J. Cryptol.
DOI: 10.1007/s00145-010-9085-7

Journal of
CRYPTOLOGY

Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches

Svetla Nikova
Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC and IBBT, Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
and
University of Twente, EEMCS-DIES, P.O. Box 217, 7500 AE Enschede, The Netherlands

Vincent Rijmen
Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC and IBBT, Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
and
Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria

Martin Schläffer
Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
martin.schlaeffer@iaik.tugraz.at

Received 1 September 2009

Abstract. Hardware implementations of cryptographic algorithms are vulnerable to side-channel attacks. Side-channel attacks that are based on multiple measurements of the same operation can be countered by employing masking techniques. Many protection measures depart from an idealized hardware model that is very expensive to meet with real hardware. In particular, the presence of glitches causes many masking techniques to leak information during the computation of nonlinear functions. We discuss a recently introduced masking method which is based on secret sharing and multi-party computation methods. The approach results in implementations that are provably resistant against a wide range of attacks, while making only minimal assumptions on the hardware. We show how to use this method to derive secure implementations of some nonlinear building blocks for cryptographic algorithms. Finally, we provide a provable secure implementation of the block cipher Noekeon and verify the results by means of low-level simulations.

Key words. DPA, Masking, Glitches, Sharing, Nonlinear functions, S-box, Noekeon.

1. Introduction

Side-channel analysis exploits the information leaked during the computation of a cryptographic algorithm. The most common technique is to analyze the power consump-

Published online: 19 October 2010

tion of a cryptographic device using differential power analysis (DPA) [17]. This side-channel attack exploits the correlation between the instantaneous power consumption of a device and the intermediate results of a cryptographic algorithm. A years-long sequence of increasingly secure designs and increasingly sophisticated attack methods breaking again these designs suggests that the problem won't be solved easily. Therefore, securing hardware implementations against advanced DPA attacks is still an active field of research.

The approach that we propose in this paper[1] is based on multi-party computation techniques, which makes it rather different from the mainstream. The most important differences are that we use more than one mask and share the data being processed through nonlinear operations such that each computation is independent of at least one input share. Further, we do not use fresh randomness after one or more steps and we make only realistic assumptions about the hardware. We provide also proofs of security against a wide range of attacks and experimental evidence to back up our claims.

The remainder of this paper is organized as follows. In Sect. 2 we discuss some related work both in side-channel attacks and defense strategies, and in multi-party computation protocols and derived protection techniques. In Sect. 3 we explain how many of the "classical" protection techniques fail when the underlying hardware is not glitch-free. We introduce our approach in Sect. 4, define three properties sufficient for security and prove our main theorems. In Sect. 5, we apply our method to some functions that form the basis of cryptographic systems, e.g. multiplication in the extension field $GF(2^{2m})/GF(2^m)$, which is often used in implementations of the AES S-box. In Sect. 6, we apply our method to the block cipher Noekeon and report on the simulations that we made in order to verify our claims. Finally, we conclude and present topics for further research in Sect. 7.

2. Related Work

In this section, we review popular countermeasures against DPA attacks. Subsequently, we briefly introduce threshold cryptography and Multi-Party Computation (MPC) protocols, which form the inspiration for our approach to protect implementations against side-channel attacks. Finally, we explain the relations between our approach and MPC protocols, and discuss related literature.

2.1. *History of Countermeasures*

In order to counteract DPA attacks several different approaches have been proposed. The general approach is to make the intermediate results of the cryptographic algorithm independent of the secret key. Circuit design approaches [36,37] try to remove the root of the side-channel leakage by balancing the power consumption of different data values. However, even small remaining asymmetries make a DPA possible. Another method is to randomize the intermediate values of an algorithm by masking them. This can be done at the algorithm level [1,5,13,25], at the gate level [15,38] or even in combination with circuit design approaches [27].

However, recent attacks have shown that masked hardware implementations (contrary to software implementations [31,32]) can still be attacked using even first-order

[1] Parts of this work appeared earlier in [23] and [24].

DPA. The problem of most masking approaches is that they were designed and proven secure in the assumption that the output of each gate switches only once per clock cycle. Instead, glitches [30] occur in combinational CMOS circuits and each signal switches several times. Due to these glitches, these circuits are vulnerable to DPA attacks [19, 20]. Furthermore, the amount of information leaked cannot be easily determined from the mathematical description of a masked function. It depends too much on the used hardware technology and the way the circuit is actually placed on a chip. All these approaches start from compact but rather insecure implementations. Subsequently the designers try to solve the known security issues by adding as little hardware as possible.

A different type of approach was proposed in [23,24] and is continued in this paper. The idea is to first start from a very secure implementation and then, make this approach more practical by minimizing the hardware requirements while still maintaining the security level. Secret sharing schemes and techniques from multi-party computation are used to construct combinational logic which is completely independent of the unmasked values. The approach holds for both FPGAs and ASICs, and the idea can also be used in software implementations. In this approach, the implementations increase with the number of shares and for each nonlinear part of a circuit at least three shares (or masks) are needed. Further, constructing secure implementations of arbitrary functions using only a small number of shares is a difficult task.

2.2. *Threshold Cryptography and Multi-Party Computation*

MPC protocols enable a set of players to securely evaluate an arbitrary function on their private inputs, but some of the players could be corrupted by an adversary. Consider s players, each player holding an input x_i. The players want to compute a function $F(x_1, \ldots, x_s) = z$ in a secure manner, which informally implies two things. The adversary cannot interrupt the computation, hence the computed value is correct. Additionally the adversary cannot learn any information about the inputs of the honest players, except of course what can be inferred from the function value. The results can be easily extended to more general types of functionality e.g. computing a function $F(x_1, \ldots, x_s) = (z_1, \ldots, z_s)$. A $(t+1, s)$ threshold system allows s parties to do secure computations when at least $t+1$ parties are needed to recover the secret. A protocol is called t-private if any set of size at most t cannot get from the protocol execution any additional information then what they already have from their shares. Hence, up to t corrupt players can be tolerated, which will learn nothing about the secret. The following theorem illustrates the power of MPC protocols.

Theorem 1 ([4, Theorem 1]). *For every function* f *and* $t < s/2$ *there exists a* t*-private protocol.*

In order to prove this result, Ben-Or *et al.* use Shamir's secret sharing scheme [33]. Let α_i denote n distinct non-zero elements. A secret x is shared by randomly selecting t elements a_i and defining the polynomial $p(y)$:

$$p(y) = x + a_1 y + a_2 y^2 + \cdots + a_t y^t.$$

Each player P_i obtains the value $x_i = p(\alpha_i)$ and the secret is equal to $p(0)$.

Let x, y be two elements that are shared using the polynomials $p(y)$, respectively $q(y)$. Then the element $x + y$ is encoded by the polynomial $(p+q)(y) = p(y) + q(y)$. Hence, in order to compute the shares for $x + y = p(0) + q(0)$, each player P_i computes $x_i + y_i = (p+q)(\alpha_i)$. In the same way, the scaling of a secret with a constant is achieved when each player scales his share by the same constant.

Let $r(y) = p(y)q(y)$. Then indeed $x \cdot y = r(0)$ as desired, but the degree of r equals $2t$, if the underlying field is of large enough order. If $2t \geq s$, then we can no longer have enough data points to recover $x \cdot y$ uniquely. Ben-Or *et al.* then define a degree reduction and randomization step which transforms the s shares $x_i \cdot y_i$ into shares $\tilde{r}(\alpha_i)$, where $\tilde{r}(y)$ is a new random polynomial of degree t and with $\tilde{r}(0) = x \cdot y$. This step requires linear multi-party computations on each of the shares $x_i \cdot y_i$, which are implemented again by defining polynomials and distributing shares to each of the players.

2.3. *Our Approach*

We construct a secret sharing scheme to share the secret variables that have to be processed by the circuit. Splitting each variable in s shares was previously proposed in [8]. Chari *et al.* analyze extensively the case $s = 2$, relating the amount of information leaking by means of side channels to the number of sequences that an attacker needs to observe in order to mount a successful attack. However, Chari *et al.* do not investigate how nonlinear operations should be implemented in such a scheme.

In this paper here, we complete this approach and propose a way to implement nonlinear functions. Namely, we divide our circuits into combinatorial blocks which are completely independent of the secret variables. We achieve this by making sure that no single combinatorial block acts on all shares.

Linking our approach to MPC protocols, one can say that we equate each combinatorial block with a party. In this paper, we investigate only the case where we need the output of all sub-circuits in order to compute the output of the circuit. This corresponds to an (s, s) threshold system. Our situation differs from the typical MPC case, because each input x_i is used by several parties (functions). Since each two functions together (possibly) use all inputs, we have a $(1, s, s)$ ramp scheme.

The functions are *corrupted* by means of side-channel attacks. A corrupt function still produces correct results, hence we have *passive corruption*. In a first-order attack, the attacker can corrupt at most one function at a time. Theorem 1 implies that given enough random values and enough rounds of communication, every function can be implemented. Our main constraint of course, is that we have to economize on the amount of randomness and extra operations. Thereby we can tolerate a loss in provable security but still try to achieve the best possible security.

A similar approach is followed in [15], where the authors try to achieve perfect security against all attackers that can measure up to t wires simultaneously. Besides the high amount of extra operations, the main drawback of the approach in [15] is that they ignore some typical aspects of real-life hardware implementations. For instance, they do not achieve security against an attack where the sum of *all* instantaneous power signals is measured, which in reality is of course a *more* easily accessible side channel than the signals on t individual wires. They also stick to the idealized hardware model without gate delays and glitches. The follow-up paper [14] looks into active probing attacks, but does not solve the issues that we mentioned here.

3. DPA Attacks on Masking

Masking is a side-channel countermeasure which tries to randomize the intermediate values of a cryptographic algorithm [22]. Then, the (randomized) power consumption does not correlate with the intermediate values anymore. The most common masking scheme is Boolean or linear masking where the mask is added by an XOR operation. However, one problem of masking is that cryptographic algorithms like AES [9] combine linear and nonlinear functions. Thus, many different hardware masking schemes and masked gates have been proposed [1,5,25,39] but all of them have been broken again [2,13,20,41]. Even though no wire carries an unmasked value, the power consumption correlates with the unmasked intermediate results of the algorithm.

3.1. *Glitches*

The problem of these hardware masking schemes is that the effect of glitches has not been considered. Glitches have first been analyzed in [19] and a technique to model glitches has been presented in [35]. Glitches occur because the signals of a combinational circuit can switch more than once if an input changes. The amount of glitches depends on the specific hardware technology, the implementation and on the input values of a combinational logic.

For example in the common CMOS technology, circuits consume very low amounts of power. The power consumption caused by glitches is relatively large compared to the power consumption caused by non-switching operation of CMOS circuits. It follows that the total power consumption of a CMOS circuit is strongly correlated to the number of glitches that occur.

The reason why most masking schemes can be attacked is that they combine masks and masked values into the same combinational logic. Since they are not processed independently, also the number of glitches and thus, the power consumption is not independent of the masks and unmasked values. It follows that the power consumption is a function of the unmasked value. Hence, it depends on the actual hardware implementation whether a design is secure. In this case, nothing about the security can be proven during the design process.

3.2. *Glitches in a Traditionally Masked AND Gate*

In this section, we study the effect of glitches in a masked AND gate. We consider a typical implementation of a masked AND gate [38], illustrated in Fig. 1. To make the analysis easier, we assume here that XOR gates exist as basic primitives and do not decompose them into smaller building blocks. We show that nevertheless, the number of glitches and thus, the power consumption depends indeed on the unmasked value.

The circuit takes 5 inputs: the two random masks m_x, m_y, the two masked inputs $x_m = m_x \oplus x$, $y_m = m_y \oplus y$, and a new random value m_z to mask the output $z = x$ AND y. The circuit outputs the output mask m_z and the masked output z_m, which is computed as follows:

$$z_m = x_m y_m \oplus \big(m_y x_m \oplus \big(m_x y_m \oplus (m_x m_y \oplus m_z)\big)\big). \tag{1}$$

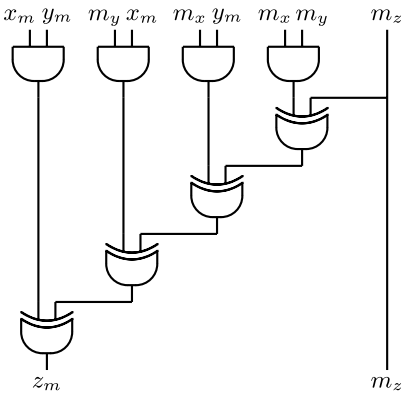

Fig. 1. Glitch propagation through a masked AND gate.

Table 1. Number of affected gates in the circuit of Fig. 1, when a glitch occurs in input x_m.

y	m_y	y_m	AND	XOR
0	0	0	0	0
1	0	1	1	1
1	1	0	1	2
0	1	1	2	2

Note that the order in which the XOR gates are evaluated, is not arbitrary. If the circuit would compute at any time the sum of any of the products, then there would be a leakage of an unmasked value. For instance, $x_m y_m \oplus m_y \cdot x_m = y x_m$, which leaks information about y. This is one of the reasons why the new random value m_z is introduced in the beginning and why all the products are added one by one to it.

Consider now what happens if a glitch occurs in input x_m. The propagation of this glitch will depend on the values of m_y and y_m. The power consumption caused by the glitch is related to the number of gates that *see* the glitch. It is clear from Table 1 that the energy consumption depends on the values of m_y and y_m. Since the mean power consumption is different for $y = 0$ and $y = 1$, the power consumption leaks information on the value of y. Similar results can be obtained by analyzing the effect of a glitch in one of the other inputs, and the cases where some of the inputs arrive delayed with respect to the other inputs [19,20]. We conclude that switching characteristics of logical circuits invalidate some of the assumptions commonly made in proofs of security against side-channel attacks.

3.3. *Simulating Attacks and Gate Delays*

Although it is difficult to verify whether a design or a masking scheme is secure, different simulation techniques have been developed to verify the security of a design. A simple method to analyze a design is by using the assumption that there is no delay at the inputs and inside of a combinational logic. In this case, each signal and output switches at most once and even simple masking schemes are secure using this model. However, in [18] it has been shown by means of computer simulations, that most masked gates can

be attacked if the input signals of the combinational logic arrive at different moments in time.

In [11] a model is used where each of the n input signals of a gate can arrive at a different time. Thus, the output can switch up to n times. Although the model does not allow delays inside the gate it takes glitches into account. In their paper a gate is defined to be G-equivalent, if there is no correlation between the number of output transitions and the unmasked value. Since it is not possible to build any nonlinear gate which is G-equivalent using standard masking, the weakened requirement of semi-G-equivalence has been defined. Using this notation it is possible to define nonlinear masked gates which can be used to build arbitrary circuits. However, the big disadvantage of this method is that semi-G-equivalent circuits have routing constraints, and it still depends on the implementation whether a circuit is secure.

Another disadvantage of the previous model is that it does not take delays inside the gate into account. Therefore, a more detailed power consumption model is to count all transitions which occur in a combinational logic. A common method is to use unit delay for all gates and an even more accurate method is to derive the delay of a circuit by back-annotated netlists [16]. In this case, different timing information for different gates and wire lengths are considered. Most secure masking schemes can be broken by performing attacks based on these simulations.

However, none of these methods can *prove* that a circuit is secure in the presence of glitches because each method takes only special cases into account. Therefore, these methods can only be used to *attack* masking schemes. In the following sections we examine a masking scheme based on secret sharing which is provable secure in the presence of glitches.

4. Sharing

In this section we introduce terminology, formulate requirements and prove results about the security of our approach. Note that both sharing (higher-order masking), as well as the uniformity requirement we give in Sect. 4.4 are not new approaches in the quest for side-channel resistant implementations. However, the main improvement in our approach is the noncompleteness requirement (see Sect. 4.3) for each computation and combinational logic of a hardware implementation. This requirement seems obvious but is rather difficult to achieve in combination with the other requirements. Nevertheless, we further show how to construct and implement basic shared Boolean functions which fulfill *all* requirements.

4.1. *Terminology*

We denote stochastic *variables* by small characters $x, y, \ldots$ and samples of these variables by capitals $X, Y, \ldots$. We denote by $\Pr(x = X)$ the probability that x takes the value X, and often abbreviate this to $\Pr(x)$. *Probability* is defined as the number of times that the variable x takes the value X, divided by the number of different values that the input of the circuit can take.

We denote a vector of s shares x_i by $\overline{x} = (x_1, x_2, \ldots, x_s)$ and split a variable x into s additive shares x_i with

$$x = \sum_i x_i.$$

We will only use (s, s) secret sharing schemes, hence all s shares are needed in order to determine x uniquely. In a *perfect* (s, s) secret sharing scheme, knowledge of up to $s - 1$ shares does not give any additional information on the value of x. Observe that a traditional masking scheme corresponds to a $(2, 2)$ secret sharing scheme.

In a function f with $p > 1$ input variables, we will use superscripts $x^1, x^2, \ldots, x^p$ to differentiate the different input variables. Let Q denote the number of different $(X^1, X^2, \ldots, X^p)$ values that the input of f can take. Then Q^s is the number of different $(\overline{X}^1, \overline{X}^2, \ldots, \overline{X}^p)$ values that the *vector of input shares* can take. In this paper, we use secret sharing schemes where

$$\Pr\left(\overline{x}^1 = \overline{X}^1, \ldots, \overline{x}^p = \overline{X}^p\right) = Q^{1-s} \Pr\left(x^1 = \sum_{i=1}^{s} X_i^1, \ldots, x^p = \sum_{i=1}^{s} X_i^p\right). \quad (2)$$

Hence we have

$$\sum_{i=1}^{s} X_i^j = \sum_{i=1}^{s} Y_i^j, \forall j \Rightarrow \Pr\left(\overline{x}^1 = \overline{X}^1, \ldots, \overline{x}^p = \overline{X}^p\right) = \Pr\left(\overline{x}^1 = \overline{Y}^1, \ldots, \overline{x}^p = \overline{Y}^p\right).$$

In words, any bias present in the joint distribution of the shares $(\overline{x}^1, \ldots, \overline{x}^p)$ is only due to a bias in the distribution of the unshared variables $x^1, \ldots, x^p$.

4.2. *Realization*

In order to implement a vector function $(z^1, \ldots, z^q) = \mathrm{f}(x^1, \ldots, x^p)$ we need a set of functions f_i which together compute the output(s) of f. We call this a *realization* and get the following property:

Property 1 (Correctness). Let $(z^1, \ldots, z^q) = \mathrm{f}(x^1, \ldots, x^p)$ be a vector function. Then the set of functions $\mathrm{f}_i(\overline{x}^1, \ldots, \overline{x}^p)$ is a *realization* of f if and only if

$$(Z_1, \ldots, Z_q) = \mathrm{f}\left(X^1, \ldots, X^p\right) = \sum_{i=1}^{s} \mathrm{f}_i\left(\overline{X}^1, \ldots, \overline{X}^p\right)$$

for all vectors of input shares $(\overline{X}^1, \ldots, \overline{X}^p)$ satisfying $\sum_{i=1}^{s} X_i^j = X^j$ with $1 \leq j \leq p$.

4.3. *Noncompleteness*

The next property is important to prove the security of a realization of a function. We denote the reduced vector $(x_1^j, \ldots, x_{i-1}^j, x_{i+1}^j, \ldots, x_s^j)$ by $\overline{x}_i^j$.

Property 2 (Noncompleteness). Every function is independent of at least one share of the input variable x and consequently, independent of at least one share of each component. Without loss of generality, we require that z_i is independent of $x_i^j, \forall j$:

$$\begin{aligned} z_1 &= \mathrm{f}_1\big(\overline{x}_1^1, \overline{x}_1^2, \ldots, \overline{x}_1^p\big) \\ z_2 &= \mathrm{f}_2\big(\overline{x}_2^1, \overline{x}_2^2, \ldots, \overline{x}_2^p\big) \\ &\ldots \\ z_s &= \mathrm{f}_s\big(\overline{x}_s^1, \overline{x}_s^2, \ldots, \overline{x}_s^p\big) \end{aligned}$$

When constructing a realization for a vector function f^j, we need to ensure that Property 2 is satisfied for each component of the output. As we have defined in Property 2, each output share with index i needs to be independent of all input shares with the same index i.

4.4. *Uniformity*

The following property will turn out to be useful in Sect. 4.8, where we introduce pipelined implementations.

Property 3 (Uniformity). A realization of $(z^1, \ldots, z^q) = \mathrm{f}(x^1, \ldots, x^p)$ is *uniform*, if the distribution of the shares of the output satisfies

$$\Pr\big(\overline{z}^1 = \overline{Z}^1, \ldots, \overline{z}^q = \overline{Z}^p\big) = Q^{1-s} \Pr\left(z^1 = \sum_{i=1}^{s} Z_i^1, \ldots, z^q = \sum_{i=1}^{s} Z_i^q\right) \tag{3}$$

provided that the distribution of the shares of the input satisfies (2).

If the function f is invertible, then Property 3 is satisfied by invertible realizations. In an invertible realization of f, every vector $(\overline{Z}^1, \ldots, \overline{Z}^q)$ is reached for exactly one input vector $(\overline{X}^1, \ldots, \overline{X}^p)$. This condition is stricter than the requirement that every output tuple $(Z^1, \ldots, Z^q)$ is reached for exactly one input tuple $(X^1, \ldots, X^p)$.

4.5. *Implementing Linear Transformations*

Consider a linear transformation $(z^1, \ldots, z^q) = \ell(x^1, \ldots, x^p)$. The easiest way to implement a linear transformation securely is to process the s shares independently. Indeed, if

$$\begin{aligned} \big(z_i^1, \ldots, z_i^q\big) &= \ell\big(x_{i+1}^1, \ldots, x_{i+1}^p\big), \quad 1 \le i < s, \\ \big(z_s^1, \ldots, z_s^q\big) &= \ell\big(x_1, \ldots, x_1^p\big), \end{aligned}$$

then by definition of a linear transformation, we have

$$\begin{aligned}\left(z^1,\ldots,z^q\right) &= \sum_{i=1}^{s}\left(z_i^1,\ldots,z_i^q\right) = \sum_{i=1}^{s}\ell\left(x_i^1,\ldots,x_i^p\right)\\ &= \ell\left(\sum_{i=1}^{s}\left(x_i^1,\ldots,x_i^p\right)\right) = \ell\left(x^1,\ldots,x^p\right).\end{aligned}$$

Such an implementation of a linear transformation does not leak information that can be used in a side-channel attack, even if glitches are taken into account [19,20]. A typical property of this implementation is that each output share z_i^j depends only on one input share of each variable.

4.6. *Implementing Nonlinear Transformations of Low Degree*

We will construct circuits for nonlinear transformations having a similar property as the secure circuits for linear transformations. Intuitively, it is clear that if a share z_i^j does not depend on input shares $x_i^1, x_i^2, \ldots$ then z_i^j cannot be correlated to $x^1, x^2, \ldots$. Neither will the computation of z_i^j leak information about the value of $x^1, x^2, \ldots$. This is formalized in the following theorem.

Theorem 2. *In a realization satisfying Property 1 and Property 2, if the distribution of the shares of the inputs satisfies (2), then each of the output shares z_i^j is statistically independent of the input variables x^j and the output variables z^j. Furthermore, the same holds for all intermediate results that are computed during the computation of the output shares and for physical quantities, like power consumption, electro-magnetic radiation etc., which are a function of these intermediate results.*

Proof. Without loss of generality, we give the proof for output variable z^1. Let $\varphi(\overline{x}_1^1,\ldots,\overline{x}_1^p)$ denote an arbitrary function of the $p\times(s-1)$ input shares x_i^j with $1\le j\le p$ and $2\le i\le s$. For instance, φ can be z_1^1, an intermediate result needed to compute z_1^1, the power consumed by the combinatorial circuit that computes $z_1^1,\ldots$

$$\begin{aligned}\Pr(\varphi=\Phi) &= \sum_{\substack{\overline{X}^1,\ldots,\overline{X}^p\\ \varphi(\overline{X}_1^1,\ldots,\overline{X}_1^p)=\Phi}} \Pr\left(\overline{x}^1=\overline{X}^1,\ldots,\overline{x}^p=\overline{X}^p\right)\\ &= \sum_{\substack{\overline{X}_1^1,\ldots,\overline{X}_1^p\\ \varphi(\overline{X}_1^1,\ldots,\overline{X}_1^p)=\Phi}} \sum_{X_1^1,\ldots,X_1^p} \Pr\left(\overline{x}^1=\overline{X}^1,\ldots,\overline{x}^p=\overline{X}^p\right).\end{aligned}$$

Since $X^j = \sum_i X_i^j$, we can change variables and replace $X_1^1, \ldots, X_1^p$ by $X^1, \ldots, X^p$. Furthermore we use (2) and obtain:

$$\Pr(\varphi = \Phi) = \sum_{\substack{\overline{X}_1^1, \ldots, \overline{X}_1^p \\ \varphi(\overline{X}_1^1, \ldots, \overline{X}_1^p) = \Phi}} Q^{1-s} \underbrace{\sum_{X^1, \ldots, X^p} \Pr\left(x^1 = X^1, \ldots, x^p = X^p\right)}_{=1}.$$

Next, we compute

$$\begin{aligned}
&\Pr\left(\varphi = \Phi \mid x^1 = X^1, \ldots, x^p = X^p\right) \\
&\quad = \sum_{\substack{\overline{X}^1, \ldots, \overline{X}^p \\ \varphi(\overline{X}_1^1, \ldots, \overline{X}_1^p) = \Phi}} \Pr\left(\overline{x}^1 = X^1, \ldots, \overline{x}^p = X^p \mid x^1 = X^1, \ldots, x^p = X^p\right) \\
&\quad = \sum_{\substack{\overline{X}_1^1, \ldots, \overline{X}_1^p \\ \varphi(\overline{X}_1^1, \ldots, \overline{X}_1^p) = \Phi}} \underbrace{\sum_{X_1^1, \ldots, X_1^p} \Pr\left(\overline{x}^1 = \overline{X}^1, \ldots, \overline{x}^p = \overline{X}^p \mid x^1 = X^1, \ldots, x^p = X^p\right)}_{A}.
\end{aligned}$$

The terms in the sum A are non-zero for exactly one combination of $X_1^1, \ldots, X_1^p$, which satisfies

$$X^j = \sum_{i=1}^{s} X_i^j.$$

It follows from (2) that in this case $\Pr(\overline{x}^1 = \overline{X}^1, \ldots, \overline{x}^p = \overline{X}^p \mid x^1 = X^1, \ldots, x^p = X^p) = Q^{1-s}$. We conclude that φ and $(x^1, \ldots, x^p)$ are statistically independent. □

In other words, a circuit of a realization satisfying the requirements of Theorem 2 leaks no information on the variables that are processed.

Corollary 1. *If a realization satisfies Property 1 and Property 2, and if the distribution of the shares of the inputs satisfies (2), then the expected value of the power consumption of a circuit implementing the realization is independent of $x^1, \ldots, x^p$ and $z^1, \ldots, z^q$, even in the presence of glitches or the delayed arrival of some inputs.*

Proof. Since the proof of Theorem 2 makes no assumption on the behavior of the circuit and/or the presence of glitches, the theorem holds for each sub-circuit computing one of the y_i^j, also in the case of delayed inputs or glitches. Furthermore, the mean power consumption of the whole circuit is the sum of the mean power consumptions of the sub-circuits and expectation is a linear operation. □

4.7. *Implementing Arbitrary Functions*

Property 1 and Property 2 impose a lower bound on the number of shares s.

Theorem 3. *The minimum number of shares required to implement a product of D variables with a realization satisfying Property 2 and 1 is given by*

$$s \geq 1 + D.$$

Proof. Multiplying D factors with s shares each can be done in the following way. Collect in the first output share all terms that do not contain the first share of any of the inputs. Collect in the second output share all terms that contain the first share of any of the inputs, but not the second share of any of the inputs. Continuing in this way, collect in output share i all the terms containing input shares $1, 2, \ldots$ and $i - 1$, but not input share i. Finally, collect in output share s the terms containing the terms with input shares $1, 2, \ldots$ and $s - 1$ but not input share s. Only if $s - 1 \geq D$, there are no terms left after step s. □

It follows that we need at least three shares in order to implement a nonlinear function. The construction used in the proof of Theorem 3 can also be used to implement more general monomials. For instance, the monomial x^3y can be implemented as a product of four variables. Because not all variables are independent, it might be that there exist other solutions with a lower number of shares. Hence, we have the following corollary.

Corollary 2. *The maximum number of shares required to implement a function* f *of u variables over* $\mathrm{GF}(2^m)$, *equals* $1 + 2^{mu}$.

Proof. Since $\forall x \in \mathrm{GF}(2^m) : x^{2^m} = x$, it is always possible to describe f as a multivariate polynomial of degree at most 2^{mu}. For instance, we can use the Lagrange interpolation formula. We construct the functions f_i for each separate monomial of f by applying the same method as in the proof of Theorem 3. Summing up the functions for each monomial, we obtain the functions for f. □

Theorem 3 shows that implementing more complicated functions typically leads to an increase in the number of shares required, as well as an increase in the number of gates required. This should not come as a big surprise, because introducing resistance against power attacks always comes at a price. For instance, in [27], the authors report an increase in area with a factor 5, for a decrease in performance with factor 0.6. The software solution proposed in [32] doubles the code size, multiplies the RAM requirements with a factor of 20 and decreases the performance with a factor 50. Other proposals add more complexity for the same security level. Nevertheless, for functions with large numbers of inputs, it is better to adopt pipelining.

4.8. *Pipelining*

Pipelining is often used to speed up hardware implementations. In order to allow large clock frequencies, combinatorial logic circuits should not be too deep. Pipelining is an implementation technique where a logical circuit with l levels is divided into two circuits with $l/2$ levels, separated by a register, which stores the intermediate result of the first stage until the active phase of the next clock cycle. As an example, the AES implementation of [42] uses a pipeline with two stages to implement the S-boxes.

Dividing a combinatorial circuit into separate pipelining stages, can also reduce the number of shares and the number of gates required for an implementation that has to be protected against side-channel attacks in the presence of glitches. By definition, a register is insensible to glitches. The registers storing the intermediate results at the end of stage bound the propagation of glitches and delays. When considered individually, each of the pipeline stages represents a mathematical function that is less complex than the full circuit: the nonlinear degree will be lower and/or the number of monomials that needs to be summed. This will typically reduced the required number of shares and gates.

We now prove a result about the security of a pipelined implementation. Consider a pipelined realization consisting of r combinatorial layers and r registers, i.e. one register for the output shares and $r-1$ for the intermediate shares. The function that computes share $y_{i,t}^{j}$ at the output of pipeline stage t is denoted by $\mathrm{f}_{i,j,t}$. The power consumption in the circuit that implements this function is denoted by $P_{i,j,t}$.

As before, we assume that the distribution of the shares of the inputs satisfies (2). Note that condition (2) needs now to be fulfilled at the input of each pipeline stage. Since the input of the next pipeline stage is formed by the output of the previous pipeline stage, we can achieve this goal by demanding that the functions $\mathrm{f}_{i,j,t}$ satisfy additionally Property 3, next to Property 1 and Property 2. We can then prove the following.

Theorem 4. *Under the conditions described above, no linear combination of the power consumptions $P_{i,j,t}$ is statistically correlated to any of the input variables x^j nor to any of the output variables y^j.*

Proof. We prove that for any choice of the linear coefficients $c_{i,j,t}$ the covariance

$$\mathrm{cov}\left(\sum_{i,j,t} c_{i,j,t} P_{i,j,t}, y^1\right) = 0.$$

The proof can easily be extended to other output variables and to input variables. We start with the definition of the covariance:

$$\begin{aligned}\mathrm{cov}\left(\sum_{i,j,t} c_{i,j,t} P_{i,j,t}, y^1\right) &= \mathrm{E}\left[y^1 \sum_{i,j,t} c_{i,j,t} P_{i,j,t}\right] - \mathrm{E}\left[y^1\right]\mathrm{E}\left[\sum_{i,j,t} c_{i,j,t} P_{i,j,t}\right] \\ &= \sum_{i,j,t} c_{i,j,t}\left(\mathrm{E}\left[y^1 P_{i,j,t}\right] - \mathrm{E}\left[y^1\right]\mathrm{E}[P_{i,j,t}]\right).\end{aligned}$$

Each of the $\mathrm{f}_{i,j,t}$ satisfies Property 1 and Property 2. Their inputs satisfy (2). Hence we can apply Corollary 1 to derive that $\mathrm{E}[y^1 P_{i,j,t}] = \mathrm{E}[y^1]\mathrm{E}[P_{i,j,t}]$. □

4.9. *Summary: What do we Achieve?*

In a circuit implementing a realization satisfying the conditions of Theorem 2, each intermediate result of the computation is statistically independent of the input and the output variables. This is a strong result. However, the required number of shares and the

typical number of gates required to implement such a circuit increases with the degree of the function that is to be implemented.

If we adopt the pipelining approach, then Property 2 is fulfilled only within each stage of the pipeline, instead of the whole realization. Hence the conditions of Theorem 2 are not fulfilled. If each of the pipeline stages satisfies the three properties, then Theorem 4 applies. The theorem implies that we achieve security against attacks that are based on correlating a secret variable or a variable derived from secret variables to the expected values of the power consumption or any other side-channel of a device or parts of a device. A foremost example of a variable derived from a secret variable is the hypothetical power consumption of a device computed by adopting a certain leakage model and a guess for some bits of the secret key [21]. We also achieve security against attacks that perform linear operations (addition, subtraction, scaling) on the side-channels before computing averages. To summarize, Theorem 4 implies first-order resistance of a shared implementation.

Alas, Theorem 4 does not cover every physical effect of a hardware implementation, nor every attack based on side-channel information. Remember that in order to prove Corollary 1, we need to make an assumption on the hardware, namely that the power consumption of each shared sub-circuit is indeed independent of the other sub-circuits. Hence, we have to make sure that for instance cross-coupling effects between different sub-circuits are negligible. However, this is at least a much easier requirement then for instance equal wire lengths, since we could place the sub-circuits separately on the chip. Further, Theorem 4 does not cover resistance against attacks using any *nonlinear* combination of shares or analyzing higher-order moments of the distribution of the power consumption. However, in practice these higher-order attacks are more difficult to perform due to the presence of noise, which is illustrated by the experimental results in Sect. 6. Further, the resistance of an implementation against higher-order attacks can be strengthened by using sharing in combination with other counter measures.

5. Implementing Nonlinear Functions Using Three Shares

Recall that Theorem 3 implies, that for any nonlinear function at least three shares are needed to fulfill Property 2. In this section we analyze which basic nonlinear functions can be shared using only three shares and present a method to construct them, such that all three properties are fulfilled. Finally, we show how the multiplication in the extension field $\mathrm{GF}(2^{2m})/\mathrm{GF}(2^m)$ (and in particular in GF(4)) can be successfully shared using three shares. This tower field approach is often used in implementations of the AES S-box [7]. Replacing the multiplication in GF(4) by a shared multiplication (which fulfills all three properties) does not immediately give pipelining stages which fulfill Property 3 as well. In general, the output distribution of combined functions is not uniform anymore. However, this could be solved by using additional random inputs at each pipelining stage.

Before we show how to find shared functions, we introduce a simplified notation which is used in the following sections. We will denote the n components of the input x by $(a, b, \dots)$ and the m components of the output z by $(e, f, \dots)$. We define the vectorial Boolean function of $z = \mathrm{f}(x)$ by

$$(e, f, \dots) = \mathrm{f}(a, b, \dots) \tag{4}$$

and its m Boolean component functions $\mathrm{f}^j(x)$ of $\mathrm{f}(x)$ as follows:

$$\begin{aligned} e &= \mathrm{f}^1(x) = \mathrm{f}^1(a, b, \dots), \\ f &= \mathrm{f}^2(x) = \mathrm{f}^2(a, b, \dots). \end{aligned} \tag{5}$$

To construct a shared implementation of the function f, each element of the input x and of the result z is divided into s shares. To divide the function f, we need to split each component function f^j into s shared functions with

$$\begin{aligned} e &= e_1 + \dots + e_s = \mathrm{f}^1\big((a_1 + \dots + a_s), (b_1 + \dots + b_s), \dots\big), \\ f &= f_1 + \dots + f_s = \mathrm{f}^2\big((a_1 + \dots + a_s), (b_1 + \dots + b_s), \dots\big). \end{aligned} \tag{6}$$

For example Property 2 requires the shares $e_i, f_i, \dots$ to be independent of $a_i, b_i, \dots$

$$\begin{aligned} e_i &= \mathrm{f}_i^1(\overline{a}_i, \overline{b}_i, \dots), \\ f_i &= \mathrm{f}_i^2(\overline{a}_i, \overline{b}_i, \dots). \end{aligned}$$

5.1. *Constructing Nonlinear Shared Functions*

In this section, we start investigating how to construct realizations that successfully "share" a given function. Here, to successfully share implies that the resulting realization of a nonlinear function satisfies all three Properties 1, 2 and 3. Note that the following steps can simply be generalized to more shares as well.

We construct nonlinear shared functions by splitting the shared function, such that only Property 1 and 2 are fulfilled first. This is always possible for any function of algebraic degree two (see Corollary 2). If we continue with the notation of Sect. 4.3 (function f^j is independent of all shares with index j), terms of degree two can only be placed in the share with the missing index. For example, the term $a_1 b_2$ can only be a part of function (or share) f_3 since f_1 has to be independent of a_1 and f_2 of b_2. However, all linear terms and quadratic terms with equal index i can be placed in one of the two shared functions f^j with $i \neq j$.

Usually, Property 3 is not fulfilled after this step. To change the output share distribution we can add other terms to the noncomplete shared functions. These *correction terms* must not violate the first two properties but can be used to fulfill Property 3. Hence, only a special set of correction terms can be added to the individual shares. To maintain Property 1, it is only possible to add the same term to an even number of different shares. This ensures that the correction terms cancel out after adding the shares. To retain Property 2, we can only add terms which are independent of at least two shares. Therefore, only linear terms and terms with equal index i can be used as correction terms.

Usually, this step is difficult to fulfill for arbitrary functions with a high algebraic degree. Two approaches can be used to simplify this step. In the first approach, we split the functions into subsequent parts with lower algebraic degree and then, try to ensure all requirements for these sub-functions first. Note that Property 3 is easier to fulfill if the resulting sub-functions are permutations (also see Sect. 6). The second approach is to add additional random inputs to the (sub-)functions. If we just remask the output

of the shared functions using fresh masks, we can always ensure Property 3. However, in our approach we try to minimize these additional masks and for simple Boolean functions, we do not need to remask at all.

5.2. *Sharing Nonlinear Functions with Two Inputs*

We first start with the most simple nonlinear Boolean functions and provide the following Theorem:

Theorem 5. *No nonlinear gate or Boolean function with two inputs and one output can be shared using three shares.*

Proof. All nonlinear Boolean functions with two inputs and one output can be defined in algebraic normal form (ANF) by the following eight functions with parameters $k_0, k_1, k_2 \in \{0, 1\}$ and index $i = k_0 \cdot 4 + k_1 \cdot 2 + k_2$:

$$f_i(a, b) = k_0 + k_1 a + k_2 b + ab. \tag{7}$$

To share these nonlinear Boolean functions using three shares, we first split the inputs a and b into three shares and get the following functions:

$$\begin{aligned} e_1 + e_2 + e_3 &= f_i(a_1 + a_2 + a_3, b_1 + b_2 + b_3) \\ &= k_0 + k_1(a_1 + a_2 + a_3) + k_2(b_1 + b_2 + b_3) \\ &\quad + (a_1 + a_2 + a_3) \cdot (b_1 + b_2 + b_3) \\ &= k_0 + k_1(a_1 + a_2 + a_3) + k_2(b_1 + b_2 + b_3) \\ &\quad + a_1b_1 + a_1b_2 + a_1b_3 + a_2b_1 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2 + a_3b_3. \end{aligned}$$

Then, terms with different indices are placed into the share with the missing index and the share for all other terms can be chosen freely.

To satisfy Property 3, the shared-output distribution of (e_1, e_2, e_3) needs to be uniform for each unshared input value (a, b). In other words, each possible shared output value has to occur equally likely. The input of the unshared functions can take the four values $(a, b) \in \{00, 01, 10, 11\}$. In the case of the shared multiplication with $f(a, b) = ab$, we get for the input $(a, b) = 00$ the output $e = e_1 + e_2 + e_3 = 0$ and the distribution of its shared output values $(e_1, e_2, e_3) \in \{000, 011, 101, 110\}$ has to be uniform.

For each of the eight nonlinear functions all possible correction terms are the constant term, the six linear terms $a_1, a_2, a_3, b_1, b_2, b_3$ and the three quadratic terms a_1b_1, a_2b_2, a_3b_3. Due to the small number of correction terms we can evaluate all possibilities and prove that no combinations leads to a uniform shared representation. It follows that a shared nonlinear function with two inputs, one output and three shares does not exist. □

5.3. *Sharing Nonlinear Functions with Three Inputs*

The result of the previous section leads to the question if there are any nonlinear functions that can be shared using three shares. To answer this question we look at the class of nonlinear Boolean functions with three inputs and one output bit:

$$f_i(a,b,c) = k_0 + k_1 a + k_2 b + k_3 c + k_4 ab + k_5 ac + k_6 bc + k_7 abc \tag{8}$$

with $k_0, \ldots, k_7 \in \{0, 1\}$. It follows from Theorem 3 that a Boolean function of algebraic degree 3 can never be shared using three shares. Therefore, we always require $k_7 = 0$. To get a nonlinear function at least one of the coefficients with degree two (k_4, k_5, k_6) needs to be non-zero and we get 112 nonlinear functions. To share these 112 functions, we split each input and output into three shares and get:

$$\begin{aligned} e_1 + e_2 + e_3 &= f_i(a_1 + a_2 + a_3, b_1 + b_2 + b_3, c_1 + c_2 + c_3) \\ &= k_0 + k_1(a_1 + a_2 + a_3) + k_2(b_1 + b_2 + b_3) + k_3(c_1 + c_2 + c_3) \\ &\quad + k_4(a_1b_1 + a_1b_2 + a_1b_3 + a_2b_1 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2 + a_3b_3) \\ &\quad + k_5(a_1c_1 + a_1c_2 + a_1c_3 + a_2c_1 + a_2c_2 + a_2c_3 + a_3c_1 + a_3c_2 + a_3c_3) \\ &\quad + k_6(b_1c_1 + b_1c_2 + b_1c_3 + b_2c_1 + b_2c_2 + b_2c_3 + b_3c_1 + b_3c_2 + b_3c_3). \end{aligned}$$

These functions can be shared using the same method as in the previous section but we can now use the following 22 correction terms:

$$\begin{aligned} &\text{linear:} && 1, a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3 \\ &\text{degree 2:} && a_1b_1, a_2b_2, a_3b_3, a_1c_1, a_2c_2, a_3c_3, b_1c_1, b_2c_2, b_3c_3 \\ &\text{degree 3:} && a_1b_1c_1, a_2b_2c_2, a_3b_3c_3 \end{aligned}$$

By adding at least three correction terms, many uniform shared functions for all of the 112 nonlinear functions can be found.

5.4. *Shared Multiplication in* $\mathrm{GF}(2^{2m})/\mathrm{GF}(2^m)$

In this section we show that the multiplication in $\mathrm{GF}(2^{2m})/\mathrm{GF}(2^m)$ and in particular GF(4) can be successfully shared using three shares. We have implemented the multiplication using normal bases, i.e. for $\mathrm{GF}(2^{2m})/\mathrm{GF}(2^m)$ the bases is defined by an element v of $\mathrm{GF}(2^{2m})$.

- Let $\{v, v^{2^m}\}$ be a normal basis of $\mathrm{GF}(2^{2m})$ over $\mathrm{GF}(2^m)$.
- Define $q = \mathrm{Tr}_{\mathrm{GF}(2^{2m})/\mathrm{GF}(2^m)}(v) = v + v^{2^m}$, so $q \in \mathrm{GF}(2^m)^*$ and
- $g = q^{-1}\mathrm{Norm}_{\mathrm{GF}(2^{2m})/\mathrm{GF}(2^m)}(v) = q^{-1}v^{2^m+1} = q^{-1}v^2 + v$, i.e. $g \in \mathrm{GF}(2^m)^*$.

Then any element x from $\mathrm{GF}(2^{2m})$ can be described by a tuple (a, b) such that $x = av + bv^{2^m}$. Let (a, b) and (c, d) be the coordinates of two elements of $\mathrm{GF}(2^{2m})$. Therefore coordinates of the product are given by the following formula:

$$(e, f) = (a, b) \times (c, d) \Leftrightarrow \begin{cases} e = (a+b)(c+d)g + qac, \\ f = (a+b)(c+d)g + qbd. \end{cases} \tag{9}$$

In the following, we will illustrate the sharing for the case $m = 1$, i.e. the multiplication in GF(4), in more detail. Formula (9) can be simplified since both constants g and q are equal to 1. Further, we use the normal basis $\{v = 01, v^2 = 10\}$.

$$(e, f) = (a, b) \times (c, d) \Leftrightarrow \begin{cases} e = (a + b)(c + d) + ac, \\ f = (a + b)(c + d) + bd. \end{cases} \tag{10}$$

To construct a shared multiplication, each of the four inputs a, b, c, and d and the results e and f are divided into three shares:

$$\begin{aligned}
&(e_1 + e_2 + e_3) \\
&\quad = (a_1 + a_2 + a_3)(c_1 + c_2 + c_3) \\
&\qquad + \big((a_1 + a_2 + a_3) + (b_1 + b_2 + b_3)\big)\big((c_1 + c_2 + c_3) + (d_1 + d_2 + d_3)\big), \\
&(f_1 + f_2 + f_3) \\
&\quad = (b_1 + b_2 + b_3)(d_1 + d_2 + d_3) \\
&\qquad + \big((a_1 + a_2 + a_3) + (b_1 + b_2 + b_3)\big)\big((c_1 + c_2 + c_3) + (d_1 + d_2 + d_3)\big).
\end{aligned}$$

After expanding the multiplication formulas, each term of the two component functions is placed into one of the three output shares. Since the multiplication consists only of quadratic terms it is always possible to fulfill Property 2:

$$\begin{aligned}
e_1 = {} & a_2d_2 + a_2d_3 + a_3d_2 \\
& + b_2c_2 + b_2c_3 + b_3c_2 \\
& + b_2d_2 + b_2d_3 + b_3d_2, \\
e_2 = {} & a_1d_3 + a_3d_1 + a_3d_3 \\
& + b_1c_3 + b_3c_1 + b_3c_3 \\
& + b_1d_3 + b_3d_1 + b_3d_3, \\
e_3 = {} & a_1d_1 + a_1d_2 + a_2d_1 \\
& + b_1c_1 + b_1c_2 + b_2c_1 \\
& + b_1d_1 + b_1d_2 + b_2d_1,
\end{aligned}
\qquad
\begin{aligned}
f_1 = {} & a_2c_2 + a_2c_3 + a_3c_2 \\
& + a_2d_2 + a_2d_3 + a_3d_2 \\
& + b_2c_2 + b_2c_3 + b_3c_2, \\
f_2 = {} & a_1c_3 + a_3c_1 + a_3c_3 \\
& + a_1d_3 + a_3d_1 + a_3d_3 \\
& + b_1c_3 + b_3c_1 + b_3c_3, \\
f_3 = {} & a_1c_1 + a_1c_2 + a_2c_1 \\
& + a_1d_1 + a_1d_2 + a_2d_1 \\
& + b_1c_1 + b_1c_2 + b_2c_1.
\end{aligned}$$

To fulfill Property 3 we need a uniform output share distribution for each of the 16 unshared input values (a, b, c, d). For example, the input $(a, b, c, d) = 0111$ results in the output $(e, f) = 01$. The shared result is uniform, if each possible value of $(e_1, e_2, e_3, f_1, f_2, f_3)$ with $e_1 + e_2 + e_3 = 0$ and $f_1 + f_2 + f_3 = 1$ occurs equally likely. We have 2^4 unshared and 2^{12} shared input values and, hence, we get $2^{12-4} = 2^8$ values for each unshared output (e, f). Since two bits of the shares $(e_1, e_2, e_3, f_1, f_2, f_3)$ have already been determined, each of the remaining 2^4 shares has to occur $2^{8-4} = 2^4$ times.

The input of the shared multiplication are the 12 variables a_i, b_i, c_i and d_i with $i \in \{1, 2, 3\}$. When searching for uniform functions, we can add only correction terms which have the same index i in all of its elements. We get one constant, four linear and

six quadratic terms, four terms of degree 3 and one term ($a_i b_i c_i d_i$) of degree 4. This gives 16 possible correction terms for each shared component function of e and f. The search space of finding a uniform representation can be reduced by allowing for only a limited number of correction terms. Further, e_i and f_i are rotation symmetric and each Boolean shared function needs to be balanced. Using at most six linear or quadratic correction terms, we have found thousands of uniform realizations of the multiplication in GF(4) using three shares. Hence, a hardware designer has still lots of freedom to choose an efficient implementation. We give one example for correction terms here:

$$
\begin{aligned}
e'_1 &= a_3 + b_2c_2 + b_3c_3 + a_2c_2,\\
f'_1 &= c_3 + d_3 + a_2c_2 + a_3c_3 + b_2d_2 + b_3d_3,\\
e'_2 &= a_1 + a_3 + d_1 + b_1c_1 + b_3c_3 + a_1d_1,\\
f'_2 &= c_3 + d_1 + d_3 + a_3c_3 + b_1d_1 + b_3d_3,\\
e'_3 &= a_1 + d_1 + b_1c_1 + b_2c_2 + a_2c_2 + a_1d_1,\\
f'_3 &= d_1 + a_2c_2 + b_1d_1 + b_2d_2.
\end{aligned}
$$

6. Secure Implementation of Noekeon

In the previous section we have analyzed which nonlinear function can be successfully shared such that all required properties are fulfilled. It turns out to be quite difficult to find a realization for more complex functions such as the AES. Especially Property 3 is difficult to achieve if no new randomness is added to the shares. However, the block ciphers Noekeon [10] and Present [6] have been designed for compact hardware implementations and consist of less complex nonlinear functions. Therefore, it is easier to find a realization for these block ciphers which is shown in this section for Noekeon and in [28] for Present.

In the following, we show a realization for the S-box of the Noekeon block cipher using three shares. We have implemented this shared function and simulate the power consumption based on back-annotated netlists, which takes data dependent glitches and timing delays into account. Then, we analyze the side-channel resistance of sharing by attacking this shared S-box and confirm its security against first-order attacks. Additionally, we analyze the higher-order resistance of the shared Noekeon S-box at the end of this section.

6.1. *Noekeon*

Noekeon is a block cipher with a block and key length of 128 bits, which has been designed to counter implementation attacks. It is an iterated cipher consisting of 16 identical rounds. In each round five simple round transformations are applied. The cipher is completely linear except for the nonlinear S-box Gamma. The linear parts can be protected against first-order DPA using one mask (two shares), whereas for the nonlinear part this is not possible.

Table 2. The substitution table of the 4-bit S-box Gamma of the block cipher Noekeon.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	7	A	2	C	4	8	F	0	5	9	1	E	3	D	B	6

The nonlinear 4-bit S-box Gamma is defined by Table 2 and consists of two equal nonlinear layers $NL(x)$, separated by a linear layer $L(x)$:

$$S(x) = NL\big(L\big(NL(x)\big)\big). \tag{11}$$

The nonlinear layer $(e, f, g, h) = NL(a, b, c, d)$, which consists of only one AND, one NOR and two XOR operations, and the linear layer $(i, j, k, l) = L(a, b, c, d)$ are defined by:

$$\begin{aligned} e &= a + (b \wedge c) = a + bc, & i &= d, \\ f &= b + \neg(c \vee d) = 1 + b + c + d + cd, & j &= b, \\ g &= c, & k &= a + b + c + d, \\ h &= d, & l &= a. \end{aligned}$$

Due to this simple structure of the S-box, it is relatively easy to find a realization using three shares, which we show in the following subsection.

6.2. *Sharing the Noekeon S-box Using Three Shares*

Since the algebraic degree of this function is 3, the whole function cannot be shared using three shares. However, if we split Gamma into two layers with algebraic degree two, we can share it using three shares again. We split Gamma after the linear layer and combine the first nonlinear layer with the linear layer to get $y = L(NL(x))$ and $z = NL(y)$. This results in less complex formulas and we get for the ANF of the resulting eight Boolean component functions $(i, j, k, l) = L(NL(a, b, c, d))$ and $(e, f, g, h) = NL(i, j, k, l)$:

$$\begin{aligned} i &= d, & e &= i + jk, \\ j &= 1 + b + c + d + cd, & f &= 1 + j + k + l + kl, \\ k &= 1 + a + b + bc + cd, & g &= k, \\ l &= a + bc, & h &= l. \end{aligned}$$

To share these functions we need to share the four inputs and outputs of each layer and get 24 shared Boolean functions. To construct these functions, we have placed the terms depending on their index into the regarding output share. This results in uniform shared functions for both layers of Gamma. The formulas for the two layers of the shared Noekeon S-box using three shares are shown in Appendix A.

We have implemented both the protected and the unprotected Noekeon S-box using a 0.35 µm standard cell library [3]. A schematic of the shared Noekeon S-box is shown in Fig. 2. In a straightforward implementation using just the ANF of the functions, the

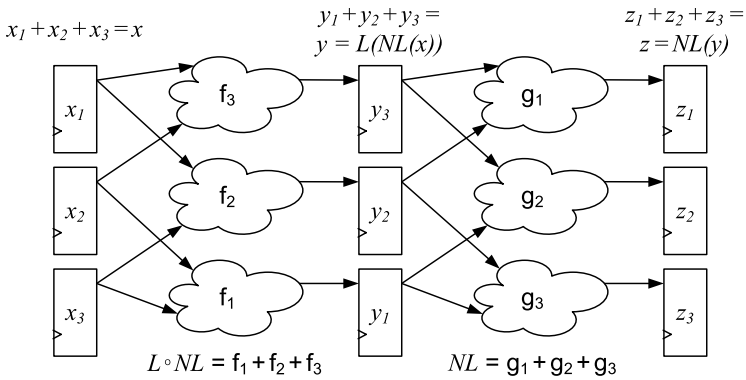

Fig. 2. A schematic of the shared Noekeon S-box using three shares.

protected S-box is approximately 3.5 times larger than the unprotected S-box (188 gate equivalents compared to 54 gate equivalents). Since there is room for further improvements and the linear parts of the Noekeon cipher can be implemented using two shares only, the relative increase in size of the implementation for the whole cipher will be smaller. This shows that shared implementations can already compete with other hardware countermeasures.

6.3. *Side-Channel Attacks Based on Simulated Power Consumptions*

To analyze the side-channel resistance of secure implementations, it is common to simulate the instantaneous power consumption of a device under attack. This is the first step to verify the effectiveness of a countermeasure. A quite accurate model is to determine the power consumption using the transition count model and a back-annotated netlist to derive the timing delays [16]. The *simulated* power consumption $L(t)$ is computed by counting the transitions at each point in time for an appropriate time quantization. The resulting power traces cover the *dynamic* switching characteristics of CMOS circuits which usually leak most information about the processed data. Note that this model takes data dependent glitches and timing delays into account and the simulated device leaks the Hamming distance (HD) between the previously and newly computed values.

In order to study the side-channel resistance of the shared S-box of Noekeon we have simulated the power consumption of the synthesized circuit using the transition count model with the timing delays of the used standard cell library. Usually, the resulting power traces depend on a secret key and a side-channel attack is applied to recover this key. We have implemented a shared Noekeon S-box with key addition at its input of the form $z = S(k \oplus d)$, where k is the secret key, d the known input data and z the output of the S-box. The shared Noekeon S-box takes as its input $x_1 = k \oplus m_1$, $x_2 = d \oplus m_2$, and $x_3 = m_1 \oplus m_2$, with m_1 and m_2 two independent random masks. The simulated power consumption at time t is denoted by $L(t)$ and examples of simulated power traces for the secret key $k = 11$ are given in Fig. 3. Note that in each clock cycle, glitches occur since many signals switch their state more than once until the output of the combinational logic is settled.

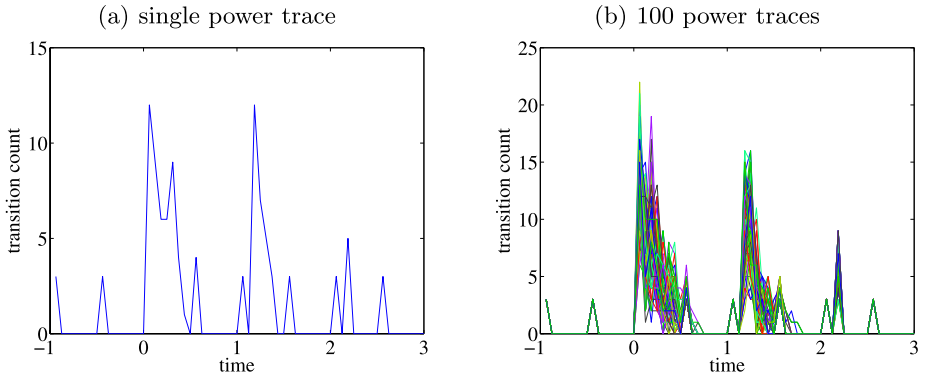

Fig. 3. A single (left) and 100 (right) simulated power traces for key $k = 11$ with a time quantization of 16 for each clock cycle. The two clock cycles of the computation (first layer: time 0-1, second layer: time 1-2) and the cycle for storing the result in the output register (time 2-3) are clearly visible.

6.4. *Correlation Attack*

In this section, we analyze the first-order resistance of the shared Noekeon S-box. We use a correlation attack to detect linear and first-order dependencies between the hypothetical and simulated power consumption. Hence, according to Theorem 4 a successful correlation attack on the shared Noekeon S-box should not be possible.

In a correlation attack, we try to correlate the *simulated* power consumption $L(t)$ with the *hypothetical* power consumption $H(k)$ derived from the different key guesses k and an appropriate power model. If the simulated device leaks the Hamming distance (HD), also the hypothetical power consumption $H(k)$ is based on the Hamming distance between the previously and newly computed hypothetical values. If the correct key guess results in a correlation value that can be statistically distinguished from the correlation values obtained using the wrong key guesses, then we declare the attack to be successful.

We have computed the correlation between the simulated power consumption $L(t)$ and the hypothetical power consumption for the first $H_y(k) = \mathrm{HD}(y', y'')$ and second layer $H_z(k) = \mathrm{HD}(z', z'')$ of the Noekeon S-box (y', z': previous values; y'', z'': new values). Figure 4 shows the results for the unprotected (unshared) S-box. The correct key guess results in a much larger correlation between $L(t)$ and $H(k)$ for each layer. The results of the shared Noekeon S-box are shown in Fig. 5. There is no point in time where the correlation between $L(t)$ and $H(k)$ for the correct key guess is distinguishable from the correlation for the wrong key guesses. These results show that there is no linear relation between the number of transitions and the unshared values. This confirms Theorem 4 in practice and that a constant mean power consumption can be achieved using the proposed sharing approach.

6.5. *Mutual Information Analysis (MIA)*

In the previous section we have verified that the mean power consumption is independent of the unshared values. Hence, we have shown that the sharing scheme resists first-order attacks using simulated power traces. In this section we perform a higher-order

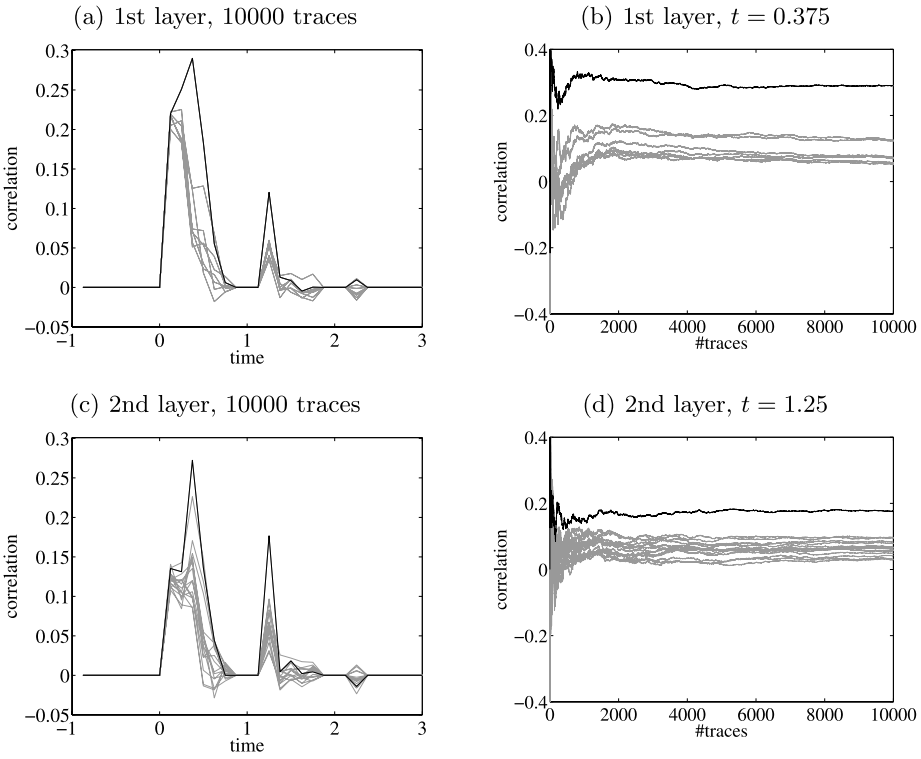

Fig. 4. Simulated correlation attack on the *unshared* Noekeon S-box (black: correct key), with a time quantization of 8 for each clock cycle which gives the best result.

attack or more specific, we analyze higher-order moments of the probability distribution functions of the unshared values. Note that we do not claim provable security against higher-order attacks in the case of the shared Noekeon S-box using three shares. Only the mean value (the first-order moment) is provably independent of the unshared values. Note that in Boolean masking, the distribution using three shares is not independent of the unshared value. For example, a 1-bit value 0 is shared by $\{000, 011, 110, 101\}$ and a 1-bit value 1 by $\{001, 010, 100, 111\}$ which clearly shows that the distributions or higher-order moments are not independent of the unshared values.

Hence, Boolean sharing (or masking) schemes are usually not resistant against higher-order attacks. If the power consumption of more shares are combined, the probability distribution functions of each unshared value can be distinguished by analyzing their higher-order moments [26,34]. To analyze higher-order dependencies of probability distribution functions, mutual information analysis (MIA) has been proposed in [12]. Indeed, mutual information analysis can lead to better side-channel attacks if the measured statistical leakage $L(t)$ is related to the hypothesis $H(k)$ in a nonlinear way [29,40]. However, using more shares the resistance of Boolean masking increases if the noise increases as well [8,34].

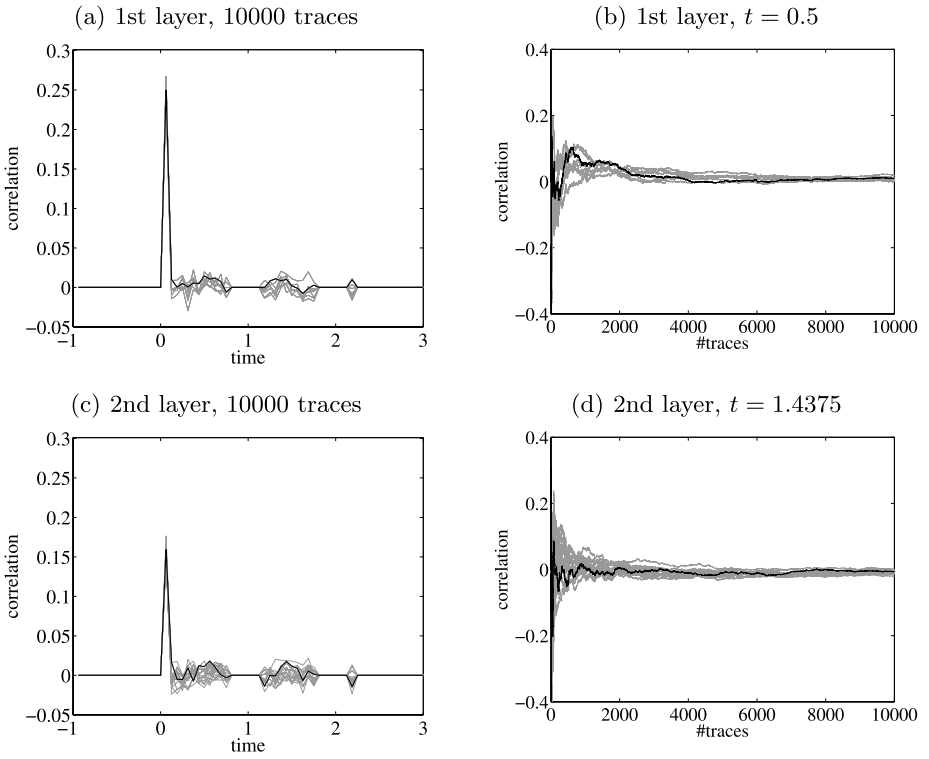

Fig. 5. Simulated correlation attack on the *shared* Noekeon S-box (black: correct key), with a time quantization of 16 for each clock cycle which gives the best result.

In the following, we first give a brief introduction to mutual information analysis. The mutual information of two random variables x and y can be defined using the (conditional) entropy of these two variables. The *entropy* $\mathrm{H}(x)$ of a random variable x is a measure of the amount of information one can obtain from an observation of x. The entropy is defined as

$$\mathrm{H}(x) = -\sum_{X \in x} \Pr(x = X) \cdot \log_2\big(\Pr(x = X)\big).$$

To determine the mutual information, we also need to compute the conditional entropy. The *Conditional Entropy* specifies the entropy of a random variable x, given a random variable y which has been obtained from some related experiment and is defined as follows:

$$\mathrm{H}(x|y) = \sum_{Y \in y} \Pr(y = Y) \cdot \mathrm{H}(x|y = Y).$$

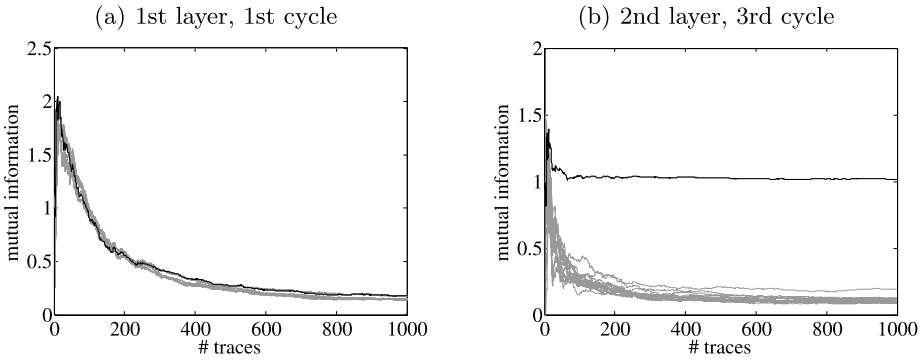

Fig. 6. Mutual information analysis on the shared Noeken S-box (black: correct key). In the first cycle only the combinational logics and in the third cycle only the registers are attacked.

Using these two definitions, the *Mutual Information* gives the amount of information that the given random variable y reveals about x and is defined as follows:

$$\mathrm{I}(x; y) = \mathrm{H}(x) - \mathrm{H}(x|y).$$

We apply a mutual information analysis to the same simulated power consumption $L(t)$ as in the correlation attack. Again, the hypothetical power consumption is the Hamming distance (HD) of the unshared values again and we get $H_y = \mathrm{HD}(y', y'')$ for the first, and $H_z = \mathrm{HD}(z', z'')$ for the second layer. Then, we evaluate $\mathrm{I}(H_y(k); L(t))$ and $\mathrm{I}(H_z(k); L(t))$ for all key guesses, which gives the mutual information between the hypothetical and simulated power consumption. We have determined the probability distribution functions (pdf) of $H(k)$ and $L(t)$ using the histogram method with 256 bins. For more details on computing the mutual information we refer to [29]. For a successful attack, the mutual information using the correct key guess should be statistically distinguishable from the mutual information using the wrong key guesses.

Figure 6 and Fig. 7 show the result of the mutual information analysis for the first and second layer of the shared Noekeon S-box at different points in time. In this attack we have used the total transition count for each clock cycle. Note that the results of the attacks do not change if a higher time quantization or more traces are used. We have attacked the three peaks of Fig. 3 at clock cycle 1, 2 and 3: Fig. 6(a) shows the first cycle (time 0-1), where only the combinational logic of the first layer is computed, and Fig. 6(b) shows the third cycle (time 2-3), where the output of the shared S-box is stored in the output registers without subsequent combinational logic. The mutual information analysis shows that an attack on merely the combinational logic (first cycle, Fig. 6(a)) is not feasible due to the high algorithmic noise of the combinational logic itself. The correct key can only be recovered in the third cycle, where only the result of the second layer is stored in the output register (Fig. 6(b)). In this case the register transitions are measured and attacked without any noise.

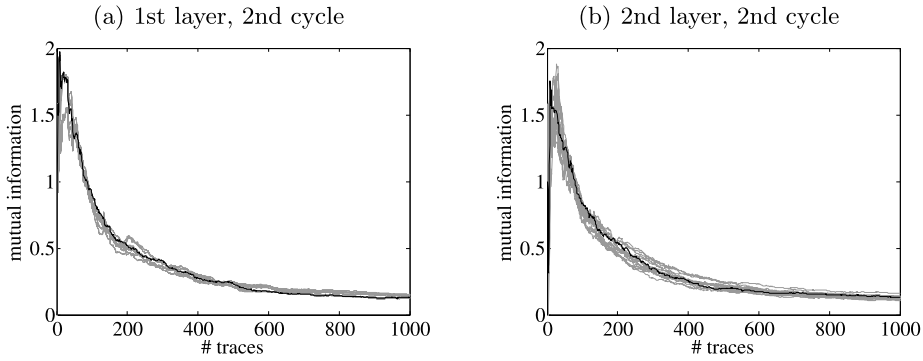

Fig. 7. Mutual information analysis on the shared Noeken S-box (black: correct key). In the second cycle, the computation of the combinational logic overlaps storing the previous results in the registers.

6.6. *Protections against Mutual Information Analysis*

Even though an attack is feasible in the third cycle, sharing is secure in the first-order setting i.e. if only the power consumption of one combinational logic or the mean of the total power consumption is analyzed. However, mutual information analysis exploits higher-order moments of the probability distribution function as well as the combination of all three shares. Note that higher-order countermeasures such as sharing are more powerful in the presence of noise. Additionally, the measurement of higher-order moments is also more sensitive to noise. Note that in practice, the transitions of the registers are overlapped by the algorithmic noise of the subsequent combinational logic which makes a higher-order comparison of probability distribution functions more difficult.

We confirm this behavior by performing a mutual information analysis on the second cycle (time 1-2), where the results of the previous computation are stored in the registers *and* the combinational logic of the second layer is computed. The results show that an attack is indeed not possible if the transitions of the registers are overlapped by the transitions of the subsequent combinational logic (second cycle, Fig. 7(a) and Fig. 7(b)). Hence, already the algorithmic noise make a mutual information analysis infeasible for shared implementations. In practice, the noise level will be even higher and it seems to be difficult that a shared implementation can be attacked in practice using mutual information analysis.

7. Conclusion

In this paper we have proposed a method to construct implementations of cryptographic functions that are secure against a large class of side-channel attacks, making only minimal assumptions on the underlying hardware. In particular, our method works also when the hardware technology is not glitch-free.

The approach is based on multi-party computation protocols. It also takes into account that in order to be used in practice, the overhead caused by protection measures should be kept under control. We have defined three properties that are sufficient to

make an implementation resistant against attacks that work by computing the correlation between the average power consumption of the device and the hypothetical power consumption using a known (guessed) key.

We have analyzed which basic nonlinear functions can be securely implemented using the minimum of three shares and presented a method to construct shared Boolean functions. We have implemented the block cipher Noekeon using only three shares by introducing pipelining stages separated by latches or registers. Finally, we have presented the first verification of this implementation method based on computer simulations.

We see several possibilities to extend our work. Firstly, it might be possible to extend our approach in order to achieve provable resistance against a *wider range of attacks*, e.g. by using more shares and/or fresh randomness after a number of steps. An alternative approach would be to introduce some assumptions on the hardware and the signal-to-noise ratio of its side-channels. For instance, in theory very powerful mutual information attack turns out to be very sensitive toward noise. Combining our approach with the addition of fresh randomness or some extra noise to the hardware circuit seems to result in a very strong protection (also see [28]).

Secondly, one might want to provide resistance also against *fault attacks*. In a fault attack, the attacker causes an error during the execution of the cryptographic functions in order to defeat some of the protection mechanisms. In some sense, a fault attack compares to an ordinary side-channel attack like a chosen-plaintext attack compares to a known-plaintext attack. We imagine that techniques from threshold cryptography could help to make a circuit recover automatically from errors, without leaking information.

A third line of future work is to securely implement *more complex nonlinear functions,* such as the AES S-boxes, which is still a mathematically challenging task. However, the construction of complex Boolean functions according to Theorem 4 can be simplified by using additional, fresh random input masks at each register level.

Acknowledgements

We thank Thomas Popp and Marcel Medwed for many useful discussions. This work has been supported in part by the IAP Programme P6/26 (BCRYPT) of the Belgian State (Belgian Science Policy), by the Research Fund K.U.Leuven and by the European Commission under contract ICT-2007-216646 (ECRYPT II). We also thank Thomas Popp and Marcel Medwed for many useful discussions.

Appendix A. Formulas for the Noekeon S-box Using Three Shares

The formulas in ANF of the shared Noekeon S-box or nonlinear function Gamma using three shares. The first step combines the first nonlinear layer with the linear layer:

$$
\begin{aligned}
i_1 &= d_2,\\
i_2 &= d_3,\\
i_3 &= d_1,\\
j_1 &= 1 + b_2 + c_2 + d_2 + c_2d_2 + c_3d_2 + c_2d_3,\\
j_2 &= b_3 + c_3 + d_3 + c_3d_1 + c_1d_3 + c_3d_3,\\
j_3 &= b_1 + c_1 + d_1 + c_1d_1 + c_2d_1 + c_1d_2,\\
k_1 &= 1 + a_2 + b_2 + b_2c_2 + b_3c_2 + b_2c_3 + c_2d_2 + c_3d_2 + c_2d_3,\\
k_2 &= a_3 + b_3 + b_3c_1 + b_1c_3 + b_3c_3 + c_3d_1 + c_1d_3 + c_3d_3,\\
k_3 &= a_1 + b_1 + b_1c_1 + b_2c_1 + b_1c_2 + c_1d_1 + c_2d_1 + c_1d_2,\\
l_1 &= a_2 + b_2c_2 + b_3c_2 + b_2c_3,\\
l_2 &= a_3 + b_3c_1 + b_1c_3 + b_3c_3,\\
l_3 &= a_1 + b_1c_1 + b_2c_1 + b_1c_2.
\end{aligned}
$$

The second step consists only of the second nonlinear layer:

$$
\begin{aligned}
e_1 &= i_2 + j_2k_2 + j_3k_2 + j_2k_3,\\
e_2 &= i_3 + j_3k_1 + j_1k_3 + j_3k_3,\\
e_3 &= i_1 + j_1k_1 + j_2k_1 + j_1k_2,\\
f_1 &= 1 + j_2 + k_2 + l_2 + k_2l_2 + k_3l_2 + k_2l_3,\\
f_2 &= j_3 + k_3 + l_3 + k_3l_1 + k_1l_3 + k_3l_3,\\
f_3 &= j_1 + k_1 + l_1 + k_1l_1 + k_2l_1 + k_1l_2,\\
g_1 &= k_2,\\
g_2 &= k_3,\\
g_3 &= k_1,\\
h_1 &= l_2,\\
h_2 &= l_3,\\
h_3 &= l_1.
\end{aligned}
$$

References

[1] M.L. Akkar, C. Giraud, An implementation of DES and AES, secure against some attacks, in *CHES*, ed. by Çetin Kaya Koç, D. Naccache, C. Paar. LNCS, vol. 2162 (Springer, Berlin, 2001), pp. 309–318

[2] M.L. Akkar, R. Bevan, L. Goubin, Two power analysis attacks against one-mask methods, in *FSE*, ed. by B.K. Roy, W. Meier. LNCS, vol. 3017 (Springer, Berlin, 2004), pp. 332–347

[3] Austria Microsystems: Standard Cell Library 0.35 µm CMOS (C35), http://asic.austriamicrosystems.com/databooks/c35/databook_c35_33
[4] M. Ben-Or, S. Goldwasser, A. Wigderson, Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract), in *STOC* (ACM, New York, 1988), pp. 1–10
[5] J. Blömer, J. Guajardo, V. Krummel, Provably secure masking of AES, in *Selected Areas in Cryptography*, ed. by H. Handschuh, M.A. Hasan. LNCS, vol. 3357 (Springer, Berlin, 2004), pp. 69–83
[6] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe, PRESENT: An ultra-lightweight block cipher, in *CHES*, ed. by P. Paillier, I. Verbauwhede. LNCS, vol. 4727 (Springer, Berlin, 2007), pp. 450–466
[7] D. Canright, A very compact S-box for AES, in *CHES*, ed. by J.R. Rao, B. Sunar. LNCS, vol. 3659 (Springer, Berlin, 2005), pp. 441–455
[8] S. Chari, C.S. Jutla, J.R. Rao, P. Rohatgi, Towards sound approaches to counteract power-analysis attacks, in *CRYPTO*, ed. by M.J. Wiener. LNCS, vol. 1666 (Springer, Berlin, 1999), pp. 398–412
[9] J. Daemen, V. Rijmen, AES proposal: Rijndael. Submitted as an AES Candidate Algorithm (2000), http://www.nist.gov/aes
[10] J. Daemen, M. Peeters, G.V. Assche, V. Rijmen, Nessie proposal: NOEKEON. Submitted as an NESSIE Candidate Algorithm (2000), http://www.cryptonessie.org
[11] W. Fischer, B.M. Gammel, Masking at gate level in the presence of glitches, in *CHES*, ed. by J.R. Rao, B. Sunar. LNCS, vol. 3659 (Springer, Berlin, 2005), pp. 187–200
[12] B. Gierlichs, L. Batina, P. Tuyls, B. Preneel, Mutual information analysis, in *CHES*, ed. by E. Oswald, P. Rohatgi. LNCS, vol. 5154 (Springer, Berlin, 2008), pp. 426–442
[13] J.D. Golic, C. Tymen, Multiplicative masking and power analysis of AES, in *CHES*, ed. by B.S. Kaliski Jr., Çetin Kaya Koç, C. Paar. LNCS, vol. 2523 (Springer, Berlin, 2002), pp. 198–212
[14] Y. Ishai, M. Prabhakaran, A. Sahai, D. Wagner, Private circuits II: Keeping secrets in tamperable circuits, in *EUROCRYPT*, ed. by S. Vaudenay. LNCS, vol. 4004 (Springer, Berlin, 2006), pp. 308–327
[15] Y. Ishai, A. Sahai, D. Wagner, Private circuits: Securing hardware against probing attacks, in *CRYPTO*, ed. by D. Boneh. LNCS, vol. 2729 (Springer, Berlin, 2003), pp. 463–481
[16] M. Kirschbaum, T. Popp, Evaluation of power estimation methods based on logic simulations, in *Austrochip*, ed. by K.C. Posch, J. Wolkerstorfer (Verlag der Technischen Universität Graz, Graz, 2007), pp. 45–51
[17] P.C. Kocher, J. Jaffe, B. Jun, Differential power analysis, in *CRYPTO*, ed. by M.J. Wiener. LNCS, vol. 1666 (Springer, Berlin, 1999), pp. 388–397
[18] S. Mangard, K. Schramm, Pinpointing the side-channel leakage of masked AES hardware implementations, in *CHES*, ed. by L. Goubin, M. Matsui. LNCS, vol. 4249 (Springer, Berlin, 2006), pp. 76–90
[19] S. Mangard, T. Popp, B.M. Gammel, Side-channel leakage of masked CMOS gates, in *CT-RSA*, ed. by A. Menezes. LNCS, vol. 3376 (Springer, Berlin, 2005), pp. 351–365
[20] S. Mangard, N. Pramstaller, E. Oswald, Successfully attacking masked AES hardware implementations, in *CHES*, ed. by J.R. Rao, B. Sunar. LNCS, vol. 3659 (Springer, Berlin, 2005), pp. 157–171
[21] S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks—Revealing the Secrets of Smart Cards* (Springer, Berlin, 2007), http://www.dpabook.org
[22] T.S. Messerges, Securing the AES finalists against power analysis attacks, in *FSE*, ed. by B. Schneier. LNCS, vol. 1978 (Springer, Berlin, 2000), pp. 150–164
[23] S. Nikova, C. Rechberger, V. Rijmen, Threshold implementations against side-channel attacks and glitches, in *ICICS*, ed. by P. Ning, S. Qing, N. Li. LNCS, vol. 4307 (Springer, Berlin, 2006), pp. 529–545
[24] S. Nikova, V. Rijmen, M. Schläffer, Secure hardware implementation of non-linear functions in the presence of glitches, in *ICISC*, ed. by P.J. Lee, J.H. Cheon. LNCS, vol. 5461 (Springer, Berlin, 2008), pp. 218–234
[25] E. Oswald, S. Mangard, N. Pramstaller, V. Rijmen, A side-channel analysis resistant description of the AES S-box, in *FSE*, ed. by H. Gilbert, H. Handschuh, LNCS, vol. 3557 (Springer, Berlin, 2005), pp. 413–423
[26] F.J. Pautot, Some formal solutions in side-channel cryptanalysis—an introduction. Cryptology ePrint Archive, Report 2008/508 (2008), http://eprint.iacr.org/
[27] T. Popp, S. Mangard, Masked dual-rail pre-charge logic: DPA-resistance without routing constraints, in *CHES*, ed. by J.R. Rao, B. Sunar. LNCS, vol. 3659 (Springer, Berlin, 2005), pp. 172–186

[28] A. Poschmann, A. Moradi, K. Khoo, C.W. Lim, H. Wang, S. Ling, Side-channel resistant crypto for less than 2,300 GE. *J. Cryptol.* Special Issues on Hardware and Security (2010). doi:10.1007/s00145-010-9086-6

[29] E. Prouff, M. Rivain, Theoretical and practical aspects of mutual information based side channel analysis, in *ACNS*, ed. by M. Abdalla, D. Pointcheval, P.A. Fouque, D. Vergnaud. LNCS, vol. 5536 (2009), pp. 499–518

[30] J.M. Rabaey, *Digital Integrated Circuits: A Design Perspective* (Prentice-Hall, Upper Saddle River, 1996)

[31] M. Rivain, E. Dottax, E. Prouff, Block ciphers implementations provably secure against second order side channel analysis, in *FSE*, ed. by K. Nyberg. LNCS, vol. 5086 (Springer, Berlin, 2008), pp. 127–143

[32] K. Schramm, C. Paar, Higher order masking of the AES, in *CT-RSA*, ed. by D. Pointcheval. LNCS, vol. 3860 (Springer, Berlin, 2006), pp. 208–225

[33] A. Shamir, How to share a secret. *Commun. ACM* **22**(11), 612–613 (1979)

[34] F.X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, S. Mangard, The world is not enough: Another look on second-order DPA. Cryptology ePrint Archive, Report 2010/180 (2010), http://eprint.iacr.org/

[35] D. Suzuki, M. Saeki, T. Ichikawa, DPA leakage models for CMOS logic circuits, in *CHES*, ed. by J.R. Rao, B. Sunar. LNCS, vol. 3659 (Springer, Berlin, 2005), pp. 366–382

[36] K. Tiri, I. Verbauwhede, Securing encryption algorithms against DPA at the logic level: Next generation smart card technology, in *CHES*, ed. by C.D. Walter, Çetin Kaya Koç, C. Paar. LNCS, vol. 2779 (Springer, Berlin, 2003), pp. 125–136

[37] K. Tiri, I. Verbauwhede, A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation, in *DATE* (IEEE Computer Society, Los Alamitos, 2004), pp. 246–251

[38] E. Trichina, T. Korkishko, K.H. Lee, Small size, low power, side channel-immune AES coprocessor: Design and synthesis results, in *AES Conference*, ed. by H. Dobbertin, V. Rijmen, A. Sowa. LNCS, vol. 3373 (Springer, Berlin, 2004), pp. 113–127

[39] E. Trichina, D.D. Seta, L. Germani, Simplified adaptive multiplicative masking for AES, in *CHES*, ed. by B.S. Kaliski Jr., Çetin Kaya Koç, C. Paar. LNCS, vol. 2523 (Springer, Berlin, 2002), pp. 187–197

[40] N. Veyrat-Charvillon, F.X. Standaert, Mutual information analysis: How, when and why? in *CHES*, ed. by C. Clavier, K. Gaj. LNCS, vol. 5747 (Springer, Berlin, 2009), pp. 429–443

[41] J. Waddle, D. Wagner, Towards efficient second-order power analysis, in *CHES*, ed. by M. Joye, J.J. Quisquater. LNCS, vol. 3156 (Springer, Berlin, 2004), pp. 1–15

[42] J. Wolkerstorfer, E. Oswald, M. Lamberger, An ASIC implementation of the AES SBoxes, in *CT-RSA*, ed. by B. Preneel. LNCS, vol. 2271 (Springer, Berlin, 2002), pp. 67–78