A Side-Channel Analysis Resistant Description of the AES S-Box*

Elisabeth Oswald[1], Stefan Mangard[1], Norbert Pramstaller[1], and Vincent Rijmen[1,2]

[1] Institute for Applied Information Processing and Communciations (IAIK), TU Graz, Inffeldgasse 16a, A–8010 Graz, Austria
[2] Cryptomathic A/S Jægergårdsgade 118, DK-8000 Århus C, Denmark
{elisabeth.oswald, stefan.mangard, norbert.pramstaller, vincent.rijmen}@iaik.tugraz.at

Abstract. So far, efficient algorithmic countermeasures to secure the AES algorithm against (first-order) differential side-channel attacks have been very expensive to implement. In this article, we introduce a new masking countermeasure which is not only secure against first-order side-channel attacks, but which also leads to relatively small implementations compared to other masking schemes implemented in dedicated hardware.

Our approach is based on shifting the computation of the finite field inversion in the AES S-box down to $GF(4)$. In this field, the inversion is a linear operation and therefore it is easy to mask.

Summarizing, the new masking scheme combines the concepts of multiplicative and additive masking in such a way that security against first-order side-channel attacks is maintained, and that small implementations in dedicated hardware can be achieved.

Keywords: AES, side-channel analysis, masking schemes.

1 Introduction

Securing small hardware implementations of block ciphers against differential side-channel attacks [8] has proven to be a challenging task. Hardware countermeasures, which are based on special leakage-resistant logic styles, typically lead to a significant increase of area and power consumption [14]. Algorithmic countermeasures also lead to a significant increase of area, if implemented in hardware. Nevertheless, algorithmic countermeasures can be tailored towards a particular algorithm, and hence, they can be optimized to a certain extent.

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT and by the FWF "Investigations of Simple and Differential Power Analysis" project (P16110).

H. Gilbert and H. Handschuh (Eds.): FSE 2005, LNCS 3557, pp. 413–423, 2005.

In case of the AES algorithm, several algorithmic countermeasures have been proposed [2], [6], and [13]. They are all based on masking, *i.e.*, the addition of a random value (the mask) to the intermediate AES values. However two of them, [2] and [13], are both susceptible to a certain type of (first-order) differential side-channel attack, the zero-value attack. The latter one has turned out to be vulnerable even to standard differential side-channel attacks as well [1]. The countermeasure presented in [6] is not suitable for hardware implementations. The weakness of these three countermeasures is the way in which they secure the intermediate values occurring in the AES **SubBytes** operation. The **SubBytes** operation is the non-linear component within AES, which makes it particularly difficult to mask.

In this article, we propose a secure masking scheme for the AES algorithm, which is particularly suited for implementation in dedicated hardware. In order to achieve security, we use a combination of additive and multiplicative masking. The most tricky part when masking AES is to mask its non-linear operation, which is the finite field inversion (short: inversion) in the S-box, *i.e.*, the **SubBytes** operation. All other operations are linear and can be masked in a straightforward manner as it is for example shown in [2]. Hence, this article focuses on the inversion operation in the S-box only.

The masking scheme for the inversion presented in this article is based on composite field arithmetic, which has already been previously used for efficient S-box implementations in hardware [15]. However, while in [15] the inversion is performed in $GF(16)$, we shift the inversion down to $GF(4)$ in this article. The motivation for this is the fact that the inversion in $GF(4)$ is a *linear* operation, which can be masked easily.

Because of that we can build a masking scheme with very nice properties. The approach presented in this article for example has a much smaller area-time product than [2]. It also has the advantage of being secure against *all* first-order differential side-channel attacks. In addition, it can be implemented in software as well.

The remainder of this article is organized as follows. In Section 2 we motivate our research by discussing zero-value attacks on multiplicative masking schemes. Our analysis shows the need for masking schemes which are secure against zero-value attacks. Such a new secure scheme is introduced in Section 3. Arguments for the security of our scheme are provided in Section 4. The efficiency in hardware compared to other masking schemes is discussed in Section 5. We conclude our research in Section 6.

2 Discussion of Multiplicative Masking Schemes

The masking schemes proposed in [2] and [13] are susceptible to so-called zero-value attacks. In this section, we analyze the effectiveness of zero-value attacks against these masking schemes.

In AES, an **AddRoundKey** operation is performed prior to the first encryption round and thus, prior to the first time when the inversion needs to be

computed. If a key byte k equals a data byte d, then the result of **AddRoundKey**, which is $x = d + k$, equals zero. This observation can readily be used in an attack which is referred to as **zero-value attack** and was introduced in [6].

Let t denote a power measurement (trace) and let the set of all traces t be denoted by T. Suppose a number of AES encryptions is executed and their power consumption is measured. Assume that the input texts are known. For all 256 possible key-bytes k', we do the following. We define a set M_1 which contains those measurements with $k' = d$ right before the **SubBytes** transformation. We also define a set M_2 which contains the measurements with $k' \neq d$ right before the **SubBytes** transformation.

$$M_1 = \{t \in T : k' = d\} \tag{1}$$

$$M_2 = \{t \in T : k' \neq d\} \tag{2}$$

If $k = k'$, then the difference-of-means trace $M_d = \overline{M_1} - \overline{M_2}$ shows a considerable peak at the point in time when the masked **SubBytes** operation has been performed. This is due to the fact that set M_1 contains the measurements in which the 0-value is manipulated in the inversion. If $k \neq k'$, then the definition of the sets is meaningless. Hence, no difference between the sets can be observed.

The difficulty in this scenario is that one needs enough traces in M_1 to reduce the variance, *i.e.* to get rid of noise. In [9] it has been estimated that around 64 times more measurements are needed in a zero-value attack than in a standard differential side-channel attack. This number indicates that zero-value attacks still pose a serious practical threat and must be avoided.

3 Combined Masking in Tower Fields

In order to thwart zero-value attacks, we have developed a new scheme which works with combinations of additive and multiplicative masks. Throughout the whole cipher, including the **SubBytes** computation, the data is concealed by an additive mask.

Before going into the details of our new scheme, we review some necessary facts about the efficient implementation of **SubBytes** first.

3.1 Inversion in $GF(256)$

Our **SubBytes** design follows the architecture we have proposed in [15] (we call this approach **S-IAIK** from now on) . This architecture is based on composite field arithmetic [5], and has very low area requirements. Thus, it is ideally suited for small hardware implementations. In this approach, each element of $GF(256)$ is represented as a linear polynomial $a_h x + a_l$ over $GF(16)$.

The inversion of such a polynomial can be computed using operations in $GF(16)$ only:

$$(a_h x + a_l)^{-1} = a'_h x + a'_l \tag{3}$$
$$a'_h = a_h \times d' \tag{4}$$
$$a'_l = (a_h + a_l) \times d' \tag{5}$$
$$d = (a_h^2 \times p_0) + (a_h \times a_l) + a_l^2 \tag{6}$$
$$d' = d^{-1} \tag{7}$$

The element p_0 is defined in accordance with the field polynomial which is used to define the quadratic extension of $GF(16)$, see [15].

In the following subsections we present the mathematical formulae for our masking scheme.

3.2 Masked Inversion in $GF(256)$

In our masking scheme for the inversion, which we call **Masked SubBytes IAIK** (short: **MS-IAIK**) from now on, all intermediate values as well as the input and the output are masked additively. In order to calculate the inversion of a masked input value, we first map the value as well as the mask to the composite field representation as defined in [15]. This mapping is a linear operation and therefore it is easy to mask. After the mapping, the value that needs to be inverted is represented by $(a_h + m_h)x + (a_l + m_l)$. Note that both elements in the composite field representation are masked additively.

Our goal is that all input and output values in the computation of the inverse are masked. Hence, we have to modify (3)-(7), by introducing functions f_{a_h}, f_{a_l}, f_d and $f_{d'}$, as follows:

$$((a_h + m_h)x + (a_l + m_l))^{-1} = (a'_h + m'_h)x + (a'_l + m'_l) \tag{8}$$
$$\begin{aligned} a'_h + m'_h &= f_{a_h}((a_h + m_h), (d' + m'_d), m_h, m'_h, m'_d) \\ &= a_h \times d' + m'_h \end{aligned} \tag{9}$$
$$\begin{aligned} a'_l + m'_l &= f_{a_l}((a'_h + m'_h), (a_l + m_l), (d' + m'_d), m_l, m'_h, m'_l, m'_d) \\ &= (a_h + a_l) \times d' + m'_l \end{aligned} \tag{10}$$
$$\begin{aligned} d + m_d &= f_d((a_h + m_h), (a_l + m_l), p_0, m_h, m_l, m_d) \\ &= a_h^2 \times p_0 + a_h \times a_l + a_l^2 + m_d \end{aligned} \tag{11}$$
$$\begin{aligned} d' + m'_d &= f_{d'}(d + m_d, m_d, m'_d) \\ &= d^{-1} + m'_d \end{aligned} \tag{12}$$

The function f_{a_h}, f_{a_l}, f_d and $f_{d'}$ are functions on $GF(16)$.

3.3 Derivation of the Functions f_{a_h}, f_{a_l}, f_d and $f_{d'}$

This section shows how to transform (4)-(7) into (9)-(12).

Transforming Equation 4 into Equation 9. Suppose that we calculate (4) with masked input values, *i.e.*, with $a_h + m_h$ instead of a_h and with $d' + m'_d$ instead of d':

$$(a_h + m_h) \times (d' + m'_d) = a_h \times d' + m_h \times d' + a_h \times m'_d + m_h \times m'_d. \quad (13)$$

Comparing the result of this calculation to (9) shows that the desired and masked result, $a_h \times d' + m'_h$, is only part of the result of (13). All the terms that occur in addition due to the masks, have to be removed. These terms can be easily removed by adding the terms $(d' + m'_d) \times m_h$, $(a_h + m_h) \times m'_d$, $m_h \times m'_d$ and m'_h. This is done by the function f_{a_h}, which takes five elements of $GF(16)$ as input, and produces an element of $GF(16)$ as output.

$$f_{a_h}(r, s, t, u, v) = r \times s + s \times t + r \times v + t \times v + u \quad (14)$$

If we choose $r = (a_h + m_h)$, $s = (d' + m'_d)$, $t = m_h$, $u = m'_h$ and $v = m'_d$ and compute $f_{a_h}((a_h + m_h), (d' + m'_d), m_h, m'_h, m'_d)$, we get the desired result $a_h \times d' + m'_h$ (see (9)).

One has to take care when adding correction terms that no intermediate values are correlated with values, which an attacker can predict. It needs to be pointed out that the formulae, which we derive in this section, do not lead to a secure implementation when directly implemented. The secure implementation of these formulae requires the addition of an independent value to the first intermediate value that is computed. This becomes clear from the security proof given in Section 4.

Another aspect, which we do not treat in this article, is the discussion of the particular choice of the masks m'_h, m'_l, m_d and m'_d. In our implementation in dedicated hardware, see [11] for details, we decided to re-use masks as often as possible. For example, in our implementation of (9) we set $v = m'_d = m_l$ and $u = m'_h = m_h$. Consequently, in our implementation we calculate the function f_{a_h} as shown in (15).

$$\begin{aligned} f_{a_h} = & \underbrace{(a_h + m_h) \times (d' + m_l)}_{dm_4} \\ & + \underbrace{(d' + m_l) \times m_h}_{c_7} + \underbrace{(a_h + m_h) \times m_l}_{c_1} \\ & + \underbrace{m_h}_{c_6} + \underbrace{m_h \times m_l}_{c_5} \end{aligned} \quad (15)$$

The term which is labelled as dm_4 refers to the masked data. The terms which are labelled as c_1 to c_7 in this equation are the so-called correction terms which are applied by the function f_{a_h}. It can be seen in the subsequent paragraphs that we can re-use several of these correction terms. This significantly reduces the area required for our implementation.

At first sight our numbering scheme for the masked-data terms and the correction terms might look erratic. However, the indices of the dm_i and c_j indicate when a certain value would be calculated during an implementation. For instance, the masked data is labelled by dm_4 in this formula, because it would be calculated only later. Equations (9)-(12) show, that the result of (12) is needed for (9) and (10). Therefore, (9) would be calculated later.

The reason why we make a difference in labelling masked-data terms and correction terms is that it makes it easier to see how many additional operations are introduced by the masking scheme. All terms labelled by dm_i have to be calculated in the original S-box design (**S-IAIK**, [15]) as well. However, all terms labelled by c_j are the corrections that we have to apply. Thus, these are the additional operations, which are introduced by the masking.

In the same style as for (4), we subsequently transform (5) and (6).

Transforming Equation 5 into Equation 10. In order to transform (5) into (10) we define a function f_{a_l} that applies the appropriate correction terms.

The function f_{a_l} takes seven elements of $GF(16)$ as input and gives one element of $GF(16)$ as output.

$$f_{a_l}(r,s,t,u,v,w,x) = r + s \times t + t \times u + s \times x + v + w + u \times x \tag{16}$$

If we choose $r = a'_h + m'_h$, $s = a_l + m_l$, $t = d' + m'_d$, $u = m_l$, $v = m'_h$, $w = m'_l$ and $x = m'_d$ we indeed get (10).

In our implementation, we set $u = w$ (*i.e.* $m'_l = m_l$) and $x = m'_d = m_h$. Hence, in our implementation, we calculate f_{a_l} as is shown in (17).

$$\begin{aligned} f_{a_l} = (a_h \times d' + m_h) &+ \underbrace{(a_l + m_l) \times (d' + m_h)}_{dm_5} \\ &+ \underbrace{(d' + m_h) \times m_l}_{c_8} + \underbrace{(a_l + m_l) \times m_h}_{c_2} \\ &+ \underbrace{m_l}_{c_9} + \underbrace{m_h}_{c_6} + \underbrace{m_l \times m_h}_{c_5} \end{aligned} \tag{17}$$

As in the previous paragraphs, the terms that are labelled by c_i are correction terms.

Transforming Equation 6 into Equation 11. In order to transform (6) into (11) we define a function f_d that applies the appropriate correction terms (as demonstrated in the previous paragraphs).

The function takes six elements of $GF(16)$ as inputs and gives an element of $GF(16)$ as result.

$$f_d(r,s,t,u,v,w) = r^2 \times t + r \times s + s^2 + r \times v + s \times u + u^2 \times t + v^2 + u \times w + u \tag{18}$$

If we choose $r = a_h + m_h$ and $s = a_l + m_l$, $t = p_0$, $u = m_h$, $v = m_l$ and $w = m_d$ then we get (11).

In our implementation we set $w = m_l$. Consequently, we calculate f_d in our implementation as shown in (19).

$$
\begin{aligned}
f_d = & \underbrace{(a_h + m_h)^2 \times p_0}_{dm_1} + \underbrace{(a_h + m_h) \times (a_l + m_l)}_{dm_2} + \underbrace{(a_l + m_l)^2}_{dm_3} \\
& + \underbrace{(a_h + m_h) \times m_l}_{c_1} + \underbrace{(a_l + m_l) \times m_h}_{c_2} \\
& + \overbrace{\underbrace{m_h^2}_{c_3} \times p_0}^{c_3'} + \underbrace{m_l^2}_{c_4} \\
& + \underbrace{m_h \times m_l}_{c_5} + \underbrace{m_h}_{c_6}
\end{aligned}
\tag{19}
$$

As in the previous paragraphs, the terms that are labelled by c_j or c_j' are correction terms.

Transforming Equation 7 into Equation 12. Calculating the inverse in $GF(16)$ can be reduced to calculating the inverse in $GF(4)$ by representing $GF(16)$ as quadratic extension of $GF(4)$.

In short, an element of $GF(4) \times GF(4)$ is a linear polynomial with coefficients in $GF(4)$, *i.e.*, $a = (a_h x + a_l)$, with a_h and $a_l \in GF(4)$. The same formulae as given in (17) – (19) can be used to calculate the masked inverse in $GF(4) \times GF(4)$. In $GF(4)$, the inversion operation is equivalent to squaring: $x^{-1} = x^2 \forall x \in GF(4)$. Hence, in $GF(4)$ we have that $(x + m)^{-1} = (x + m)^2 = x^2 + m^2$; the inversion operation preserves the masking in this field.

4 Security of Our Masking Scheme

In this section, we show that all the operations discussed in Section 3, are secure. We follow the security notion that has been introduced in [4] and strengthened by [3]:

Definition 1. An algorithm is said to be secure if for all adversaries A and all realizable distributions M_1 and M_2, M_1 equals M_2.

This definition is equivalent to the *perfect masking* condition given in [3] for standard differential SCA. Counteracting higher-order differential SCA is not within the scope of this article.

In the following paragraphs we will show that all data-dependent intermediate values that occur in (17) – (19) fulfill Definition 1. These values are masked data $a+m$, masked multiplications $(a+m_a) \times (b+m_b)$, multiplications of masked values with masks $(a + m_a) \times m_b$, and masked squarings $(a + m_a)^2$ and $(a + m_a)^2 \times p$.

Definition 1 does imply that regardless of the hypotheses, which an attacker can make, the distributions, which are derived by using these hypotheses, are identical. Consequently, we must proof that every operation that is performed in our masking scheme, leads to an output whose distribution does not depend (in a statistical sense) on the input.

Our proof is divided into two parts. In the first part, which consists of the Lemmas 1 to 4, we show that the data-dependent intermediate values are all secure. In the second part, which consists of Lemma 5, we show that also the summation of the intermediate results can be done securely.

We re-use the Lemmas 1 and 2 of [3]:

Lemma 1. Let $a \in GF(2^n)$ be arbitrary. Let $m \in GF(2^n)$ be uniformly distributed in $GF(2^n)$ and independent of a. Then, $a + m$ is uniformly distributed regardless of a. Therefore, the distribution of $a + m$ is independent of a.

Lemma 2. Let $a, b \in GF(2^n)$ be arbitrary. Let $m_a, m_b \in GF(2^n)$ be independently and uniformly distributed in $GF(2^n)$. Then the probability distribution of $(a + m_a) \times (b + m_b)$ is

$$\Pr((a + m_a) \times (b + m_b) = i) = \begin{cases} \frac{2^{n+1}-1}{2^{2n}} & \text{, if } i = 0, \textit{ i.e.}, \text{ if } m_a = a \text{ or } m_b = b \\ \frac{2^n-1}{2^{2n}} & \text{, if } i \neq 0. \end{cases}$$

Therefore, the distribution of $(a + m_a) \times (b + m_b)$ is independent of a and b.

These two lemmas cover almost all data-dependent operations in our masking scheme. The operation $(a + m_a) \times m_b$ is covered by Lemma 3.

Lemma 3. Let $a \in GF(2^n)$ be arbitrary. Let $m_a, m_b \in GF(2^n)$ be independently and uniformly distributed in $GF(2^n)$. Then the distribution of $(a+m_a) \times m_b$ is

$$\Pr((a + m_a) \times m_b = i) = \begin{cases} \frac{2^{n+1}-1}{2^{2n}} & \text{if } i = 0, \textit{ i.e.}, \text{ if } m_a = a \text{ or } m_b = 0 \\ \frac{2^n-1}{2^{2n}} & \text{if } i \neq 0. \end{cases}$$

Therefore, the distribution of $(a + m_a) \times m_b$ is independent of a.

Lemma 3 is a special case ($b = 0$) of Lemma 2. The proof is therefore omitted.

The two remaining operations that occur in our masking scheme, $(a + m_a)^2$ and $(a + m_a)^2 \times p$ are covered by Lemma 4.

Lemma 4. Let $a \in GF(2^n)$ be arbitrary and $p \in GF(2^n)$ a constant. Let $m_a \in GF(2^n)$ be independently and uniformly distributed in $GF(2^n)$.

Then, the distribution of $(a + m_a)^2$ and $(a + m_a)^2 \times p$ is independent of a.

Proof. According to Lemma 1, $a + m_a$ is uniformly distributed in $GF(2^n)$. This is straightforward because for an arbitrary but fixed a, $a+m_a$ is a permutation of $GF(2^n)$. Hence, $(a+m_a)^2$ gives all quadratic residues of $GF(2^n)$, regardless of a. This implies that the distribution of $(a+m_a)^2$ is independent of a. Consequently, also the distribution of $(a + m_a)^2 \times p$ (p is a constant) is independent of a.

The Lemmas 1 to 4 show that all major operations of our masking scheme are secure. However, more intermediate results occur in the masking scheme because we add the major operations, and thus, produce implicitly more intermediate results than are directly visible from the formulae.

Lemma 5 shows that these intermediate results can be added in a secure way.

Lemma 5. Let $a_i \in GF(2^n)$ be arbitrary and $M \in GF(2^n)$ be independent of all a_i and uniformly distributed in $GF(2^n)$.

Then the distribution of $\sum_i a_i + M$ is (pairwise) independent of a_i.

Proof. The proof for this lemma follows directly from Lemma 1.

Lemma 5 shows that for secure implementations, the order in which the terms of a sum are added, is important! In particular, every summation of variables must start with the addition of an independent mask M.

5 Comparison of Masking Schemes

A high-level comparison of the three masking schemes, **S-Akkar**, **S-Blömer** and **MS-IAIK** shows that in terms of area, our scheme leads to the smallest implementation. Table 1 lists the number of high-level operations (multiplication, multiplication with a constant and square) in $GF(16)$ of each of the three schemes.

Table 1. High-level comparison of masking schemes

	Mult	MultConst	Square
S-Akkar	18	6	4
S-Blömer	12	1	2
MS-IAIK	9	2	2

We have not included a count of the $GF(4)$ operations for **S-Blömer** and **MS-IAIK** because they do not contribute significantly to the area. We have also not included an XOR count in $GF(16)$ because the number of XORs is highly dependent on the amount of fresh masks which are available. In the following, we discuss hardware implementations of **S-Akkar** and **S-Blömer** in more detail.

5.1 S-Akkar

S-Akkar makes use of multiplications in finite fields. In particular, 4 multiplications, 1 inversion and 2 XORs in addition to the original inversion have to be computed. All operations are performed in the finite field with 256 elements. For a fair comparison, we assume that all multipiers are based on the same, optimized multipliers which we consider for the implementation of **S-IAIK**. Because **S-IAIK** uses composite field arithmetic, three $GF(16)$ multipliers and one $GF(16)$ constant-coefficient multiplier had to be combined according to [10] in order to build a $GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$ multiplier. Counting only the largest component of this circuit, which are the $GF(16)$ multipliers, we see that this implementation requires $4 \times 3 = 12$ $GF(16)$ multipliers. Hence, the implementation of **S-Akkar** is much bigger than an implementation of our masking scheme.

5.2 S-Blömer

S-Blömer suggests a comparable technique for counteracting (single-order) differential side-channel attacks as we do. In this article, an implementation strategy for a dedicated hardware implementation is outlined. In this strategy, the architecture of the **SubBytes** operation is based on [12]. In [12], the authors have used the Itho-Tsujii algorithm [7] for computing the multiplicative inverse over a finite field. The Itho-Tsujii inversion algorithm leads to the same inversion formulae as used in [15].

The major difference between [15] and [12] is that in [15], field polynomials have been chosen that lead to particularly efficient finite field arithmetic. In [12], other field polynomials have been chosen that lead to a less efficient finite field arithmetic. Thus, **S-Blömer** suffers from this drawback.

S-Blömer leads to a larger **SubBytes** implementation than our proposal **MS-IAIK**. This is based on the fact that in **S-Blömer** high-level operations such as the finite field multiplication and the finite field squaring are masked. As a consequence, correction terms are computed more often than necessary. In particular, correction terms which involve multiplications cannot be re-used. This, in combination with the less efficient finite field arithmetic, leads to an increase in area.

S-Blömer requires the computation of three masked multiplications in $GF(16)$. One masked multiplication requires 4 ordinary $GF(16)$ multiplications. Hence, **S-Blömer** requires 12 $GF(16)$ multipliers according to [12].

6 Conclusions

In this article, we have presented a new secure and efficient scheme for masking the intermediate value of an AES **SubBytes** implementation. To motivate our research, we have discussed zero-value attacks on multiplicative masking schemes first. Zero-value attacks pose a serious practical threat. Therefore we have introduced a new masking scheme which does not succumb to these attacks. We have given arguments for the security of our scheme. In addition, we have compared the number of operations needed in our scheme with other masking schemes; our scheme requires the least amount of operations.

References

1. M.-L. Akkar, R. Bevan, and L. Goubin. Two Power Analysis Attacks against One-Mask Methods. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, 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, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

3. Johannes Blömer, Jorge Guajardo Merchan, and Volker Krummel. Provably Secure Masking of AES. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2004/101, 2004.
4. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
5. John Horton Conway. On Numbers and Games. 2nd edition, AK Peters, 2001
6. Jovan D. Golić and Christophe Tymen. Multiplicative Masking and Power Analysis of AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2535 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2003.
7. Toshiya Itoh and Shigeo Tsujiis. A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. *Information and Computation*, 78(3):171–177, September 1988.
8. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
9. Elisabeth Oswald, Stefan Mangard, and Norbert Pramstaller. Secure and Efficient Masking of AES – A Mission Impossible? Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2004/134, 2004.
10. Christof Paar. *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields*. PhD thesis, Institute for Experimental Mathematics, University of Essen, 1994.
11. Norbert Pramstaller, Frank K. Gürkaynak, Simon Haene, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner. Towards an AES Crypto-chip Resistant to Differential Power Analysis. In *Proccedings 30th European Solid-State Circuits Conference - ESSCIRC 2004, Leuven, Belgium, Proceedings - to appear*, 2004.
12. Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, pages 239–254. Springer, 2001.
13. 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, *Cryptographic Hardware and Embedded Systems – CHES 2002*, pages 187–197. Springer, 2003.
14. Kris Tiri and Ingrid Verbauwhede. Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, pages 125–136. Springer, 2003.
15. Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. An ASIC implementation of the AES SBoxes. In Bart Preneel, editor, *Topics in Cryptology - CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2002.