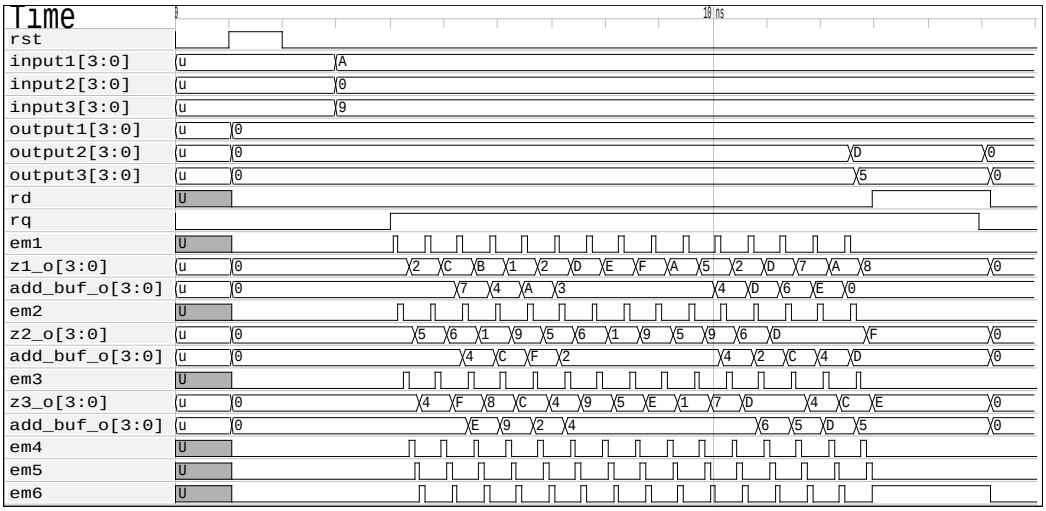

Figure 3.8: Timing diagram for the regular mCrypton

version has also not yet been implemented. Again, a mistake was made in the design of a previous folded mCrypton S-box, therefore this new design architecture is proposed.

3.3.3 Area and performance

The area in GE of the components and the total S-boxes are given in Table 3.3. The mCrypton S-box shows to be more compact than the PRESENT S-box. The es_i controlled registers that could be dropped in the mCrypton design amount for aroung half of the difference. The results come from heuristics so the differences do not necessarily add up. For the folded designs, and solely based on the difference in the shared multiplier, we have 1433 GE for the PRESENT S-box and 1356 GE for the mCrypton S-box. The shared multiplier can be estimated to require 702 GE. The estimates for the multiplexers are obtained from the analysis of a previous folded design, where a 9:3 multiplexer results in 41 GE and a 6:2 multiplexer accounts for 28 GE. Both regular implementations need 89 clock cycles from the activation of the request signal to the output of all three shares. For the folded designs, this becomes 269 clock cycles. The secure evaluation of the PRESENT S-box requires 156-bits of randomness. For the mCrypton S-box, 132 random bits are needed. When linear squaring would be used, this randomness drops to 36-bits and 24-bits for respectively the PRESENT and mCrypton. For the folded designs, these numbers stay the same. The impact on the area resulting from a higher number of shares still needs to be investigated, but from the area of the individual components, it can be expected that the folded designs promise a more compact result. The numbers are obtained from synopsis using the NanGate Open Cell Library [1].

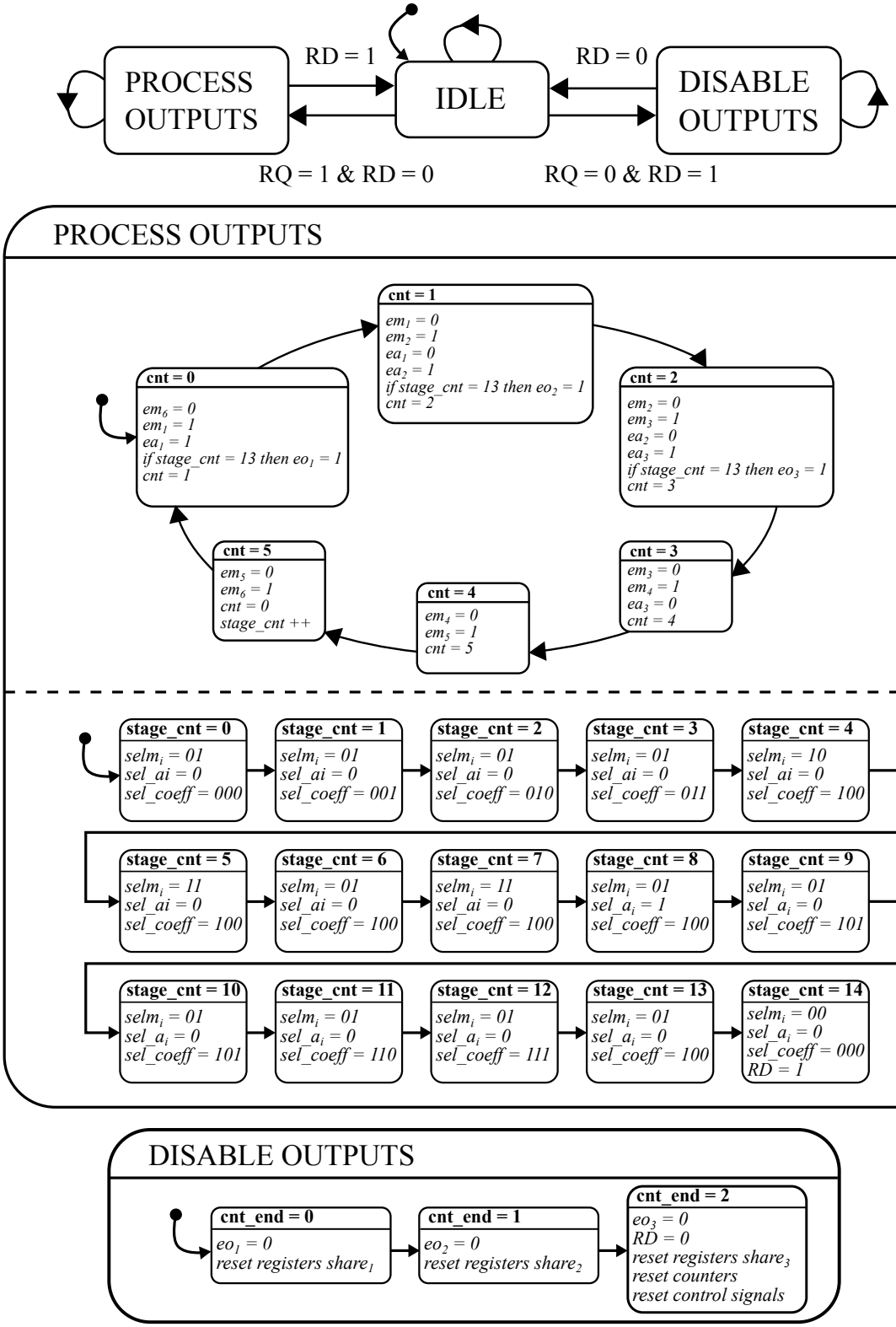

Figure 3.9: Finite state machines for the regular mCrypton implementation

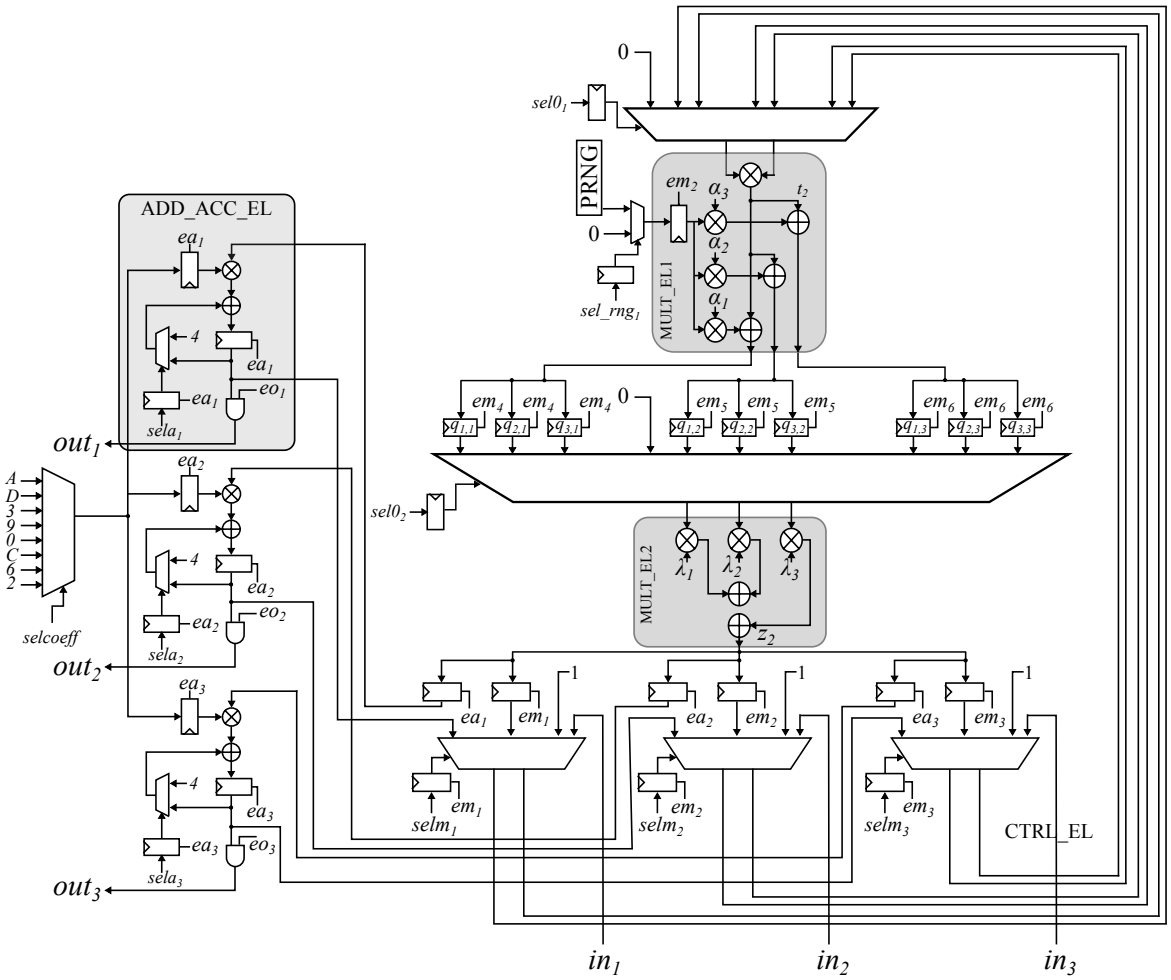

Figure 3.10: Architecture diagram for the folded mCrypton implementation

Component	Area (GE)	
	PRESENT	mCrypton
Multiplier	47	47
edge_triggered_Dff	8	8
reg4	28	28
muxer_2_1	10	10
muxer_4_2	26	27
muxer_8_1	47	47
mult_el1	233	233
mult_el2	232	232
shared_mult	1360	1360
add_acc_el	127	124
add_acc	379	379
ctrl_el	120	94
ctrl	352	275
S-box	3594	3407

Table 3.3: Area in GE of the regular implementations

Chapter 4

Side-channel Evaluation

For the practical evaluation of the side-channel leakage during the S-box calculations, a SASEBO-G (Side-channel Attack Standard Evaluation Board) is used [25]. It features two FPGAs: a cryptographic FPGA (Xilinx Virtex-2 Pro xc2vp7-fg456-5) and a control FPGA (Xilinx Virtex-2 xc2vp30-fg676-5). The power consumption traces are measured with a Tektronix DPO 7254C oscilloscope. The board is clocked with an external frequency of $3.072MHz$.

The host PC sends the following to the control FPGA through an RS-232 serial port: two sharings of two plaintext inputs, the public coefficients $\alpha_1, \alpha_2, \alpha_3$ and the corresponding $\lambda_1, \lambda_2, \lambda_3$, a 128-bit plaintext for the AES, the desired amount of traces and the mode of operation. Six modes of operation are supported:

1. Fixed *vs.* Fixed Input, Masks On
2. Fixed *vs.* Random Input, Masks On
3. Random *vs.* Random Input, Masks On
4. Fixed *vs.* Fixed Input, Masks Off
5. Fixed *vs.* Random Input, Masks Off
6. Random *vs.* Random Input, Masks Off

The modes without masking are used to check the measurement setup. The device should not pass the tests as the leakage countermeasure is disabled. In that case, the DUT essentially evaluates a non-masked S-box three times. After validating the test setup, the masks can be turned on. When the DUT does not leak, there is confidence that this absence of leakage is resulting from the masking countermeasure.

After receiving the data from the PC, the control FPGA executes two AES encryptions. During the first encryption, the received plaintext is used, while in all consecutive encryptions, the ciphertext from the previous encryption is input as plaintext . The key is fixed in the FPGA during the evaluation. After the AES

outputs are calculated and stored, they are used as pseudo random numbers. The received input sharings are remasked and one sharing is forwarded to the cryptographic FPGA, along with the necessary amount of random nibbles, the public coefficients $\alpha_{1\leq i\leq 3}$ and the first row $\lambda_{1\leq i\leq 3}$ of the inverse Vandermonde matrix. The control FPGA sets a request signal *RQ* high and goes into an idle state before the S-box evaluation is started. When the outputs are ready, the crypto FPGA sets a ready signal *RD* high, which causes the control FPGA to go out of its idle state. After the outputs are saved, the request signal *RQ* is deactivated. The crypto FPGA disables its outputs and resets its internal registers. Before the next calculation is started, the control FPGA waits for an adjustable amount of clock cycles to allow the *fastframing* feature of the oscilloscope to work. This way traces can be collected at much faster rates. This process is repeated until the desired amount of evaluations are done. When the last evaluation is finished, the output shares and the last ciphertext of the AES are send back to the PC for validation. To check the correctness of the results, an implementation of the expected behaviour in MATLAB is run with the same settings. The code was written in verilog by adapting an already existing framework. The finite state machines of both FPGAs are given in figure 4.1.

The input sharing sent to the S-box is chosen by a coin flip. The last bit of the AES ciphertext is used as the outcome of this tossing. Depending on the mode of operation, both input sharings are changed in a certain way before one of them is transmitted to the crypto FPGA. In the fixed *vs.* fixed operation mode, the two reconstructed inputs keep their same value but their sharings are remasked before each evaluation. In the fixed *vs.* random operation mode, the first reconstructed input is fixed at all times but the sharing is refreshed before every evaluation. The second input shares are obtained by masking a random 4-bit value that represents a new reconstructed input. The random *vs.* random operation mode masks two random 4-bit inputs to create two new sharings before evaluation.

By starting the measurement with the request signal *RQ* and stopping the measurement when the ready signal *RD* is activated, the instantaneous power consumption of the S-box is traced. The power traces are divided in sets based on an intermediate value in the calculation, *e.g.* the input or output. The mean and variance of each set of power traces is calculated. The t-test is then performed between two different sets and the confidence threshold is checked.

4.1 Results

The erroneous folded mCrypton design, that is not presented in this thesis, was analysed and shown to leak in the fixed *vs.* fixed mode without masking. The next evaluation run, the masks were enabled. The design showed 1^{st}-order leakage when the amount of power traces were small, but disappeared when the amount of evaluations increased. This could be due to the absence of reinitialization between shared operations in the multiplier circuit. In addition, the behaviour of the control

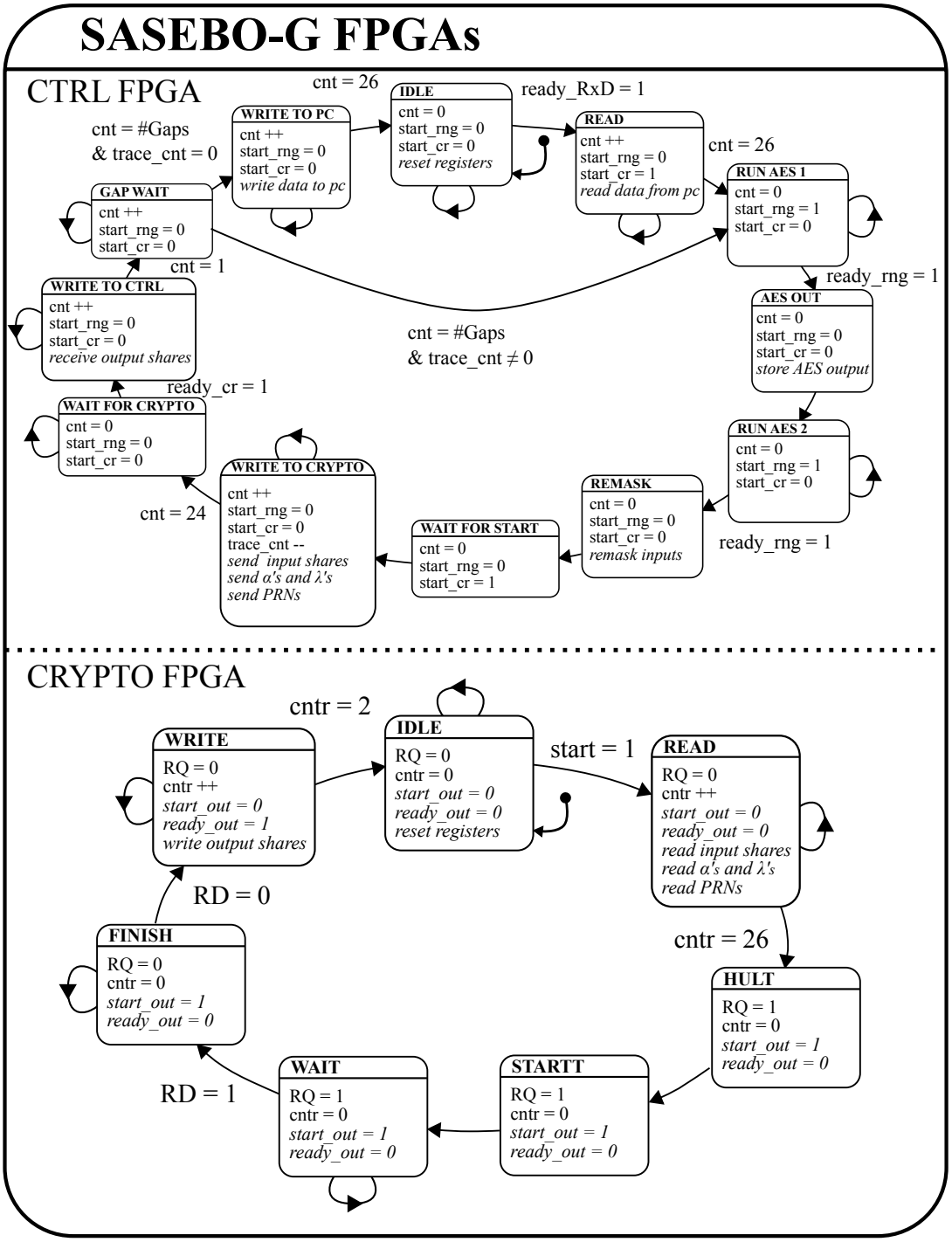

Figure 4.1: Finite state machines for the control and cryptographic FPGAs

FPGA at the time of the evaluation was assumed to have to much correlation between subsequent shares. Instead of the earlier described behaviour of the operation modes, the output shares were either discarded (fixed input) or used to overwrite the input shares chosen for evaluation (random input). As nothing was remasked, all shares were dependent. This mistake has been corrected to match the desired behaviour.

The SCA evaluation has not yet been performed with the updated assignation and remasking of input shares. No conclusions can be made about the security of the designs, although theoretically, the conditions are fulfilled, countermeasures can only be verified by physical testing.

Chapter 5

Conclusion

In this thesis, two 4-bit S-boxes were implemented on a hardware platform. After the setting and the problem description were introduced, an overview was given of the masking principles, the S-boxes and the method for practical security evaluation. The PRESENT S-box and the mCrypton S_0 S-box each were secured against 1^{st}-order glitch attacks by means of the polynomial masking scheme introduced in [26]. For each S-box, two designs were then proposed. A regular design based on the guidelines in [20] and a folded design that trades off speed with area to enable a less costly higher-order implementation.

The regular designs result in 3594 GE for the PRESENT S-box and 3407 GE for the mCrypton S-box. An estimation of the folded designs gives 1433 GE for the PRESENT S-box and 1356 GE for the mCrypton S-box. Both regular implementations need 89 clock cycles to output the shares. For the folded designs, this increases to a total of 269 clock cycles. In both the regular and folded designs, secure evaluation requires 156-bits of randomness for the PRESENT S-box and 132 random bits are needed for the mCrypton S-box. When linear squaring would be used, this randomness drops to 36-bits and 24-bits for respectively the PRESENT and MCRYPTON.

5.1 Future Work

Work for the near future includes: implementing the folded designs, validating all design by checking their 1^{st}-order leakage. Then, when the designs are proven to be secure, implementations for higher orders can be considered. Accurate values regarding the amount of gate equivalents can bring insights in the practical realisability of higher order glitches freeness.
The glitch resistant polynomial masking scheme can be applied to the whole block ciphers. This way, the impact on the whole cipher can be assessed. When the footprint results in too expensive designs, a weaker adversary model *w.r.t* glitches can be considered [26]. Another possibility for reducing the cost is to investigate less generic and more efficient SMC protocols [26]. This way glitch free designs might become feasible for lightweight applications. The advantage of introducing a squarer

in the designs is the reduction of the needed randomness. This however increases the required area. To compensate this, other, *e.g.* smaller multiplier architectures can be considered.

Bibliography

[1] Nangate open cell library. http://www.nangate.com/.

[2] J. Balasch, S. Faust, B. Gierlichs, and I. Verbauwhede. Theory and Practice of a Leakage Resilient Masking Scheme. In X. Wang and K. Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 758–775. Springer, 2012.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM.

[4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[5] C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and M. Rivain. Higher-Order Masking Schemes for S-Boxes. In A. Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.

[6] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[7] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche. Higher-Order Side Channel Security and Mask Refreshing. In *Proceedings of FSE 2013, to appear.*

[8] Y. Crama and P. L. Hammer. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering.* Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[9] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. P.: A testing methodology for side-channel resistance validation, niat, 2011.

[10] J. Guo, T. Peyrin, and A. Poschmann. The PHOTON family of lightweight hash functions. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.

[11] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED block cipher. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[12] M. Karpinskyy, L. Korkishko, and A. Furmanyuk. Masked encryption algorithm mCrypton for resource-constrained devices. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007. IDAACS 2007. 4th IEEE Workshop on*, pages 628–633, Sept 2007.

[13] P. C. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.

[14] S. Kutzner, P. H. Nguyen, A. Poschmann, and H. Wang. On 3-share threshold implementations for 4-bit s-boxes. *IACR Cryptology ePrint Archive*, 2012:509, 2012. informal publication.

[15] C. H. Lim. CRYPTON: A new 128-bit block cipher - specification and analysis, 1998.

[16] C. H. Lim and T. Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In J. Song, T. Kwon, and M. Yung, editors, *WISA*, volume 3786 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2005.

[17] S. Mangard, T. Popp, and B. M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[18] S. Mangard, T. Popp, and B. M. Gammel. Side-channel leakage of masked CMOS gates. In A. Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[19] S. Mangard, N. Pramstaller, and E. Oswald. Successfully attacking masked AES hardware implementations. In J. R. Rao and B. Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[20] A. Moradi and O. Mischke. On the simplicity of converting leakages from multivariate to univariate - (case study of a glitch-resistant masking scheme). In *CHES*, pages 1–20, 2013.

[21] S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In P. Ning, S. Qing, and N. Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[22] S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.

[23] A. Poschmann, S. Ling, and H. Wang. 256 bit standardized crypto for 650 GE - GOST revisited. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2010.

[24] A. Poschmann, A. Moradi, K. Khoo, C.-W. Lim, H. Wang, and S. Ling. Side-channel resistant crypto for less than 2, 300 GE. *J. Cryptology*, 24(2):322–345, 2011.

[25] Research Center for Information Security, National Institute of Advanced Industrial Science and Technology. *Side-channel Attack Standard Evaluation Board SASEBO-G Specification.*

[26] T. Roche and E. Prouff. Higher-order glitch free implementation of the AES using Secure Multi-Party Computation protocols - Extended version. *J. Cryptographic Engineering*, 2(2):111–127, 2012.

[27] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[28] B. L. Welch. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, 1947.

Master thesis filing card

Student: Thomas De Cnudde

Title: Higher Order Glitch Free Implementation of 4-bit Optimal S-boxes

Dutch title: Hogere Orde Glitch Vrije Implementatie van 4-bit S-boxen

UDC: 681.3*I20

Abstract:
The increasing deployment of small-scale wireless embedded systems calls for proper security measures. While cyptographic algorithms provide the necessary security, practical cryptographic implementations have shown to leak information that is supposed to be secret. This information is leaked through side-channels and can be observed using side-channel analysis. The order of the analysis is defined by the amount of observations one uses to compromise the system. Masking is one popular way to thwart attacks that exploit the side-channels. A masking scheme gives directions on how to divide a secret variable into shares and how to perform operations on these shares. Non-linear operations pose the difficulty in these schemes. Therefore, we focus on the non-linear S-boxes of block ciphers. A new side-channel occuring from CMOS transistor switching during computations, has shown to deteriorate the effect of masking. Therefore, extra restrictions need to be applied to a masking scheme to provide glitch resistance. One such a glitches free masking scheme has been proposed in the literature[26]. This scheme is provably-secure and can be implemented to thwart any order of side-channel analysis. The sharing in the proposed masking scheme is based on Shamir's secret sharing scheme. In addition, it uses a secure multi-party computation protocol introduced by Ben-Or *et al.* [3] for the secure operations on these shares. The glitch resistance in this scheme is achieved by temporal or spatial separation. Temporal separation means that no operations on different shares that mask the same hidden variable, may be done at the same time. Spatial separation requires the implementation of each multi-party player to be on a different platform. This thesis considers a 1^{st}-order glitch resistant hardware implementations for the PRESENT S-box and the mCrypton S_0 S-box. Two implementations are proposed, one is based on the guidelines in [20], the other is an adaptation of this design in order to reduce the area. One of these designs is implemented for both S-boxes and their costs are compared *w.r.t.* the area, the speed and the amount of randomness.

Thesis submitted for the degree of Master of Science in Artificial Intelligence

Thesis supervisor: Prof. dr. Vincent Rijmen

Assessors: Prof. dr. Liesbet Van der Perre
dr. Svetla Nikova

Mentors: Begül Bilgin
Oscar Repáraz