Towards a Third Order Side Channel Analysis Resistant Table Recomputation Method

Guillaume Fumaroli, Louis Goubin, Sylvain Lachartre, and Ange Martinelli

{guillaume.fumaroli, sylvain.lachartre, jean.martinelli}@fr.thalesgroup.com,
louis.goubin@prism.uvsq.fr

Abstract. SCA exploits the instantaneous leakage from a physical implementation at a time t to recover some information about the data it processes at time t. Countermeasures against (first-order) SCA consists in splitting the internal state of the algorithm in several random shares and processing this shares individually. However, d-th order SCA that leverage the information leaked at d times can be mounted to bypass countermeasures that provide $(d-1)$-th order SCA resistance. While higher order SCA has been thoroughly investigated during the last decade, countermeasures against these attacks received much less attention. In this paper we propose an extension of a known second order countermeasure to order 3 and we give some implementation results.
Keywords: Side-Channel Attacks, CPA, MIA, Masking, Matrix Product

1 Introduction

Side Channel Analysis (*SCA* for short) is a cryptanalytic technique that exploits information leaked, during the execution of a cryptographic algorithm on an embedded system, through physical channel as power consumption or electromagnetic emanations. SCA makes use of the fact that this leakage is statistically dependent on the intermediate variables that are processed. In particular, some of these variables, called *sensitive*, are related to secret data, and recovering information on them allows efficient cryptanalysis [3, 1, 2]. We call dth-*order SCA* (written dO-SCA) a SCA that exploits leakages at d different times, which are respectively associated with d different intermediate variables. An implementation of a cryptographic algorithm is said to provide dth-*order resistance* if no d-tuple of its intermediate variables is sensitive, *i.e.* is statistically dependent of secret data and plaintext. If not, the implementation possess a *d-order flaw*. First order countermeasures are now well known and implemented. In [5] Rivain *et al.* present a S-box recomputation method proven secure against second order SCA. The authors left it as an open problem to design a third order SCA secure S-box recomputation method. This paper presents an almost third order SCA secure S-box recomputation method.

2 S-box recomputation secured against third order SCA

2.1 From second order S-box recomputation to third order

In order to extend the S-box recomputation algorithm given in [5] by Rivain *et al.* to order 3, a straightforward method consists in using two random masks where one was previously used. We denote by (α, n)-LUT a look-up table (LUT for short) with α n-bit entries. Listing the intermediate variables of this algorithm, we can state that such an extension is clearly unsecured. In order to secure the masking of the S-box output, and the manipulation of the temporary mask r', two additional temporary masks must be employed. It leads to Algorithm 1.

Algorithm 1
INPUT: $\tilde{x} = x \oplus r_1 \oplus r_2 \oplus r_3 \in \mathbb{F}_2^n$, $(r_1, r_2, r_3) \in (\mathbb{F}_2^n)^3$, $(s_1, s_2, s_3) \in (\mathbb{F}_2^m)^3$, a LUT for the (n, m)-S-box S
OUTPUT: $S(x) \oplus s_1 \oplus s_2 \oplus s_3$

1. $r_4, r_5, r_6 \leftarrow rand(\mathbb{F}_2^n)$
2. $r_7 \leftarrow rand(\mathbb{F}_2^m)$

3. $r' \leftarrow (r_1 \oplus r_4) \oplus (r_2 \oplus r_5) \oplus r_3$
4. **for** $a = 0$ to $2^n - 1$ **do**
5. $\quad a' \leftarrow a \oplus r'$
6. $\quad T[a'] \leftarrow ((S(\tilde{x} \oplus (a \oplus r_6)) \oplus (s_1 \oplus r_7)) \oplus s_2) \oplus s_3$
7. Return $T[r_4 \oplus r_5 \oplus r_6] \oplus r_7$

In spite of the previous modification, listing the intermediate variables I_j, we can state that there exist a unique third order flaw with the triplet (I_1, I_2, I_3) as $I_1 \oplus I_2 \oplus I_3 = x$:

- $I_1 = a \oplus r_1 \oplus r_4 \oplus r_2 \oplus r_5 \oplus r_3$ [step 5]
- $I_2 = r_4 \oplus r_5 \oplus r_6$ [step 7]
- $I_3 = x \oplus r_1 \oplus r_2 \oplus r_3 \oplus a \oplus r_6$. [step 6]

Let $\alpha_1 = r_1 \oplus r_2 \oplus r_3$, $\alpha_2 = r_4 \oplus r_5$ and $\alpha_3 = r_6$, then we can re-write the previous equations as :

- $I_1 = a \oplus \alpha_1 \oplus \alpha_2$
- $I_2 = \alpha_2 \oplus \alpha_3$
- $I_3 = x \oplus \alpha_1 \oplus a \oplus \alpha_3$.

In order to confirm that this is the only flaw, we used a program that checks all the second and third order dependencies to x given the list of all the intermediate variables. This flaw cannot be secured with only exclusive or (*xor* for short) masking. Indeed, masks are used to cover the progression of the sensitive variable x through the computation of $S(x)$ (steps 5,6), so x has to be unmasked at the output of the algorithm (step 7). More precisely we want to compute $S(x) \oplus s_1 \oplus s_2 \oplus s_3$ using a table T in order to avoid manipulating x directly. We then compute, for all a in the input space, $T[a] = S(x \oplus a \oplus M)$, where M is a random mask. To recover $S(x)$ we then evaluate $T[M]$. In order to give a higher security, we also mask x, a, and M with random temporary masks $(r_i)_{0 \leq i \leq n_3}$:

$$T[a] = S\left((x \bigoplus_{i=0}^{n_1} r_i) \oplus (a \bigoplus_{i=n_1}^{n_2} r_i) \oplus (M \bigoplus_{i=n_2}^{n_3} r_i) \right).$$

Moreover we have that $\bigoplus_{i=0}^{n_3} r_i = 0$ to achieve that $T[M] = S(x)$. Hence knowing $I_1 = M$, $I_2 = (a \bigoplus_{i=n_1}^{n_2} r_i) \oplus (M \bigoplus_{i=n_2}^{n_3} r_i)$, and $I_3 = x \bigoplus_{i=0}^{n_1} r_i$, we have a third order flaw.

2.2 Core Idea

In order to increase the security given by Algorithm 1 we have to introduce another operation than *xor* . The idea of the countermeasure is to mask masks with an operation invertible and distributive over *xor*, e.g. product with a well distributed invertible matrix. In the sequel we give Algorithm 2 and discuss its performances and resistance against third order SCA.

Algorithm 2

INPUT: $\tilde{x} = x \oplus r_1 \oplus r_2 \oplus r_3 \in \mathbb{F}_2^n$, $(r_1, r_2, r_3) \in (\mathbb{F}_2^n)^3$, $(s_1, s_2, s_3) \in (\mathbb{F}_2^m)^3$, a LUT for the (n, m)-S-box S, matrices R and R^{-1} in $\mathcal{M}_n^*(\mathrm{GF}(2))$
OUTPUT: $S(x) \oplus s_1 \oplus s_2 \oplus s_3$

1. $r_4, r_5, r_6 \leftarrow rand(\mathbb{F}_2^n)$
2. $r_7 \leftarrow rand(\mathbb{F}_2^m)$
3. $r' \leftarrow (r_1 \oplus r_4) \oplus (r_2 \oplus r_5) \oplus r_3$
4. **for** $a = 0$ to $2^n - 1$ **do**
5. $\quad a' \leftarrow a \oplus R^{-1} \cdot r'$
6. $\quad T[a'] \leftarrow ((S(\tilde{x} \oplus R(a \oplus R^{-1} \cdot r_6)) \oplus (s_1 \oplus r_7)) \oplus s_2) \oplus s_3$
7. $T[R^{-1} \cdot r_4 \oplus (R^{-1} \cdot r_5 \oplus R^{-1} \cdot r_6)] \oplus r_7$

The matrices R and R^{-1} are generated once at the beginning of each computation and used for every S-boxes through the algorithm.

3 Efficiently implementing the matrix product

In Algorithm 2 every operation is easy to compute, except for the matrix product. Hence it is necessary to devise an efficient algorithm for inverting and computing a matrix-vector product. In the sequel of this section we propose to simultaneously generate two LUT T and T^{-1} to represent this operation and its inverse. Computing the product then consists in evaluating $T[x]$.

Recall that a n-bit Gray code $\mathcal{C}$ is a binary encoding where two successive values (called *codewords*) differ in only one bit. Let $c_i \in \mathcal{C}$, with $0 \leq i \leq 2^n - 1$, denote the i-th Gray $n-$bit codeword, where $c_0 = 0$ and $c_1 = 1$. Let $C_i = \log_2(c_i \oplus c_{i+1})$, with $0 \leq i \leq 2^n - 1$, denote the position of the modified bit between Gray codewords c_i and c_{i+1}.

Let us denote by R_i the $(i+1)$-th column of R. Based on the properties of the Gray code, it can be checked that $R \cdot c_{i+1} = (R \cdot c_i) \oplus R_{C_i}$. This suggests the iterative Algorithm 3 for building the LUT of the product $x \mapsto R \cdot x$. Let $N = 2^n - 1$, the binary complexity of Algorithm 3 is $2N$ *xor* operations, N shifts and $6N + 2$ table lookups, and each product involves one table look-up. Online computation for its part, involves for each computation n^2 *and* operations and n^2 *xor*, and inverting R has a complexity in $O(n^3)$.

Algorithm 3

INPUT: $R = (R_0, R_1, \ldots, R_{n-1}) \in \mathcal{M}_n^*(\mathrm{GF}(2))$ where R_i denotes the $(i+1)$-th column of R, C: the $(2^n - 1, \log_2(n))$-LUT such that $C[i] = \log_2(c_i \oplus c_{i+1})$ where $(c_i)_i$ are the $n-$bit Gray codewords with $c_0 = 0$ and $c_1 = 1$

OUTPUT: The $(2^n, n)$-LUTs T and T^{-1} such that $T[x] = R \cdot x$ and $T^{-1}[x] = R^{-1} \cdot x$ for all $x \in \mathrm{GF}(2)^n$

1. $T[0] \leftarrow 0$
2. $T^{-1}[0] \leftarrow 0$
3. $c' \leftarrow 0$
4. **for** $i = 0$ to $2^n - 2$ **do**
5. $\quad c \leftarrow c' \oplus 2^{C[i]}$
6. $\quad T[c] \leftarrow T[c'] \oplus R_{C[i]}$
7. $\quad T^{-1}[T[c]] \leftarrow c$
8. $\quad c' \leftarrow c$
9. Return (T, T^{-1})

4 Security of the scheme

4.1 Security against third order SCA

In order to assess the security of Algorithm 2 against third order SCA, we list all the intermediate variables involved in the computation and check the dependency between each 3-tuple of these variables and a sensitive variable, i.e. x or $S(x)$.

We can check that the three intermediates variables involved in the flaw of Algorithm 1 do not appear in the computation of Algorithm 2. Indeed, the three intermediate variables corresponding in Algorithm 2 now are :

- $I_1 = a \oplus R^{-1}(r_1 \oplus r_4 \oplus r_2 \oplus r_5 \oplus r_3)$ [step 5]
- $I_2 = R^{-1}(r_4 \oplus r_5 \oplus r_6)$ [step 7]
- $I_3 = x \oplus r_1 \oplus r_2 \oplus r_3 \oplus R(a \oplus R^{-1}r_6)$ [step 6]

Or using the notation introduced in section 2.1 :

- $I_1 = a \oplus R^{-1}(\alpha_1 \oplus \alpha_2)$
- $I_2 = R^{-1}(\alpha_2 \oplus \alpha_3)$
- $I_3 = x \oplus \alpha_1 \oplus R(a \oplus R^{-1}\alpha_3)$

As a is a loop variable, it is equivalent for the adversary to set $a = 0$, then

$$I_1 \oplus I_2 \oplus I_3 = x \oplus \alpha_1 \oplus \alpha_3 \oplus R^{-1}(\alpha_1 \oplus \alpha_3).$$

Let us define $r = r_1 \oplus r_2 \oplus r_3 \oplus r_6 = \alpha_1 \oplus \alpha_3$, and let Id_n be the identity matrix of size n, we can state that the protection is assumed by $m = r \oplus R^{-1}r = (Id_n \oplus R^{-1})r$. We want to know the distribution of m. Remark that if m is uniformly distributed, then m cannot be distinguished from random, and the countermeasure is secure. In order to evaluate this distribution we generate all invertible matrices R of size n, and every n-bit vector r, then we count the occurrences of each value $m' = r \oplus R \cdot r$. We can state that the repartition is uniform for non-zero values, but 0 appear with a probability $1/2^{n-1}$. This result implies that the countermeasure is not theoretically secured. Let us now evaluate the practical impact of this bias upon the security of the scheme.

4.2 Attack simulation

We have seen in section 4.1 that the countermeasure is theoretically unsecured against third order SCA. We want to evaluate the practical security of the scheme against known attacks, and compare it to the security given by a straightforward extension. The simulated attacks target a masked 8-bit S-box computation (*e.g.* the AES S-box). In both case, the targeted variables are given in table 4.2.

Variable	Straightforward extension	This paper
I_1	$a \oplus \alpha_1 \oplus \alpha_2$	$R^{-1}(\alpha_1 \oplus \alpha_2)$
I_2	$\alpha_2 \oplus \alpha_3$	$R^{-1}(\alpha_2 \oplus \alpha_3)$
I_3	$x \oplus \alpha_1 \oplus a \oplus \alpha_3$	$x \oplus \alpha_1 \oplus R(R^{-1}\alpha_3)$

Table 1. Simulated variables targeted by the attacks.

Eventually, we perform two attacks: a third order CPA with normalized product combining and the Hamming weight as prediction function, and a third order MIA with the Hamming weight as prediction function. For third-order MIA, we use the histogram method with Scott's rule for the probability density estimation [4].

All the attack simulations are based on a Hamming weight leakage model with additional Gaussian noise. Namely, the leakage measurements are simulated as samples of the random variables $L_i = \varphi(I_i) + B_i$ for φ being the Hamming weight function and the B_i's being independent Gaussian random variables with mean 0 and standard deviation $\sigma \in \{0, \sqrt{2}, 2, \sqrt{10}, 2\sqrt{5}\}$ (which corresponds to SNR values respectively $+\infty$, 1, 1/2, 1/5 and 1/10). The number of leakage measurements needed to perform the attacks are summarized in Table 2. In particular, in the case of real noise setting (SNR $\geq 1/2$), the attack is clearly impractical.

Attack \ SNR	$+\infty$	1	1/2	1/5	1/10
3O-CPA on Straightforward extension	5500	8000	30000	250000	$> 10^6$
3O-CPA on This paper	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
3O-MIA on Straightforward extension	30000	350000	$> 10^6$	$> 10^6$	$> 10^6$
3O-MIA on This paper	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$

Table 2. Number of leakage measurements for a 90% success rate.

5 Conclusion

In this paper, we introduced a method aiming to thwart third order SCA. The core idea is to use matrix product to enable the extension of Rivain *et al.*'s countermeasure to third order SCA. While this algorithm is not theoretically secure against third order SCA, it can be proven theoretically secure against second order SCA, and practically secure against third order SCA in real world noisy setting.

Acknowledgements. The authors wish to thank Emmanuel Prouff for his helpful comments and discussions about preliminary versions of this work.

References

1. É. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.
2. Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.
3. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M.J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
4. Emmanuel Prouff and Matthieu Rivain. Theoretical and Practical Aspects of Mutual Information Based Side Channel Analysis. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security – ANCS 2009*, volume 5536 of *Lecture Notes in Computer Science*, pages 499–518. Springer, 2009.
5. Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. Cryptology ePrint Archive, Report 2008/021, 2008. http://eprint.iacr.org/.