Secure Hardware Implementation of Non-Linear Functions in the Presence of Glitches *

Svetla Nikova[1] and Vincent Rijmen[1,2] and Martin Schläffer[2]

Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC and IBBT,
Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium
Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
martin.schlaeffer@iaik.tugraz.at

Abstract. Hardware implementations of cryptographic algorithms are still 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. In the presence of glitches, most of the currently known masking techniques still leak information during the computation of non-linear functions. We discuss a recently introduced masking method which is based on secret sharing and results in implementations that are provable resistant against first-order side-channel attacks, even in the presence of glitches. We reduce the hardware requirements of this method and show how to derive provable secure implementations of some non-linear building blocks for cryptographic algorithms. Finally, we provide a provable secure implementation of the block cipher Noekeon and verify the results.

Keywords: DPA, masking, glitches, sharing, non-linear 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 consumption of a cryptographic device using differential power analysis (DPA) [13]. 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.

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

* The work in this paper has been supported in part by the Austrian Government (BMVIT) through the research program FIT-IT Trust (Project ARTEUS) and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

P.J. Lee and J.H. Cheon (Eds.): ICISC 2008, LNCS 5461, pp. 218–234, 2008.
The original publication is available at http://www.springerlink.com/content/v17r43218758t805/

[26,27] 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 [2,4,9,20], at the gate level [10,28] or even in combination with circuit design approaches [21].

However, recent attacks have shown that masked hardware implementations (contrary to software implementations [24,23]) can still be attacked using even first-order 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 [22] occur in combinational CMOS circuits and each signal switches several times. Due to these glitches, these circuits are vulnerable to DPA attacks [15,16]. 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. In [19] and in this paper, a different type of approach is followed. 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. The approach is based on only a single assumption about the implementation technology, namely: the existence of memory cells that completely isolate the switching characteristics of their inputs from their outputs (e.g. registers). Therefore, it holds for both, FPGAs and ASICs. Secret sharing schemes and techniques from multiparty computation are used to construct combinational logic which is completely independent of the unmasked values.

In this approach the hardware requirements increase with the number of shares and for each non-linear part of a circuit at least three shares (or masks) are needed. Constructing secure implementations of arbitrary functions using only a small number of shares, is a difficult task. In [19] the inversion in GF(16) has been implemented using 5 shares. In this paper we analyze which basic non-linear functions can be securely implemented using only three shares and present a method how to construct such shared Boolean functions. We show that the multiplication in GF(4) and the block cipher Noekeon can be implemented using three shares as well and present the first verification of this method based on computer simulations.

In the next section we give a short overview on simulation based DPA attacks to motivate sharing. Section 3 reformulates and extends the sharing approach of [19]. In Sect. 4 we show which non-linear functions can be securely implemented using three shares. In Sect. 5 we give a provable secure implementation of the S-box of the block cipher Noekeon using three shares. Computer simulations confirm that this shared S-box is not vulnerable to DPA, even in the presence of glitches.

2 DPA Attacks on Masking

Masking is a side-channel countermeasure which tries to randomize the intermediate values of a cryptographic algorithm [18]. 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 [7] or ARIA [14] combine linear and non-linear functions. Thus, many different hardware masking schemes and masked gates have been proposed [2,4,20,29] but all of them have been broken already [1,9,16,30]. Even though no wire carries an unmasked value, the power consumption correlates with the unmasked intermediate results of the algorithm.

2.1 Glitches

The problem of these hardware masking schemes is that the effect of glitches has not been considered. Glitches have first been analyzed in [15] and a technique to model glitches has been presented in [25]. 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 implementation and on the input values of a combinational logic. Since the power consumption of CMOS circuits strongly depends on the amount of glitches, it depends on all inputs as well. 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, it depends on the actual hardware implementation whether a design is secure and cannot be proven during the design process.

2.2 Simulation Based 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 [17] 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 [8] 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 non-linear 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 non-linear masked gates which can be used to build arbitrary circuits. However, the big disadvantage of this method is that semi-G-equivalent

circuits still have routing constraints and it 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 [12]. 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.

3 Sharing

In this section we recall the most important elements of the approach in [19]. The idea is to build combinational blocks which are completely independent of the unmasked values. This can be achieved by avoiding that masked values and masks are used as an input to the same combinational circuit. Each function is shared into combinational blocks which are independent of at least one share. Hence, also the number of glitches and the power consumption is independent of any unmasked value. This method can be extended to counter even higher-order attacks by increasing the number of shares [19].

3.1 Terminology

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 $x = \bigoplus_i x_i$. Since we are only using linear or XOR masking, the addition $x + y$ is always computed over $\mathrm{GF}(2^n)$ in the remainder of this paper. Further, we will only use (s, s) secret sharing schemes, hence all s shares are needed in order to determine x uniquely. More precisely, we use secret sharing schemes where the conditional probability distribution $\Pr(\overline{x}|x)$ is uniform for every possible sharing vector $\overline{X} = (X_1, X_2, \ldots, X_s)$:

$$\Pr(\overline{x} = \overline{X}) = c \Pr(x = \bigoplus_i X_i) \tag{1}$$

with c a normalization constant, which ensures that $\sum_{\overline{X}} \Pr(\overline{x} = \overline{X}) = 1$. In words, any bias present in the joint distribution of the shares $\overline{x}$ is only due to a bias in the distribution of the unshared variable x. This means, that for each unshared value of x, all possible sharing vectors $\overline{X} = (X_1, X_2, \ldots, X_s)$ with $x = \bigoplus_i X_i$ need to occur equally likely.

3.2 Realization

Assume we want to implement a vectorial Boolean function $z = \mathrm{f}(x)$ with the n-bit input x and the m-bit output z. Then we need a set of functions f_i which together compute the output of f. We call this a *realization* and get the following property:

Property 1 (Correctness [19]). Let $z = \mathrm{f}(x)$ be a vectorial Boolean function. Then the set of functions $\mathrm{f}_i(\overline{x})$ is a *realization* of f if and only if

$$z = \mathrm{f}(x) = \bigoplus_{i=1}^{s} \mathrm{f}_i(\overline{x}) \tag{2}$$

for all shares $\overline{x}$ satisfying $\bigoplus_{i=1}^{s} x_i = x$.

Alternatively, we will denote the n components of x by $(a, b, \dots)$ and the m components of z by $(e, f, \dots)$. We define the vectorial Boolean function of $z = \mathrm{f}(x)$ by:

$$(e, f, \dots) = \mathrm{f}(a, b, \dots) \tag{3}$$

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

$$\begin{aligned} e &= \mathrm{f}^1(x) = \mathrm{f}^1(a, b, \dots) \\ f &= \mathrm{f}^2(x) = \mathrm{f}^2(a, b, \dots) \end{aligned} \tag{4}$$

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:

$$\begin{aligned} e &= e_1 + \cdots + e_s = \mathrm{f}^1((a_1 + \cdots + a_s), (b_1 + \cdots + b_s), \dots) \\ f &= f_1 + \cdots + f_s = \mathrm{f}^2((a_1 + \cdots + a_s), (b_1 + \cdots + b_s), \dots) \end{aligned} \tag{5}$$

3.3 Non-completeness

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

Property 2 (Non-completeness [19]). 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:

$$\begin{aligned} z_1 &= \mathrm{f}_1(x_2, x_3, \dots, x_s) &&= \mathrm{f}_1(\overline{a}_1, \overline{b}_1, \dots) \\ z_2 &= \mathrm{f}_2(x_1, x_3, \dots, x_s) &&= \mathrm{f}_2(\overline{a}_2, \overline{b}_2, \dots) \\ &\dots \\ z_s &= \mathrm{f}_s(x_1, x_2, \dots, x_{s-1}) &&= \mathrm{f}_s(\overline{a}_s, \overline{b}_s, \dots) \end{aligned} \tag{6}$$

Theorem 2 and Theorem 3 of [19] are essential. These theorems state that the shared components f_i of a function $z = \mathrm{f}(x)$ are independent of the input x and output z, as long as the shared realization satisfies Property 2 and the input vectors satisfy (1). Therefore, also the mean power consumption, or any other characteristic of an implementation of each component f_i is independent of the unmasked values x and z, even in the presence of glitches or the delayed arrival time of some inputs. More general, to counter DPA attacks of order r, each shared component function needs to be independent of at least r shares.

When partitioning each Boolean component function f^j, we need to ensure that Property 2 is satisfied for each component. 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, namely independent of a_i, b_i, ...

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

3.4 Uniform

In principle, Property 2 is sufficient to construct secure implementations of arbitrary (vectorial) Boolean functions. However, the number of required shares, and thereby the size of the circuit, grows rapidly with the algebraic degree of the function. This can be reduced by splitting the function into stages. The only requirement is that the switching characteristics between each stage are isolated. This can be achieved by a pipelined implementation, where the different stages are separated by registers or latches. In order to design the different stages separately, we need to ensure (1) for the input shares $\overline{x}$ of each stage. Since every output of a stage is used as the input in the next stage we need to make assumptions about the probability distribution of the output shares $\overline{z}$ of a shared function as well. The following property ensures that if the input-share distribution satisfies (1), also the output-share distribution does:

Property 3 (Uniform [19]). [1] A realization of $z = \mathrm{f}(x)$ is *uniform*, if for all distributions of the inputs x and for all input share distributions satisfying (1), the conditional probability

$$\Pr(\overline{z} = \overline{Z} | z = \bigoplus_i Z_i) \tag{7}$$

is constant.

4 Sharing Non-Linear Functions using 3 Shares

In [19] it has been proven, that at least three shares are needed to fulfill Property 2 for any non-linear function. However, in their paper the best result was a

[1] This property is called *Balance* instead of *Uniform* in [19].

uniform implementation for the inversion in GF(16) using 5 shares. In this section we analyze which basic non-linear 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 GF(4), which is often used in the implementation of the AES S-box [5], can be successfully shared using three shares.

4.1 Constructing Non-linear Shared Functions

We construct non-linear shared functions by splitting the shared function, such that only Property 1 and 2 are fulfilled first. In [19] it has been proven that this is always possible for any function of algebraic degree two. If we continue with the notation of Sect. 3.3 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 non-complete 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.

4.2 Sharing Non-linear Functions with 2 Inputs

In [19] it has been shown that no shared AND gate using three shares exists which fulfills all three properties. Note that using *any* single non-linear gate we would be able to build arbitrary circuits. Hence, we generalize the idea in this section.

Theorem 1. *No non-linear gate or Boolean function with two inputs and one output can be shared using three shares.*

Proof. All non-linear Boolean functions with two inputs and one output can be defined in algebraic normal form (ANF) by the following 8 functions with parameters $k_0, k_1, k_2 \in \{0, 1\}$ and index $i = k_0 \cdot 4 + k_1 \cdot 2 + k_2$:

$$f_i(a, b) = k_0 + k_1 a + k_2 b + ab. \tag{8}$$

To share these non-linear 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 4 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 8 non-linear functions all possible correction terms are the constant term, the 6 linear terms $a_1, a_2, a_3, b_1, b_2, b_3$ and the 3 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 non-linear function with 2 inputs, one output and 3 shares does not exist. □

4.3 Sharing Non-linear Functions with 3 Inputs

The result of the previous section leads to the question if there are any non-linear functions that can be shared using three shares. To answer this question we look at the class of non-linear Boolean functions with 3 inputs and one output bit:

$$f_i(a, b, c) = k_0 + k_1a + k_2b + k_3c + k_4ab + k_5ac + k_6bc + k_7abc \tag{9}$$

with $k_0, \ldots, k_7 \in \{0, 1\}$. As shown in [19, Theorem 1], a Boolean function of algebraic degree 3 can never be shared using three shares. Therefore, we always require $k_7 = 0$. To get a non-linear 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 non-linear 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:

linear:	$1, a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$
degree 2:	$a_1b_1, a_2b_2, a_3b_3, a_1c_1, a_2c_2, a_3c_3, b_1c_1, b_2c_2, b_3c_3$
degree 3:	$a_1b_1c_1, a_2b_2c_2, a_3b_3c_3$

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

4.4 Shared Multiplication in GF(4)

In this section we show that the multiplication in GF(4) can be successfully shared using three shares. We have implemented the multiplication in GF(4) using normal bases. The used normal basis is (v, v^2) and the two elements are represented by $v = 01$ and $v^2 = 10$. The zero element is represented by 00 and the one element by 11. We define the multiplication in GF(4) using this normal basis by:

$$\begin{aligned}(e, f) &= (a, b) \times (c, d)\\ e &= ac + (a + b)(c + d)\\ f &= bd + (a + b)(c + d)\end{aligned}$$

and get the following multiplication tables:

$\times$	0	1	v	v^2
0	0	0	0	0
1	0	1	v	v^2
v	0	v	v^2	1
v^2	0	v^2	1	v

$\times$	00	11	01	10
00	00	00	00	00
11	00	11	01	10
01	00	01	10	11
10	00	10	11	01

To construct a shared multiplication in GF(4), each of the 4 inputs a, b, c and d and the results e and f are divided into three shares:

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

After expanding the multiplication formulae, each term of the two component functions is placed into one of the three output shares (see App. A). Since the multiplication in GF(4) consists only of quadratic terms it is always possible to fulfill Property 2.

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 1 constant, 4 linear and 6 quadratic terms, 4 terms of degree 3 and 1 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 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 6 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 and we give an example for found correction terms in App. A.

5 Noekeon

Noekeon [6] 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 5 simple round transformations are applied. The cipher is completely linear except for the non-linear S-box Gamma. The linear parts can be protected against first-order DPA using one mask (two shares), whereas for the non-linear part this is not possible. In this section we show how the non-linear S-box Gamma can be successfully shared using 3 shares. Finally, we show that this shared function is secure in the presence of glitches by performing a simulation based on a back-annotated netlist.

5.1 The S-box Gamma

The non-linear 4-bit S-box Gamma is defined by Table 1) and consists of two equal non-linear layers $NL(x)$, separated by a linear layer $L(x)$:

$$S(x) = NL(L(NL(x)) \tag{10}$$

The non-linear 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:

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

Table 1. 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

5.2 Sharing the Noekeon S-box using 3 Shares

Since the algebraic degree of this function is 3, the whole function cannot be shared using 3 shares. However, if we split Gamma into two stages with algebraic degree two, we can share it using 3 shares again. We split Gamma after the linear layer and combine the first non-linear layer with the linear layer to get $y = L(NL(x))$ and $z = NL(y)$. This results in less complex formulae and we get for the ANF of the resulting 8 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 4 inputs and outputs of each layer and get 24 shared Boolean functions. We place the terms depending on their index into the regarding output share which results already in uniform shared functions for both stages of Gamma. The formulae for the two steps of the shared Noekeon S-box using three shares are shown in App. B. We have implemented both the protected and the unprotected Noekeon S-box using a $0.35\mu m$ standard cell library [3]. A schematic of the shared Noekeon S-box is shown in Fig. 1. In a straight forward implementation using just the ANF of the functions, the 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 overall size of the cipher can still be less than 3.5 times larger. This shows that shared implementations can already compete with other hardware countermeasures.

5.3 Simulation based on the Transition Count Model

Any shared implementation of these two stages of the S-box with registers in between is secure even in the presence of glitches. Further, we do not have any timing constraints or the need for balanced wires. The resulting shared component functions can be implemented on any hardware and optimized in any form, as long as there is a glitch resistant layer in between and the functionality stays the same. In order to study the resistance of this implementation in the presence of glitches, we have synthesized the circuit and performed a simulation of an

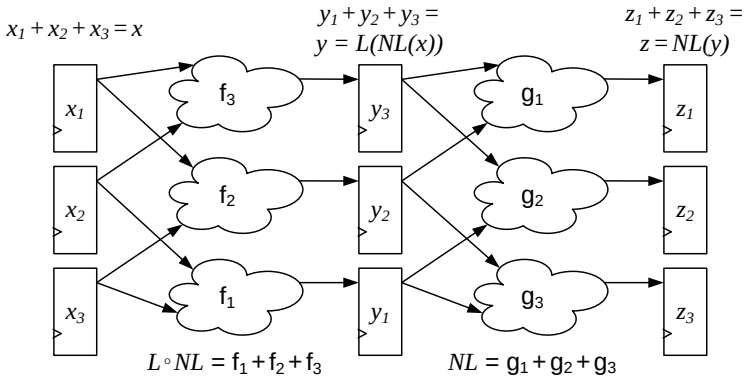

Fig. 1. A schematic of the shared Noekeon S-box using three shares.

attack using the transition count model. We have used a back-annotated netlist to derive the timing delays. Note that this is only one example of an implementation. However, by computing the correlation coefficient between the unmasked values and the number of transitions, we can show that the implementation with registers in between is secure in this model, whereas the implementation without registers between the two stages is not.

The shared S-box has 12 inputs and outputs. To verify if a circuit is secure we count the total number of transitions for each possible input transition. Every input can perform one out of 4 transitions, $0 \to 0, 0 \to 1, 1 \to 0, 1 \to 1$. Thus, we need to simulate $4^{12} = 16.777.216$ transitions. We do not need to simulate different arrival times since glitches occur due to the internal gate delays. Figure 2 shows the correlation between the number of transitions and the Hamming Distance (HD) of the unmasked values before and after the input transition. The correlation for both implementations with and without registers between the two stages is shown. These results demonstrate the DPA resistance of a shared implementation using three shares in the presence of glitches. However, the final confirmation for the security of this method can only be provided by an on-chip implementation which is clearly out of scope of this paper.

6 Conclusion

In the side-channel resistant implementation method proposed in [19] the number of shares grows with the complexity of the function to be protected. Consequently, also the number of gates required increases. In this paper we have analyzed which basic non-linear 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.

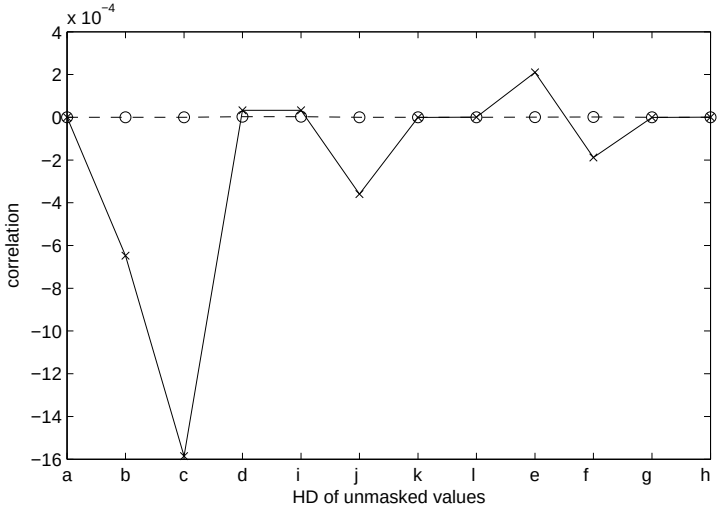

Fig. 2. The correlation of the computer simulated attack on the two implementations of the Noekeon S-box. The solid line shows the result for the S-box without registers, and the dashed line for the S-box with registers in between the two stages.

In this work we have shown that it is possible to implement cryptographic functions using much less hardware requirements than proposed in [19]. By varying the number of shares between non-linear and linear layers, the hardware requirements for a full cipher can even be further reduced. This makes the shared implementation method more competitive with other countermeasures while maintaining its high security level. Future work is to further reduce the hardware requirements and securely implement more complex non-linear functions such as the AES or ARIA S-boxes [11], which is still a mathematically challenging task. A reasonable trade-off between the number of shares and pipelining stages needs to be found. Further analysis is required to investigate the practical resistance of shared implementations against higher-order and template attacks.

References

1. Mehdi-Laurent Akkar, Régis Bevan, and Louis Goubin. Two Power Analysis Attacks against One-Mask Methods. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *LNCS*, pages 332–347. Springer, 2004.
2. Mehdi-Laurent Akkar and Christophe Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES*, volume 2162 of *LNCS*, pages 309–318. Springer, 2001.

3. Austria Microsystems. Standard Cell Library $0.35\mu m$ CMOS (C35). Available online at `http://asic.austriamicrosystems.com/databooks/c35/databook_c35_33`.
4. Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably Secure Masking of AES. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *LNCS*, pages 69–83. Springer, 2004.
5. David Canright. A Very Compact S-Box for AES. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *LNCS*, pages 441–455. Springer, 2005.
6. Joan Daemen, Michael Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: NOEKEON. Submitted as an NESSIE Candidate Algorithm, 2000. `http://www.cryptonessie.org`.
7. Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. Submitted as an AES Candidate Algorithm. Submitted as an AES Candidate Algorithm, 2000. `http://www.nist.gov/aes`.
8. Wieland Fischer and Berndt M. Gammel. Masking at Gate Level in the Presence of Glitches. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *LNCS*, pages 187–200. Springer, 2005.
9. Jovan Dj. Golic and Christophe Tymen. Multiplicative Masking and Power Analysis of AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *LNCS*, pages 198–212. Springer, 2002.
10. Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.
11. ChangKyun Kim, Martin Schläffer, and SangJae Moon. Differential Side Channel Analysis Attacks on FPGA Implementations of ARIA. *Electronics and Telecommunications Research Institute (ETRI) in Daejeon, South Korea*, 30(2):315–325, April 2008.
12. Mario Kirschbaum and Thomas Popp. Evaluation of Power Estimation Methods Based on Logic Simulations. In Karl Christian Posch and Johannes Wolkerstorfer, editors, *Proceedings of Austrochip 2007, October 11, 2007, Graz, Austria*, pages 45–51. Verlag der Technischen Universität Graz, October 2007. ISBN 978-3-902465-87-0.
13. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
14. Daesung Kwon, Jaesung Kim, Sangwoo Park, Soo Hak Sung, Yaekwon Sohn, Jung Hwan Song, Yongjin Yeom, E-Joong Yoon, Sangjin Lee, Jaewon Lee, Seongtaek Chee, Daewan Han, and Jin Hong. New Block Cipher: ARIA. In Jong In Lim and Dong Hoon Lee, editors, *ICISC*, volume 2971 of *LNCS*, pages 432–445. Springer, 2003.
15. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *LNCS*, pages 351–365. Springer, 2005.
16. Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *LNCS*, pages 157–171. Springer, 2005.
17. Stefan Mangard and Kai Schramm. Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *LNCS*, pages 76–90. Springer, 2006.

18. Thomas S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In Bruce Schneier, editor, *FSE*, volume 1978 of *LNCS*, pages 150–164. Springer, 2000.
19. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.
20. Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-Channel Analysis Resistant Description of the AES S-Box. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *LNCS*, pages 413–423. Springer, 2005.
21. Thomas Popp and Stefan Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *LNCS*, pages 172–186. Springer, 2005.
22. Jan M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.
23. Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 127–143. Springer, 2008.
24. Kai Schramm and Christof Paar. Higher Order Masking of the AES. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *LNCS*, pages 208–225. Springer, 2006.
25. Daisuke Suzuki, Minoru Saeki, and Tetsuya Ichikawa. DPA Leakage Models for CMOS Logic Circuits. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *LNCS*, pages 366–382. Springer, 2005.
26. Kris Tiri and Ingrid Verbauwhede. Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2779 of *LNCS*, pages 125–136. Springer, 2003.
27. Kris Tiri and Ingrid Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE*, pages 246–251. IEEE Computer Society, 2004.
28. Elena Trichina, Tymur Korkishko, and Kyung-Hee Lee. Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results. In Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors, *AES Conference*, volume 3373 of *LNCS*, pages 113–127. Springer, 2004.
29. Elena Trichina, Domenico De Seta, and Lucia Germani. Simplified Adaptive Multiplicative Masking for AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *LNCS*, pages 187–197. Springer, 2002.
30. Jason Waddle and David Wagner. Towards efficient second-order power analysis. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *LNCS*, pages 1–15. Springer, 2004.

A Formulas for the Multiplication in GF(4)

An example of the formulae in ANF for the shared Multiplication in GF(4) using 3 shares together with the correction terms e'_1, e'_2, e'_3 and f'_1, f'_2, f'_3:

$$\begin{aligned} e_1 = {} & a_2d_2 + a_2d_3 + a_3d_2 + \\ & b_2c_2 + b_2c_3 + b_3c_2 + \\ & b_2d_2 + b_2d_3 + b_3d_2 \end{aligned} \qquad \begin{aligned} f_1 = {} & a_2c_2 + a_2c_3 + a_3c_2 + \\ & a_2d_2 + a_2d_3 + a_3d_2 + \\ & b_2c_2 + b_2c_3 + b_3c_2 \end{aligned}$$

$$\begin{aligned} e_2 = {} & a_1d_3 + a_3d_1 + a_3d_3 + \\ & b_1c_3 + b_3c_1 + b_3c_3 + \\ & b_1d_3 + b_3d_1 + b_3d_3 \end{aligned} \qquad \begin{aligned} f_2 = {} & a_1c_3 + a_3c_1 + a_3c_3 + \\ & a_1d_3 + a_3d_1 + a_3d_3 + \\ & b_1c_3 + b_3c_1 + b_3c_3 \end{aligned}$$

$$\begin{aligned} e_3 = {} & a_1d_1 + a_1d_2 + a_2d_1 + \\ & b_1c_1 + b_1c_2 + b_2c_1 + \\ & b_1d_1 + b_1d_2 + b_2d_1 \end{aligned} \qquad \begin{aligned} f_3 = {} & a_1c_1 + a_1c_2 + a_2c_1 + \\ & a_1d_1 + a_1d_2 + a_2d_1 + \\ & b_1c_1 + b_1c_2 + b_2c_1 \end{aligned}$$

$$\begin{aligned} e'_1 &= a_3 + b_2c_2 + b_3c_3 + a_2c_2 \\ e'_2 &= a_1 + a_3 + d_1 + b_1c_1 + b_3c_3 + a_1d_1 \\ e'_3 &= a_1 + d_1 + b_1c_1 + b_2c_2 + a_2c_2 + a_1d_1 \end{aligned}$$

$$\begin{aligned} f'_1 &= c_3 + d_3 + a_2c_2 + a_3c_3 + b_2d_2 + b_3d_3 \\ f'_2 &= c_3 + d_1 + d_3 + a_3c_3 + b_1d_1 + b_3d_3 \\ f'_3 &= d_1 + a_2c_2 + b_1d_1 + b_2d_2 \end{aligned}$$

B Formulas for the Noekeon S-box using 3 Shares

The formulae in ANF of the shared Noekeon S-box or non-linear function Gamma using 3 shares. The first step combines the first non-linear 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 non-linear layer:

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