Secure Multiplicative Masking of Power Functions

Laurie Genelle[1], Emmanuel Prouff[1], and Michaël Quisquater[2]

[1] Oberthur Technologies
{l.genelle,e.prouff}@oberthur.com
[2] University of Versailles
michael.quisquater@prism.uvsq.fr

Abstract. Side Channel Analysis (SCA) is a powerful key recovery attack that efficiently breaks block ciphers implementations. In software, it is usually counteracted by applying a technique called masking, that combines the key dependent variables with random values. When the block cipher to protect mixes affine functions and power functions, a natural strategy is to additively mask the first category of functions and to multiplicatively mask the second one. Several works that follow this strategy have been proposed in the literature, but all of them have been proved to be flawed or very costly. The main difficulty comes from the multiplicative masking of the zero value in a finite field. In this paper, we propose a scheme to multiplicatively mask power functions in such a way that the security against first-order SCA is maintained. We moreover show how to securely combine additive masking of affine transformations with multiplicative masking of power functions. We then apply our method to protect the AES implementation and we show that our proposal offers good timing/memory performances.

1 Introduction

Originally, cryptographic algorithms were designed to provide resistance against logical attacks. These attacks try to recover the key from ciphertexts related to known or unknown plaintexts. While an algorithm may be considered as perfect from a logical point of view, its implementation may leak information. This very old idea was actually already applied to the *one-time pad* by distinguishing the electrical trace of a zero resulting from the addition of two one's and of two zero's. In the late 90's, the development of the smart card industry instigated the research community to develop attacks against software (and hardware) implementations of algorithms. Those attacks take advantage of the correlation between the manipulated secret key and physical measures such as the running time, the power consumption or the electromagnetic emanation of the algorithm processing. Cryptanalyses based on physical measures are today commonly referred to as *side channel attacks*. For such attacks, the idea consists in targeting internal processed values from which it is often possible to derive information on the key. The family of side channel analyzes can be split into two main categories: the *Simple Power Analysis* (SPA) and the *Differential Power Analysis*

J. Zhou and M. Yung (Eds.): ACNS 2010, LNCS 6123, pp. 200–217, 2010.

(DPA). SPA consists in directly interpreting power consumption measurements and in identifying the execution sequence. In a DPA, the attacker focuses on the power consumption of a single instruction and performs statistical tests to reveal some correlations between the distribution of the measurement values and the *sensitive data* (*i.e.* depending on a secret value) manipulated by the instruction. Since the publication of the first SPA and DPA, many papers describing either countermeasures or attack improvements have been published (see [2, 4, 5, 13] for example). Among these improvements, *higher-order SCA attacks* are of particular interest. They extend the SPA and DPA by considering a set of several instructions instead of a single one. The number d of instructions targeted by the attack is called *order* of the SCA. Today's implementations are expected to be resistant against first-order SCA (1O-SCA for short), higher order being difficult to mount in practice. That's why only resistance against SCA of order one (including the classical DPA) is considered in this paper. We investigate this problematic for software implementations of block ciphers that mix affine transformations with power functions (*e.g.* the AES).

1.1 Related Work

A way to thwart SCA involves random values (called masks) to de-correlate the leakage signal from the sensitive data that are manipulated. This way of securing the implementation is called *masking*. It is today considered as the most effective one and is therefore privileged by the smart card industry. The masking countermeasures that have been proposed in the literature may be divided into two global strategies.

The first one consists in masking additively internal values whatever the transformation of the block cipher is applied to. For linear operations, dealing with additive masks propagation and correction is straightforward. In contrast, dealing with additive masking for the non-linear steps of the algorithm (*e.g.* the S-box transformations) is more difficult. This issue was addressed in several ways. Table re-computation [5, 12] enables to mask any function at the cost of a high memory complexity. It may be a good solution when the device on which is embedded the implementation is not too limited in `RAM` memory. In the context of the S-boxes of the AES, Courtois and Goubin [7] have improved the memory performances of the method by using the compact representation of *homographic functions*. The main other methods concern S-boxes with a simple polynomial representation and they are essentially based on the decomposition of the evaluation of those functions. Blömer *et al.* [3] proposed to evaluate them using a *square-and-multiply* algorithm while propagating additive mask at each step. In the same spirit, several researchers have applied the so-called *tower field methods* [19, 10, 14, 15, 16, 18] that evaluate functions defined on quadratic extensions of a finite field from the elements of the subfields while propagating the mask from one representation to another.

The second global strategy was initially proposed by Akkar and Giraud [2] and is dedicated to block ciphers that mix additive operations with multiplicative ones. The core of the strategy is to take the best part of each world using

additive masking to secure linear operations and multiplicative masking to deal with power functions. Originally, the tricky part of this approach was believed to be the conversion of additive masked values into multiplicative ones and Akkar and Giraud [2] proposed a solution to this problematic. However Golić and Tymen [9] exhibited a flaw in the scheme, observing that the sensitive variable is not masked when it equals zero. Indeed, in this case the multiplicatively masked variable equals zero whatever the value of the mask. As a result and as confirmed afterwards in several papers, it turned out that the main difficulty of the second strategy is to multiplicatively mask the value zero in a finite field.

In order to bypass this difficulty, several solutions were proposed:

- Golić *et al.* [9] proposed to shift the computation of the power function up to a ring embedding the finite field on which the power function is defined. This procedure enables to map the zero element of the finite field to non-zero elements of the ring. This procedure doubles the size of the elements and thus induces both a memory and a computational overhead. Moreover, the scheme does not perfectly protect the data against first-order SCA.
- Trichina *et al.* [20] simplified Akkar and Giraud's scheme and proposed some tricks to correct the zero-value flaw. Unfortunately, the whole scheme was shown to be vulnerable to DPA attacks [1] because some masks are biased.
- Another solution would be to detect the zero value masking processing and to apply a particular treatment in this case. As for instance suggested in [8], this could be done by using conditional branches. However, this solution does not give satisfaction because it is vulnerable to SPA. Trichina and Korkishko [21] proposed a trick based on pre-computed tables in order to avoid the use of conditional branches. However, with such a procedure the processing of the AES S-box is simply wrong when applied to zero or one as noticed by Oswald and Schramm [16]. The latter authors tried to repair the schema but conditional branches were always necessary.

Eventually none of the techniques proposed in the literature to apply multiplicative masking to sensitive data is perfectly secure against first-order SCA. This led people to work in the direction of the first strategy even if the second one is actually more natural for block ciphers such as AES.

1.2 Our Results

The method we describe in this paper circumvents the difficulties encountered so far to combine additive masking with multiplicative one. Our solution may be outlined as follows. We first mask the sensitive variable additively and we compute the propagation of this mask through the affine transformations. We then convert this additive masking into a multiplicative masking, mapping (masked) sensitive data equal to zero into non-zero values. We keep track of this modification if applied and we compute the propagation of the multiplicative mask of this non-zero masked value through the power function. The multiplicative mask is then converted into an additive mask taking into account the potential modification of a sensitive variable at the preceding step. These steps are

repeated to eventually obtain both the additively masked ciphertext and the corresponding mask. In the paper we propose two algorithms to convert an additive masking into a multiplicative masking and conversely. Those algorithms result in two methods to secure the implementation of block ciphers. We compared their performances with those of the main techniques proposed in the literature [5, 12, 14, 15, 18, 21]. The timing complexity of our first proposal is ranked second after the table re-computation method [5, 12] and is at least 2.5 times faster than the others [14, 15, 18, 21]. It requires as much memory as the re-computation table method, but it enables to change the masks frequently during the processing with only little overhead. This is a strong advantage since this specificity can be mandated to ensure that no simple higher-order SCA is possible [9]. The second method we propose is an optimization of the first one for devices with little `RAM` and with at least 32 bytes of bit-addressable memory (as for instance the `80C251` architecture microcontrollers). In memory constraints environment, the re-computation method can no longer be used and our solution is the fastest first-order secure alternative to it. We may further conclude that this second proposal achieves the best timing/memory trade-off while having the advantage to enable frequent change of the masks during the computation with only little overhead.

1.3 Paper Organization

The paper is organized as follows. Section 2 deals with the definitions and the concepts we use in the paper. Section 3 and 4 respectively describe the core idea and the algorithms enabling to convert additive masks to multiplicative ones and conversely. Section 5 compares our solution to other implementations in the case of the AES. Section 6 concludes the paper.

2 Theoretical Framework for Security Analysis

We briefly give in what follows some definitions and concepts used to describe our proposal and to analyze its security.

We shall view an implementation of a cryptographic algorithm as the processing of a sequence of *intermediate data*, as defined by Blömer *et al.* [3]. Those data will be associated with *random variables* (r.v. for short) denoted by capital letters, X for instance, while lowercase letters, x for instance, will denote a particular value. The *probability* of the event $\{X \in B\}$ shall be denoted by $\mathrm{P}(X \in B)$. When the event is the singleton $\{x\}$, we will write $\mathrm{P}(X = x)$. If the random variable X has the law of probability $\mathcal{L}(S)$ on the set S, we will write $X \sim \mathcal{L}(S)$. In particular, the uniform law will be denoted by $\mathcal{U}(S)$. We introduce below the concept of *dependency* with respect to $\mathrm{P}(\cdot)$.

Definition 1 (Independency). *Two discrete random variables X_1 and X_2, defined over finite sets S_1 and S_2 respectively, are independent if and only if*

$$\mathrm{P}(X_1 = x_1, X_2 = x_2) = \mathrm{P}(X_1 = x_1) \cdot \mathrm{P}(X_2 = x_2)$$

for any pair $(x_1, x_2) \in S_1 \times S_2$.

Throughout this paper we will often use the following results (see Theorems 3.3.1 and 3.3.2 in [6]).

Proposition 1. *Let $X_1, X_2, \cdots, X_n$ be independent random variables taking respectively their values in the finite sets $S_1, S_2, \cdots, S_n$. If $T_1, T_2, \cdots, T_n$, U and V denote arbitrary sets, then*

1. *$g_1(X_1), g_2(X_2), \cdots, g_n(X_n)$ are independent for any applications $g_i : S_i \rightarrow T_i$, $i = 1...n$,*
2. *$f(X_1, \cdots, X_k)$ and $g(X_{k+1}, \cdots, X_n)$ are independent for any applications $f : S_1 \times \cdots \times S_k \rightarrow U$ and $g : S_{k+1} \times \cdots \times S_n \rightarrow V$.*

We now state a proposition that will allow us to derive the next results. It generalizes analyzes published in [3, 15].

Proposition 2. *Consider a finite group $(G, \star)$, a finite set S and an application $g : S \rightarrow G$. Let X_1 and X_2 be two independent random variables over G and S, respectively. If X_1 is uniform, then $X_3 = X_1 \star g(X_2)$ is a uniform random variable over G which is independent of X_2.*

Proof. See Appendix A.1. □

In the next proposition, we apply Proposition 2 to the additive group $(\mathrm{GF}(2^n), \oplus)$ and the multiplicative group $(\mathrm{GF}(2^n)^\star, \cdot)$. These results will be useful to prove the security of our scheme in Sect. 4.

Proposition 3. *In what follows, $\mathcal{L}(S)$ denotes a law of probability on S which is not specified.*

1. *If $X_1 \sim \mathcal{U}(\mathrm{GF}(2^n))$ and $X_2 \sim \mathcal{L}(\mathrm{GF}(2^n))$ are independent r.v's and g is an application from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2^n)$, then*

$$X_3 = X_1 \oplus g(X_2) \sim \mathcal{U}(\mathrm{GF}(2^n)) \text{ and } X_3 \text{ is independent of } X_2\,.$$

2. *If $X_1 \sim \mathcal{U}(\mathrm{GF}(2^n)^\star)$ and $X_2 \sim \mathcal{L}(\mathrm{GF}(2^n))$ are independent r.v's and g is an application from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2^n)^\star$, then*

$$X_3 = X_1 \cdot g(X_2) \sim \mathcal{U}(\mathrm{GF}(2^n)^\star) \text{ and } X_3 \text{ is independent of } X_2\,.$$

3. *If $X_1 \sim \mathcal{L}(\mathrm{GF}(2^n)^\star)$, $X_2 \sim \mathcal{U}(\mathrm{GF}(2^n))$ and $X_3 \sim \mathcal{L}(\mathrm{GF}(2^n))$ are independent r.v's and g is an application from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2^n)$, then*

$$X_4 = X_1 \cdot (X_2 \oplus g(X_3)) \sim \mathcal{U}(\mathrm{GF}(2^n)) \text{ and } X_4 \text{ is independent of } X_3\,.$$

Proof. See Appendix A.2. □

In the rest of the paper, an intermediate variable shall be said to be *sensitive* if it is dependent on the secret key. The following definition gives a formal definition of the security against first-order SCA.

Definition 2 (First-Order SCA Security). *A cryptographic algorithm is said to be secure against first-order SCA if every intermediate variable is independent of every sensitive variable.*

In the next sections, we present the core idea of our proposal and we prove that it is first-order SCA resistant.

3 Core Idea of Our Proposal

The core idea of our contribution deals with the first-order SCA resistant implementation of block ciphers mixing affine transformations and power functions in a finite field. In what follows, Op denotes either an affine transformation or a power function, depending on the context.

In what follows, we assume that input/output of the affine operations of the block cipher (*e.g.* the AES individual affine transformations) are always masked additively. Namely, every such operation Op is implemented such that it takes as input the masked data $x \oplus m_{\text{in}} \in \text{GF}(2^8)$, where m_{in} is randomly generated over $\text{GF}(2^8)$ and x is a sensitive value. Since Op is affine, then the corresponding output equals $Op(x) \oplus Op(m_{\text{in}}) \oplus c$, where c is a constant and $Op(m_{\text{in}}) \oplus c$ is the new mask. In this case, it is obvious that neither the input, nor the output, nor any intermediate data during the computation is sensitive.

Let us now consider the case when Op is a power function (not linear) defined over $\text{GF}(2^8)$. To take advantage of the multiplicative structure of Op, we would like to secure its implementation with multiplicative masks. For such a purpose, since Op is likely to operate on the output of additively masked affine transformations, we first need to convert an additively masked value $x \oplus m_{\text{in}}$ into a multiplicatively one in the form $x \cdot b$, where b is a random value in $\text{GF}(2^8)^\star$. Secondly, we need to define a scheme such that Op operates securely on $x \cdot b$, and outputs data in the form $Op(x) \cdot b'$. Eventually, since an affine transformation is likely to operate on the multiplicatively masked output of Op, we need to convert it back into an additively masked value in the form $Op(x) \oplus m_{\text{out}}$, where m_{out} is randomly generated over $\text{GF}(2^8)$. The masking conversions and the secure scheme for the power function must be defined by taking into account the so-called *zero-value problem* identified in several papers [7,9,20,21]: the value $x = 0$ cannot be masked multiplicatively.

The core idea of this paper is to convert additive masking into multiplicative masking *via* an algorithm AMToMM in such a way that the sensitive value $x = 0$ is mapped into 1, keeping trace of this transformation. The power function is then applied to this non-zero multiplicatively masked value. The result is eventually converted into an additive masked value *via* an algorithm MMToAM, taking into account the potential mapping of the zero sensitive value in the first step.

For any y, let us denote by δ_y the function defined by $\delta_y(x) = 1$ if $x = y$ and $\delta_y(x) = 0$ otherwise. The most tricky part in this method is to define AMToMM such that it computes $b \cdot (x \oplus \delta_0(x))$ in a secure way from $x \oplus m_{\text{in}}$, m_{in} and b. This indeed implies the SCA-secure computation of $\delta_0(x)$ from $x \oplus m_{\text{in}}$ and m_{in}. It can first be noticed that we have $\delta_0(x) = \delta_{m_{\text{in}}}(x \oplus m_{\text{in}})$. Based on this remark, we propose in the following section a masked implementation of $\delta_{m_{\text{in}}}(x \oplus m_{\text{in}})$ designed such that:

$$\widetilde{\delta}_{m_{\text{in}}}(x \oplus m_{\text{in}}) = \begin{cases} r \oplus 1 & \text{if } x = 0 \\ r & \text{otherwise} \end{cases}$$

where r is a random value in GF(2^8) (Proposal 1 in Sect(s). 4.1 and 4.3) or in GF(2) (Proposal 2 in Sect. 4.4). To implement it securely and efficiently we choose to tabulate the function thanks to a pre-computed table T defined such that $T[i] = r \oplus 1$ if $i = m_{\text{in}}$ and $T[i] = r$ otherwise. Indeed, for such a table T we have $\widetilde{\delta}_{m_{\text{in}}}(x \oplus m_{\text{in}}) = T[x \oplus m_{\text{in}}]$. The whole procedure is summarized in Fig. 1.

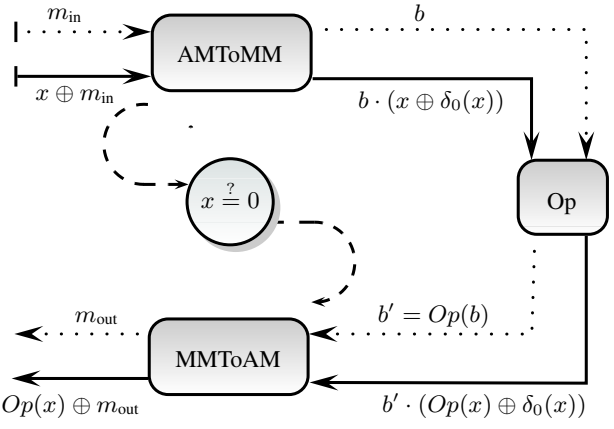

Fig. 1. Multiplicative Masking of a Power Function Op

In the next section the three algorithms AMToMM, Op and MMToAM are described and their efficiency and security are both discussed.

4 Algorithmic of Our Proposal

In the following description of Alg(s). AMToMM, Op and MMToAM we will keep the same notations involved above. Namely, we shall denote by x a sensitive value, by m_{in} an additive mask, by $b \in \text{GF}(2^8)^\star$ a multiplicative mask, by r a random value over GF(2^8) and by T a table of 256 bytes/bits such that $T[x \oplus m_{\text{in}}]$ takes the value $r \oplus 1$ if $x = 0$ and r otherwise. The value r and the table T are regenerated at each execution of the algorithm. Values m_{in} and b can be regenerated several times per algorithm processing. We shall denote $(x)_{m_{\text{in}}} = x \oplus m_{\text{in}}$ and $[x]_b = b \cdot (x \oplus \delta_0(x))$. The output $Op(x) \oplus \delta_0(x)$ shall be denoted by x' and the associated multiplicative mask $Op(b)$ shall be denoted by b'.

4.1 From Additive Masking (AM) to Multiplicative Masking (MM)

To transform every $(x)_{m_{\text{in}}}$ into $[x]_b$ without leaking information about x, we suggest to use the following algorithm which has been decomposed into several

elementary operations (left column), each manipulating an intermediate result (right column) computed from $x \oplus m_{\text{in}}$, m_{in}, b and r.

Algorithm 1. SCA-resistant AMToMM

INPUTS: The table T, the random value r, the additively masked value $(x)_{m_{\text{in}}}$, the additive mask m_{in}, the multiplicative mask b
OUTPUT: The multiplicatively masked value $[x]_b$ and the updating of the global variable mem with $T[(x)_{m_{\text{in}}}]$

Pseudo-Code	Data
1. $res \leftarrow r$	$res = r$
2. $res \leftarrow res \oplus (x)_{m_{\text{in}}}$	$res = r \oplus x \oplus m_{\text{in}}$
3. $res \leftarrow res \oplus m_{\text{in}}$	$res = r \oplus x$
4. $res \leftarrow b \cdot res$	$res = b \cdot (r \oplus x)$
5. $tmp \leftarrow (x)_{m_{\text{in}}}$	$tmp = x \oplus m_{\text{in}}$
6. $mem \leftarrow T[tmp]$	$mem = r \oplus \delta_0(x)$
7. $tmp \leftarrow b \cdot mem$	$tmp = b \cdot (r \oplus \delta_0(x))$
8. $res \leftarrow res \oplus tmp$	$res = b \cdot (x \oplus \delta_0(x))$

Correctness of Alg. 1. Algorithm 1 processes the following computations. Additions are performed in the order from left to right to insure that the calculation is first-order SCA resistant:

$$b \cdot (r \oplus (x)_{m_{\text{in}}} \oplus m_{\text{in}} \oplus T[(x)_{m_{\text{in}}}]) .$$

Substituting the data according to their definitions, this computation simplifies to:

$$b \cdot (x \oplus \delta_0(x)) = [x]_b .$$

Moreover we can see that Alg. 1 keeps the track of the possible mapping into 1 of $x = 0$ by saving $T[(x)_{m_{\text{in}}}]$ in a global variable denoted by mem.

Security Analysis. In the following, we denote by X, M_{in}, B and R the random variables modeling the values x, m_{in}, b and r respectively. From the description of Alg. 1, we obtain in Table 1 the list of all intermediate variables $I_1, \ldots, I_7$ that involve X in their construction. We prove below that each of them is independent of X.

We remind that M_{in} and R are uniformly distributed over $\text{GF}(2^8)$, and that B is uniformly distributed over $\text{GF}(2^8)^\star$. Moreover M_{in}, B, R and X are by definition mutually independent.

Proposition 4. *Algorithm 1 is first-order SCA resistant.*

Proof. We have $I_1 = (R \oplus M_{\text{in}}) \oplus X$, $I_2 = R \oplus X$, $I_4 = X \oplus M_{\text{in}}$ and $I_5 = R \oplus \delta_0(X)$, where R, M_{in} and X are mutually independent random variables. We moreover have $M_{\text{in}} \sim \mathcal{U}(\text{GF}(2^8))$ and $R \sim \mathcal{U}(\text{GF}(2^8))$. Thus, the random variable $R \oplus M_{\text{in}}$ satisfies $R \oplus M_{\text{in}} \sim \mathcal{U}(\text{GF}(2^8))$ according to the first statement

Table 1. Intermediate variables of Alg. 1

j	I_j
1	$R \oplus X \oplus M_{\text{in}}$
2	$R \oplus X$
3	$B \cdot (R \oplus X)$
4	$X \oplus M_{\text{in}}$
5	$R \oplus \delta_0(X)$
6	$B \cdot (R \oplus \delta_0(X))$
7	$B \cdot (X \oplus \delta_0(X))$

of Proposition 3 and it is independent of X according to Statement 2 of Proposition 1. The three r.v's M_{in}, R and $R \oplus M_{\text{in}}$ being uniform over $\mathcal{U}(\text{GF}(2^8))$ and independent of X, it follows from Statement 1 of Proposition 3 that I_1, I_2, I_4 and I_5 are also independent of X.

We observe now that $I_7 = B \cdot (X \oplus \delta_0(X))$ is the product of a uniform random variable B defined over $\text{GF}(2^8)^*$ with a function of X taking its values in $\text{GF}(2^8)^*$. Noting that B and X are independent, it follows from the second statement of Proposition 3 that I_7 is independent of X.

Eventually, we have $I_3 = B \cdot (R \oplus X)$ and $I_6 = B \cdot (R \oplus \delta_0(X))$, where $B \sim \text{GF}(2^8)^*$, $R \sim \mathcal{U}(\text{GF}(2^8))$ and X are mutually independent random variables. We conclude that I_3 and I_6 are independent of X according to the third statement of Proposition 3.

We have proved that all intermediate variables in Alg. 1 are independent of X. From Definition 2, we can then conclude that Alg. 1 is first-order SCA resistant. □

4.2 Multiplicative Masking of Power Functions

We describe hereafter how to secure the processing of a power function Op with multiplicative masking.

Algorithm 2. SCA-resistant Power Function Op

INPUTS: The multiplicative masked value $[x]_b$ and the multiplicative mask b
OUTPUT: The masked value $[Op(x)]_{b'}$ and the output mask b'

1. output $\leftarrow Op([x]_b)$
2. $b' \leftarrow Op(b)$

Correctness of Alg. 2. We apply the power function Op to the multiplicatively masked value $[x]_b$ and we obtain:

$$Op([x]_b) = Op(b) \cdot Op(x \oplus \delta_0(x)) = b' \cdot x' \,,$$

where we recall that x' and b' respectively denote $Op(x \oplus \delta_0(x))$ and $Op(b)$. Observing that x' is always non-zero, we deduce that $\delta_0(x') = 0$ and we have:

$$Op([x]_b) = b' \cdot (x' \oplus \delta_0(x')) = [x']_{b'} .$$

Security Analysis. In Alg. 2, the following intermediate variables appear: $I_1 = [x]_b$, $I_2 = Op([x]_b)$ and $I_3 = Op(b)$. Among them only I_1 and I_2 involve X in their definition and we prove below that each of them is independent of X.

Proposition 5. *Algorithm 2 is first-order SCA resistant.*

Proof. Let g denote the function defined from $\mathrm{GF}(2^8)$ into $\mathrm{GF}(2^8)^\star$ by $g(X) = X \oplus \delta_0(X)$. The r.v. I_1 is the product of $g(X)$ with a random variable $B \sim \mathcal{U}(\mathrm{GF}(2^n)^\star)$ which is independent of X. The second statement of Proposition 3 thus implies that I_1 and X are independent. Since I_2 is a function of the r.v. I_1 which is independent of X, it is itself independent of X according to Statement 1 of Proposition 1. We eventually conclude that all the intermediate variables of Alg. 2 are independent of X. Definition 2 thus implies that the latter one is secure against first-order SCA. □

4.3 From Multiplicative to Additive Masking

We recall that we have $x' = Op(x \oplus \delta_0(x))$, that is $x' = Op(x) \oplus \delta_0(x)$ since Op is a power function (and thus $Op(0) = 0$ and $Op(1) = 1$). We assume that m_{out} is a random value generated over $\mathrm{GF}(2^8)$.

The following algorithm describes a method to securely convert every $[x']_{b'}$ into $(Op(x))_{m_{\text{out}}}$.

Algorithm 3. SCA-resistant MMToAM

INPUTS: The multiplicatively masked value $[x']_{b'}$, the multiplicative mask $b' \in \mathrm{GF}(2^8)^\star$, the additive mask $m_{\text{out}} \in \mathrm{GF}(2^8)$, the global variable $mem = T[(x)_{m_{\text{in}}}]$
OUTPUT: The additively masked value $(Op(x))_{m_{\text{out}}}$

Pseudo-Code	Data
1. $res \leftarrow mem$	$res = r \oplus \delta_0(x)$
2. $res \leftarrow res \oplus m_{\text{out}}$	$res = r \oplus \delta_0(x) \oplus m_{\text{out}}$
3. $res \leftarrow res \oplus r$	$res = \delta_0(x) \oplus m_{\text{out}}$
4. $res \leftarrow b' \cdot res$	$res = b' \cdot (\delta_0(x) \oplus m_{\text{out}})$
5. $tmp \leftarrow [x']_{b'}$	$tmp = b' \cdot (Op(x) \oplus \delta_0(x))$
6. $res \leftarrow res \oplus tmp$	$res = b' \cdot (Op(x) \oplus m_{\text{out}})$
7. $res \leftarrow b'^{-1} \cdot res$	$res = Op(x) \oplus m_{\text{out}}$

Correctness of Alg. 3. Algorithm 3 processes the following computations where the operation order is defined from the left to the right between each pair of brackets:

$$b'^{-1} \cdot (b' \cdot (T[(x)_{m_{\text{in}}}] \oplus m_{\text{out}} \oplus r) \oplus [x']_{b'}) .$$

Substituting the variables according to their definitions and observing that $Op(x \oplus \delta_0(x)) = Op(x) \oplus \delta_0(x)$, this computation simplifies to:

$$\begin{aligned}\delta_0(x) \oplus r \oplus m_{\text{out}} \oplus r \oplus x' \oplus \delta_0(x') &= \delta_0(x) \oplus m_{\text{out}} \oplus Op(x \oplus \delta_0(x)) \\ &= m_{\text{out}} \oplus Op(x) \\ &= (Op(x))_{m_{\text{out}}}\end{aligned}$$

Security Analysis. We use the same notations as introduced in Sect. 4.1. Additionally we denote by M_{out} and B' the random variables corresponding respectively to the values m_{out} and b'. Table 2 lists all the intermediate variables $I_1, \ldots, I_7$ of Alg. 3. We prove below that each of them is independent of X.

Table 2. Intermediate variables of Alg. 3

j	I_j
1	$R \oplus \delta_0(X)$
2	$R \oplus \delta_0(X) \oplus M_{\text{out}}$
3	$\delta_0(X) \oplus M_{\text{out}}$
4	$B' \cdot (\delta_0(X) \oplus M_{\text{out}})$
5	$B' \cdot (Op(X) \oplus \delta_0(X))$
6	$B' \cdot (Op(X) \oplus M_{\text{out}})$
7	$Op(X) \oplus M_{\text{out}}$

By definition M_{out} is uniformly distributed over $\text{GF}(2^8)$, and B' is uniformly distributed over $\text{GF}(2^8)^\star$. Moreover we remind that M_{out}, B' and X are mutually independent.

Proposition 6. *Algorithm 3 is first-order SCA resistant.*

Proof. We have, $I_1 = R \oplus \delta_0(X)$, $I_2 = (M_{\text{out}} \oplus R) \oplus \delta_0(X)$, $I_3 = M_{\text{out}} \oplus \delta_0(X)$ and $I_7 = Op(X) \oplus M_{\text{out}}$ where R, M_{out} and X are mutually independent. After noting that $M_{\text{out}} \sim \mathcal{U}(\text{GF}(2^8))$ and $R \sim \mathcal{U}(\text{GF}(2^8))$, we deduce respectively from the first Statement of Proposition 3 and the second Statement of Proposition 1 that $R \oplus M_{\text{out}}$ satisfies $R \oplus M_{\text{out}} \sim \mathcal{U}(\text{GF}(2^8))$ and is independent of X. We conclude from Statement 1 of Proposition 3 that I_1, I_2, I_3 and I_7 are independent of X.

Let us observe now that $I_5 = B' \cdot (Op(X) \oplus \delta_0(X))$ is the product of a uniform random variable defined over $\text{GF}(2^8)^*$ and a function of X with values in $\text{GF}(2^8)^\star$. It follows from Statement 2 of Proposition 3 that I_5 is independent of X.

Eventually, we have $I_4 = B' \cdot (\delta_0(X) \oplus M_{\text{out}})$ and $I_6 = B' \cdot (Op(X) \oplus M_{\text{out}})$, where $B' \sim \mathcal{U}(\text{GF}(2^8)^*)$, $M_{\text{out}} \sim \mathcal{U}(\text{GF}(2^8))$ and X are mutually independent. These intermediate variables are independent of X according to Statement 3 of Proposition 3.

All intermediate variables of Alg. 3 are independent of X. We conclude that Alg. 3 is secure against first-order SCA according to Definition 2. □

4.4 Optimization

Here we propose an alternative algorithm to convert an additive masking into a multiplicative masking. Compared with Alg. 1, it involves a `RAM`-table T of 256 bits (stored in 32 bytes) instead of a `RAM`-table T of 256 bytes, at the cost of only two additional bitwise additions (Steps 3 and 9 in Alg. 4). This version is therefore of particular interest when the device on which is implemented the countermeasure makes it easy to manipulate bits in memory (*e.g.* has bit-addressable memory).

In what follows, we denote by γ a random bit and we define T by $T[i] = \gamma \oplus 1$ if $i = m_{\text{in}}$ and $T[i] = \gamma$ otherwise.

Algorithm 4. SCA-resistant AMToMM using a bit-table

INPUTS: The table T, the random bit γ, the additively masked value $(x_i)_{m_{\text{in}}}$, the additive mask m_{in}, the multiplicative mask b
OUTPUT: The multiplicatively masked value $[x]_b$ and the updating of the global variable mem with $T[(x)_{m_{\text{in}}}]$

Pseudo-Code	Data
1. $res \leftarrow \gamma$	$res = \gamma$
2. $rand \leftarrow$ `RNG`	$rand$ is random value
3. $res \leftarrow res \oplus rand$	$res = \gamma \oplus rand$
4. $res \leftarrow res \oplus (x)_{m_{\text{in}}}$	$res = \gamma \oplus rand \oplus x \oplus m_{\text{in}}$
5. $res \leftarrow res \oplus m_{\text{in}}$	$res = \gamma \oplus rand \oplus x$
6. $res \leftarrow b \cdot res$	$res = b \cdot (\gamma \oplus rand \oplus x)$
7. $tmp \leftarrow (x)_{m_{\text{in}}}$	$tmp = x \oplus m_{\text{in}}$
8. $mem \leftarrow T[tmp]$	$mem = \gamma \oplus \delta_0(x)$
9. $tmp \leftarrow mem \oplus rand$	$tmp = \gamma \oplus \delta_0(x) \oplus rand$
10. $tmp \leftarrow b \cdot tmp$	$tmp = b \cdot (\gamma \oplus \delta_0(x) \oplus rand)$
11. $res \leftarrow res \oplus tmp$	$res = b \cdot (x \oplus \delta_0(x))$

We have described the optimization for Alg. AMToMM only but the modification must obviously also be done for Alg. MMToAM. Since adapting the optimization above to the latter algorithm is straightforward, we chose to not describe it.

5 Application to the AES

To compare the efficiency of our proposals with that of other methods in the literature, we applied them to protect an implementation of the AES-128 algorithm in encryption mode. We wrote the codes in assembly language for an `8051` based 8-bit architecture without bit-addressable memory. This context was not ideally suitable for our Proposal 2, but as it can be observed in Table 3, its performances are already good for this architecture and actually achieve the best timing/memory trade-off. For this reason, we didn't chose to move to a bit-addressable target[1] for our comparison.

[1] After a rough analysis of the assembly code for our Proposal 2, we think that a 10% loss of performances is due to the fact that no bit-addressable memory is available.

In Table 3, we have listed the timing/memory performances of the different implementations. Memory performances correspond to the number of bytes allocations and cycles numbers correspond to multiple of 10^3. We moreover have dissociated `RAM` consumption inherent to the method from `RAM` consumption related to the implementation choice (in brackets) which can highly vary from a code to another. A column has been added to alert on the fact that some of the listed countermeasures can be easily adapted (without significant timing/memory performances overhead) to involve different masks to secure each S-box calculation. This specificity is referred to as MM (for *Multi-Masking*) in the table and is discussed in more details in [9, 18]. We also added a column to point out that some of the listed countermeasures do not achieve first-order SCA resistance as defined in Definition 2. This fact is discussed in further details in the next paragraph.

Table 3. Comparison of AES implementations

	Method	Ref.	Cycles	RAM	ROM	MM	1O-SCA
Unprotected Implementation							
1.	No Masking	Na.	2	0 (+32)	1150	Na.	No.
Masking by Addition *vs* First-Order SCA							
2.	Re-computation	[12]	10	256 (+35)	1553	No.	Yes.
3.	Tower Field in GF(2^2)	[14, 15]	77	0 (+42)	3195	Yes.	Yes.
4.	Tower Field in GF(2^4)	[16]	29	0 (+36)	3554	Yes.	No.
5.	Masking *on-the-fly*	[18]	82	0 (+39)	2948	Yes.	Yes.
Masking by Addition-and-Multiplication *vs* First-Order SCA							
6.	Log-ALog Tables	[21]	55	256 (+44)	1900	No.	No.
7.	Proposal 1	Na.	26	256 (+40)	2795	Yes.	Yes.
8.	Proposal 2	Na.	28	32 (+40)	2960	Yes.	Yes.

Outlines of the Methods and Main Differences. The AES implementations listed in Table 3 only differ in their approaches to protect the S-box access. The linear steps and the key-scheduling of the AES have been implemented in the same way: the key-scheduling has not been masked and the internal sensitive data manipulated during the linear steps have been protected by bitwisely adding a random value. We moreover chose to protect all the rounds of the AES processing.

Since the linear steps are protected by additive masking, the AES S-Box denoted by S must be modified to securely deal with such a masking of its inputs/outputs. To answer this issue, the re-computation method computes the table of the function $x \mapsto S[x \oplus m_{\text{in}}] \oplus m_{\text{out}}$ for two pre-defined values of input/output masks and stores it in `RAM`. For the other methods, S is split into its affine part and its non-linear part Op which is the power function $x \in \text{GF}(2^8) \mapsto x^{254} \in \text{GF}(2^8)$. Under this representation, dealing with the mask propagation for S amounts to deal with the mask propagation for Op. The methods of Oswald *et al.* [14, 15] and of Prouff *et al.* [18] start by representing

Op over an extension field of GF(2^4) of degree 2 (such a technique is usually called *tower field method*) and they only differ in the ways of securely computing an inversion in GF(2^4). In [15, 14], the inversion is performed by going down to GF(2^2) where this operation is linear. In [18], the inversion is performed for every element of GF(2^4) and the correct result is saved in a special location in memory according to the value of a comparison test. This method is referred to as the *Masking on-the-fly* method in Table 3. All the methods mentioned above achieve perfect security against first-order side channel analysis, which is not the case of those of Trichina-Korkishko [21] and Oswald-Schramm [16] that respectively correspond to the 6th and 4th method presented in Table 3.

In Trichina *et al.*'s method, a *primitive element* of GF(2^8) is computed and every non-zero element of GF(2^8) is expressed as a power of that element. To resolve the zero-value problem, Trichina *et al.* use slightly modified discrete logarithm and exponentiation tables that are pre-computed at the beginning of the processing to evaluate the AES power function. As argued in [16], the method has a faulty behavior when some intermediate values are null and no sound correction of the method has been published until now in the literature. As in [14, 15], the method proposed by Oswald-Schramm [16] is based on the tower field method but some operations are processed by accessing look-up tables at addresses that depend on both the mask and the masked data. Since such a variable is dependent on the sensitive variable, the scheme cannot be considered as first-order SCA resistant with respect to Definition 2 and attacks such as those exhibited in [17, 22] are possible. Despite the imperfect security of the two methods discussed above, we chose to implement them for comparison since they counteract almost all first-order SCA when no pre-processing is performed (which is for instance the case for the classical DPA [11] and CPA [4]).

Discussion about the Implementations Results. First, since the performances have been measured for a particular implementation on a particular architecture, we draw reader's attention that Table 3 does not aim at arguing that a method is better than another but aims at enlightening the main particularities (timing performances and `ROM`/`RAM` requirements) of each method.

As expected, when 256 bytes of `RAM` memory are available, then the re-computation method achieves the best timing performances. In this context, our first proposal is also promising (ranked second behind the re-computation method). It can even be a valuable alternative to the re-computation method, when the input/output masks of the S-box calculations are required to change during the AES processing. As argued for instance by Golić and Tymen [9] or by Prouff and Rivain [18], such a security requirement can be laid down in order to increase the resistance against simple higher-order SCA. In this case, the re-computation becomes prohibitive whereas the performances of our first proposal stay almost unchanged (the values taken by the input/output mask in Alg. $1-4$ are not assumed to be fixed during the AES processing).

As `RAM` memory is a very sensitive resource in the area of embedded devices, it is often preferable to value memory allocation reduction over timing reduction. Except for our first proposal, all the countermeasures listed in Table 3

are less efficient than the re-computation method but they all require much less RAM allocation. In memory constrained devices, they therefore are preferred to the re-computation method. Among them, our second proposal is the most efficient one. Only the method of Oswald-Schramm has close performances, but at the cost of imperfect security *versus* 1O-SCA. When compared to the methods achieving perfect resistance against first-order SCA, our second proposal is at least 2.5 times faster.

To conclude about the experiment results reported in Table 3, the choice between the implementations that offer perfect resistance against first-order SCA essentially depends on two parameters: the size of the RAM memory available on the device and the necessity to change masks frequently. We sum up our conclusions in Table 4, where we give for different contexts (amount of RAM available and chosen masking methods) the method(s) which is(are) the most efficient one(s).

Table 4. Distribution of the 1O-SCA-resistant methods *vs* the available RAM memory

Method	Cycles ($\times 10^3$)	ROM (in bytes)	Multi-Masking
Device with Large RAM memory (> 256 bytes)			
Re-computation Table	10	1553	No.
Proposal 1	26	2795	Yes.
Device with Medium RAM memory (between 40 and 256 bytes)			
Proposal 2	28	2960	Yes.
Device with Small RAM memory (< 40 bytes)			
Tower Field in $GF(2^2)$	77	3195	Yes.
Masking *on-the-fly*	82	2948	Yes.

6 Conclusion

In this paper, we have proposed a solution to the zero-value problem in the multiplicative masking. By introducing a scheme that embeds the multiplicative masking in $GF(2^8)$ into a multiplicative masking in $GF(2^8)^\star$ we obtained a secure method to protect the implementation of power functions. We proposed two methods with different timing/memory trade-offs to implement our scheme and we proved the security of both methods against first-order SCA. We moreover compared the new solutions with the existing ones for the AES. Based on our experiments, we argued that our solutions offer very valuable timing/memory performances. When a large amount of RAM memory can be used, our proposal is ranked second among the implemented methods and offers better resistance to simple higher-order SCA than the first ranked method. In memory constrained devices, our second proposal is ranked first. To the best of our knowledge, it has the best timing/memory overhead and is therefore a valuable alternative to the existing methods.

References

1. Akkar, M.-L., Bévan, R., Goubin, L.: Two Power Analysis Attacks against One-Mask Methods. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 332–347. Springer, Heidelberg (2004)
2. Akkar, M.-L., Giraud, C.: An Implementation of DES and AES, Secure against Some Attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 309–318. Springer, Heidelberg (2001)
3. Blömer, J., Merchan, J.G., Krummel, V.: Provably Secure Masking of AES. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 69–83. Springer, Heidelberg (2004)
4. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
5. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)
6. Chung, K.L.: A Course in Probability Theory. Academic Press, London (2001)
7. Courtois, N., Goubin, L.: An Algebraic Masking Method to Protect AES against Power Attacks. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 199–209. Springer, Heidelberg (2006)
8. Damgard, M., Keller, M.: Secure Multiparty AES. In: Financial Cryptography (to appear, 2010)
9. Golić, J., Tymen, C.: Multiplicative Masking and Power Analysis of AES. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 198–212. Springer, Heidelberg (2003)
10. Gueron, S., Parzanchevsky, O., Zuk, O.: Masked Inversion in GF(2^n) Using Mixed Field Representations and its Efficient Implementation for AES. In: Nedjah, N., Mourelle, L.M. (eds.) Embedded Cryptographic Hardware: Methodologies and Architectures, pp. 213–228. Nova Science Publishers, Bombay (2004)
11. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 388–397. Springer, Heidelberg (1999)
12. Messerges, T.: Securing the AES Finalists against Power Analysis Attacks. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 150–164. Springer, Heidelberg (2001)
13. Messerges, T.: Using Second-order Power Analysis to Attack DPA Resistant Software. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000)
14. Oswald, E., Mangard, S., Pramstaller, N.: Secure and Efficient Masking of AES – A Mission Impossible? Cryptology ePrint Archive, Report 2004/134 (2004)
15. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A Side-Channel Analysis Resistant Description of the AES S-box. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 413–423. Springer, Heidelberg (2005)
16. Oswald, E., Schramm, K.: An Efficient Masking Scheme for AES Software Implementations. In: Song, J.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 292–305. Springer, Heidelberg (2006)
17. Prouff, E., McEvoy, R.P.: First-Order Side-Channel Attacks on the Permutation Tables Countermeasure. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 81–96. Springer, Heidelberg (2009)

18. Prouff, E., Rivain, M.: A Generic Method for Secure SBox Implementation. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 227–244. Springer, Heidelberg (2008)
19. Rudra, A., Bubey, P.K., Jutla, C.S., Kumar, V., Rao, J., Rohatgi, P.: Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 171–184. Springer, Heidelberg (2001)
20. Trichina, E., DeSeta, D., Germani, L.: Simplified Adaptive Multiplicative Masking for AES. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 187–197. Springer, Heidelberg (2003)
21. Trichina, E., Korkishko, L.: Secure and Efficient AES Software Implementation for Smart Cards. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 425–439. Springer, Heidelberg (2005)
22. Waddle, J., Wagner, D.: Towards Efficient Second-Order Power Analysis. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 1–15. Springer, Heidelberg (2004)

A Appendix

A.1 Proof of Proposition 2

Proof. We first observe that for any $x_2 \in S$ and $x_3 \in G$,

$$\mathrm{P}(X_1 \star g(X_2) = x_3, X_2 = x_2) = \mathrm{P}(X_1 \star g(x_2) = x_3, X_2 = x_2)\,.$$

Since X_1 and X_2 are independent and X_1 is uniform over G,

$$\mathrm{P}(X_1 \star\ g(x_2) = x_3, X_2 = x_2) = \mathrm{P}(X_1 \star\ g(x_2) = x_3) \cdot \mathrm{P}(X_2 = x_2) = \frac{1}{\mid G \mid} \cdot \mathrm{P}(X_2 = x_2)\,,$$

for any $x_2 \in S$ and $x_3 \in G$. It follows that for any $x_2 \in S$ and $x_3 \in G$,

$$\mathrm{P}(X_1 \star\ g(X_2) = x_3, X_2 = x_2) = \frac{1}{\mid G \mid} \cdot \mathrm{P}(X_2 = x_2)\,. \tag{1}$$

Let us now prove that $X_3 = X_1 \star\ g(X_2)$ is a uniform random variable over G. According to the law of total probability, for any $x_3 \in G$

$$\mathrm{P}(X_1 \star\ g(X_2) = x_3) = \sum_{x_2 \in S} \mathrm{P}(X_1 \star\ g(X_2) = x_3 \mid X_2 = x_2) \cdot \mathrm{P}(X_2 = x_2)\,.$$

By definition of the conditional probability and (1), for any $x_3 \in G$

$$\mathrm{P}(X_1 \star\ g(X_2) = x_3) = \sum_{x_2 \in S} \frac{1}{\mid G \mid} \cdot \mathrm{P}(X_2 = x_2)\,.$$

Noting that $\sum_{x_2 \in S} \mathrm{P}(X_2 = x_2) = 1$, we conclude that $X_3 = X_1 \star\ g(X_2)$ is a uniform random variable over G. According to (1) and the uniformity of X_3, we have for every $(x_2, x_3) \in S$:

$$\mathrm{P}(X_1 \star\ g(X_2) = x_3, X_2 = x_2) = \mathrm{P}(X_1 \star\ g(X_2) = x_3) \cdot \mathrm{P}(X_2 = x_2)\,.$$

This means that $X_3 = X_1 \star\ g(X_2)$ and X_2 are independent. □

A.2 Proof of Proposition 3

Proof. The first and the second statement follow immediately from Proposition 2 particularized to the groups $G = (\mathrm{GF}(2^n), \oplus)$ and $G = (\mathrm{GF}(2^n)^*, \cdot)$ respectively. Let us prove the third statement. Let us first prove that if we have two independent random variables $A \sim \mathcal{L}(\mathrm{GF}(2^n)^*)$ and $B \sim \mathcal{U}(\mathrm{GF}(2^n))$ then $C = A \cdot B \sim \mathcal{U}(\mathrm{GF}(2^n))$. Let us denote by A^{-1} the inverse of A in $\mathrm{GF}(2^n)^*$. For any $c \in \mathrm{GF}(2^n)$, we have

$$\mathrm{P}(C = c) = \mathrm{P}(A \cdot B = c) = \mathrm{P}(B = c \cdot A^{-1}) = \mathrm{P}(B \oplus c \cdot A^{-1} = 0)\,.$$

Note that $B \oplus c \cdot A^{-1}$ is the sum of uniform random variable over $\mathrm{GF}(2^n)$ and a random variable that may be considered over $\mathrm{GF}(2^n)$. According to the first statement we have $B \oplus c \cdot A^{-1} \sim \mathcal{U}(\mathrm{GF}(2^n))$. It follows that $C \sim \mathcal{U}(\mathrm{GF}(2^n))$.

Consider now the independent random variables $X_1 \sim \mathcal{L}(\mathrm{GF}(2^n)^*)$, $X_2 \sim \mathcal{U}(\mathrm{GF}(2^n))$ and $X_3 \sim \mathcal{L}(\mathrm{GF}(2^n))$. Note that $X_2 \oplus g(X_3)$ is uniform over $\mathrm{GF}(2^n)$ according to the first statement of Proposition 3. Applying the result above to $A = X_1$ and $B = X_2 \oplus g(X_3)$, we deduce that

$$X_4 = X_1 \cdot (X_2 \oplus g(X_3)) \sim \mathcal{U}(\mathrm{GF}(2^n))\,.$$

We are left to prove that $X_4 = X_1 \cdot (X_2 \oplus g(X_3))$ is independent of X_3. For any $x_3, x_4 \in \mathrm{GF}(2^n)$, we have:

$$\mathrm{P}(X_1 \cdot (X_2 \oplus g(X_3)) = x_4, X_3 = x_3) = \mathrm{P}(X_1 \cdot (X_2 \oplus g(x_3)) = x_4, X_3 = x_3)\ .$$

Since X_1, X_2 and X_3 are independent random variables, it follows that $X_1 \cdot (X_2 \oplus g(x_3))$ is independent of X_3 for any $x_3 \in \mathrm{GF}(2^n)$ (see for example Theorem 3.3.2 in Chung [6] p.54). Therefore, we have for any $x_3, x_4 \in \mathrm{GF}(2^n)$

$$\mathrm{P}(X_1 \cdot (X_2 \oplus g(X_3)) = x_4, X_3 = x_3) = \mathrm{P}(X_1 \cdot (X_2 \oplus g(x_3)) = x_4) \cdot \mathrm{P}(X_3 = x_3)\,. \tag{2}$$

Now, applying the result above to the random variables $A = X_1$ and $B = X_2$, we have $X_1 \cdot X_2 \sim \mathcal{U}(\mathrm{GF}(2^n))$. Therefore, considering $X_1 \cdot g(x_3)$ as a random variable over $\mathrm{GF}(2^n)$, we have according to the first statement of this proposition that

$$X_1 \cdot (X_2 \oplus g(X_3)) = X_1 \cdot X_2 \oplus X_1 \cdot g(x_3) \sim \mathcal{U}(\mathrm{GF}(2^n))\,.$$

We conclude that

$$\mathrm{P}(X_1 \cdot (X_2 \oplus g(x_3)) = x_4) = \mathrm{P}(X_1 \cdot (X_2 \oplus g(X_3) = x_4)$$

for any $x_3, x_4 \in \mathrm{GF}(2^n)$. Due to (2), we have for any $x_3, x_4 \in \mathrm{GF}(2^n)$,

$$\mathrm{P}(X_1 \cdot (X_2 \oplus g(X_3)) = x_4, X_3 = x_3) = \mathrm{P}(X_1 \cdot (X_2 \oplus g(X_3)) = x_4) \cdot \mathrm{P}(X_3 = x_3)\,.$$

The result follows. □