Multiplicative Masking and Power Analysis of AES

Jovan D. Golić[1]* and Christophe Tymen[2,3]

[1] Rome CryptoDesign Center, Gemplus
Via Pio Emanuelli 1, 00143 Rome, Italy
jovan.golic@gemplus.com
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
christophe.tymen@gemplus.com
[3] École Normale Supérieure
45 rue d'Ulm, 75230 Paris Cedex 05, France

Abstract. The recently proposed multiplicative masking countermeasure against power analysis attacks on AES is interesting as it does not require the costly recomputation and RAM storage of S-boxes for every run of AES. This is important for applications where the available space is very limited such as the smart card applications. Unfortunately, it is here shown that this method is in fact inherently vulnerable to differential power analysis. However, it is also shown that the multiplicative masking method can be modified so as to provide resistance to differential power analysis of nonideal but controllable security level, at the expense of increased computational complexity. Other possible random masking methods are also discussed.

Keywords. AES, differential power analysis, countermeasures, multiplicative masking.

1 Introduction

Side-channel attacks on software or hardware implementations of various cryptosystems aim at recovering the secret key information from certain physical measurements performed on the electronic device during the computation such as the power consumption, the time, and the electromagnetic radiation. Power analysis attacks [9] are very powerful as they do not require expensive resources and as most implementations without specific countermeasures incorporated are vulnerable to such attacks. Among them, the (first-order) differential power analysis (DPA) attacks are particularly impressive, because they use relatively simple mathematical tools and techniques that are independent of the implementation of the cryptographic algorithm. Moreover, the countermeasures are typically costly in terms of speed performance and memory requirements.

* A preliminary version of this work was presented at the Gemplus Quarterly meeting, La Ciotat, France, October 30–31, 2001.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 198–212, 2003.

The goal of simple power analysis (SPA) attacks is to deduce some information about the secret key, such as the Hamming weight of some parts of the key, from a single power consumption curve. This may be possible if, for example, there are branches in the computation that depend on the secret key. More generally, one can also collect a large training set of power consumption curves from different secret keys (and possibly different input data) and then use appropriate statistical hypothesis testing methods in order to identify traces or signatures of the parts of the secret key hidden in the curves. For example, key scheduling algorithms for block ciphers may especially be vulnerable in this regard, due to the absence of the randomization effect of input data. However, the statistical techniques to be used may be complicated and dependent on the particular implementation.

The DPA attack [9] requires a set of power consumption curves obtained by running the cryptographic algorithm a number of times for the same secret key and different inputs. A necessary algorithmic condition, the so-called *fundamental hypothesis*, for the DPA attack to be effective is the existence of one or more intermediate variables in the algorithm that can be expressed as or are correlated to functions depending on a small number of key bits and on known input or output data. The key bits involved may then be reconstructed by partitioning the set of curves according to the value of the chosen intermediate variable corresponding to the key bits guessed and to the input or output data known and by computing and comparing some simple statistic, such as the average, on the partitioned curves at individual points in time. The attack is successful if the correct guess about the key bits results in a significant difference between the computed average curves at one or more points in time. For other possibilities, see [4]. What makes the attack practically very interesting is that many cryptographic algorithms satisfy the fundamental hypothesis. For example, the intermediate variables in the first or the last few rounds of practical block ciphers are especially vulnerable.

A higher-order DPA attack is a generalization of the (first-order) DPA attack in which the power consumption curves are analyzed by using a joint statistic applied to collections of points in time. The general attack is more powerful, but may be more complex and considerably more complicated as the choice of these points and possibly also of the joint statistic is likely to depend on the particular implementation.

In principle, the complexity of the side-channel attacks can be increased by introducing physical or algorithmic countermeasures. A general strategy to render the SPA and DPA attacks more difficult to mount is to balance and randomize elementary computations involving the secret key, e.g., by randomly introducing dummy operations and timing shifts, as well as by randomizing the order of elementary computations and the computations themselves. A general technique to prevent the first-order or higher-order DPA attacks is random data splitting [7], [3], especially for the computation of intermediate variables satisfying the fundamental hypothesis. It is pointed out in [10] that for the (first-order) DPA, instead of splitting the data into two parts one may as well apply random masks to data which are easier to implement. Of course, one has to be careful to mask the data completely and thus avoid weaknesses such as the one shown in [5] for a masking technique from [10]. Also, the computations involved in masking have to be performed in a secure way, which itself is not vulnerable to DPA. Random masks have to be generated for each new run of the cryptographic algorithm, but may be repeated within the algorithm. The repetitions generally increase the vulnerability to higher-order DPA. The random masks can be combined with data by using (quasi)group operations such as the bitwise addition or modular integer addition.

If an affine transformation is applied to masked data and if the masking operation is the same as the corresponding linear operation, then only the additive constant has to be recomputed for each new mask in order to maintain the equivalence of the data transformations. However, this is generally not the case with nonlinear transformations. They typically have to be recomputed and stored depending on the mask and this can be very costly for many cryptographic algorithms including AES [6]. In AES, the only nonlinear transformations are nonlinear parts of 8×8-bit S-boxes which perform the multiplicative inversion in GF(256). In [2], a masking technique is proposed which combines the usual binary additive masking with the multiplicative masking of data, using the multiplication in GF(256), thus avoiding the costly recomputation and RAM storage of S-boxes.

In this paper, it is shown that this multiplicative masking technique is vulnerable to the first-order DPA attack and in some sense even more than AES without masking. This is essentially because the all-zero input to the S-boxes is not effectively masked by the multiplicative mask and the binary additive mask is first removed in order to apply the multiplicative mask.[1] Moreover, it is argued that the weakness is inherent to the multiplicative masking and therefore cannot be remedied so as to achieve ideal security. In addition, the so-called embedded multiplicative masking technique which can achieve approximate security with a controllable security level is introduced. It is also pointed out that the masking technique [10] in which only one S-box is recomputed and stored in RAM and used repeatedly during one execution of AES is vulnerable to a relatively simple second-order DPA attack.

The main lines of the DPA attack [9] applied to AES are described in more detail in Section 2 and the multiplicative masking technique is presented in Section 3. The inherent weakness is explained in Section 4, the embedded multiplicative masking technique is proposed in Section 5, and conclusions are given in Section 6.

[1] A similar power analysis attack, although not in the DPA form, is independently given in the unpublished manuscript “Time and memory efficiency in protecting AES against higher order power attacks,” by N. T. Courtois and M.-L. Akkar, from April 2002.

2 Differential Power Analysis of AES

AES is a product block cipher composed of a number of rounds each consisting of a reversible nonlinear transformation providing *confusion* and a reversible linear transformation providing *diffusion*, where the linearity is with respect to the binary field, GF(2). The expanded secret key is bitwise added to the plaintext and to the output of each round. The nonlinear transformation consists of identical 8×8-bit S-boxes each performing the byte substitution ByteSub defined as the multiplicative inversion in GF(256), leaving the all-zero input intact, followed by an affine 8×8-bit transformation. The linear transformation consists of a permutation of output bytes of S-boxes denoted as ShiftRow followed by a linear transformation denoted as MixColumn, which is removed from the last round. More details can be found in [6], but are irrelevant for our analysis.

According to [9], the DPA attack on AES consists of the following stages. The intermediate variables satisfying the fundamental hypothesis are the output bytes of the S-boxes or of just the nonlinear parts of the S-boxes in the first round. Each of them is a function of the input byte which itself is a bitwise sum of the corresponding plaintext and expanded key bytes. Accordingly, if the plaintext byte is known and the key byte is guessed correctly, then the S-box output byte can be computed correctly. The objective of the attack is to recover the expanded key in a byte-by-byte divide-and-conquer manner.

In the first stage, a sufficient number, N, of curves are obtained by measuring the power consumption during the execution of (the first round of) AES for the same secret key and N different plaintexts. The average C of these N curves is then computed. The second stage is performed for each of the S-boxes in the first round. For each of 256 possible values of the targeted expanded key byte K, a subset of M (on average, $M = N/2^m$) plaintexts resulting in a chosen fixed m-bit value of the partial output byte of the considered S-box are identified, the corresponding M curves are extracted, and their average $C(K)$ computed. For example, the chosen fixed value may have maximal or minimal Hamming weight (all-one or all-zero m-bit words). More generally, if one knows good power consumption models of the involved components, an optimal subset of M curves can be chosen according to a set of 2^{8-m} or of any other number of output byte values best distinguished from the others with respect to power consumption, as proposed in [1] for $m = 1$.

A value K is then assumed to be correct if the difference between the two average curves, $C(K)$ and C, contains one or more noticeable peaks. The peaks are due to the same value of the S-box output being computed at the same time for each of the extracted M curves, if the value K is correct, and to unbalanced power consumption associated with different S-box output values. *This is the main point of the DPA attack.* On the other hand, if the value K is incorrect, then the outputs of the S-box will vary and the peaks will not be observed. More precisely, this is the case for $m < 8$. For $m = 8$, as the S-boxes are reversible, a fixed output byte value implies that the input byte value is also fixed. Therefore, even if the guessed value K is incorrect, then for the extracted curves both the input and output bytes will have fixed values, different for different K, also giving rise to observable peaks, possibly of different magnitudes for different K. As a consequence, the reliable statistical distinction of the correct K may be infeasible.

If m decreases, then M increases, but the impact of the repeated computation on each of the M extracted power consumption curves becomes statistically less significant. Also, it is not clear how one can simultaneously use more than just one fixed output m-bit value in order to increase the statistical distinction between the correct and incorrect key values. Nonetheless, this may be possible.

According to the key schedule in AES, if the key size is not bigger than the plaintext block size, then the recovered expanded key bytes directly specify the secret key. Otherwise, the DPA attack should also be applied to the second round of AES in order to recover the whole secret key.

3 Multiplicative Masking of AES

The starting idea of the method proposed in [2] in order to prevent the DPA attack on AES is to use the binary additive mask which is compatible with the binary linear or affine transformations in AES. Accordingly, as far as the affine transformations are concerned, only the additive constants are affected by this mask. However, if the additive mask is applied to the input of the nonlinear part of an S-box in AES, then this nonlinear part has to be recomputed for each new mask used. Recall that the nonlinear part, F, of the S-box transformation ByteSub is the multiplicative inversion in GF(256) extended by mapping the all-zero input into the all-zero output. For simplicity, F is called the inversion in GF(256). The main idea from [2] is to use the nonzero multiplicative mask, with respect to the multiplication in GF(256), for the data passing through F, without having to recompute and store F. To this end, one has to convert the additive mask into the multiplicative mask at the input of each F and to reproduce the additive mask from the multiplicative mask at the output of each F. A way, secure with respect to DPA, of converting the masks is suggested in [2]. More details are given below.

Fig. 1 shows the data flow in the i-th round of AES without and with the masking countermeasure. A general rule in all the figures presented is that the expressions for input, output, and all intermediate data are displayed within the rectangular blocks. It is assumed that the ByteSub and Modified ByteSub transformations act on all the bytes in a block. Note that the expanded key is bitwise added to the plaintext to form the input to the first round, that the MixColumn transformation is removed from the last round, and that the additive mask is not produced at the output of the last round. According to [2], the additive mask X is the same in every round. In fact, keeping the same additive mask in every round would matter if the S-boxes had to be recomputed for each new mask, because in that case the same recomputed S-boxes could be used in each round. So, restoring the same mask X in the last step of each round is not really needed. Instead, one can just add the expanded key K_i and thus effectively obtain the output mask $X^{(3)}$. Only in the last round, the output mask has to be removed. Here, the masks are transformed by the linear transformations as $X^{(1)} = L(X)$, $X^{(2)} = \text{ShiftRow}(X^{(1)})$, and $X^{(3)} = \text{MixColumn}(X^{(2)})$, where L denotes the linear part of the affine transformation of ByteSub combined for all the bytes in a block. So, essentially only the ByteSub transformation has to be modified, because of the nonlinear part contained.

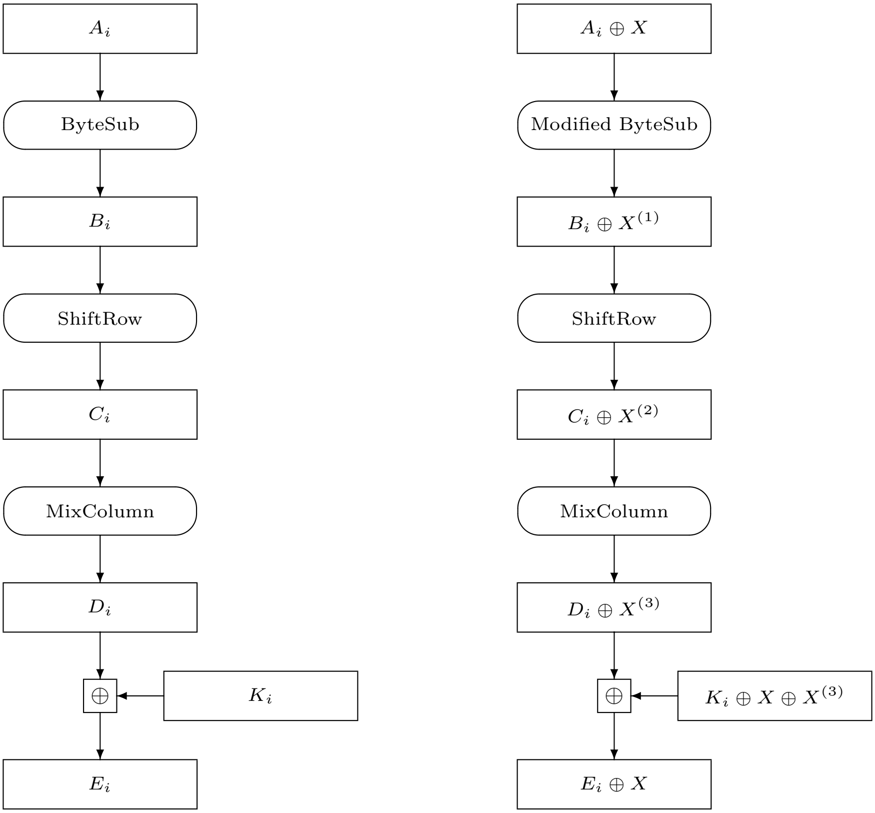

Fig. 1. The round i of AES without and with masking countermeasure.

The data flow through the original and modified ByteSub transformations, acting on bytes, is shown in Fig. 2 (the index j stands for a particular byte in a block and the index i stands for a particular round). The affine transformation is unchanged, and only the nonlinear transformation, F, has to be modified. This is achieved by using a nonzero multiplicative mask $Y_{i,j}$ in a way displayed in Fig. 3, which is self-explanatory.

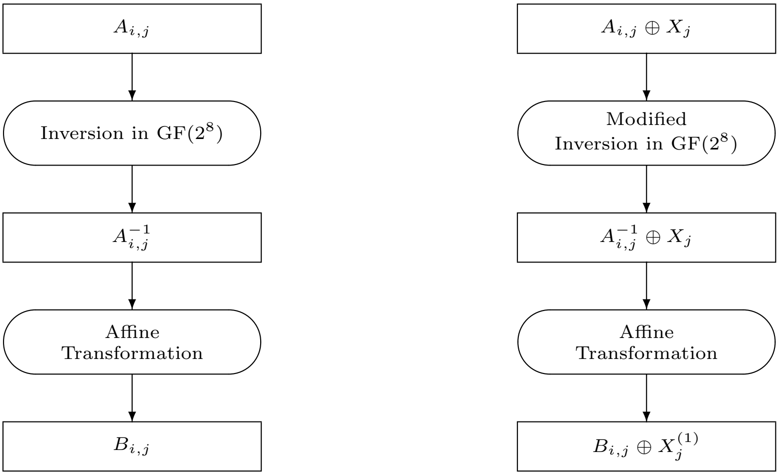

Fig. 2. The ByteSub transformation without and with masking countermeasure.

Recall that the addition in GF(256) is the same as the bitwise addition. It follows that F does not have to be recomputed and stored in a look-up table for each new mask $Y_{i,j}$. This is due to the multiplication in GF(256) being compatible with F or, more precisely, to the equality

$$F(A \otimes Y) \;=\; F(A) \otimes F(Y) \tag{1}$$

where $F(Y) \neq 0$ if $Y \neq 0$, so that $F(A)$ can be recovered from $F(A) \otimes F(Y)$. In other words, if a masked input is transformed by F itself, then the masked desired output is obtained. So, one just has to convert the multiplicative into the additive mask and vice versa, and that can be done by one more inversion, four multiplications, and two additions in GF(256).

Note that in general, if two, possibly different, group operations $*$ and $\bullet$ are used for masking the input and output data for a transformation F, respectively, then the masked data should be transformed by the modified transformation F' satisfying $F'(A * Y_1) = F(A) \bullet Y_2$. Equivalently, F' is defined by

$$F'(A) \;=\; F(A * Y_1^{-1}) \bullet Y_2, \tag{2}$$

where Y_1 and Y_2 are the input and output masks, respectively, which can be mutually related. In order to resist DPA, F' should not be directly implemented by using F and (2). For example, a secure way would be to use a look-up table for F', but it has to be recomputed and stored in RAM for every new pair of masks Y_1 and Y_2.

The multiplicative masks $Y_{i,j}$ and the additive masks X_j can be randomly chosen so as to be uniformly distributed and mutually independent. Also, $Y_{i,j}$ can be the same for each round i and possibly related to X_j, but this generally increases the vulnerability to higher-order DPA. Since all the intermediate variables in Fig. 3 are masked, it is claimed in [2] that the masked AES should be resistant to the (first-order) DPA attack. This masking method is important, because it avoids the recomputation and storage of S-boxes for each new run of AES, which would, for example, require 256 × 16 bytes of RAM for the 128-bit AES if all the S-boxes in a round are masked by mutually independent masks.

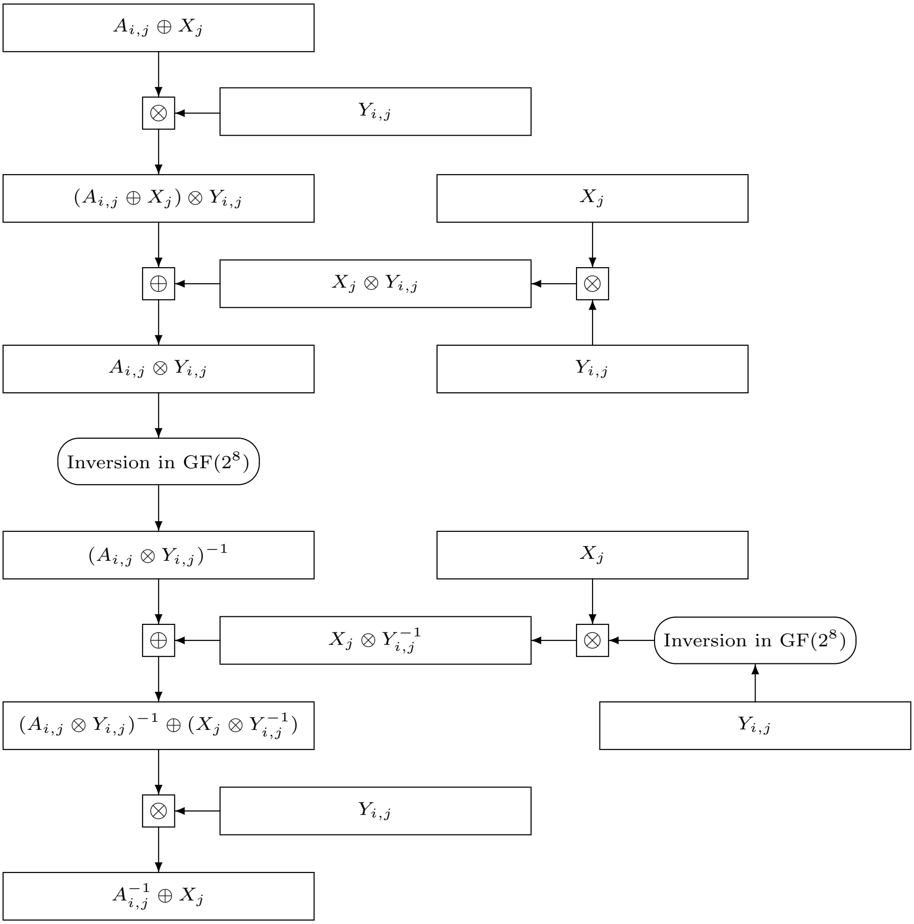
$A_{i,j} \oplus X_j$
$\otimes$
$Y_{i,j}$
$(A_{i,j} \oplus X_j) \otimes Y_{i,j}$
X_j
$\oplus$
$X_j \otimes Y_{i,j}$
$\otimes$
$A_{i,j} \otimes Y_{i,j}$
$Y_{i,j}$
Inversion in GF(2^8)
$(A_{i,j} \otimes Y_{i,j})^{-1}$
X_j
$\oplus$
$X_j \otimes Y_{i,j}^{-1}$
$\otimes$
Inversion in GF(2^8)
$(A_{i,j} \otimes Y_{i,j})^{-1} \oplus (X_j \otimes Y_{i,j}^{-1})$
$Y_{i,j}$
$\otimes$
$Y_{i,j}$
$A_{i,j}^{-1} \oplus X_j$

Fig. 3. Modified inversion in GF(256) with multiplicative masking countermeasure.

4 Differential Power Analysis of Masked AES

In this section, a subtle security flaw of the masking method [2] described in Section 3 is pointed out. In addition, it is argued that the multiplicative masking for AES is inherently vulnerable to the DPA attack.

The basic problem with the multiplicative mask is that it does not mask the all-zero byte value of data, that is, the all-zero byte remains unchanged after masking by a multiplicative mask. On the other hand, the all-zero (intermediate) data bytes cannot be avoided in AES. As a consequence, there are intermediate variables in the modified inversion scheme from Fig. 3 that are not masked completely and satisfy the fundamental hypothesis for DPA by being correlated to a function depending on only 8 key bits and 8 plaintext bits.

More precisely, in the first round of the masked AES, the vulnerable intermediate variables are the input byte $Z_{1,j} = A_{1,j} \otimes Y_{1,j}$ and the output byte $Z_{2,j} = F(A_{1,j} \otimes Y_{1,j})$ of the block implementing the inversion in GF(256). Note that the data byte $A_{1,j}$ is given as $A_{1,j} = P_j \oplus K_{0,j}$ where P_j and $K_{0,j}$ are the corresponding plaintext and expanded key bytes, respectively. It follows that

$$K_{0,j} = P_j \;\Rightarrow\; Z_{1,j} = 0 \;\Rightarrow\; Z_{2,j} = 0. \tag{3}$$

So, interestingly, it turns out that the (first-order) DPA attack on the masked AES can be mounted in essentially the same way as on the original AES without masking, which is described in Section 2. The difference is that one has to target the all-zero input byte or, equivalently, the all-zero output byte of F. In other words, for each of 256 possible values of the corresponding expanded key byte $K_{0,j}$, the power consumption curves for which $P_j = K_{0,j}$ are extracted and used for identifying the correct key. To this end, appropriately chosen plaintexts can reduce the required number of power curves. Since $m = 8$, the DPA attack on AES without masking would not be effective, as explained in Section 2.

However, for the masked AES, the DPA attack will be able to distinguish between correct and incorrect guesses of the expanded key byte, because of the randomization effect provided by the random multiplicative mask. Observe that if $K_{0,j}$ is guessed correctly, then the peaks will appear because of the repeated simultaneous computation of not only the all-zero output byte $Z_{2,j}$, but also the all-zero input byte $Z_{1,j}$. Altogether, the DPA attack may be more effective than the one on AES without masking, especially if one cannot find an effective way to simultaneously use more than just one fixed target m-bit value in the DPA attack on AES, where $m < 8$, or, more generally, a way to use (possibly optimal) partitions of power consumption curves into more than just two sets, provided that the power consumption models are available.

Now, the question is if the described DPA attack can be somehow prevented by using some other implementation of the multiplicative masking. For example, one may try to replace the modified inversion performed on $A_{i,j}$ by the modified inversion performed on some nonzero input byte whenever $A_{i,j} = 0$, and then to replace the computed output value by the desired one. However, detecting whether $A_{i,j} = 0$ and replacing the computed output value require specific computations that are themselves vulnerable to the DPA attack. In conclusion, it appears that the weakness of the multiplicative masking for AES is hard to remove ideally. Nevertheless, there may exist measures for reducing the weakness.

Of course, it would be practically important, especially for applications where the space is very limited, to find another masking method that will not require the recomputation and storage of S-boxes for every new run of AES. To this end, one has to use group or, more generally, quasigroup operations for masking the whole range of possible byte values which would at least simplify the secure computation and/or storage of F' according to (2), where F is the inversion in GF(256). However, this does not appear to be very likely.

In the next section, we propose an approximate, nonideal solution to the problem which is based on a random embedding of GF(256) into a larger algebraic structure so that the zero value is mapped into a set of values and all the operations remain compatible with GF(256).

5 Embedded Multiplicative Masking

5.1 Overview of Countermeasure

We represent the field GF(256) as the ring of binary polynomials in x modulo an irreducible polynomial $P(x)$ of degree 8. Let $Q(x)$ be another binary irreducible polynomial that is coprime to $P(x)$ and has degree k. Then GF(256) is a subring of the ring $\mathbf{R} = \mathrm{GF}(2)[x]/(PQ)$, which is isomorphic to $\mathrm{GF}(256) \times \mathrm{GF}(2^k)$, with the isomorphism $U \mapsto (U_P, U_Q)$, where the two coordinates are defined as $U_P = U \mod P$ and $U_Q = U \mod Q$.

To repair the multiplicative masking described above, we suggest to use the random mapping $\rho : \mathrm{GF}(256) \to \mathbf{R}$ defined by

$$\rho(U) \ = U + RP \tag{4}$$

where R is a randomly chosen polynomial of degree less than k (a k-bit word). Our basic idea relies on the fact that the zero in GF(256) is mapped onto 2^k possible values in $\mathbf{R}$ and should hence be more difficult to detect when k increases. Since $\rho(U)_P = U$, ρ only randomizes the second coordinate, so that choosing R of degree k or larger and taking the result modulo PQ will not increase the randomization effect.

Let $F' : \mathbf{R} \to \mathbf{R}$ be a mapping defined by $F'(U) = U^{254}$. Then, because of $(F'(U)_P, F'(U)_Q) = (U_P^{254}, U_Q^{254})$, F' coincides with F on GF(256), and if 7 does not divide k, then U_Q^{254} is an 1-1 function of U_Q, so that F' does not deteriorate the randomization induced by ρ (for $k = 7$, U^{509} will do). The embedded multiplicative masking countermeasure then consists in modifying the data path in Fig. 3 so that the input data $A_{i,j} \oplus X_j$ is mapped through ρ into $\mathbf{R}$, the first multiplication and two additions are taken modulo PQ, and F' is substituted for F. The second multiplication along the data path and all other operations involving the additive and multiplicative masks remain the same, that is, modulo

P. Accordingly, in mathematical terms, the modified method is the same as the original method with respect to the first coordinate, and the only difference is in the introduced randomized second coordinate. Here, the k-bit word R essentially acts as an additional mask. Of course, in a secure implementation the two coordinates should not be computed explicitly.

5.2 Efficient Implementation

As k grows, it quickly becomes difficult to securely implement F' using a look-up table. For $k = 8$, for instance, 2^{20} bits of (ROM) memory space are required, which is unacceptable in many practical situations. As a software alternative, it is possible to evaluate F' using the traditional "square-and-multiply" method, with about 8 squarings and 4 multiplications in $\mathbf{R}$. This solution can be made more efficient by chosing a specific representation for $\mathbf{R}$, as it is shown now.

Recall that the AES standard specifies usage of the polynomial $P_0(x) = 1 + x + x^3 + x^4 + x^8$ to represent GF(256). The idea is to choose a different polynomial that is more suitable for performing the multiplication. In particular, since in GF(2)[x]

$$1 + x^{17} = (1 + x)(1 + x^3 + x^4 + x^5 + x^8)(1 + x + x^2 + x^4 + x^6 + x^7 + x^8), \tag{5}$$

the choice $P(x) = 1+x^3+x^4+x^5+x^8$ instead of $P_0(x)$, and $Q(x) = 1+x+x^2+x^4+x^6+x^7+x^8$, $k = 8$, yields a particularly efficient encoding. The conversion between the coordinates in the two corresponding bases is achieved by applying the linear transformations determined by the 8×8 binary matrices

$$M = \begin{pmatrix} 1&1&0&1&1&0&0&1 \\ 0&0&1&1&0&1&1&1 \\ 0&0&1&1&0&1&0&0 \\ 0&0&1&1&1&0&0&1 \\ 0&0&1&0&0&1&1&1 \\ 0&1&1&0&0&1&1&0 \\ 0&1&0&0&0&1&1&0 \\ 0&0&0&1&1&1&0&0 \end{pmatrix} \quad \text{and} \quad M^{-1} = \begin{pmatrix} 1&1&1&0&0&0&1&1 \\ 0&1&1&1&0&1&0&1 \\ 0&0&0&0&0&1&1&0 \\ 0&1&0&0&1&0&0&0 \\ 0&0&1&0&0&1&1&1 \\ 0&1&1&0&1&1&1&0 \\ 0&0&0&1&1&0&0&1 \\ 0&1&1&1&1&0&0&1 \end{pmatrix} \tag{6}$$

with respect to the LSB-first representation. More precisely, the input and output bytes in Fig. 3 should be multiplied as binary vectors (one-column matrices) by M and M^{-1}, respectively, and the additive mask used should be multiplied by M, because of $M(A_{i,j} \oplus X_j) = MA_{i,j} \oplus MX_j$. Note that the output multiplication by M^{-1} restores the same additive mask. In fact, the explicit output multiplication by M^{-1} can be avoided by incorporating M^{-1} into the affine part of the ByteSub transformation shown in Fig. 2.

Let us look at how multiplication works in $\mathbf{R}_{16} = \mathrm{GF}(2)[x]/(PQ)$ (see also [13]). As all the elements of $\mathbf{R}_{16}$ can be represented as 16-bit words, let U and V be two words representing two elements of $\mathbf{R}_{16}$, with the LSB-first convention. We compute $W = U \otimes V$ in $\mathbf{R}_{16}$ by performing

```
$W_1, W_2 \leftarrow$ MULX $U, V$
$W = \ll W_2$
$W \oplus = W_1$
IF $(W_2)_0 = 1$ THEN $W \oplus = \texttt{FFFF}$ ELSE $W_1 \oplus = \texttt{FFFF}$
```

where MULX denotes the polynomial multiplication, $\oplus$ denotes the 16-bit XOR operation, and $\ll$ the 16-bit leftshift operation. The last operation $W_1 \oplus = \texttt{FFFF}$ is here only to ensure that the code runs in time independent of the input. Concerning the square operation, let us consider more generally the mapping $s_i : U \mapsto U^{2^i}$ in $\mathbf{R}_{16}$. As $U(x) \mapsto U(x)^{2^i} \mod (1 + x^{17}) = U(x^{2^i \mod 17})$ is simply a permutation of the coefficients of $U(x)$, s_i can easily be implemented in hardware by first permuting the bits of U, considering that the 16th bit of U is set to zero (this operation requires no logical operations), and then XOR-ing the result with FFFF if the resulting 16th bit is equal to 1. Hence, computing $s_i(U)$ requires basically one 16-bit XOR operation in hardware. In software, s_i can be evaluated by using a table look-up, also with a complexity of one 16-bit XOR.

The total complexity of evaluating F' can now be estimated as follows. From the decomposition $254 = 2(1 + 2 + 2^2 + 2^3(1 + 2 + 2^2) + 2^6)$, $V = U^{254}$ can be evaluated by using the following sequence of operations :

$$\begin{aligned}
&V \leftarrow s_1(U)\\
&V \leftarrow V \otimes U\\
&V \leftarrow s_1(V)\\
&V \leftarrow V \otimes U\\
&V \leftarrow V \otimes s_3(V)\\
&V \leftarrow V \otimes s_6(U)\\
&V \leftarrow s_1(V)
\end{aligned}$$

with the total cost of four multiplications and five calls to some s_i. This yields a total complexity of roughly 4 MULX, 17 elementary 16-bit word operations, and between 4 and 9 branching instructions. Besides the fact that our method offers some resistance to DPA, it is much faster than GCD-based algorithms, like the binary GCD of [8] or a variant of [12], which would require at least about 100 16-bit word operations. It is especially interesting for software implementations on 16-bit or 32-bit microprocessors as well as for hardware implementations.

5.3 Security Analysis

We consider a power consumption model based on the Hamming weight, that is, we assume that an attacker has access at any time to the Hamming weight of the registers of the microprocessor through the power curves. The strategy of the attacker consists in averaging the Hamming weight of the registers in order to discriminate between the case $\rho(U) = 0 \mod P$ and the case $\rho(U) \neq 0 \mod P$. The inversion algorithm presented above involves 25 elementary 16-bit word manipulations (5 different 16-bit values per multiplication and 1 per s_i).

$U \neq 0 \mod P$	$U = 0 \mod P$	$U \neq 0 \mod P$	$U = 0 \mod P$
8.01562	8.03137	7.59191	7.78039
7.00973	7.27843	6.51737	6.71373
7.92926	8.59608	8.27146	7.8902
8.06434	8.28235	7.50376	7.67059
7.00063	7.10588	6.49996	6.58824
7.98235	7.85882	8.26128	8.33725
8.00197	8.03137	7.48426	7.56078
6.97292	7.0902	6.48111	6.52549
7.92929	8.59608	8.32587	7.95294
8.01562	8.03137	7.49336	7.70196
6.99863	7.3098	6.49944	6.81569
7.99385	7.95294	8.25166	8.20784
8.00083	8.03137		

Fig. 4. Average Hamming weight of each 16-bit register used in modified inversion.

Software simulation allows us to compute exactly the average Hamming weight of each of the 25 registers, as shown in Fig. 4.

Looking at the difference of average Hamming weights between the two cases, one observes a maximum difference of about 8.5%. This is a convincing empirical argument that the proposed randomization technique is sound with respect to DPA, and we emphasize that the security level increases with k. Furthermore, as the recomputation and storage of S-boxes are not needed, in order to reduce the vulnerability to higher-order DPA one should use as many mutually independent masks as practically feasible, especially in the first and the last few rounds. In particular, the additive masks used in different rounds can be made mutually independent by using two mutually independent additive masks in the upper and lower halves of Fig. 3.

6 Conclusions

Although the proposed embedded multiplicative masking countermeasure may suffice for many applications, a possibly more secure alternative is to use random binary additive masks and accordingly recomputed S-boxes stored in RAM, for each new run of AES. In fact, it is proposed in [10] to recompute only one S-box and use it repeatedly during one execution of AES. In general, if two intermediate variables both satisfy the fundamental hypothesis for DPA and are masked by the same mask, then their mutual correlation can be used to mount a second-order DPA attack similar to the one proposed in [11]. In order to avoid this attack, the input and output masks for an S-box should be mutually independent.

In principle, increasing the number of mutually independent random masks increases the resistance against higher-order DPA as well as against more sophisticated statistical analysis of power consumption curves. If different masks are generated pseudorandomly, then the security has to be examined more carefully.

With respect to the first-order and higher-order DPA, it is critical to protect the first and the last few rounds of AES by random masks, whereas the protection of intermediate rounds may be useful with respect to more sophisticated statistical power analysis. In this regard, it is safer to repeat the masks in intermediate rounds rather than in the first or the last few rounds.

Even if the same recomputed S-box is used throughout the whole AES, the (first-order) DPA attack is still prevented as it targets the individual points of power consumption curves in time. However, such a solution is vulnerable to a relatively simple second-order DPA attack, especially for implementations in which the executions of S-box transformations are well separated in time (e.g., in software or limited-space hardware).

More precisely, one can identify the execution times of any two S-box transformations in the first and/or the last round of AES, and then compare the power consumption curves at the two points when the S-box outputs (or inputs) are computed by using some simple statistic such as the average absolute value or variance of the difference. The attack is enabled by the fact that the output (or input) values of the two S-boxes are masked by the same mask. The corresponding two expanded key bytes are guessed simultaneously in order to compute the two values and the curves are then partitioned according to the bitwise XOR of these values. To increase the security, it is then desirable to randomize the order of S-box computations within a round, with preferably mutually independent randomizations in the first and the last round.

References

1. M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, "Power analysis, what is now possible...," Advances in Cryptology – Asiacrypt 2000, *Lecture Notes in Computer Science*, vol. 1976, pp. 489–502, 2000.
2. M.-L. Akkar and C. Giraud, "An implementation of DES and AES, secure against some attacks," Cryptographic Hardware and Embedded Systems – CHES 2001, *Lecture Notes in Computer Science*, vol. 2162, pp. 309–318, 2001.
3. S. Chari, C. Jutla, J. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," Advances in Cryptology – CRYPTO '99, *Lecture Notes in Computer Science*, vol. 1666, pp. 398–412, 1999.
4. J.-S. Coron, P. Kocher, and D. Naccache, "Statistics and secret leakage," Financial Cryptography – FC 2000, *Lecture Notes in Computer Science*, vol. 1962, pp. 157–173, 2001.
5. J.-S. Coron and L. Goubin, "On Boolean and arithmetic masking against differential power analysis," Cryptographic Hardware and Embedded Systems – CHES 2000, *Lecture Notes in Computer Science*, vol. 1965, pp. 231–237, 2000.
6. J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999, available at `http://www.nist.gov/aes/`.
7. L. Goubin and J. Patarin, "DES and differential power analysis: The duplication method," Cryptographic Hardware and Embedded Systems – CHES '99, *Lecture Notes in Computer Science*, vol. 1717, pp. 158–172, 1999.

8. D. E. Knuth, *The Art of Computer Programming*, Vol. 2, Addison-Wesley, Reading, MA, 1973.
9. P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” Advances in Cryptology – CRYPTO ’99, *Lecture Notes in Computer Science*, vol. 1666, pp. 388–397, 1999.
10. T. Messerges, “Securing the AES finalists against power analysis attacks,” Fast Software Encryption - FSE 2000, *Lecture Notes in Computer Science*, vol. 1978, pp. 150–164, 2001.
11. T. Messerges, “Using second-order power analysis to attack DPA resistant software,” Cryptographic Hardware and Embedded Systems – CHES 2000, *Lecture Notes in Computer Science*, vol. 1965, pp. 238–251, 2000.
12. R. Schroeppel, H. Orman, S. O’Malley, and O. Spatscheck, “Fast key exchange with elliptic curve systems,” Advances in Cryptology - CRYPTO ’95, *Lecture Notes in Computer Science*, vol. 963, pp. 43–56, 1995.
13. J. H. Silverman, “Fast multiplication in finite fields $\mathrm{GF}(2^N)$,” Cryptographic Hardware and Embedded Systems - CHES ’99, *Lecture Notes in Computer Science*, vol. 1717, pp. 122–134, 1999.