

Threshold Implementations Against Side-Channel Attacks and Glitches^{*}

Svetla Nikova¹, Christian Rechberger², and Vincent Rijmen²

¹ Department Electrical Engineering, ESAT/COSIC,
Katholieke Universiteit Leuven, Belgium
`Svetla.Nikova@esat.kuleuven.be`

² Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Austria
`{Christian.Rechberger,Vincent.Rijmen}@iaik.tugraz.at`

Abstract. Implementations of cryptographic algorithms are vulnerable to side-channel attacks. Masking techniques are employed to counter side-channel attacks that are based on multiple measurements of the same operation on different data. Most currently known techniques require new random values after every nonlinear operation and they are not effective in the presence of glitches. We present a new method to protect implementations. Our method has a higher computational complexity, but requires random values only at the start, and stays effective in the presence of glitches.

Key words: Masking, secret sharing, side-channel attacks

1 Introduction

Several approaches to design circuits that counteract side-channel attacks, have been published recently. A popular approach is to make the intermediate results of the cryptographic algorithm being executed independent of the secret key. This can be done both at the algorithm level [2, 5, 10, 18] and at the gate level [12, 25]. These approaches have in common that they require the use of random values in order to *mask* the data that is being processed. A common feature of all these approaches is that in order to implement nonlinear circuits, they require the introduction of additional (*fresh*) random values. Among other reasons, these additional random values are needed in order to mask the intermediate results computed by the circuits. Without the fresh random values, these intermediate results would cause leakage of information. Additionally, it has been shown that

^{*} The work described in this paper has been supported in part by the European Commission under contract IST-2002-507932 (ECRYPT) and through the Austrian Science Fund (FWF) under grant number P16110-N04. The information in this paper is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

the occurrence of glitches can lead to side-channel information in circuits that were previously believed to be secure [15, 16]. Recently, also other ways to break gate level masking schemes have been devised [24].

In terms of software implementations, in [19], it is shown that many existing masking proposals are not secure against higher-order attack. A first proposal to secure AES software implementations against higher-order attacks is given in [22]. Gate-level solutions are proposed in [9, 20]. Our method addresses this issue at the circuit level.

In this paper we describe a new approach to mask, based on secret sharing [3, 23], threshold cryptography [8] and multi-party computation protocols [27]. The main contributions of this paper are the following. Firstly, we show circuits that resist side-channel attacks, even in the presence of glitches. Secondly, we achieve provable security against first-order side-channel attacks. Thirdly, our circuits also resist higher-order attacks that are based on a comparison of mean power consumption.

Compared to traditional masking approaches, our approach uses more random values during the setup. A related approach was presented in [6]. It requires new random values for remasking during the computation, which is costly in many environments. Furthermore, remasking can be effective only if the delays of all the circuits are fully controlled and if there are no glitches [15, 16, 18]. An alternative approach to avoid the generation of fresh random values is presented in [2]: they derive new masking values from the old ones by applying linear functions. Another related approach was presented in [11]. Also there, the effect of glitches is not considered.

We introduce notation and terminology in Section 2. In Sections 3–5 we present the main contribution of this paper: a new theory for designing implementations that are provably secure against first-order side-channel attacks, even in the presence of glitches. In Section 6 we illustrate our approach by discussing an important application: the implementation of the multiplicative inverse in the field $\text{GF}(2^m)$. We briefly discuss the resistance of our schemes against template attacks in Section 7 and conclude with suggestions for future work in Section 8.

2 Notation and terminology

We use \oplus, \bigoplus to denote addition in the field $\text{GF}(2^m)$ (XOR) and $+, \sum$ to denote addition of real numbers. A vector (x_1, x_2, \dots, x_n) is also denoted by \bar{x} , and the reduced vector $(x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ by \bar{x}_i . We denote by $\Pr(t(\bar{x}) = T)$ the probability that the variable t takes value T , i.e. the number of times that $t(\bar{x}) = T$ divided by the number of values the input of the circuit can take.

In our approach, the data is not masked by only one random value, but by two or more. Hence, during setup we need typically more random values than with traditional approaches. Our approach is inspired by methods used in secret sharing and threshold computing systems. We say that a variable x is split into

n shares x_i if

$$x = \bigoplus_{i=1}^n x_i . \quad (1)$$

We will only use (n, n) secret sharing schemes, hence all n shares are needed in order to determine x uniquely. In a *perfect* (n, n) secret sharing scheme, knowledge of up to $n - 1$ shares doesn't give any additional information on the value of x . Observe that a $(2, 2)$ secret sharing scheme corresponds to a traditional masking scheme. In a (k, t, n) *ramp scheme* [4], t honest parties are needed to recover the secret, but more than k malicious parties can already obtain information about the secret. In this paper, we use $(1, n, n)$ ramp schemes and secret sharing schemes where the conditional probability distribution $\Pr(\bar{X}|X)$ is uniform and hence:

$$\forall \bar{X} : \Pr(\bar{x} = \bar{X}) = c \Pr(x = \bigoplus_{i=1}^n X_i), \quad (2)$$

with c a normalization constant, which ensures that $\sum_{\bar{X}} \Pr(\bar{x} = \bar{X}) = 1$.

3 Basic principle

We introduce our approach to implement linear and non-linear transformations in a secure way. We point out how this idea is related to threshold cryptography and we give an example that we will use in the remainder of this paper.

3.1 Linear transformations

Consider a transformation $z = L(x)$ over $\text{GF}(2^m)$, which is linear over $\text{GF}(2)$. The easiest way to implement a linear transformation securely is to process the n shares independently. Indeed, if

$$z_i = L(x_i), \quad 0 \leq i < n, \quad (3)$$

then by definition of a linear transformation, we have

$$z = \bigoplus_{i=1}^n z_i = \bigoplus_{i=1}^n L(x_i) = L \left(\bigoplus_{i=1}^n x_i \right) = L(x) . \quad (4)$$

Linear transformations taking more inputs can be treated in the same way. For a linear transformation $z = LL(x, y, \dots)$, we take all $f_i = LL$.

Such an implementation of a linear transformation doesn't leak information that can be used in a side-channel attack, even if presence of glitches is taken into account [15, 16]. A typical property of this implementation is that each output share z_i depends only on one input share of each variable (x_i, y_i, \dots) .

3.2 Non-linear transformations

Our idea is to construct circuits for non-linear transformations having a similar property as the secure circuits for linear transformations discussed in the previous section. Intuitively, it is clear that if a share z_i doesn't depend on the value of an input share x_i, y_i, \dots , then z_i can't be correlated to x, y, \dots . Neither will the computation of z_i leak information about the value of x, y, \dots . By imposing additional constraints, we will also ensure that no correlation to the output z exists. In this section, we introduce two properties. In the next section, we will show that with functions satisfying these properties, we can construct secure circuits.

Let $z = N(x, y, \dots)$ denote a transformation over $\text{GF}(2^m)$ which is not linear over $\text{GF}(2)$. Let f_1, f_2, \dots, f_n be a set of functions satisfying the following properties:

Property 1 (Non-completeness). Every function is independent of at least one share of each of the input variables x, y, \dots

$$\begin{aligned} z_1 &= f_1(x_2, x_3, \dots, x_n, y_2, y_3, \dots, y_n, \dots) &= f_1(\bar{x}_1, \bar{y}_1, \dots) \\ z_2 &= f_2(x_1, x_3, \dots, x_n, y_1, y_3, \dots, y_n, \dots) &= f_2(\bar{x}_2, \bar{y}_2, \dots) \\ &\dots & \\ z_n &= f_n(x_1, x_2, \dots, x_{n-1}, y_1, y_2, \dots, y_{n-1}, \dots) &= f_n(\bar{x}_n, \bar{y}_n, \dots) \end{aligned} \quad (5)$$

Property 2 (Correctness). The sum of the output shares gives the desired output.

$$z = \bigoplus_{i=1}^n z_i = \bigoplus_{i=1}^n f_i(\dots) = N(x). \quad (6)$$

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

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

$$n \geq 1 + s .$$

Proof. Multiplying s factors with n shares each can be done in the following way. Collect in the first output share all terms that don't 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 $1, 2, \dots$ and $i - 1$, but not input share i . Finally, collect in output share n the terms containing the terms with input shares $1, 2, \dots$ and $n - 1$ but not input share n . Only if $n - 1 \geq s$, there are no terms left after step n . \square

It follows that we need at least 3 shares in order to implement a non-linear function. The construction used in the proof of Theorem 1 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 1. *The maximum number of shares required to implement a function N of u variables over $GF(2^m)$, equals $1 + 2^{mu}$.*

Proof. Since $\forall x \in GF(2^m) : x^{2^m} = x$, it is always possible to describe N as a multi-variate 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 N by applying the same method as in the proof of Theorem 1. Summing up the functions for each monomial, we obtain the functions for N . \square

3.3 Effects on the power consumption

By definition, knowledge of up to $n - 1$ shares of an input variable, doesn't reveal any information on this input variable. In a circuit satisfying Property 1, each share z_i of the output z is independent of at least one share of each input variable. Consequently, we have that the output shares are uncorrelated to the input variables. Such a circuit has the following advantages:

1. Each intermediate result of the computation is uncorrelated to the input variables. Hence, no additional random values are needed for masking the intermediate results of the computation.
2. Even the presence of glitches doesn't result in the leakage of information, provided that we can restrict an attacker to look at only one f_i at a time. We will discuss in Section 4.2 what we can do in case an attacker can measure the consumption of more than one f_i simultaneously.

We generalize now condition (2) in the following way:

$$\Pr(\bar{x} = \bar{X}, \bar{y} = \bar{Y}, \dots) = c \Pr(x = \bigoplus_i X_i, y = \bigoplus_i Y_i, \dots). \quad (7)$$

In words, this means that any bias present in the joint distribution of \bar{x} and \bar{y} is due to biases in the joint distribution of x and y . Under this condition, we can prove the following.

Theorem 2. *In a circuit implementing a set of functions satisfying Property 1 and Property 2, when the input satisfies (7), all the intermediate results are independent of the inputs x, y, \dots and the output z . Also the power consumption, or any other characteristic of each individual function f_i are independent of x, y, \dots and z .*

The proof is given in Appendix A.

Example 1. Consider the multiplication of two operands in a finite field with characteristic 2: $z = N(x, y) = xy$. Let the number of shares $n = 3$ and define the 3 functions f_i as follows:

$$\begin{aligned} z_1 &= f_1(x_2, x_3, y_2, y_3) = x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \\ z_2 &= f_2(x_1, x_3, y_1, y_3) = x_3y_3 \oplus x_1y_3 \oplus x_3y_1 \\ z_3 &= f_3(x_1, x_2, y_1, y_2) = x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \end{aligned} \quad (8)$$

The functions f_i satisfy Property 1. Furthermore, since

$$z_1 \oplus z_2 \oplus z_3 = (x_1 \oplus x_2 \oplus x_3)(y_1 \oplus y_2 \oplus y_3) = xy = z, \quad (9)$$

also Property 2 is satisfied and the functions f_i form a secure realization of $N(x, y)$.

Example 2. The following example illustrates why we need condition (7). Consider the linear transformation $z = L(x, y) = x \oplus y$. The realization

$$z_i = f(x_{i+1}, y_{i+1}) = x_{i+1} \oplus y_{i+1} \quad (10)$$

satisfies Properties 1 and 2. If $x = y$, then $z = \bigoplus_i z_i = 0$. Suppose now that $\Pr(\bar{x} = \bar{X}, \bar{y} = \bar{Y}) = \Pr(\bar{x} = \bar{X})$ if $\bar{X} = \bar{Y}$, and zero otherwise. With this dependency between \bar{x} and \bar{y} , we get *always* $z_i = 0, \forall i$, and each z_i is perfectly correlated to z .

3.4 Relation to multi-party computation and threshold cryptography

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 n players, each player holding an input x_i . The players want to compute a function $F(x_1, \dots, x_n) = z$ 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 $F(x_1, \dots, x_n) = (z_1, \dots, z_n)$.

A (t, n) threshold system allows n parties to do secure computations when at least t parties are honest. We equate each function f_i with a party, thus we have an (n, n) 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 an $(1, n, n)$ ramp scheme.

The functions are *corrupt* 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 2 shows we achieve perfect security against first-order attacks.

If the attacker can corrupt several functions simultaneously, then the attack is called a *higher-order attack* [13, 17]. We discuss issues that arise in this setting in Section 4.2.

4 Glitches

In this section, we first illustrate how glitches can cause leakage of information. Subsequently, we examine one traditional masking scheme and we show that our approach leads to improved security.

CMOS circuits consume very low amounts of power. Consequently, the power consumption caused by glitches is relatively large compared to the power consumption caused by the “normal” operation of CMOS circuits. Most masking schemes³ presented in the cryptographic literature don’t take the presence of glitches into account [5, 18, 25]. It has been shown that for many of these traditional masking schemes the presence of glitches or delays in logical circuits causes side-channel leakage in the power consumption [15, 16].

Firstly, consider a simple AND gate, with inputs x, y and output z . Assume now that a glitch occurs in x , or that input x becomes stable significantly later than input y . If input y equals 1, then a variation or glitch in input x will cause the AND gate to temporarily change state, because $z = x$. However, if $y = 0$, then $z = 0$ and changes at input x will not affect the output. Consequently, the power consumption caused by glitches in input x depends on the value of input y . In the next subsection, we will study the effect of glitches in a masked AND gate.

4.1 Glitches in a traditionally masked AND-gate

We consider a typical implementation of a masked AND gate [25], illustrated in Figure 1. To make the analysis easier, we assume here that XOR gates exist as basic primitives: we don’t decompose them into smaller building blocks.

The circuit takes 5 inputs: the two random masks a, b , the two masked inputs $\tilde{x} = a \oplus x, \tilde{y} = b \oplus y$, and a new random value c to mask the output $z = x \text{ AND } y$. The circuit outputs the output mask and the masked output, which is computed as follows:

$$\tilde{z} = \tilde{x}\tilde{y} \oplus (b\tilde{x} \oplus (a\tilde{y} \oplus (ab \oplus c))) . \quad (11)$$

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 leakage. For instance, $\tilde{x}\tilde{y} \oplus b \cdot \tilde{x} = y\tilde{x}$, which leaks information about y . This is one of the reasons why the new random value c 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 \tilde{x} . The propagation of this glitch will depend on the values of b and \tilde{y} . 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 b and \tilde{y} . 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 [15, 16].

We conclude that switching characteristics of logical circuits may invalidate some of the assumptions commonly made in proofs of security against side-channel attacks. A slightly frustrating aspect of the findings in [15] is that it remains unclear how to construct security proofs that do take into account the

³ In this section we use the terms “mask” and “masked gate” in order to stay close to the original description of the schemes.

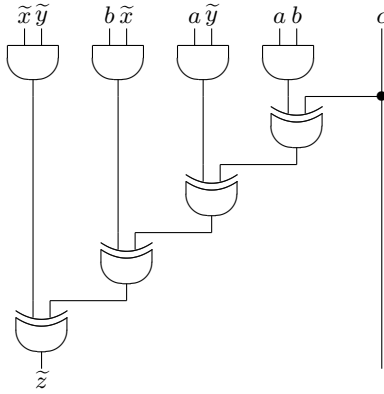


Fig. 1. Glitch propagation through a masked AND gate.

Table 1. Number of affected gates in the circuit of Figure 1, when a glitch occurs in input \tilde{x} .

$b \tilde{y}$	AND	XOR
0 0	0	0
0 1	1	1
1 0	1	2
1 1	2	2

presence of glitches. In order to avoid exhaustive analysis of all the possible combinations of signal arrival times, it seems beneficial to use a circuit that has Property 1.

4.2 Glitches in a shared AND-gate

The realization (8) can be used to implement multiplication in a finite field with characteristic two. Multiplication in $GF(2)$ corresponds to the logical AND operation. Hence, the circuit can be used as a masked AND gate in order to implement arbitrary Boolean functions.

Theorem 3. *If the distributions of the input shares \bar{x}, \bar{y} satisfy (7), then the mean power consumption of a circuit implementing realization (8) is independent of x, y and z , even in the presence of glitches or the delayed arrival of some inputs.*

Proof. Theorem 2 states that all characteristics of the circuits implementing one of the functions f_i are independent of x, y , and z . Since no assumption is made on the behavior of the circuit and/or the presence of glitches, the theorem also holds in this case. Consequently, also the mean power consumption of each individual circuit is independent of x, y, z , even in the presence of glitches. Since the mean power consumption of the whole circuit equals the sum of the mean

power consumptions of the individual functions, it is also independent of x, y, z .

□

Theorem 3 only applies to the mean power consumption of the circuit. We don't achieve indistinguishable distributions of the power consumption as demanded in [5]. Nevertheless, since the mean power consumption of the circuit is always the same, it resists the type of higher-order attacks that are based on the mean value of the addition or subtraction of the power consumption traces of the different circuits [17]. Although one can theoretically devise side-channel attacks that don't require a difference in the mean power consumption, such attacks have not been demonstrated in practice yet.

If the used logic style prevents the occurrence of glitches, then not only the mean power consumption, but also the variance are independent of the values of x, y, z . This can be shown by simply going through all possible state transitions.

5 Implementing arbitrary functions

Theorem 1 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. As a rough rule of thumb, going from 1 share to n shares will increase the number of gates with a factor n^2 . This shouldn't come as a big surprise, because introducing resistance against power attacks always comes at a price. For instance, in [20], 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 [22] 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.

Pipelining is often used to speed up hardware implementations. In order to allow large clock frequencies, combinatorial logic circuits shouldn't be many levels 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 [26] 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 a secure implementation. 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.

If the mean power consumption of each pipeline stage is constant, then also the mean of the total power consumption is constant, and the circuit is secure against first-order differential power attacks. Note that condition (7) needs now

to be fulfilled at the input of each pipeline stage in order for Theorem 3 to hold. 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 circuits satisfy an additional balance property.

Property 3 (Balance). A realization of $z = N(x, y, \dots)$ is balanced if for all distributions of the inputs x, y, \dots , and for all input share distributions satisfying (7) the conditional probability

$$\Pr(\bar{z} = \bar{Z} | z = \bigoplus_i Z_i)$$

is constant.

If the function N is invertible, then Property 3 is satisfied by invertible realizations. In an invertible realization of $z = N(x)$, every vector \bar{z} is reached for exactly one input vector \bar{x} . This condition is stricter than the requirement that every value z is reached for exactly one input x .

Example 3. The realization (8) of the multiplication in Example 1 doesn't have Property 3. In fact, there is no realization for multiplication satisfying this property with 3 shares only. The following realization with 4 shares satisfies Property 1, 2 and 3:

$$\begin{aligned} z_1 &= (x_3 \oplus x_4)(y_2 \oplus y_3) \oplus y_2 \oplus y_3 \oplus y_4 \oplus x_2 \oplus x_3 \oplus x_4 \\ z_2 &= (x_1 \oplus x_3)(y_1 \oplus y_4) \oplus y_1 \oplus y_3 \oplus y_4 \oplus x_1 \oplus x_3 \oplus x_4 \\ z_3 &= (x_2 \oplus x_4)(y_1 \oplus y_4) \oplus y_2 \oplus x_2 \\ z_4 &= (x_1 \oplus x_2)(y_2 \oplus y_3) \oplus y_1 \oplus x_1. \end{aligned} \tag{12}$$

Property 1 and 2 can be verified with pen and paper. Property 3 was verified by direct computation of all conditional probabilities.

6 Example: inversion over finite fields

Finite field inversion is an important map, for instance because of its use in the AES. As illustration, we study here inversion in $\text{GF}(16)$.

Firstly, let $\text{GF}(4)$ be represented as $\text{GF}(2)[t]/(t^2+t+1)$. Operations in $\text{GF}(4)$ then correspond to:

$$\begin{aligned} (at \oplus b) \oplus (ct \oplus d) &= (a \oplus c)t \oplus (b \oplus d) \\ (at \oplus b) \times (ct \oplus d) &= (ad \oplus bc \oplus ac)t \oplus (bd \oplus ac) \\ (at \oplus b)^{-1} &= at \oplus (a \oplus b) \\ (at \oplus b)^3 &= ab \oplus a \oplus b. \end{aligned} \tag{13}$$

Secondly, let $\text{GF}(16)$ be represented by $\text{GF}(4)[s]/(s^2 \oplus s \oplus \alpha)$. Inversion in $\text{GF}(16)$ then becomes:

$$(as \oplus b)^{-1} = a(a^2\alpha \oplus ab \oplus b^2)^{-1}s \oplus (a \oplus b)(a^2\alpha \oplus ab \oplus b^2)^{-1}.$$

Defining $c = a \oplus b$, we obtain:

$$\begin{aligned} (as \oplus (c \oplus a))^{-1} &= a(a^2\alpha \oplus ac \oplus c^2)^{-1}s \oplus c(a^2\alpha \oplus ac \oplus c^2)^{-1} \\ &= (a^2\alpha^2 \oplus a^3c^2 \oplus ac)s \oplus (aca^2 \oplus a^2c^3 \oplus c^2). \end{aligned} \quad (14)$$

Combining (13) with (14) and choosing $\alpha = t$, we obtain

$$((xt \oplus y)s \oplus (zt \oplus v))^{-1} = (ft \oplus g)s \oplus (ht \oplus k),$$

where f , g , h , and k are Boolean functions defined as follows:

$$\begin{aligned} f &= x \oplus y \oplus xv \oplus xyz \\ g &= y \oplus xv \oplus yz \oplus xyz \oplus xyv \\ h &= y \oplus xv \oplus yv \oplus xzv \\ k &= z \oplus v \oplus xz \oplus yz \oplus yv \oplus xzv \oplus yzv. \end{aligned}$$

Theorem 1 predicts we need at least 4 shares to implement these functions. Exhaustive search revealed that no realization with 4 shares can satisfy Property 3. We give a realization with 5 shares for f, g, h and k in Appendix B.

7 Considering template attacks

In the previous sections of this paper, we ignored the possibility of simple power attacks [14]. We assumed that an attacker can't use a single measurement to obtain a meaningful signal. Provided that a few basic rules are followed during the implementation, this is usually a realistic assumption, which is commonly made when discussing masking schemes. However, more sophisticated methods have been developed since then.

7.1 Template attacks

Template attacks were introduced in [7], as an extension of simple power attacks. A template attack starts with a *profiling phase*, during which the attacker has at his disposal a freely programmable device which is identical to the targeted device. This device is used to build a model (templates) of its state while performing different operations on (parts of) the secret key. Afterwards, in the *hypothesis-testing phase*, these templates are used to classify the single trace of the targeted device which in turn reduces the entropy of the key. In order to improve classification results, multivariate instead of univariate statistics is employed to yield practical classification results.

Recent advances in template attacks highlight the importance of this topic in the context of masking schemes [1]. These results are summarized as follows. Using single-bit templates even masked implementations can be broken. The new, but in many practical circumstances reasonable setting is that an attacker can get hold of a device with a biased RNG which is generating the used masks. Using templates generated from such a card, it is shown that even devices with perfectly unbiased masks can be attacked.

7.2 Resistance against template attacks

Our proposal to protect implementations does not prevent this type of attack if a biased RNG is in the hands of an attacker. However it makes it more difficult to implement the attack, because the parallel computation of the n shares lowers the signal-to-noise ratio. For example, consider a Hamming weight based leakage model. Let w denote the bit-width of a share. Ignoring noise, the relative leakage l is given by:

$$l = \frac{nw - \log_2\left(\sum_{i=0}^{nw} \frac{\binom{nw}{i}^2}{2^{n \cdot w}}\right)}{nw}. \quad (15)$$

Since we assume that all shares are uncorrelated, the number of shares n effectively multiplies the bit-width. As a consequence the number of samples needed in the profiling step is greatly increased. If this number of samples can't be taken, then the classification results will get worse. In turn, the gain for the attacker, namely the reduction of the key entropy, is less.

8 Conclusions and open problems

We presented a new method to design implementations that counteract side-channel attacks. The big advantages are that we don't need fresh random values after every nonlinear transformation and that we achieve provable security against first order attacks, even in the presence of glitches. The scheme also resists certain types of higher-order attacks. To illustrate the design method, we applied it to the computation of the multiplicative inverse over $\text{GF}(2^4)$.

Disadvantages are the increased data storage requirements due to the higher number of shares, and the corresponding increase in computational complexity.

Investigating whether other attacks are feasible and how to protect against them is proposed as topic of further research. Future work also includes the design of more complicated circuits. Clearly, implementing more complex circuits in one go, will increase the complexity of our circuits dramatically. We propose to employ techniques from proactive secret sharing schemes [21] in order to reduce the circuit complexity. This has the added benefit to limit an attacker's possibilities even further, but has as disadvantage that now fresh random values are required.

References

1. Dakshi Agrawal, Josyula R. Rao, Pankaj Rohatgi, Kai Schramm, "Templates as Master Keys", J.R. Rao, B. Sunar, Eds., CHES 2005, LNCS 3659, Springer-Verlag, 2005, pp. 15–29.
2. Mehdi-Laurent Akkar, Christophe Giraud, "An implementation of DES and AES, secure against some attacks", Ç Koç, D. Naccache, Ch. Paar, Eds., CHES 2001, LNCS 2162, Springer-Verlag, 2001, pp. 309–318.
3. George Blakley, Safeguarding cryptographic keys, *AFIPS* 48, 1979, pp. 313-317.

4. George Blakley, Catherine Meadows, "Security of ramp schemes", CRYPTO '84, LNCS 196, Springer-Verlag, 1984, pp. 242–268.
5. Johannes Blömer, Jorge Guajardo Merchan, Volker Krümmel, "Provably Secure Masking of AES", H. Handschuh, M. Anwar Hasan, Eds., Selected Areas in Cryptography (SAC 2004), LNCS 3357, Springer-Verlag, 2004, pp. 69–83.
6. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, Pankaj Rohatgi, "Towards Sound Approaches to Counteract Power- Analysis Attacks" CRYPTO '99, LNCS 1666, Springer-Verlag, 1999, pp. 398–412.
7. Suresh Chari, Josyula R. Rao, Pankaj Rohatgi, "Template Attacks", B. Kaliski, Ç Koç, Ch. Paar, Eds., CHES 2002, LNCS 2523, Springer-Verlag, 2003, pp. 13–28.
8. Yvo Desmedt, "Some recent research aspects of threshold cryptography", E. Okamoto, G. Davida, M. Mambo, Eds., Information Security, LNCS 1396, Springer-Verlag, 1997, pp. 158–173.
9. Wieland Fischer, Berndt M. Gammel, "Masking at Gate Level in the Presence of Glitches," J.R. Rao, B. Sunar, Eds., CHES 2005, LNCS 3659, Springer-Verlag, 2005, pp. 187–200.
10. Jovan D. Golić, Christophe Tymen, "Multiplicative masking and power analysis", B. Kaliski, Ç Koç, Ch. Paar, Eds., CHES 2002, LNCS 2523, Springer-Verlag, 2003, pp. 198–212.
11. Louis Goubin, Jacques Patarin, "DES and differential power analysis – the duplication method", Ç. Koç, Ch. Paar, Eds., CHES '99, LNCS 1717, Springer-Verlag, 1999, pp. 158–172.
12. Yuval Ishai, Amit Sahai, David Wagner, "Private circuits: securing hardware against probing attacks", D. Boneh, Ed., CRYPTO 2003, LNCS 2729, Springer-Verlag, 2003, pp. 463–481.
13. Marc Joye, Pascal Paillier, Berry Schoenmakers, "On second-order differential power analysis," J.R. Rao, B. Sunar, Eds., CHES 2005, LNCS 3659, Springer-Verlag, 2005, pp. 293–308.
14. Paul Kocher, Joshua Jaffe, Benjamin Jun, "Differential Power Analysis", M. Wiener, Ed., CRYPTO '99, LNCS 1666, Springer-Verlag, 1999, pp. 388–397.
15. Stefan Mangard, Thomas Popp, Berndt M. Gammel, "Side-channel leakage of masked CMOS gates", A.J. Menezes, Ed., CT-RSA 2005, LNCS 3376, Springer-Verlag, 2005, pp. 351–365.
16. Stefan Mangard, Norbert Pramstaller, Elisabeth Oswald, "Successfully attacking masked AES hardware implementations," J.R. Rao, B. Sunar, Eds., CHES 2005, LNCS 3659, Springer-Verlag, 2005, pp. 157–171.
17. Thomas S. Messerges, "Using second-order power analysis to attack DPA resistant software," Ç.K. Koç, Ch. Paar, Eds., CHES 2000, LNCS 1965, Springer-Verlag, 2000, pp. 238–251.
18. Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, Vincent Rijmen, "A Side-Channel Analysis Resistant Description of the AES S-box", FSE 2005, LNCS 3557, Springer-Verlag, pp. 413–423.
19. Elisabeth Oswald, Stefan Mangard, Christoph Herbst, Stefan Tillich, "Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers", CT-RSA 2006, LNCS 3860, Springer-Verlag, pp. 192–207.
20. Thomas Popp, Stefan Mangard, "Masked dual-rail pre-charge logic: DPA-resistance without routing constraints," J.R. Rao, B. Sunar, Eds., CHES 2005, LNCS 3659, Springer-Verlag, 2005, pp. 172–186.
21. Rafail Ostrovsky, Moti Yung, "How to withstand mobile virus attacks", Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC), 1991, pp. 51–59.

22. Kai Schramm, Christof Paar, “Higher Order Masking of the AES”, CT-RSA 2006, LNCS 3860, Springer-Verlag, pp. 208–225.
23. Adi Shamir, “How to share a secret”, *Commun. ACM* 22, 1979, pp. 612-613.
24. Kris Tiri, Patrick Schaumont, “Changing the Odds against Masked Logic”, Selected Areas of Cryptography 2006 (SAC), LNCS, Springer-Verlag, to appear.
25. Elena Trichina, Tymur Korkishko, “Small size, low power, side channel immune AES coprocessor design and synthesis results”, Proc. fourth conference on the Advanced Encryption Standard (AES4), LNCS 3373, Springer-Verlag, 2005, pp. 113–127.
26. Johannes Wolkerstorfer, Elisabeth Oswald, Mario Lamberger, “An ASIC implementation of the AES S-boxes”, B. Preneel, Ed., CT-RSA 2002, LNCS 2271, Springer-Verlag, 2002, pp. 67–78.
27. Andrew Yao, “Protocols for secure computation”, *FOCS’82*, 1982, pp. 160-164.

A Proof of Theorem 2

For sake of readability, we give the proof for the case of two input variables. The general case follows straightforwardly.

Let $t(\bar{x}_1, \bar{y}_1)$ be any intermediate result or any physical characteristic of the circuit implementing f_1 . The only assumption we make about t is that it doesn’t depend on the values of x_1 and y_1 . Let $\delta(X, Y)$ be the function that is equal to 1 if $X = Y$ and 0 otherwise. Let A, A_1 denote the following sets:

$$A = \{(\bar{X}, \bar{Y}) \mid t(\bar{X}_1, \bar{Y}_1) = T\}$$

$$A_1 = \{(\bar{X}_1, \bar{Y}_1) \mid t(\bar{X}_1, \bar{Y}_1) = T\} .$$

and let B denote the set of possible values for (x_1, y_1) . Since t doesn’t depend on the values of x_1 and y_1 , we have $A = B \times A_1$. By definition and using (2):

$$\Pr(t = T) = \sum_{\bar{X} \in A} \Pr(\bar{x} = \bar{X}) = \sum_{(\bar{X}, \bar{Y}) \in A} c \Pr(x = \bigoplus_i X_i, y = \bigoplus_i Y_i)$$

Splitting up the summation results in

$$\Pr(t = T) = \sum_{(\bar{X}_1, \bar{Y}_1) \in A_1} c \sum_{(X_1, Y_1) \in B} \Pr(x = \bigoplus_i X_i, y = \bigoplus_i Y_i) .$$

Since the shares $X_2, \dots, X_n, Y_2, \dots, Y_n$ give no information on X, Y , the latter summation equals 1 and hence:

$$\Pr(t = T) = \sum_{(\bar{X}_1, \bar{Y}_1) \in A_1} c \tag{16}$$

Similarly, we obtain:

$$\begin{aligned} \Pr(t = T, x = X, y = Y) &= \sum_{(\bar{X}_1, \bar{Y}_1) \in A_1} \Pr(\bar{x} = (X \oplus \bigoplus_{i=2}^n X_i, \bar{X}_1), \bar{y} = (Y \oplus \bigoplus_{i=2}^n Y_i, \bar{Y}_1)) \\ &= \sum_{(\bar{X}_1, \bar{Y}_1) \in A_1} c \Pr(x = X, y = Y) \end{aligned}$$

Using (16), we obtain:

$$\Pr(t = T, x = X, y = Y) = \Pr(t = T) \Pr(x = X, y = Y).$$

Hence t is independent of x and y . Let $C(Z)$ denote the set

$$C(Z) = \{(X, Y) \mid N(X, Y) = Z\} = \{(X, Y) \mid \delta(N(X, Y), Z) = 1\}.$$

Then, we can write for z :

$$\Pr(z = Z) = \sum_{(X, Y) \in C(Z)} \Pr(x = X, y = Y).$$

$$\begin{aligned} \Pr(t = T, z = Z) &= \sum_{(\bar{X}, \bar{Y}) \in A} \delta(N(\bigoplus_i X_i, \bigoplus_i Y_i), Z) \Pr(\bar{x} = \bar{X}, \bar{y} = \bar{Y}) \\ &= \sum_{(\bar{X}_1, \bar{Y}_1) \in A_1} \sum_{(X_1, Y_1) \in B} \delta(N(\bigoplus_i X_i, \bigoplus_i Y_i), Z) c \Pr(x = \bigoplus_i X_i, y = \bigoplus_i Y_i). \end{aligned}$$

Since the shares $X_2, \dots, X_n, Y_2, \dots, Y_n$ give no information on X, Y , we can rewrite this as follows:

$$\begin{aligned} \Pr(t = T, z = Z) &= \sum_{(\bar{X}_1, \bar{Y}_1) \in A_1} c \sum_{(X, Y) \in C(Z)} \Pr(x = X, y = Y) \\ &= \Pr(t = T) \Pr(z = Z). \end{aligned}$$

B Realization of inversion in GF(16) with 5 shares for f, g, h, k

$$\begin{aligned} f_1 &= x_2 \oplus y_2 \oplus (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(v_2 \oplus v_3 \oplus v_4 \oplus v_5) \\ &\quad \oplus (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(y_2 \oplus y_3 \oplus y_4 \oplus y_5)(z_2 \oplus z_3 \oplus z_4 \oplus z_5) \\ f_2 &= x_3 \oplus y_3 \oplus x_1(v_3 \oplus v_4 \oplus v_5) \oplus v_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1 v_1 \\ &\quad \oplus x_1(y_3 \oplus y_4 \oplus y_5)(z_3 \oplus z_4 \oplus z_5) \oplus y_1(x_3 \oplus x_4 \oplus x_5)(z_3 \oplus z_4 \oplus z_5) \\ &\quad \oplus z_1(x_3 \oplus x_4 \oplus x_5)(y_3 \oplus y_4 \oplus y_5) \oplus x_1 y_1(z_3 \oplus z_4 \oplus z_5) \oplus x_1 z_1(y_3 \oplus y_4 \oplus y_5) \\ &\quad \oplus y_1 z_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1 y_1 z_1 \\ f_3 &= x_4 \oplus y_4 \oplus x_2 v_1 \oplus x_1 v_2 \oplus x_1 y_1 z_2 \oplus x_1 y_2 z_1 \oplus x_2 y_1 z_1 \oplus x_1 y_2 z_2 \oplus x_2 y_1 z_2 \oplus x_2 y_2 z_1 \\ &\quad \oplus x_1 y_2 z_4 \oplus x_2 y_1 z_4 \oplus x_1 y_4 z_2 \oplus x_2 y_4 z_1 \oplus x_4 y_1 z_2 \oplus x_4 y_2 z_1 \oplus x_1 y_2 z_5 \oplus x_2 y_1 z_5 \\ &\quad \oplus x_1 y_5 z_2 \oplus x_2 y_5 z_1 \oplus x_5 y_1 z_2 \oplus x_5 y_2 z_1 \\ f_4 &= x_5 \oplus y_5 \oplus x_1 y_2 z_3 \oplus x_1 y_3 z_2 \oplus x_2 y_1 z_3 \oplus x_2 y_3 z_1 \oplus x_3 y_1 z_2 \oplus x_3 y_2 z_1 \\ f_5 &= x_1 \oplus y_1 \end{aligned}$$

$$\begin{aligned}
g_1 &= (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(y_2 \oplus y_3 \oplus y_4 \oplus y_5)(z_2 \oplus z_3 \oplus z_4 \oplus z_5) \\
&\oplus (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(y_2 \oplus y_3 \oplus y_4 \oplus y_5)(v_2 \oplus v_3 \oplus v_4 \oplus v_5) \\
&\oplus (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(v_2 \oplus v_3 \oplus v_4 \oplus v_5) \\
&\oplus (y_2 \oplus y_3 \oplus y_4 \oplus y_5)(z_2 \oplus z_3 \oplus z_4 \oplus z_5) \oplus y_2 \\
g_2 &= x_1(y_3 \oplus y_4 \oplus y_5)(z_3 \oplus z_4 \oplus z_5) \oplus y_1(x_3 \oplus x_4 \oplus x_5)(z_3 \oplus z_4 \oplus z_5) \\
&\oplus z_1(x_3 \oplus x_4 \oplus x_5)(y_3 \oplus y_4 \oplus y_5) \oplus x_1y_1(z_3 \oplus z_4 \oplus z_5) \oplus x_1z_1(y_3 \oplus y_4 \oplus y_5) \\
&\oplus y_1z_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1y_1z_1 \oplus x_1(y_3 \oplus y_4 \oplus y_5)(v_3 \oplus v_4 \oplus v_5) \\
&\oplus y_1(x_3 \oplus x_4 \oplus x_5)(v_3 \oplus v_4 \oplus v_5) \oplus v_1(x_3 \oplus x_4 \oplus x_5)(y_3 \oplus y_4 \oplus y_5) \\
&\oplus x_1y_1(v_3 \oplus v_4 \oplus v_5) \oplus x_1v_1(y_3 \oplus y_4 \oplus y_5) \oplus y_1v_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1y_1v_1 \\
&\oplus x_1(v_3 \oplus v_4 \oplus v_5) \oplus v_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1v_1 \oplus y_1(z_3 \oplus z_4 \oplus z_5) \\
&\oplus z_1(y_3 \oplus y_4 \oplus y_5) \oplus y_1z_1 \oplus y_3 \\
g_3 &= x_1y_1z_2 \oplus x_1y_2z_1 \oplus x_2y_1z_1 \oplus x_1y_2z_2 \oplus x_2y_1z_2 \oplus x_2y_2z_1 \oplus x_1y_2z_4 \oplus x_2y_1z_4 \\
&\oplus x_1y_4z_2 \oplus x_2y_4z_1 \oplus x_4y_1z_2 \oplus x_4y_2z_1 \oplus x_1y_2z_5 \oplus x_2y_1z_5 \oplus x_1y_5z_2 \oplus x_2y_5z_1 \\
&\oplus x_5y_1z_2 \oplus x_5y_2z_1 \oplus x_1y_1v_2 \oplus x_1y_2v_1 \oplus x_2y_1v_1 \oplus x_1y_2v_2 \oplus x_2y_1v_2 \oplus x_2y_2v_1 \\
&\oplus x_1y_2v_4 \oplus x_2y_1v_4 \oplus x_1y_4v_2 \oplus x_2y_4v_1 \oplus x_4y_1v_2 \oplus x_4y_2v_1 \oplus x_1y_2v_5 \oplus x_2y_1v_5 \\
&\oplus x_1y_5v_2 \oplus x_2y_5v_1 \oplus x_5y_1v_2 \oplus x_5y_2v_1 \oplus x_2v_1 \oplus x_1v_2 \oplus y_2z_1 \oplus y_1z_2 \oplus y_4 \\
g_4 &= x_1y_2z_3 \oplus x_1y_3z_2 \oplus x_2y_1z_3 \oplus x_2y_3z_1 \oplus x_3y_1z_2 \oplus x_3y_2z_1 \oplus x_1y_2v_3 \oplus x_1y_3v_2 \\
&\oplus x_2y_1v_3 \oplus x_2y_3v_1 \oplus x_3y_1v_2 \oplus x_3y_2v_1 \oplus y_5 \\
g_5 &= y_1
\end{aligned}$$

$$\begin{aligned}
h_1 &= (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(v_2 \oplus v_3 \oplus v_4 \oplus v_5)(z_2 \oplus z_3 \oplus z_4 \oplus z_5) \oplus y_3 \oplus v_2 \\
&\oplus (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(v_2 \oplus v_3 \oplus v_4 \oplus v_5) \\
&\oplus (y_2 \oplus y_3 \oplus y_4 \oplus y_5)(v_2 \oplus v_3 \oplus v_4 \oplus v_5) \\
h_2 &= x_1(v_3 \oplus v_4 \oplus v_5)(z_3 \oplus z_4 \oplus z_5) \oplus v_1(x_3 \oplus x_4 \oplus x_5)(z_3 \oplus z_4 \oplus z_5) \\
&\oplus z_1(x_3 \oplus x_4 \oplus x_5)(v_3 \oplus v_4 \oplus v_5) \oplus x_1v_1(z_3 \oplus z_4 \oplus z_5) \oplus x_1z_1(v_3 \oplus v_4 \oplus v_5) \\
&\oplus v_1z_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1v_1z_1 \oplus x_1(v_3 \oplus v_4 \oplus v_5) \oplus v_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1v_1 \\
&\oplus y_1(v_3 \oplus v_4 \oplus v_5) \oplus v_1(y_3 \oplus y_4 \oplus y_5) \oplus y_1v_1 \oplus y_4 \oplus v_1 \oplus v_5 \\
h_3 &= x_1v_1z_2 \oplus x_1v_2z_1 \oplus x_2v_1z_1 \oplus x_1v_2z_2 \oplus x_2v_1z_2 \oplus x_2v_2z_1 \oplus x_1v_2z_4 \oplus x_2v_1z_4 \\
&\oplus x_1v_4z_2 \oplus x_2v_4z_1 \oplus x_4v_1z_2 \oplus x_4v_2z_1 \oplus x_1v_2z_5 \oplus x_2v_1z_5 \oplus x_1v_5z_2 \oplus x_2v_5z_1 \\
&\oplus x_5v_1z_2 \oplus x_5v_2z_1 \oplus x_2v_1 \oplus x_1v_2 \oplus y_2v_1 \oplus y_1v_2 \oplus y_5 \oplus v_1 \oplus v_2 \oplus v_5 \\
h_4 &= x_1v_2z_3 \oplus x_1v_3z_2 \oplus x_2v_1z_3 \oplus x_2v_3z_1 \oplus x_3v_1z_2 \oplus x_3v_2z_1 \oplus y_1 \oplus v_1 \\
h_5 &= y_2 \oplus v_1
\end{aligned}$$

$$\begin{aligned}
k_1 &= (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(v_2 \oplus v_3 \oplus v_4 \oplus v_5)(z_2 \oplus z_3 \oplus z_4 \oplus z_5) \\
&\quad \oplus (v_2 \oplus v_3 \oplus v_4 \oplus v_5)(y_2 \oplus y_3 \oplus y_4 \oplus y_5)(z_2 \oplus z_3 \oplus z_4 \oplus z_5) \\
&\quad \oplus (x_2 \oplus x_3 \oplus x_4 \oplus x_5)(z_2 \oplus z_3 \oplus z_4 \oplus z_5) \\
&\quad \oplus (y_2 \oplus y_3 \oplus y_4 \oplus y_5)(z_2 \oplus z_3 \oplus z_4 \oplus z_5) \\
&\quad \oplus (y_2 \oplus y_3 \oplus y_4 \oplus y_5)(v_2 \oplus v_3 \oplus v_4 \oplus v_5) \oplus z_2 \oplus v_2 \\
k_2 &= x_1(v_3 \oplus v_4 \oplus v_5)(z_3 \oplus z_4 \oplus z_5) \oplus v_1(x_3 \oplus x_4 \oplus x_5)(z_3 \oplus z_4 \oplus z_5) \\
&\quad \oplus z_1(x_3 \oplus x_4 \oplus x_5)(v_3 \oplus v_4 \oplus v_5) \oplus x_1v_1(z_3 \oplus z_4 \oplus z_5) \oplus x_1z_1(v_3 \oplus v_4 \oplus v_5) \\
&\quad \oplus v_1z_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1v_1z_1 \oplus v_1(y_3 \oplus y_4 \oplus y_5)(z_3 \oplus z_4 \oplus z_5) \\
&\quad \oplus y_1(v_3 \oplus v_4 \oplus v_5)(z_3 \oplus z_4 \oplus z_5) \oplus z_1(v_3 \oplus v_4 \oplus v_5)(y_3 \oplus y_4 \oplus y_5) \\
&\quad \oplus v_1y_1(z_3 \oplus z_4 \oplus z_5) \oplus v_1z_1(y_3 \oplus y_4 \oplus y_5) \oplus y_1z_1(v_3 \oplus v_4 \oplus v_5) \oplus v_1y_1z_1 \\
&\quad \oplus x_1(z_3 \oplus z_4 \oplus z_5) \oplus z_1(x_3 \oplus x_4 \oplus x_5) \oplus x_1z_1 \oplus y_1(z_3 \oplus z_4 \oplus z_5) \\
&\quad \oplus z_1(y_3 \oplus y_4 \oplus y_5) \oplus y_1z_1 \oplus y_1(v_3 \oplus v_4 \oplus v_5) \oplus v_1(y_3 \oplus y_4 \oplus y_5) \oplus y_1v_1 \\
&\quad \oplus z_3 \oplus v_3 \\
k_3 &= x_1v_1z_2 \oplus x_1v_2z_1 \oplus x_2v_1z_1 \oplus x_1v_2z_2 \oplus x_2v_1z_2 \oplus x_2v_2z_1 \oplus x_1v_2z_4 \oplus x_2v_1z_4 \\
&\quad \oplus x_1v_4z_2 \oplus x_2v_4z_1 \oplus x_4v_1z_2 \oplus x_4v_2z_1 \oplus x_1v_2z_5 \oplus x_2v_1z_5 \oplus x_1v_5z_2 \oplus x_2v_5z_1 \\
&\quad \oplus x_5v_1z_2 \oplus x_5v_2z_1 \oplus v_1y_1z_2 \oplus v_1y_2z_1 \oplus v_2y_1z_1 \oplus v_1y_2z_2 \oplus v_2y_1z_2 \oplus v_2y_2z_1 \\
&\quad \oplus v_1y_2z_4 \oplus v_2y_1z_4 \oplus v_1y_4z_2 \oplus v_2y_4z_1 \oplus v_4y_1z_2 \oplus v_4y_2z_1 \oplus v_1y_2z_5 \oplus v_2y_1z_5 \\
&\quad \oplus v_1y_5z_2 \oplus v_2y_5z_1 \oplus v_5y_1z_2 \oplus v_5y_2z_1 \oplus x_2z_1 \oplus x_1z_2 \oplus y_2z_1 \oplus y_1z_2 \oplus y_2v_1 \\
&\quad \oplus y_1v_2 \oplus z_4 \oplus v_4 \\
k_4 &= x_1v_2z_3 \oplus x_1v_3z_2 \oplus x_2v_1z_3 \oplus x_2v_3z_1 \oplus x_3v_1z_2 \oplus x_3v_2z_1 \oplus v_1y_2z_3 \oplus v_1y_3z_2 \\
&\quad \oplus v_2y_1z_3 \oplus v_2y_3z_1 \oplus v_3y_1z_2 \oplus v_3y_2z_1 \oplus z_5 \oplus v_5 \\
k_5 &= z_1 \oplus v_1
\end{aligned}$$