Simplified Adaptive Multiplicative Masking for AES

Elena Trichina, Domenico De Seta, and Lucia Germani

Cryptographic Design Center, Gemplus Technology R & D
Via Pio Emanuelli 1, 00143 Rome, Italy
{elena.trichina,domenico.deseta,lucia.germani}@gemplus.com

Abstract. Software counter measures against side channel attacks considerably hinder performance of cryptographic algorithms in terms of memory or execution time or both. The challenge is to achieve secure implementation with as little extra cost as possible. In this paper we optimize a counter measure for the AES block cipher consisting in transforming a boolean mask to a multiplicative mask prior to a non-linear Byte Substitution operation (thus, avoiding S-box re-computations for every run or storing multiple S-box tables in RAM), while preserving a boolean mask everywhere else. We demonstrate that it is possible to achieve such transformation for a cost of two additional multiplications in the field.
However, due to an inherent vulnerability of multiplicative masking to so-called zero attack, an additional care must be taken to securize its implementation. We describe one possible, although not perfect, approach to such an implementation which combines algebraic techniques and partial re-computation of S-boxes. This adds one more multiplication operation, and either occasional S-box re-computations or extra 528 bytes of memory to the total price of the counter measure.

1 Introduction

With the increasing research endeavors in the field of *side-channel attacks* both hardware and software implementations of cryptosystems have to take into account various counter measures. The main techniques are timing attacks [10], simple (SPA) and differential power analysis (DPA) [11], and electromagnetic attacks [7] . A particularly worrying factor is that the first three attacks can be mounted using cheap resources. The last one requires more sophisticated set-up, including the design of special probes and development of advanced measurements methods.

In what follows we do not describe how the attacks work; papers [10,11,9,13] provide an excellent study of this topic; we just outline their main principles. Side-channel attacks work because there is a correlation between the physical measurements taken during computations (e.g., power consumption, computing time, EMF radiation, etc.) and the internal state of the processing device, which is itself related to the secret key. An SPA is an attack where the adversary can

© Springer-Verlag Berlin Heidelberg 2003

directly use a single power consumption signal to break a cryptosystem. For example, if an implementation of a cryptographic primitive includes branches that depend on the secret data, particularly, if the bodies of the *'then'* and *'else'* branches differ, an SPA attack can be successfully mounted with very inexpensive resources.

A DPA attack uses statistical analysis to extract information from a collection of power consumption curves obtained by running an algorithm many times with different inputs. Then the analysis of the probability distribution of points on the curves is carried on. The DPA uses a correlation between power consumption and specific key-dependent bits which appear at known steps of the encryption computations. For example, a selected bit b at the output of one S-box of the first round of the Advanced Encryption Standard (AES) [6] will depend on the known input message and 8 unknown bits of the key. The correlation between power consumption and b can be computed for all the 256 values of 8 unknown bits of the key. The correlation is likely to be maximal for the correct guess of the 8 bits of the key. Then the attack can be repeated for the remaining S-boxes.

It has been claimed that all "naive" implementations can succumb to attacks by power analysis technique. The only solution is to re-implement cryptosystems taking into account a wide range of counter measures, although the cost in terms of performance and memory usage can be high. General strategies to combat side-channel attacks are [9]:

- de-correlate the output traces on individual runs (e.g., by introducing random timing shifts and wait states, inserting dummy instructions, randomization of the execution of operations, etc.);
- replace critical assembler instructions with ones whose "consumption signature" is hard to analyze, or re-engineer the critical circuitry which performs arithmetic operations or memory transfers;
- make algorithmic changes to the cryptographic primitives so that attacks are provably inefficient on the obtained implementation, e.g., masking data and key with random mask generated at each run.

It had been shown [3,9] that among these, algorithmic techniques are the most versatile, all-pervasive, and may be the most powerful. Also, in many contexts it is the cheapest to put in place.

In [1], Akkar and Giraud described a practical implementation of the AES using a new *adaptive masking method.* The idea is the following: the message is masked by means of a traditional XOR operation with some random X at the beginning of the algorithm; and thereafter everything is almost as usual. The XOR operation is compatible with the AES structure except for an inversion in the field; hence the mask must be arithmetic on $GF(2^8)$. For this, the authors devised a technique of transforming a boolean mask into a multiplicative mask, namely a *modified byte substitution.* Of course, the value of the mask at some fixed step (e.g., at the end of the round) must be known in order to re-establish the expected value at the end of the execution. Fig. 1 illustrates the difference between one round of the AES with and without masking counter measure. In what follows we review the proposed counter measure and suggest a new solution

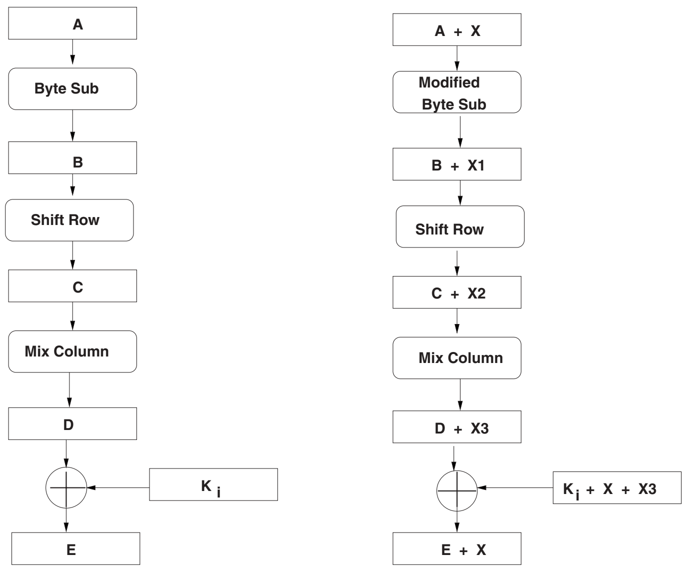

Fig. 1. One round of the AES with and without multiplicative masking counter measure.

based on the same idea; a solution that significantly simplifies the structure of the algorithm and reduces the number of expensive field operations. After which we conduct a security analysis of the simplified method and propose some techniques for its secure implementation.

2 Adaptive Masking Method for AES

The block cipher Rijndael [6] became an official new advanced encryption standard (AES) in 2001. It means that the AES will be used as the standard cryptographic algorithm for financial transactions, for telecommunication applications, and in many areas where DES is currently used. The large potential market is making it worthwhile for chip manufactures to run AES on their Smart Card micro-controllers. Since Smart Cards are easy victims to side-channel attacks, implementation of counter measures is mandatory. However, a price to pay must not be prohibitive for devices such as Smart Cards that have limited memory; and their on-line usage requires reasonable time performance.

2.1 The Rijndael Round

For simplicity, we consider the 128-bit block- and and key sizes version on the basis that the cryptanalytic study of the Rijndael during the standardization process was primarily focused on this version. For a complete mathematical specification of the Rijndael algorithm we refer readers to [6]. An encryption module is shown in Fig. 2. The total number of rounds (counting the extra round performed at the end of enciphering) is 10, the key block length and data block length are both equal to 4.

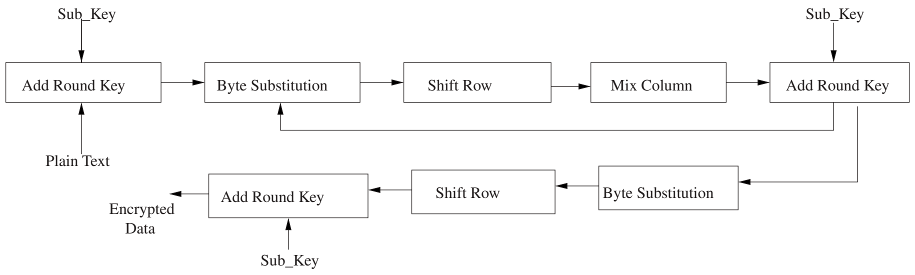

Fig. 2. The main flow of the algorithm.

In the Rijndael, the 128-bit data block is considered as a 4×4 array of bytes. The algorithm consists of an initial data/key addition, 9 full rounds (when the key length is 128 bits), and a final (modified) round. A separate key scheduling module is used to generate all the sub-keys, or *round keys*, from the initial key; a sub-key is also represented as 4×4 array of bytes. The full Rijndael round involves four steps.

The *Byte Substitution* step replaces each byte in a block by its substitute in an S-box. The S-box is an invertible substitution table which is constructed by a composition of two transformations, as Fig. 3 illustrates:

– First, each byte $A_{i,j}$ is replaced with its reciprocal in $GF(2^8)$ (except that 0, which has no reciprocal, is replaced by itself).
– Then, an affine transformation f is applied. It consists of
  - a bitwise matrix multiply with a fixed 8×8 binary matrix M,
  - after which the resultant byte is XOR-ed with the hexadecimal number $'63'$.

The S-box is usually implemented as a look-up table consisting of 256 entries; each entry is 8 bits wide; but it also can be computed “on-a-fly”. Although the latter takes more time, it saves memory.

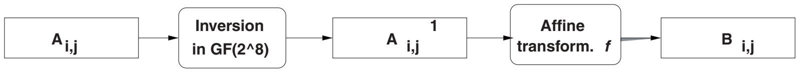

Fig. 3. Two steps of Byte Substitution transformation.

Next comes the *Shift Row* step. Each row in a 4×4 array of bytes of data is shifted 0, 1, 2 or 3 bytes to the left in a round fashion, producing a new 4×4 array of bytes.

In the *Mix Column* step, each column in the resultant 4×4 array of bytes is considered as polynomial over $GF(2^8)$ and multiplied modulo x^4+1 with a fixed polynomial $c(x) =' 03'x^3 +' 01'x^2 +' 01'x +' 02'$. The operation of a multiplication with a fixed polynomial $a(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0$ can be written as a matrix multiplication where the matrix is a circular matrix with the first row equal to a_0, a_3, a_2, a_1, each subsequent row is obtained by a circular shift of the previous one by 1 position to the left. Since multiplication is carried out in $GF(2^8)$, the product is calculated modulo irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$, or $'11B'$ in hexadecimal representation.

The final step, *Add Round Key*, simply XOR-es the result with the sub-key for the current round.

In parallel to the round operation, the round key is computed in the *Key Scheduling Block*. The round key is derived from the cipher key by means of key expansion and round key selection.

Round keys are taken from the expanded key (which is a linear array of 4-byte words) in the following way: the first round key consists of the first N_b words, the second of the following N_b words, etc. The first N_k words are filled in with the cipher key. Every following word $W[i]$ is obtained by XOR-ing the words $W[i-1]$ and $W[i-N_k]$.

For words in positions that are multiples of N_k, the word is first rotated by one byte to the left; then its bytes are transformed using the S-box from the *Byte Substitution* step, after which XOR-ed with the round-dependent constant.

2.2 Adaptive Multiplicative Masking

It is easy to see that the problem of implementing a masking counter measure comes from the *Byte Substitution* transformation, which is the only non-linear part. One known solution [13] consists in masking a table look-up T which implements the S-box with two boolean masks, the input mask R_{in} and the output mask R_{out}, as follows: $T[A_{i,j}] = T'[A_{i,j} \oplus R_{in}] \oplus R_{out}$. This implies that the masked table must be computed for each pair R_{in}, R_{out}. If done "on-a-fly", it takes time. Another solution is to fix a pair R_{in}, R_{out} prior each run, and pre-compute table look-ups for all such pairs. If one wants to mask every byte in 128-bit data, it would require as much as 256×16 bytes, or 4K of memory, which is not desirable for memory-limited devices like Smart Cards.

[1] suggests a method that allows to obtain the scheme without S-box re-computations. The message is masked at the beginning of the algorithm by XOR-ing it with a random value, generated for every new run; and thereafter everything is nearly as usual.

Since the mask must be arithmetic on $GF(2^8)$, the transformation *"boolean mask to multiplicative mask"* is devised such that the first step of the *Byte Substitution*, namely, the inversion in $GF(2^8)$, produces a masked multiplicative inverse of the input data, as shown in Fig. 4. Here $X_{i,j}$ is an 8-bit random

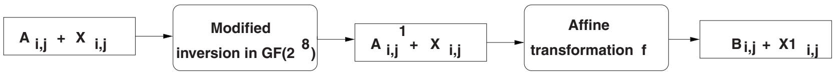

Fig. 4. Modified Byte Substitution with masking counter-measure.

value which masks data $A_{i,j}$ and $X1_{i,j} = f(X_{i,j})$ (this comes from the affine property of f). Then, the reverse transformation *"multiplicative mask to boolean mask"* is performed to restore an additive mask on the inverse data before an affine transformation f takes place. The full scheme of the modified inversion

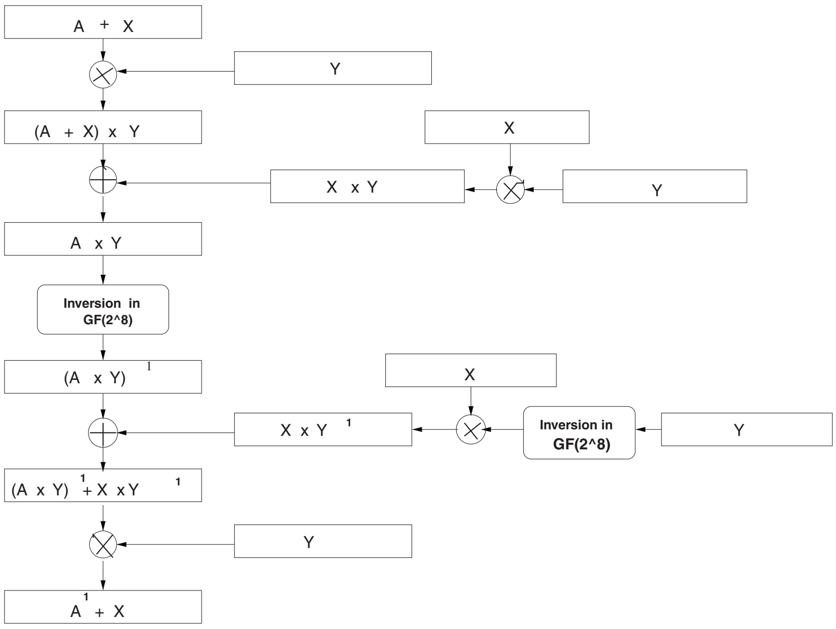

Fig. 5. Modified inversion in $GF(2^8)$ with masking counter measure.

is depicted in Fig. 5. As one can see, it requires an additional random variable, $Y_{i,j}$, one additional inversion, and 4 extra multiplications in the field. During all stages of the modified inversion, intermediary values seem to be independent from $A_{i,j}$.

While the masking methods, like in [13], must respect a masking condition at each step of the algorithm, using the transformed method one only needs to know the value of the mask at a fixed step (e.g., at the end of the round, or at the end of a non-linear part). The expected value is re-established after the computations at the end of the algorithm.

3 Simplified Multiplicative Masking

The idea of the simplified transformed masking is the same as the one described above. At the beginning of the byte substitution, the input value is $A_{i,j} \oplus X_{i,j}$, where $X_{i,j}$ is a random byte (we can safely drop indices i, j). We found a very efficient method that allows us to have $A^{-1} \oplus X$ at the end of the inversion without compromising value A. It can be described using solely algebraic laws for operations in finite fields.

Let us approach our goal from two directions simultaneously: from the input $A \oplus X$ working forwards to get $A^{-1} \oplus X^{-1}$, and from the output $A^{-1} \oplus X$ working backwards, also towards $A^{-1} \oplus X^{-1}$.

1. Suppose, we managed to obtain $A \otimes X$ from $A \oplus X$ without compromising A; then applying inversion in $GF(2^8)$, we get $A^{-1} \otimes X^{-1}$. Here how it can be done.
   a) We want to have a multiplicative masking $A \otimes X$ from an additive masking $A \oplus X$. A distributivity law $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ gives us the idea. Substituting A for a and X for c, we get
   $$(A \oplus X) \longrightarrow (A \oplus X) \otimes X = A \otimes X \oplus X^2.$$
   b) To obtain a pure multiplicative mask, we have to get rid of X^2. An algebraic law $a \oplus a = 0$ can be applied here
   $$A \otimes X \oplus X^2 \longrightarrow A \otimes X \oplus X^2 \oplus X^2 = A \otimes X.$$
   c) At this stage, one can safely apply inversion in $GF(2^8)$
   $$A \otimes X \longrightarrow (A \otimes X)^{-1} = A^{-1} \otimes X^{-1}.$$
2. Now we face a symmetric task of obtaining the additive mask $A^{-1} \oplus X$ from the multiplicative mask $A^{-1} \otimes X^{-1}$. The algebraic law $x^{-1} \otimes x = 1$ will help to get rid of the X^{-1}; but before doing so, an additional step must be taken.
   a) Ensure that the value A^{-1} will not be revealed in the process:
   $$A^{-1} \otimes X^{-1} \longrightarrow A^{-1} \otimes X^{-1} \oplus 1.$$
   b) Now we can get rid of X^{-1}:
   $$A^{-1} \otimes X^{-1} \oplus 1 \longrightarrow (A^{-1} \otimes X^{-1} \oplus 1) \otimes X = A^{-1} \otimes 1 \oplus X = A^{-1} \oplus X.$$

Fig. 6 depicts this method graphically. Compare it with Fig. 5. As one can see, ours is a significant simplification of the solution in [1]; it requires no extra inversion in $GF(2^8)$, and only two extra multiplications and one squaring.

One can argue that the first step in the “boolean mask to multiplicative mask” transformation, can jeopardize the security of the masking because $(a \oplus x) \otimes x$ is not fully random for a random x. Indeed, as has been pointed out in [5], the equation $(a \oplus x) \otimes x = y$ has either zero solution, or two solutions; namely,

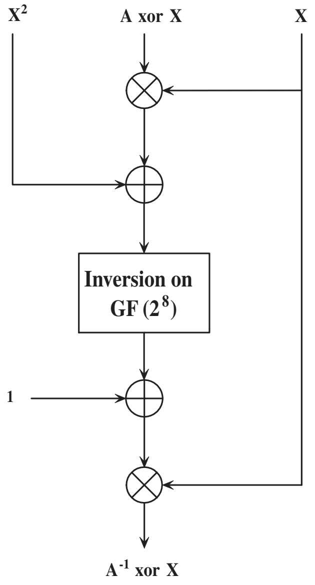

Fig. 6. Inversion in $GF(2^8)$ with simplified multiplicative masking.

substituting x with $z \otimes a$ in the equation above, we obtain $z^2 \oplus z = y/a^2$. We know that the equation $z^2 \oplus z = b$ has zero solution if $Trace(b) = 1$, and two solutions if $Trace(b) = 0$. So $(a \oplus x) \otimes x$ reaches only half of the elements in $GF(2^8)$. However, it is sufficiently random to serve the purpose.

If the original additive mask is restored at the end of the round, as illustrated in Fig. 1, there are no limitations on the choice of the random 128-bits mask X apart from the requirements that none of its bytes $X_{i,j}$ is equal to zero.

However, one can imagine the computation scheme, where, instead of restoring the original mask $X_{i,j}$ at the end of each round, one simply goes on with the computations, taking the value $X1_{i,j} = f(X_{i,j})$ as the mask for the second round, $X2_{i,j} = f(X1_{i,j})$ for the third, etc.

Only at the end of the computations the data is unmasked. The corresponding "correction" on the mask has to be carried out in parallel with the main algorithm. In this case, due to the nature of the *Mix Column* and *Shift Row* operations, some of the random bytes $XK_{i,j}$ for the K-th round, $K = 2, ...9$, can turn to zeros.

A mathematical analysis of the effect of the *Mix Column* and *Shift Row* operations on bytes of the mask indicates, and a simple computer experiment with all possible choice of random bytes for a 128-bit random confirms, that the sufficient condition which effectively prevents this from happening is that no two bytes of the initial 128-bit random X should be the same.

4 Securized Implementation of the Simplified Multiplicative Masking

Let us ask a question: how to implement the simplified multiplicative masking in a secure way? A straightforward implementation can lead to a potential security flaw, as had been pointed out in [2,4,8]. The flaw consists in the fact that multiplicative mask masks only **non-zero values**, i.e., zero input value is mapped into zero by the inversion. In other words, if an attacker can detect that the value before (i.e., $A_{i,j} \otimes X_{i,j}$) and after (i.e., $(A_{i,j} \otimes X_{i,j})^{-1}$) the inversion is 0, he/she gets an information on $A_{i,j}$. An attacker can exploit this fact and mount the first order DPA attack as if no masking has been applied.

Hence, not to reveal the weakness, nowhere in the implementation there should be a moment where both, $A_{i,j} \otimes X_{i,j}$ and $(A_{i,j} \otimes X_{i,j})^{-1}$, are read or written in clear. In other words, the counter measure shown in Fig. 6 must be implemented in a completely protected environment. Currently we are working on such an implementation. The obvious solution stems from the nature of our simplified masking.

Indeed, since the constant 1 is to be added to every entry of the inverse table (see Fig. 6), it can be done in advance, while creating the table itself. This, however, may not protect from the attacker who now, instead of looking for 0 as the resulting value of the table lookup will look for 1. The situation can be remedied as follows.

– Prior to a run of the AES algorithm, all entries of a table are XOR-ed with some random constant value K. Then the result of the table lookup for $(A_{i,j} \otimes X_{i,j})$ will be $(A_{i,j} \otimes X_{i,j})^{-1} \oplus K$.
– The subsequent multiplication with $X_{i,j}$ produces $((A_{i,j})^{-1} \oplus (K \otimes X_{i,j}))$, which either can be carried to the affine transformation f with $K \otimes X_{i,j}$ as a new random or can be replaced with $X_{i,j}$ by further XOR-ing the table lookup result with $X_{i,j} \oplus (K \otimes X_{i,j})$.

This adds 256 bytes and one more field multiplication to the implementation; however, computing a simplified multiplicative masking is still more efficient than re-computing S-boxes for every run.

Still, the problem remains: how not to reveal $A_{i,j} \otimes X_{i,j}$ during computations? We do not have a good answer to this question yet, which undoubtedly weakens our counter measure. To prevent reading $A_{i,j} \otimes X_{i,j}$ in clear from the inverse table T, the upper-most operation $\oplus$ with X^2 in Fig. 6 must never be actually performed.

One solution is that operation $\oplus$ with $(X_{i,j}^2 \oplus M)$ for some a-priori chosen M is carried out, simultaneously updating T in such a way that for a new table T': $T'[B \oplus M] = T[B]$. M could be chosen so that this re-computation amounts to simple re-shufling of the indices; for example, if $M = 1$, T' is obtained from T by simply “swapping” each even and odd entries. Obviously, from time to time, M and, respectively, the table T' must be re-newed.

Another solution brings us back to S-box re-computations: instead of recovering $A_{i,j} \otimes X_{i,j}$ from $A_{i,j} \otimes X_{i,j} \oplus X_{i,j}^2$ a new table T' such that $T'[A_{i,j} \otimes X_{i,j} \oplus$

$X_{i,j}^2] = T[A_{i,j} \otimes X_{i,j}]$ is computed. Since only the first and the last rounds are most vulnerable to the DPA, it seems enough to apply re-computations (or store two extra pre-computed S-boxes) only for these rounds.

A general algorithm to compute such T' given some table T and a random value X is described below.

Look-up table re-computation.

```
Input:  table T;
        random X = (x_7, ..., x_1, x_0)
Output: table T' such that T'[b+X] = T[b] for b = 0..255
   T' := T;
   For every x_i from (x_7, ..., x_0) in random order do:
      If x_i = 1 then
         (1) split T' into blocks, each block containing 2^(x_i)
             subsequent elements from T;
         (2) swap pairwise j-th and j+1-st blocks;
         (3) assign the result to T';
   Return T'
```

Notice, that the algorithm reads bits of X at random which provides some protection from an attacker during re-computations.

The proposed securized implementation of the simplified adaptive masking is computationally more efficient than full S-box re-computations, thus representing a compromise between cost and security.

5 Conclusion

We have shown that the *Modified Byte Substitution* can be implemented in a way that to some degree avoids a severe security flaw paying a price of additional multiplications and RAM usage.

However, many security features are a matter of trade-offs. Described in this paper implementation of the simplified multiplicative masking provides similar protection as an S-box re-computation but has lower implementation costs, in terms of both, memory and execution time; namely, the price to pay is, apart from numerous XOR operations, only three to four extra multiplications in $GF(2^8)$ per round plus an occasional re-shuffling of the inverse table stored in ROM. While may not ensuring a complete protection from a sophisticated attacker due to an inherent vulnerability of a multiplicative masking in the field, the method increases the number of power curves acquisitions and thus can be sufficient for a low-end line of Smart Cards.

Another, quite different solution would be not to work in the field $GF(2^8)$, but in the field $GF(2^8 + a)$ where a is chosen such that $2^8 + a$ is prime. The we can avoid zero by replacing it by $2^8 + 1$. This is what had been done in IDEA cipher as a solution to the problem of inverting a number modulo 2^{16} during decryption [12].

References

1. Akkar, M., Giraud, C.: An implementation of DES and AES, secure against some attacks. Proc. *Cryptographic Hardware and Embedded Systems: CHES 2001*, LNCS **2162** (2001) 309–318
2. Akkar, M., Goubin, L.: A generic protection against high-order differential power analysis. *Manuscript*
3. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. Proc. *Advances in Cryptology – Crypto'99*, LNCS **1666** (1999) 398–412
4. Courtois, N., Akkar, M.: Time and memory efficiency in protecting against higher order power attacks. *Manuscript*
5. Coron, J.-S., personal communication
6. Daemen, J., Rijmen, V.: *The design of Rijndael: AES – The Advanced Encryption Standard*. Springer-Verlag Berlin Heidelberg, 2002
7. Gandolfi, K., Mourtel, C., Oliver, F.: Electromagnetic analysis: concrete results. Proc. *Cryptographic Hardware and Embedded Systems: CHES 2001*, LNCS **2162** (2001) 251–261
8. Golic, J., Tymen, Ch.: : Multiplicative masking and power analysis of AES. Proc. *Cryptographic Hardware and Embedded Systems: CHES 2002*, LNCS **2523** (2002) *These proceedings.*
9. Goubin, L., Patarin, J.: DES and differential power analysis. Proc. *Cryptographic Hardware and Embedded Systems: CHES'99*, LNCS **1717** (1999) 158–172
10. Kocher, P.: Timing attacks on implementations of Diffie-Hellmann, RSA, DSS, and other systems. Proc. *Advances in Cryptology – Crypto'96*, LNCS **1109** (1996) 104–113
11. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. Proc. *Advances in Cryptology – Crypto'99*, LNCS **1666** (1999) 388–397
12. Massey, J. L., Lai, X.: Device for the conversion of a digital block and use of same. U.S. Patent # 5,214,703, 25 May 1993
13. Messerges, T.: Securing the AES finalists against power analysis attacks. Proc. *Fast Software Encryption Workshop 2000* LNCS **1978** (2000) 150–165