An Implementation of DES and AES, Secure against Some Attacks

Mehdi-Laurent Akkar[1] and Christophe Giraud[2]

[1] Schlumberger CP8,
68 route de Versailles, 78431 Louveciennes, France.
ml.akkar@free.fr
[2] Oberthur Card Systems,
25, rue Auguste Blanche, 92800 Puteaux, France.
c.giraud@oberthurcs.com

Abstract. Since Power Analysis on smart cards was introduced by Paul Kocher [7], many countermeasures have been proposed to protect implementations of cryptographic algorithms. In this paper we propose a new protection principle: the transformed masking method. We apply this method to protect two of the most popular block ciphers: DES and the AES Rijndael. To this end we introduce some transformed S-boxes for DES and a new masking method and its applications to the non-linear part of Rijndael.

Keywords: AES, Rijndael, DES, Transformed mask, Multiplicative mask, Power analysis, DPA, SPA, Smart Cards.

1 Introduction

Since Kocher, Jaffe and Jun introduced Differential Power Analysis, many countermeasures have been proposed to protect the card against power analysis type attacks (SPA, DPA, HODPA):

- insertion of dummy instructions;
- randomization of operations;
- transformation of the data (i.e. Duplication Method [6]);
- masking of the data [2,3]: boolean, arithmetic...

In this paper we present a practical implementation of DES ([10]) and AES ([5]) using some of these countermeasures combined with new methods. We will essentially use a new idea -an adapted masking method- combined with a bit-per-bit randomization of many operations during the computation.

2 Transformed Masking Method

2.1 Principle

The idea is the following: the message is masked at the beginning of the algorithm and after this, everything (or nearly) is as usual. Most of the previous proposed methods must respect a masking condition at each step of the

Ç.K. Koç, D. Naccache, and C. Paar (Eds.): CHES 2001, LNCS 2162, pp. 309–318, 2001.

algorithm, but here we only need to know the value of the mask at a fixed step (for example at the end of a round or at the end of a non-linear part) and we reestablish the expected value at the end of the algorithm.

It is easy to see that the problem of implementing a masking countermeasure comes from the non-linear parts of the algorithm. Furthermore the security of a symmetric cryptographic algorithm is essentially based in these parts.

2.2 DES

In the case of the DES, the most appropriate mask is a boolean mask X which is applied before the Initial Permutation IP (we XOR the 64-bit message M with a 64-bit value X). The only non-linear part of the DES is the S-Box, so when using a masking countermeasure, we use a modified S-Box. This also enables us to reestablish the mask value. It is only after the Final Permutation FP that the mask will be removed to obtain the right result.

2.3 AES

For this algorithm, the method is slightly different. We still use a XOR operation as a masking countermeasure, but now the mask is arithmetic on GF(2^8). This operation is compatible with the AES structure except for the inversion in GF(2^8). For this, we use a new technique to transform the boolean mask into a multiplicative mask. This allows us to keep the same level of security throughout the algorithm. As for the DES, the mask will be reestablished at the end of each round and the value will be unmasked at the end of the algorithm.

3 Applications

3.1 Securing the DES

DES Structure We want to cipher a 64-bit message M with the DES. We choose a 64-bit mask X which will be XOR-ed with the message M at the beginning of the DES. Then we start with the value $M \oplus X$.

Just before the S-box, it is easy to see (fig.1) that we have an intermediary value masked with $X2 = EP(X1_{32-63})$ where:

- $X1$ represents the 64-bit value $IP(X)$;
- IP represents the initial permutation;
- $X1_{0-31}$ (respectively $X1_{32-63}$) represents the 32-bit low-weight (respectively high-weight) part of the 64-bit mask X;
- $X2$ represents the 48-bit value $EP(X1_{32-63})$;
- EP represents the expansive permutation of a DES round.

The method chosen in the case of the DES is to reestablish the mask $X1$ at each round. To obtain this result, we will use a modified S-box, denoted SM-Box. The output of the SM-Box, after the permutation P and after being XOR-ed

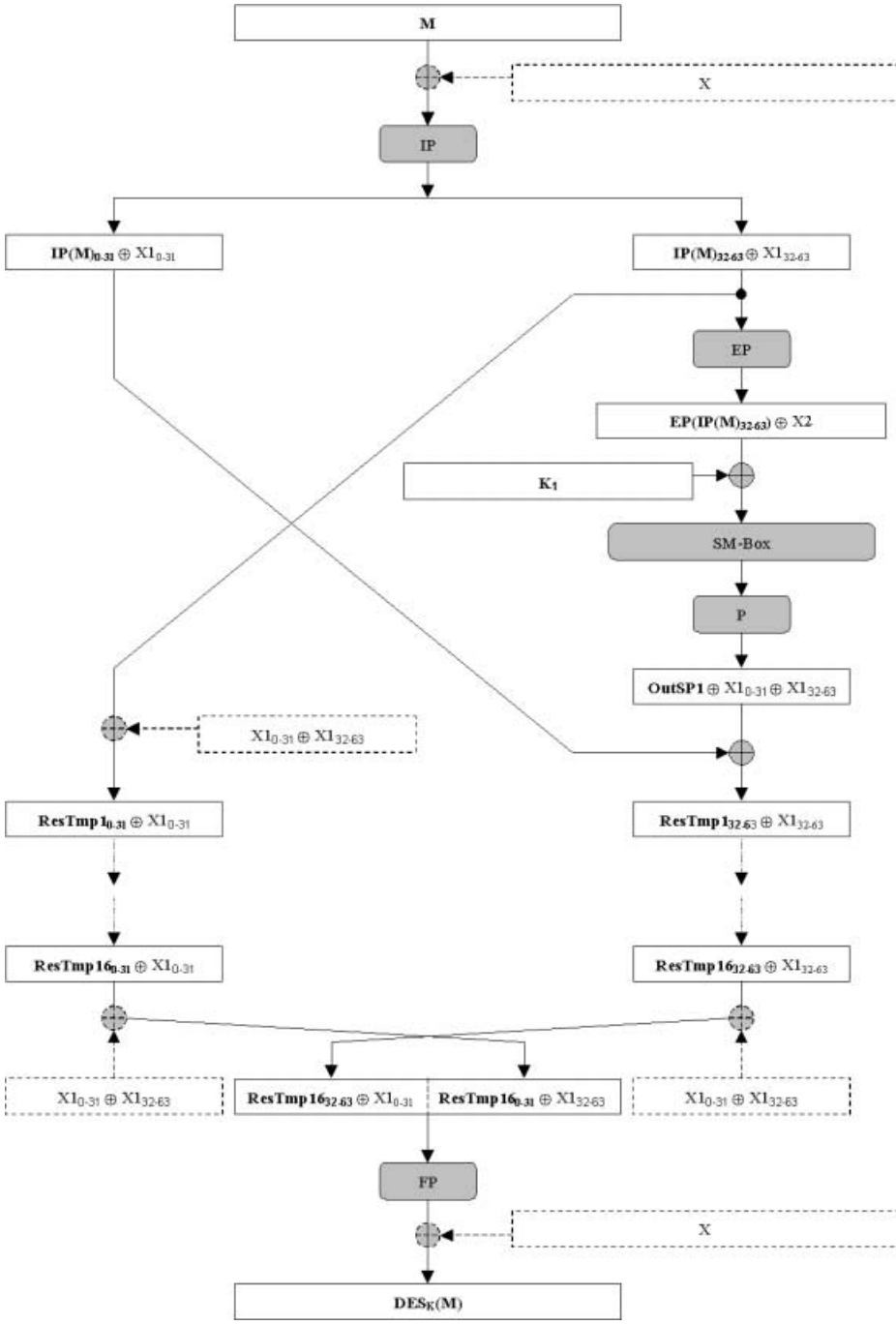

Fig. 1. Differences between a DES without countermeasure and a DES with masking countermeasure.

with the left part of the message, must have a mask corresponding to $X1_{32-63}$. So, the SM-box is now defined by:

$$SM\text{-}Box(A) = S\text{-}Box(A \oplus X2) \oplus P^{-1}(X1_{0-31} \oplus X1_{32-63}) \quad (1)$$

where P^{-1} is the inverse of the permutation P applied after the S-Box.

It is also necessary to modify the left part of the message, we XOR it with $X1_{0-31} \oplus X1_{32-63}$. So, at the end of the round, the mask $X1$ will be preserved.

After the last round, the two 32-bit parts are interchanged, so it will be necessary to XOR these two parts with the 32-bit value $X1_{0-31} \oplus X1_{32-63}$ before the final permutation FP. The correct cipher is obtained after the final permutation by unmasking the value with the 64-bit mask X.

To sum up, the following scheme represents the differences between a DES without countermeasure and a DES with masking countermeasure. Operations added to implement the masking countermeasure are represented with dotted lines.

Mask Preparation It is important to note that the preparation of the different values used as mask during the cipher (X, $X1$ and $X2$) must be computed in a very secure way. Indeed, if an attacker succeeds in finding X, $X1$ or $X2$ then he will be able to break our countermeasure. So the method used is to compute these values with a randomized bit-per-bit calculation. This computation is slow but is done only once at the beginning of each DES and guarantees high-level security. In the worst case, the attacker will learn only the Hamming weight of the mask.

Put into Practice We have implemented this countermeasure twice. The first implementation was done entirely in C using a 32-bit risc CPU and the second was done in assembly code using another 32-bit risc CPU with specialized assembly instructions to facilitate a DES implementation.

For the first implementation we obtain the following results:

Type of DES	Timing at 5 Mhz	Space of ROM in bytes	Space of RAM in bytes
Normal DES	9.4 ms	1540	42
DES with CM1	18.6 ms	2660	187
DES with CM2	21.2 ms	2656	452

Fig. 2. Timings and memory space used for non-optimized C code.

And for the second implementation we obtain:

Type of DES	Timing at 5 Mhz	Space of ROM in bytes	Space of RAM in bytes
Normal DES	46.2 μs	596	16
DES with CM2	237.6 μs	2017	272

Fig. 3. Timings and memory space used for assembly code.

Where:

- Normal DES : a non-optimized implementation without countermeasure. This DES served as a basis for the construction of the "secure" implementations,
- CM1 : classical countermeasure (cf. [1]) where the message or its complement is ciphered,
- CM2 : DES with the masking countermeasure on the message and with randomization.

3.2 AES

For the AES algorithm, the method is close to the one used for the DES when we want to secure the affine and linear parts of the algorithm: we simply keep the same mask at each round. Next, we show how to deal with the non-linear parts.

The first part of an AES round is the *ByteSub* transformation which is the only non-linear part of the AES. It is an S-Box which is the composition of two transformations (a multiplicative inversion in GF(2^8) and an affine transformation f) applied on each byte $A_{i,j}$ of the input A:

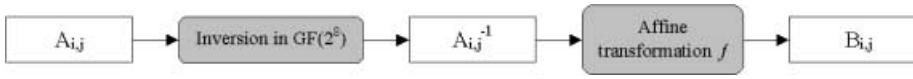

Fig. 4. The ByteSub transformation.

With the masking countermeasure, we want to obtain the following scheme where $X_{i,j}$ is the 8-bit value which masks $A_{i,j}$ and $X1_{i,j} = f(X_{i,j}) \oplus 0x63$ (this comes from the affine property of f):

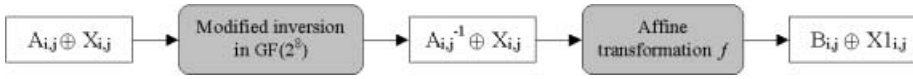

Fig. 5. The ByteSub transformation with masking countermeasure.

We must resolve the following problem: how to obtain ${A_{i,j}}^{-1} \oplus X_{i,j}$ when we have $A_{i,j} \oplus X_{i,j}$ without compromising the 8-bit value $A_{i,j}$.

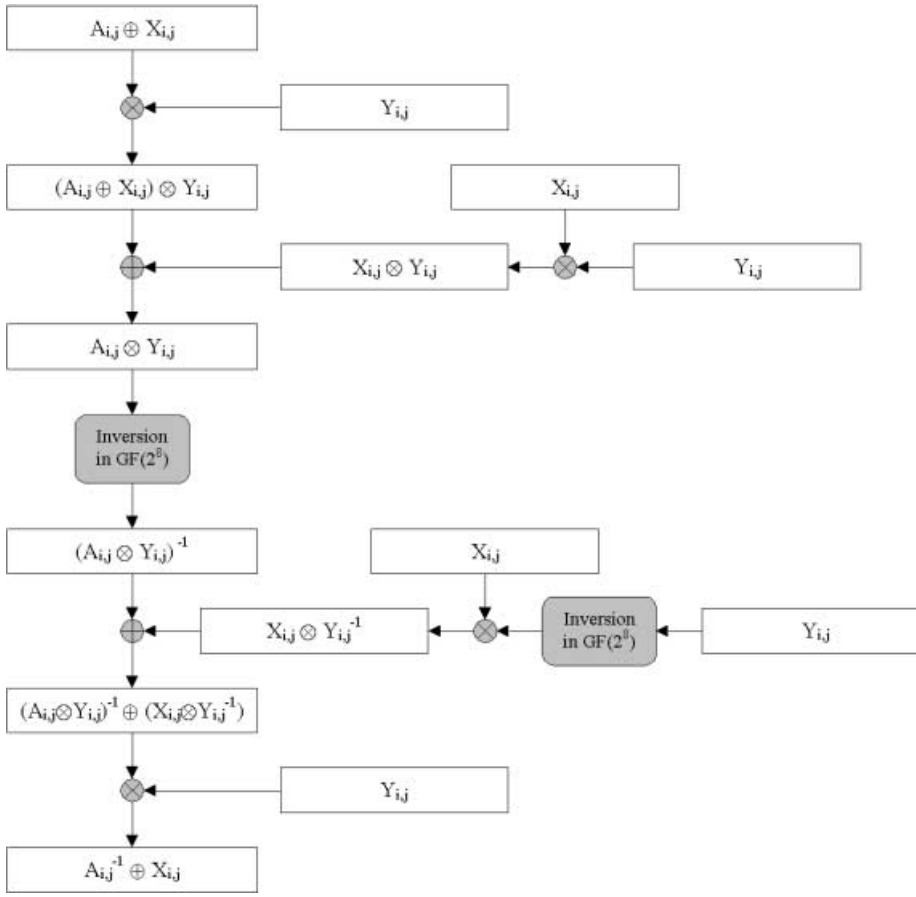

Fig. 6. Modified inversion in GF(2^8) with masking countermeasure.

The first idea is, like in the DES algorithm, to use a modified S-Box computed each time we start an AES. However, in the AES case, the size of the table goes from 256 bytes to $(256 * 16)$ bytes, equal to 4 Ko when we choose a 128-bit message. This solution is not possible when working in a smart card environment (this table is dynamic and must be located in RAM).
The approach selected is the following: an operation compatible with inversion is multiplication, so we obtain the trivial formula: $(X.Y)^{-1} = X^{-1}.Y^{-1}$. The problem lies in transforming a boolean mask into a multiplicative mask. If we denote by $Y_{i,j}$ an 8-bit random different from zero and by $\otimes$ the multiplication in GF(2^8) using the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ as modulus, the mask transformation is obtained as follows:

During all stages of the transformation boolean mask / multiplicative mask, intermediary values are independent of $A_{i,j}$:

1. we multiply with a non-zero 8-bit random $Y_{i,j}$,
2. and we XOR with $X_{i,j} \otimes Y_{i,j}$.

After the inversion in GF(2^8) we have a multiplicative mask and to reestablish the boolean mask we use values independent of $A_{i,j}$:

1. we XOR with $X_{i,j} \otimes {Y_{i,j}}^{-1}$,
2. and we multiply with $Y_{i,j}$.

Now, let us see the difference between a round of the AES without countermeasure and a round with masking countermeasure (fig.7).

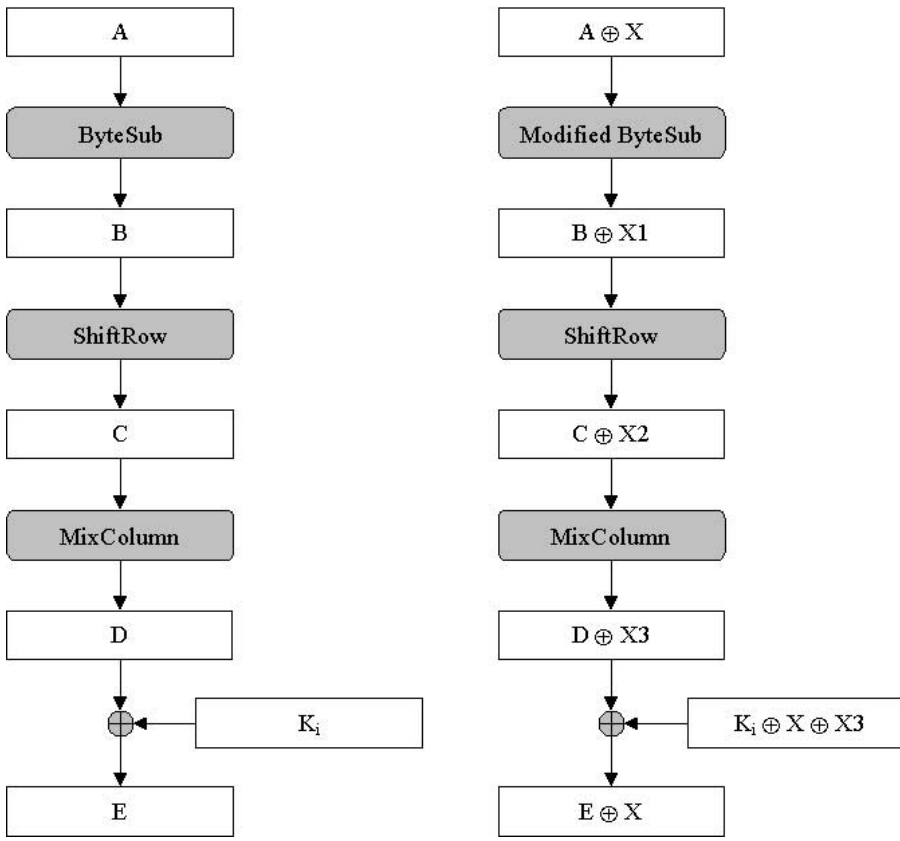

Fig. 7. The round i of the AES without and with masking countermeasure.

Where:

- X represents the mask applied;
- $X1 = f1(X)$ where $f1$ is the linear part of the affine transformation f of $ByteSub$;
- $X2 = ShiftRow(X1)$;
- $X3 = MixColumn(X2)$;
- K_i represents the round key i.

With this, it is possible to compute an AES and to keep the same random mask at each round.

Put into practice. The following timings come from the encryption of a 128-bit message using a 128-bit key. The implementation was done in assembly code using a 8-bit CPU.

Type of AES	Timing at 5 Mhz	Space of ROM in bytes	Space of RAM in bytes
Normal AES	18.1 ms	730	41
AES with CM2	58.7 ms	1752	121

Fig. 8. Timings and memory space used for assembly code.

Where:

- Normal AES : a non-optimized implementation without countermeasure,
- AES with CM2 : AES with masking countermeasure.

4 Some Security Considerations

4.1 Against SPA

All the permutations have been randomized so (presumably) the only thing the attacker can read is the Hamming weight (HW) of the permuted value. Moreover the values are masked meaning that an attacker would need to know the value of the mask too.

During the key scheduling of DES, the attacker could get the HW of the round keys K_i, construct linear equations on the 16 rounds and acquire information about the key. To avoid such problems the bits of the entire key (56 bits or even 64) are processed at each round. Therefore the only information that the attacker could obtain is the HW of the key.

During the other operations (XOR, load, store), we think that the 32-bit operations prevent an attacker from being able to get the precise 32-bit value that is loaded in one operation. But due to high-order DPA (HODPA) it would be better to randomly load, store and XOR these values bit-per-bit.

Of course, at the beginning of the DES while computing $X1$ and $X2$, one should be careful and use randomization too. It is still true for the AES; moreover the operations involved in the AES are well adapted to usual processors and the problem of the bit-permutation does not exist here.

4.2 Against DPA

We will not present the general DPA attack but simply consider the following fact: an ordinary DPA attack is based on the prediction of one intermediate value of the computation during the algorithm. Based on this fact, it seems that our implementation is fully protected against such a -simple- attack. Indeed, from the beginning of the DES (input message) to the end of it (cipher text) none of the "real" intermediate values appear.

Due to the general random mask, we are not vulnerable to the kind of attacks described in either [4] or [1]. Indeed, at each computation, unless one knows the mask used, the output of the non-linear part (S-Box for DES / *ByteSub* transformation for the AES) is random and not just masked by 0x00 or 0xFF.

Fundamentally we are subject to second-order DPA (see [8,9]) due to the method of masking. But if we consider a real high-order DPA, some other aspects appear: due to the randomization of all operations, the place, (i, j) for example, showing correlation in a HODPA attack, changes a lot. Indeed, in the general case the value i and j will both have 32 possibilities (if we consider that the bit in question is in a 32-bit value). Therefore this gives 1024 positions for the DPA peak and considerably increases a "normal" HODPA.

5 Conclusion

We have described some new ideas for a practical implementation of DES and AES: adapted mask, modified S-Box, transformation boolean mask / multiplicative mask. As is seen from the timings of our implementations, these countermeasures against SPA and DPA can be implemented in a smart-card environment where the memory space is restricted and the processor speed is slow.

References

1. M.-L. Akkar, R. Bévan, P. Dischamp, and D. Moyart. Power analysis, what is now possible. *Asiacrypt*, 2000.
2. S. Chari, C. Jutla, J.R. Rao, and P. Rohatgi. A cautionary note regarding evaluation of aes candidates on smart-cards. *The Second AES Candidate Conference*, 1999.
3. S. Chari, C. Jutla, J.R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. *Crypto*, 1999.
4. J.-S. Coron and L. Goubin. On boolean and arithmetic masking against differential power analysis. *CHES*, 2000.
5. Joan Daemen and Vincent Rijmen. The block cipher rijndael. *Web Page: http://www.esat.kuleuven.ac.be/∼rijmen/rijndael/*, 2000.
6. L. Goubin and J. Patarin. Des and differential power analysis, the duplication method. *CHES*, 1999.
7. P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. *Web Site: www.cryptography.com/dpa*, 1998.
8. P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. *Crypto*, 1999.

9. T.S. Messerges. Using second-order power analysis to attack dpa resistant software. *CHES*, 2000.
10. National Bureau of Standards. The data encryption standard. *FIPS PUB 46*, 1977.