Masking with Randomized Look Up Tables
Towards Preventing Side-Channel Attacks of All Orders

François-Xavier Standaert*, Christophe Petit**, Nicolas Veyrat-Charvillon***

Université catholique de Louvain, Crypto Group, Belgium.

Abstract. We propose a new countermeasure to protect block ciphers implemented in leaking devices, at the intersection between One-Time Programs and Boolean masking schemes. First, we show that this countermeasure prevents side-channel attacks of all orders during the execution of a protected block cipher implementation, given that some secure precomputations can be performed. Second, we show that taking advantage of the linear diffusion layer in modern block ciphers allows deriving clear arguments for the security of their implementations, that can be easily interpreted by hardware designers. Masking with randomized look up tables allows fast execution times but its memory requirements are high and, depending on the block cipher to protect, can be prohibitive. We believe this proposal brings an interesting connection between former countermeasures against side-channel attacks and recent formal solutions to cope with physical leakage. It illustrates the security vs. performance tradeoff between these complementary approaches and, as a result, highlights simple design guidelines for leakage resilient ciphers.

Introduction

More than a decade after the introduction of Differential Power Analysis [19], masking cryptographic implementations remains one of the most frequently considered solutions to increase security against such attacks. Its underlying principle is to randomize the sensitive data, by splitting it into d shares, where $d-1$ usually denotes the order of the masking scheme. The masked data and individual mask(s) are then propagated throughout the cryptographic implementation, so that recovering secret information from a side-channel trace should at least require to combine the leakage samples corresponding to these d shares. This is an arguably more difficult task than targeting single samples separately because (1) more "points of interests" have to be identified in the leakage traces, (2) if the masking scheme is properly designed, the mutual information between a secret data and its physical leakage decreases with the amount of shares.

In practice, three main ways of mixing some input data and mask(s) have been proposed in the literature. The first solution, usually referred to as Boolean masking, is to use a bitwise XOR [5, 12]. Multiplicative masking was then proposed as an efficient alternative for the AES, but suffers from some weaknesses,

* Research associate of the Belgian Fund for Scientific Research (FNRS - F.R.S.).
** Postdoctoral researcher of the Belgian Fund for Scientific Research (FNRS - F.R.S.).
*** Postdoctoral researcher funded by the Walloon region SCEPTIC project.

due to the easily distinguishable leakage when multiplying by zero [15]. Finally, the affine masking introduced in [39], and further analyzed in [11], allows combining the advantages of Boolean and multiplicative masking from a security point of view, but it implies costly re-computations during the encryption process.

Attacks against masked implementations range in two main categories. On the one hand, physical imperfections such as glitches can lead to easily exploitable leakage, e.g. in the case of hardware implementations [20, 21]. On the other hand, and more systematically, higher-order attacks that combine the leakage of multiple shares can be applied [23]. Non-profiled higher-order attacks using Pearson's correlation coefficient are discussed in [30] and their profiled counterpart using templates proved their effectiveness in [28]. A careful information theoretic and security analysis of higher-order Boolean masking can be found in [38]. In view of these results, an important issue for circuit designers is to develop efficient higher-order masking schemes. Schramm and Paar proposed one in [35], purposed for software implementations, but it was subsequently shown to be secure only for $d = 2$ [8]. Solutions based on Look Up Tables (LUT) are described in [29], but they are hardly practical (for performance reasons) for any $d > 2$. More recently, a provably secure and reasonably efficient higher-order masking of the AES was proposed at CHES 2010 [34], which can be viewed as an adaptation of Ishai et al.'s private circuits. Note that this state-of-the-art is not exhaustive and many other variations of masking have been proposed, bringing different tradeoffs between efficiency and security, e.g. [1, 16, 26, 27, 33].

In this paper, inspired by two recent works published at FOCS 2010 [4, 10], we propose a new type of masking scheme, extending the power of precomputed LUT. We show that it is possible to obtain security against side-channel attacks of all orders, if some secure refreshing of the tables can be performed prior to the encryption of the data. More specifically, we first show that the combination of an input and mask can be secure against attacks of all orders in this case. Then, we show that the use of Randomized Look Up Tables (RLUT) allows us to extend this security guarantee to the implementation of any S-box. Finally, we show that it is possible to design a substitution-permutation network relying on these principles. Intuitively, these results are possible because in a RLUT design, one of the shares is only manipulated during the secure precomputation, and not during the encryption process. The key advantages of this approach are:

1. Contrary to all previous masking schemes, our proposal leads to secure implementations, even if implemented as a stand-alone solution. In particular, it does not require to be combined with physical noise and the leakage function can leak the full intermediate values during a cryptographic computation.
2. The only randomness to generate online (i.e. after the plaintext has been chosen) is a single n-bit mask, where n is the block cipher bit size.
3. After precomputation is performed, the execution time of an encryption is only moderately increased and similar to the one of first-order masking.

Quite naturally, our proposal also comes with two main drawbacks:

1. The use of randomized tables implies a high memory cost, which strongly depends on the block cipher size, and structure. For a number of modern ciphers (and in particular, the AES Rijndael), it leads to unrealistic overheads. On the positive side, we show that it is possible to design ciphers for which these overheads can be realistic for certain embedded devices.
2. The strong security argument that we prove relies on the strong assumption that secure precomputations are performed in order to refresh random tables. However, we note that this requirement is not completely unrealistic, and could typically correspond to a practical setting where a smart card is operated in a safe environment between different transactions.

Interestingly, our proposed masking scheme is reminiscent of different recent ideas in the area of secure implementations. First, it has remarkable similarities with the One Time Programs (OTP) presented at Crypto 2008 [14], of which a practical implementation has been analyzed at CHES 2010 [18]. As OTP, RLUT exploit a significant precomputation power: randomized tables can in fact be seen as analogous to a masked program, for which one only focuses on preventing side-channel attacks (while the goal of OTP is more general). For this purpose, we consider a practical scenario of challenge-response protocol, where the inputs are provided by an untrusted environment, allowing their masking to be performed online by the leaking device. By contrast, the implementation of [18] assumed securely masked inputs. In addition, we argue that, besides security proofs that require to precompute tables in a perfectly secure environment, the refreshing of RLUT masking is also inherently easy to protect with heuristic countermeasures like shuffling [16]. This allows a large range of tradeoffs, between the formal security guarantee offered by a completely secure precomputation, and different levels of practical security, if the refreshing of the randomized tables is partially leaking. In this respect, we remark that exploiting a partially leaky precomputation would anyway require sophisticated techniques, similar to Side-Channel Analysis for Reverse Engineering (SCARE) [9, 31], that are an interesting scope for further research. Second, the proposal in this paper shares some design principles with white box cryptography, and its intensive use of precomputed tables [40]. Examples of white box DES and AES designs can be found in [6, 7]. Attacks against these white box designs can be found in [2, 13]. Note that these attacks against white-box designs do not imply side-channel key-recovery, because of the relaxed adversarial power we consider. Essentially, a RLUT implementation corresponds to a partially white box design, where all intermediate computations can be leaked to the adversary, but where some memory needs to remain secret. Third, RLUT masking can be seen as a variation of the threshold implementations proposed by Nikova et al. [24, 25]. As in these papers, we require three shares in our protected implementations, in order to ensure independence between secret keys and physical leakages.

Summarizing, masking with randomized look up tables is an appealing connection between practical countermeasures against side-channel attacks and recent solutions designed to prevent physical leakages with the techniques of modern cryptography. By combining parts of the advantages of both worlds, our

analysis leads to clear security arguments with majorly simplified proofs. Admittedly, in terms of performances, it is not straightforwardly applicable to standard algorithms and devices. But our results open the way towards new design principles for low cost block ciphers, in which the complete countermeasure could be implemented for certain sensitive applications requiring high security levels.

1 Masking keyed permutations

Most present block ciphers combine small keyed permutations $\mathsf{p}_k : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$. A usual way to implement such keyed permutations is to use a non-linear S-box s, and to define $\mathsf{p}_k(x) = \mathsf{s}(x \oplus k)$. In this section, we start by re-calling classical masking schemes to protect an S-box implementation.

The idea of masking, intuitively pictured in Figure 1, is to generate a (secret) random mask m on chip and to combine it with an input x, using a mask function g. Then, during the execution of the (e.g. S-box) computations, only the masked values and masks are explicitly manipulated by the device. For this purpose, a correction function c needs to be implemented, so that at the end of the computations, it is possible to remove the mask and output the correct ciphertext. For

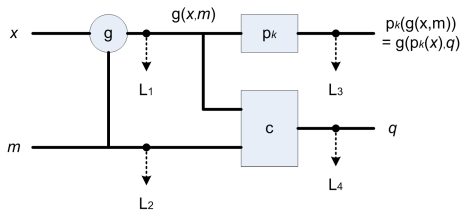

Fig. 1. Masking keyed permutations.

example, in case of the keyed permutation in Figure 1, the correction function is defined such that for all $(x, m) \in \{0,1\}^{2n}$, the following condition is respected:

$$\mathsf{p}_k(\mathsf{g}(x,m)) = \mathsf{g}(p_k(x), \mathsf{c}(\mathsf{g}(x,m), m)).$$

We denote functions with sans serif fonts, random variables with capital letters and sample values with small caps. For simplicity, we also denote the output mask of an S-box as $q = \mathsf{c}(\mathsf{g}(x,m), m)$. Following this description, three main types of masking schemes have been proposed in the literature:

1. Boolean masking [5, 12], in which: $\mathsf{g}(x,m) = x \oplus m$,
2. multiplicative masking [15], in which:: $\mathsf{g}(x,m) = x \cdot m$,
3. affine masking [39, 11], in which: $\mathsf{g}_a(x,m) = a \cdot x \oplus m$,

where $\oplus$ and $\cdot$ denote the addition and multiplication in the field $GF(2^n)$.

2 Second-order side-channel attacks

It is easy to see that masking improves security against side-channel attacks. For example, in Boolean masking, the distribution of the random variable $x \oplus M$, where x is fixed and M is uniformly distributed, is independent of x. This means that if no information is leaked about M, nothing can be learned about x. It is similarly easy to see that higher-order attacks that target the joint distribution $(x \oplus M, M)$ overcome this limitation, since this distribution is not uniform over $\{0,1\}^{2n}$ and depends on x. In order to quantify how much masking reduces the leakage, we perform the information theoretic evaluation proposed in [37]. For this purpose, let us assume that some information is leaked about both $g(x,m)$ and m in Figure 1. As a case study, we can consider the frequently assumed Hamming weight leakage, which gives rise to the variables L_1 and L_2 defined as:

$$L_1 = \mathrm{W_H}(g(x,m)) + N,$$
$$L_2 = \mathrm{W_H}(m) + N,$$

where $\mathrm{W_H}$ is the Hamming weight function and N is a normally distributed random variable, with mean 0 and standard deviation σ_n, representing the measurement noise. In this context, it is possible to evaluate the mutual information:

$$\mathrm{I}(X; L_1, L_2) = -\sum_x \Pr[x] \int_{l_1} \int_{l_2} \Pr[l_1, l_2 | x] \log_2 \Pr[x | l_1, l_2] \; dl_1 dl_2.$$

The results of this information theoretic analysis for a 4-bit S-box are in Figure 2. Note that for affine masking, a third-order attack exploiting the leakage of a could also be applied. It leads to the following observations. First, multiplicative masking has a significantly higher information leakage, due to the "zero problem" detailed in [15]. Second, when noise increases, the slope of the information curves becomes identical for Boolean and affine masking. This confirms previous analyzes in [34, 38], where it is shown that this slope essentially depends on smallest order of a successful attack. The offset between both curves also exhibits the better mix of leakage distributions that affine masking provides.

Note that this information theoretic analysis evaluates the leakage $\mathrm{I}(X; L_1, L_2)$. But analyzing the S-box output leakage $\mathrm{I}(\mathsf{p}_k(X); L_3, L_4)$ would give rise to exactly the same curves as in Figure 2. And assuming known plaintexts and a secret key k, it can also be turned into key leakage $\mathrm{I}(K; X, L_3, L_4)$.

3 Randomized Look Up Tables

The previous higher-order attacks essentially take advantage of the fact that all the shares in a masking scheme are manipulated by a leaking device when computing, e.g. an S-box. Hence, by combining the leakage corresponding to these shares, one can recover secret information. In this section, we show that the use of carefully precomputed tables can significantly improve this situation.

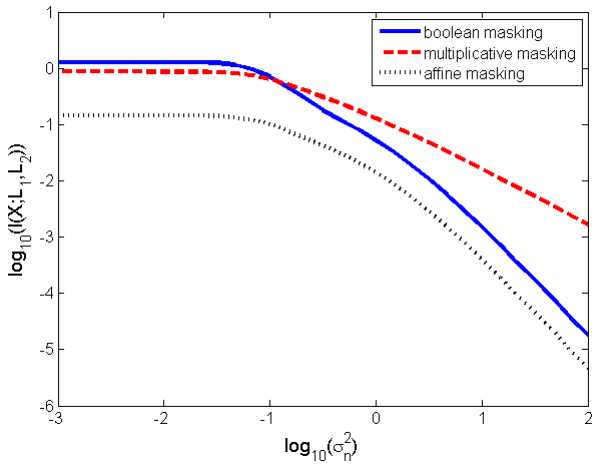

Fig. 2. Information theoretic evaluation of masking schemes.

As a starting point, just observe that any Boolean function of n_1 bits to n_2 bits can be implemented as a $(2^{n_1} \times n_2)$-bit table. Let us then consider a (Boolean-inspired) masking scheme in which $\mathsf{g}_a(x, m) = x \oplus m \oplus a$. As in the case of multiplicative masking in the previous section, we assume that the parameter a is secret, with no information leakage. This can be simply implemented with a $(2^{2n} \times n)$-bit table. If a fresh new table (corresponding to a fresh random a) is precomputed in a leakage-free environment, prior to any S-box computation, a side-channel adversary would only be able to observe leakage of the form $(x \oplus M \oplus A, M)$. The use of such randomized tables has two main advantages:

1. No higher-order attack can be applied, because one of the shares will never be manipulated by the leaking device during the encryption process.
2. Most attacks exploiting parasitic effects in the hardware (e.g. glitches [20, 21]) are discarded, because only memory accesses are performed during the encryption process (i.e. all leaky computations are precomputed).

3.1 Security model

Our security model is similar to the one introduced independently by Brakerski et al. and Dodis et al. at FOCS 2010. As in these works, we consider that some secure precomputation can be performed in a leakage-free environment. But contrary to [18], this precomputation does not require the knowledge of the input x. Once the precomputation task is finished, the actual computation can then be launched in a leaking environment. As illustrated in Figure 3, for an S-box execution, it only requires to perform three memory accesses, corresponding to functions G_1, R and C. In addition, our model for the leakage during this evaluation phase is very general: any input/output of the tables that is accessed during the S-box computation can be given to the adversary. In other words,

there is no restriction on the amount of information leakage during the S-box evaluation: even leakage functions $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{L}_4$ that output the exact values of the tables' inputs and outputs should still give rise to a secure implementation. On the other hand, memory cells that are not accessed during the execution of the block cipher are assumed to be perfectly secure. Note finally that, as a refreshing of the tables is performed prior to every S-box evaluation, the functions G_1, R, C and G_2 appear as random ones to the adversary.

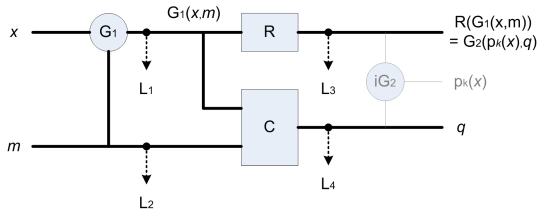

Fig. 3. Masking with Randomized LUT.

In practice, different alternatives are possible to refresh the tables. In the rest of this paper, we mainly focus on a simple solution with minimum (but still significant) randomness requirements. Namely, we refresh all random tables by XORing a random mask at their output and then re-compute the corresponding correction function. Intuitively, refreshing g_1 and g_2 is required to avoid second-order leakages based on $(\mathsf{L}_1, \mathsf{L}_2)$ or $(\mathsf{L}_3, \mathsf{L}_4)$; having g_1 independent of g_2 is required to avoid fourth-order leakages taking advantage of the tables' inputs and outputs; and having a randomized permutation R is required to hide the key. Our table refreshing and S-box evaluation are specified as follows[1]:

Algorithm 1 - Table refreshing.

- **input:** p_k.
1. Pick $a_1 \xleftarrow{R} \{0,1\}^n$;
2. Pick $a_2 \xleftarrow{R} \{0,1\}^n$
3. Pick $a_3 \xleftarrow{R} \{0,1\}^n$;
4. Precompute $\mathsf{g}_1(I, J) = I \oplus J \oplus a_1$;
5. Precompute $\mathsf{r}(I) = \mathsf{p}_k(I) \oplus a_2$;
6. Precompute $\mathsf{g}_2(I, J) = I \oplus J \oplus a_3$;
7. Precompute $\mathsf{c}(I, J) = \mathsf{r}(I) \oplus \mathsf{p}_k(I \oplus J \oplus a_1) \oplus a_3$;

- **output:** $\mathsf{g}_1, \mathsf{r}, \mathsf{g}_2, \mathsf{c}$.

Note that g_2 is not used explicitly during the S-box evaluation but it is necessary to keep it in memory for unmasking, after a secure computation is completed.

[1] Other, more expensive, types of randomization techniques exist and could be considered (e.g. pick up random permutations for r and use latin squares for g).

Algorithm 2 - S-box evaluation on input x.

- **input:** $\mathsf{g}_1, \mathsf{r}, \mathsf{c}$.
1. Pick $m \xleftarrow{R} \{0,1\}^n$;
2. Compute $\mathsf{g}_1(x, m)$;
3. Compute $\mathsf{r}(\mathsf{g}_1(x, m))$;
4. Compute $\mathsf{c}(\mathsf{g}_1(x, m), m)$;
- **output:** $\mathsf{r}(\mathsf{g}_1(x, m))$, $\mathsf{c}(\mathsf{g}_1(x, m), m)$

3.2 Secure S-box computation

The security of the S-box evaluation in Figure 3 is formalized as follows:

Lemma 1. *Let* $\mathsf{g}_1 : \{0,1\}^{2n} \to \{0,1\}^n$, $\mathsf{r} : \{0,1\}^n \to \{0,1\}^n$, $\mathsf{c} : \{0,1\}^{2n} \to \{0,1\}^n$ *and* $\mathsf{g}_2 : \{0,1\}^{2n} \to \{0,1\}^n$ *be four random tables generated according to Algorithm 1. Let* $\mathsf{g}_1(x, m)$, m, $\mathsf{g}_2(\mathsf{p}_k(x), q)$ *and* q, *with* $q = \mathsf{c}(\mathsf{g}_1(x, m), m)$, *be the four intermediate computations produced during the execution of Algorithm 2. Then, we have that the following (4-dimensional) distribution:*

$$\Big(\mathsf{G}_1(x, M), M, \mathsf{G}_2(\mathsf{p}_k(x), Q), Q\Big),$$

taken over the randomized functions $\mathsf{G}_1, \mathsf{G}_2, \mathsf{R}$, *with uniformly distributed* A_1, A_2, A_3 *and* M *in Algorithms 1 and 2, is independent of* (X, K) *and uniform over* $\{0,1\}^{4n}$.

Proof. Let us suppose that K and X are fixed to k and x. Let us also define:

$$\begin{aligned}
L_1 &:= \mathsf{G}_1(x, M) = x \oplus M \oplus A_1,\\
L_2 &:= M,\\
L_3 &:= r(\mathsf{G}_1(x, M)) = \mathsf{p}_k(\mathsf{G}_1(x, M)) \oplus A_2 = \mathsf{p}_k(L_1) \oplus A_2,\\
L_4 &:= L_3 \oplus \mathsf{p}_k(x) \oplus A_3.
\end{aligned}$$

Clearly, L_2 is uniformly distributed. Since A_1 is distributed uniformly and independently of M, L_1 is distributed uniformly and independently of L_2. Since A_2 is distributed uniformly and independently of M and A_1, L_3 is distributed uniformly and independently of L_1, L_2. Since A_3 is distributed uniformly and independently of M, A_1 and A_2, L_4 is distributed uniformly and independently of L_1, L_2, L_3. Therefore, for any key k and input x, the variable (L_1, L_2, L_3, L_4) is uniform. In particular, it is independent of the variables K, X. □

This lemma guarantees that even a complete leakage of the intermediate computations during the execution of Algorithm 2 does not allow to recover any information at all on the actual S-box input x and secret key k.

4 Towards leakage resilient block ciphers

The previous section showed that precomputation allows implementing an S-box securely in the model of Section 3.1. Interestingly, it provides a very simple counterpart to OTP for such an elementary computation. Hence, a natural question

is to know whether this secure S-box computation can be extended towards a full block cipher. In this section, we answer this question positively and provide a proof of security for a quite generic substitution-permutation network. We start with the following definition of an iterated block cipher, from [32]:

Definition 1. *An iterated block cipher is an algorithm that transforms a plaintext block of fixed size n into a ciphertext of identical size, under the influence of a key k, by the repeated application of an invertible transformation ρ, called the round transformation. Denoting the plaintext with x_1 and the ciphertext with x_{N_r+1}, the encryption operation can be written as:*

$$x_{i+1} = \rho_{k_i}(x_i), \qquad i = 1, 2, \ldots N_r,$$

where the different k_i are the round keys generated by a key scheduling algorithm.

We then define a slightly more specific iterated block cipher with linear diffusion:

Definition 2. *An $[n, m, N_r]$ iterated block cipher with linear diffusion is an N_r-round, n-bit iterated block cipher in which the round functions are defined as:*

$$\rho_{k_i}(x_i) = \mathsf{d}(\mathsf{p}_{k_i}(x_i)),$$

where d denotes a bijective transform that is linear over $GF(2^n)$, i.e. $\mathsf{d}(x \oplus y) = \mathsf{d}(x) \oplus \mathsf{d}(y)$, and $\mathsf{p}_{k_i}(x_i)$ consists in the parallel application of an m-bit S-box s:

$$\mathsf{p}_{k_i}(x_i) = \mathsf{s}(x_{i1} \oplus k_{i1}) || \mathsf{s}(x_{i2} \oplus k_{i2}) || \ldots || \mathsf{s}(x_{i\frac{n}{m}} \oplus k_{i\frac{n}{m}}),$$

with x_{ij} representing the jth m-bit block of the input vector x_i. We additionally assume that the block size n is a multiple of the S-box bit size m.

Note that Definition 2 is not a strong restriction as most present block ciphers (e.g. the AES Rijndael) are iterated ones with linear diffusion. Note also that such block ciphers are mainly determined by two parameters: the number of rounds N_r and the number of S-boxes per round $N_s = \frac{n}{m}$. We now show that, independently of the linear diffusion transform, it is possible to implement an iterated block cipher with linear diffusion securely, in the model of Section 3.1.

Figure 4 illustrates a secure implementation of block cipher with $N_s = 2$ and $N_r = 2$. It is essentially a straightforward extension of the previous secure S-box computations, in which each S-box is protected with an independent set of precomputed tables r and c. The S-boxes in the first round are protected exactly as in the previous section. The S-boxes in the second (and following) round(s) are protected according to the slightly modified algorithms 3 and 4 in appendix. The main difference is that the input mask and masking function in the second round are not picked up randomly but provided by the first round.

Theorem 1. *An iterated block cipher with linear diffusion in which the S-boxes in the first round are implemented following Algorithm 2, with the secure precomputation in Algorithm 1, the S-boxes in the other rounds are implemented following Algorithm 4, with the secure precomputation in Algorithm 3, and such that:*

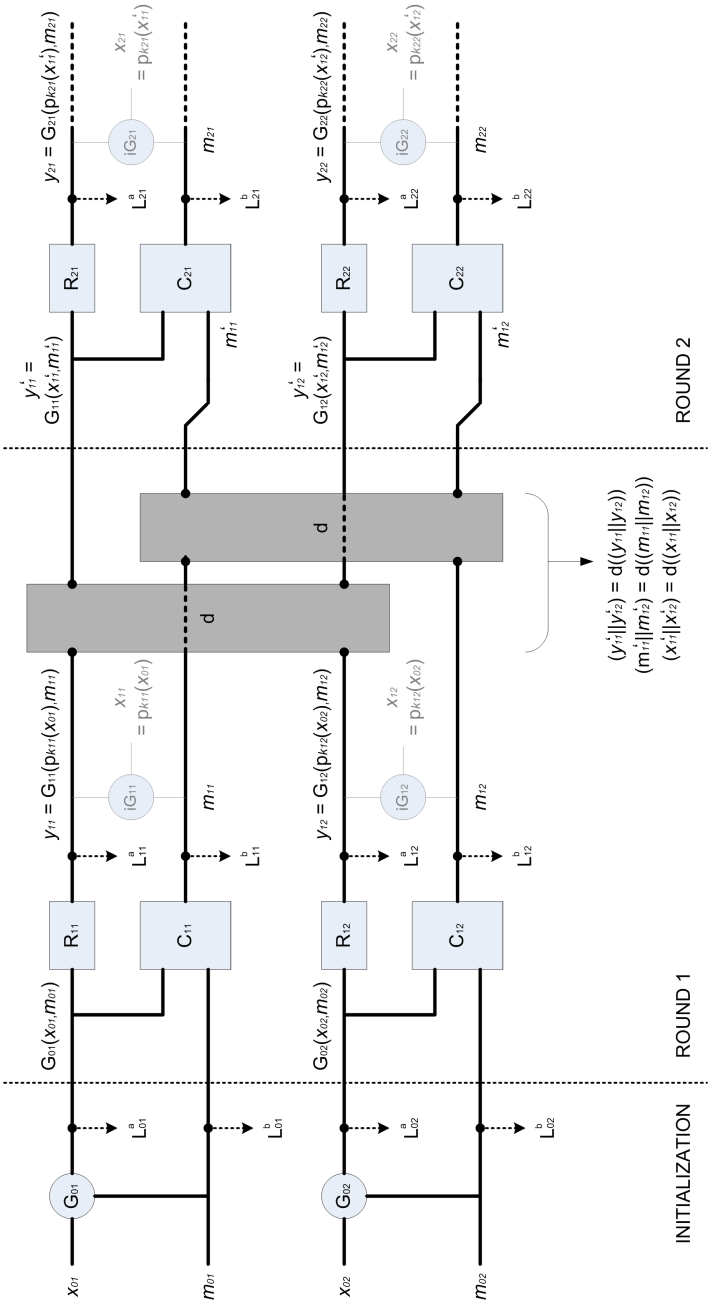

Fig. 4. Leakage resilient Substitution-Permutation Network.

– *The masking function* g *is the same before and after any linear transform* d*,*
– *Given an output mask* m *after the S-box in round* i *(i.e. before the* d *transform), the input mask in round* $i+1$ *is computed as* $m' = \mathsf{d}(m)$*,*

has all its intermediate values generated by computations in Algorithms 2 and 4 uniformly distributed and independent of the input x *and round keys* k_i*.*

Proof. For simplicity, we prove Theorem 1 using the notations of Figure 4, in the case where $N_s = 2$ and $N_r = 2$. The proof trivially extends to larger number of S-boxes and rounds. For this purpose, let us suppose that the inputs x_{0j}'s and round keys k_{ij}'s in Figure 4 are fixed. We show that the leakage $(L^a_{01}, ..., L^b_{22})$ is independent of these inputs and keys and uniformly distributed over $\{0,1\}^{12}$.

We first denote the LSB and MSB parts of the diffusion function output as $\mathsf{d}(i||j) = \mathsf{d}_H(i)||\mathsf{d}_L(j)$. Using Lemma 1, we directly obtain that the leakage vectors $(L^a_{01}, L^b_{01}, L^a_{11}, L^b_{11})$ and $(L^a_{02}, L^b_{02}, L^a_{12}, L^b_{12})$ are uniformly distributed. The two distributions are also independent, as they are produced by independently generated tables. Let us now denote the fresh masks used in the generation of tables r_{21} and r_{22} with Algorithm 3 as $a_{21,2}$ and $a_{22,2}$. And let us denote the fresh masks used in the generation of tables $\mathsf{g}_{21,3}$ and $\mathsf{g}_{22,3}$ with Algorithm 3 as $a_{21,3}$ and $a_{22,3}$. Then, for any particular value $(\ell^a_{01}, \ell^b_{01}, \ell^a_{02}, \ell^b_{02})$ of the initialization leakage vector and $(\ell^a_{11}, \ell^b_{11}, \ell^a_{12}, \ell^b_{12})$ of the first round leakage vector, the leakage vector in the second round can be written as follows:

$$
\begin{aligned}
L^a_{21} &:= \mathsf{p}_{k_{21}}(\mathsf{d}_H(\ell^a_{11}, \ell^a_{12})) \oplus A_{21,2},\\
L^b_{21} &:= L^a_{21} \oplus \mathsf{p}_{k_{21}}(x_{11}) \oplus A_{21,3},\\
L^a_{22} &:= \mathsf{p}_{k_{22}}(\mathsf{d}_L(\ell^a_{11}, \ell^a_{12})) \oplus A_{22,2},\\
L^b_{22} &:= L^a_{22} \oplus \mathsf{p}_{k_{22}}(x_{12}) \oplus A_{22,3}.
\end{aligned}
$$

Again, these variables are independent and uniformly distributed, since $A_{21,2}$, $A_{21,3}$, $A_{22,2}$ and $A_{22,3}$ are independent and uniformly distributed. □

This theorem guarantees that even a complete leakage of the intermediate computations when executing Algorithms 2, 4 and the diffusion transform in Figure 4 does not allow to recover any information on the actual intermediate values x_{ij}'s and round keys k_{ij}'s. Intuitively, this result derives from the fact that, due to its linearity, the diffusion transform is applied independently to the masked intermediate values and masks. As applying a known bijection d to a known value does not reveal any additional information, it has no impact on our proof.

5 Performance analysis

5.1 Memory requirements

Let us consider a block cipher with linear diffusion to be protected with the RLUT countermeasure. Let us also consider the previously introduced parameters: n-bit block size, N_r rounds, m-bit S-boxes, $N_s = \frac{n}{m}$ S-boxes per round. The

implementation of this block cipher will essentially be made of two parts: a table map and a masked program. Intuitively, the table map contains all the randomness that is necessary to refresh the tables in a protected implementation. And the masked program only contains the tables that will actually be used during encryption. As detailed in the previous sections, our security proof only holds if these values are refreshed prior to the encryption of any new plaintext.

The table map is illustrated in Figure 5 for a 3-round cipher with 5 S-boxes per round. Each item in the map represents an m-bit value to be kept in memory. Circles represent the random m-bit values used to generate the g tables. Squares represent the random m-bit values used to generate the r tables. And stars represent the round keys. As the figure illustrates, the total memory required to store such a table map corresponds to $(N_s \cdot N_r) \cdot 3 + N_s$ strings of m bits. For example, the memory cost of the table map for a realistic cipher with parameters $n = 96$, $m = 3$, $N_r = 32$, $N_s = 32$ is 9,312 bits. And the same memory cost for a larger cipher with parameters $n = 128$, $m = 4$, $N_r = 32$, $N_s = 32$ is 12,416 bits.

Next to the table map, the masked program only contains the r and c tables used for encrypting a plaintext. Note that no g function will be explicitly used in the inner round computations. And for the initial masking and final unmasking, it is possible to use a XOR operation (rather than a g table), as the plaintext and ciphertext are supposed to be given to the adversary. This means storing $N_s \cdot N_r$ tables of size $2^m \cdot m$ for the r functions, and $N_s \cdot N_r$ tables of size $2^{2m} \cdot m$ for the c functions. For example, the memory cost of the masked program for a realistic cipher with parameters $n = 96$, $m = 3$, $N_r = 32$, $N_s = 32$ is 221,184 bits. And the same memory cost for a larger cipher with parameters $n = 128$, $m = 4$, $N_r = 32$, $N_s = 32$ is 1,114,112 bits. Clearly, the storage of the c tables is the most memory-consuming, because it has doubled-sized inputs.

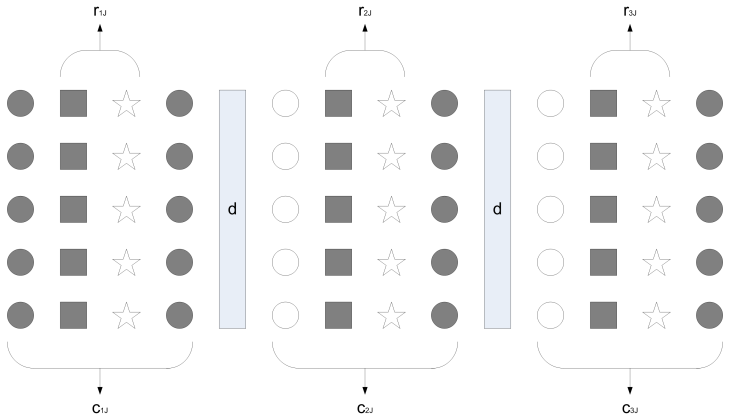

Fig. 5. Masked tables map: circles represent the random m-bit values used to generate the g tables, squares represent the random m-bit values used to generate the r tables, stars represent round keys. Dark grey items require fresh randomness before encryption.

5.2 Time complexity

Next to the memory requirements of a RLUT implementation, another important parameter is the time complexity when refreshing it. First, the refreshing of the table map requires two random strings per S-box and per round. This implies a total of $(N_s \cdot N_r) \cdot 2 + N_s$ random number generations and memory accesses in order to overwrite previous data. Second, the refreshing of the masked program requires to update the r and c tables. This task can be done in 2^m and 2^{2m} bitwise XOR operations per S-box, respectively. Roughly speaking, this suggests that the complete refreshing of a RLUT implementation can be done in approximately $((N_s \cdot N_r) \cdot 2 + N_s) + (N_s \cdot N_r) \cdot 2^m + (N_s \cdot N_r) \cdot 2^{2m}$ elementary operations. Taking the previous exemplary ciphers with 3-bit and 4-bit S-boxes, it means 75,808 and 280,608 elementary operations, respectively. Quite naturally, the exact number of clock cycles required to perform these operations highly depends on the available hardware. Interestingly, these refreshing tasks are inherently parallelizable, which can be an advantage in certain devices.

5.3 On-chip randomization with shuffling

As discussed in the previous section, our proofs only hold if the refreshing of Algorithms 1 and 3 can be performed in a safe environment. However, from a practical point of view, it is interesting to observe that the RLUT countermeasure also has convenient features to remain practically secure, even if its refreshing is partially leaky. For example, this refreshing can be easily combined with the shuffling countermeasure described in [16]. If we denote the number of m-bit memory cells in a table map as N_c, the use of a random pointer $p \xleftarrow{R} [1; N_c]$ allows to refresh the table in a randomized way[2]. The practical security of the refreshing could also be improved by replacing the randomized table $\mathsf{r}(I) = \mathsf{p}_k(I) \oplus a$, with $a \xleftarrow{R} \{0,1\}^m$, by a random permutation $\mathsf{r} \xleftarrow{R} \mathcal{P}$: this would considerably increase the workload of a guessing strategy in side-channel attack. In general, targeting the (partially leaking) refreshing of a block cipher implementation protected with RLUT masking would require to combine higher-order attacks with techniques similar to SCARE [9, 31]. We leave the precise evaluation of this advanced scenario as an interesting scope for further research.

In the same lines, we note that in practical implementations where also the block cipher execution is partially leaking (while our proofs tolerate a full leakage of the intermediate values), it could be possible to refresh only parts of an implementation. As most practical attacks preferably exploit the leakage in the first or last rounds of a block cipher, this is a very convenient feature in order to trade security for performance. One could also trade time for memory, by re-computing some tables in the masked program "on-the-fly", rather than storing them all. Summarizing, there exists a wide range of adaptations of RLUT masking that could provide different levels of practical security.

[2] In practical implementations, the generation of these random strings, as well as those in a table map, could be performed with a low cost pseudorandom number generator.

5.4 Comparison with other approaches

We first mention that comparisons with other countermeasures against side-channel attacks are difficult, as the present proposal is not efficiently applicable to all ciphers. For example, plugging 8-bit S-boxes such as the ones of the AES Rijndael into the performance formulas of the previous section would lead to unrealistic memory requirements. On the other hand, low cost block ciphers are a very active research area and designs based on 4-bit S-boxes, such as PRESENT [3], or even 3-bit S-boxes, such as SEA [36], have been proposed in the literature. So the implementation of RLUT masking is possible for existing ciphers.

Besides, and as already mentioned in the introduction, our proposal has strong connections with the OTP implementation described in [18]. We similarly build masked programs, with two main differences. First, we consider random tables rather than random gates. This makes the execution of a block cipher protected with the RLUT countermeasure (which only requires $N_s \cdot N_r$ table lookups and N_r applications of the d transform) much faster than a OTP. But it implies larger memory requirements, as the cost of masking a $m \times m$-bit table grows with $2^{2m} \cdot m$. Second, we take advantage of the specificities of modern block ciphers, e.g. the linearity of their diffusion transform, that makes their protection easier. As a side-result, we also obtain very simple security arguments and proofs.

The RLUT masking also shares objectives with the higher-order masking scheme of Rivain and Prouff at CHES 2010. Their work describes protected AES implementations for masking of orders $d = 1, 2, 3$. It is an interesting counterpart to our proposal as it implies different types of overheads. [34] only increases the code size moderately. Its main drawback is the execution time. For $d = 3$, it multiplies the one of an unprotected implementation in an 8-bit smart card by more than 100. By contrast, RLUT masking requires a large memory and allows fast online encryption. The main difference between these approaches is their security model: RLUT directly tackle attacks of all orders, while the higher-order masking scheme in [34] uses the order d as a security parameter, which allows to adapt the security level and performances depending on the applications.

Conclusion & open problems

Inspired by previous models and designs in the area of physically observable cryptography, we proposed a new type of masking scheme, based on the use of randomized look up tables. It illustrates that relatively simple principles can lead to strong security guarantees. Admittedly, our solution is not directly applicable to standard algorithms such as the AES Rijndael. But it highlights that the structure of a block cipher has a strong impact on the possibilities to protect its implementations against side-channel attacks. Hence, one important consequence of these results is the statement of clear design principles for leakage-resilient block ciphers. Namely, our analysis gives a strong motivation for designing low cost block ciphers, with small (e.g. 3-bit, 4-bit) S-boxes, and powerful linear transforms, achieving complete diffusion in a small number of rounds.

References

1. M.-L. Akkar, C. Giraud, *An Implementation of DES and AES, Secure against Some Attacks*, CHES 2001, LNCS, vol 2162, pp 309-318, Paris, France, May 2001.
2. O. Billet, H. Gilbert, C. Ech-Chatbi, *Cryptanalysis of a White Box AES Implementation*, SAC 2004, LNCS, vol 3357, pp 227-240, Waterloo, Canada, August 2004.
3. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe, *PRESENT: An Ultra-Lightweight Block Cipher*, CHES 2007, LNCS, vol 4727, pp 450-466, Vienna, Austria, September 2007.
4. Z. Brakerski, Y. Tauman Kalai, J. Katz, V. Vaikuntanathan, *Overcoming the Hole in the Bucket: Public-Key Cryptography Resilient to Continual Memory Leakage*, FOCS 2010, Las Vegas, NV, USA, October 2010.
5. S. Chari, C.S. Jutla, J.R. Rao, P. Rohatgi, *Towards Sound Approaches to Counteract Power-Analysis Attacks*, CRYPTO 1999, LNCS, vol 1666, pp 398-412, Santa Barbara, CA, USA, August 1999.
6. S. Chow, P.A. Eisen, H. Johnson, P.C. van Oorschot, *A White-Box DES Implementation for DRM Applications*, DRM 2002, LNCS, vol 2696, pp 1-15, Washington DC, USA, November 2008.
7. S. Chow, P.A. Eisen, H. Johnson, P.C. van Oorschot, *White-Box Cryptography and an AES Implementation*, SAC 2002, LNCS, vol 2595, pp 250-270, St. John's, Newfoundland, Canada, August 2002.
8. J.-S. Coron, E. Prouff, M. Rivain, *Side Channel Cryptanalysis of a Higher Order Masking Scheme*, CHES 2007, LNCS, vol 4727, pp 28-44, Vienna, Austria, September 2007.
9. R. Daudigny, H. Ledig, F. Muller, F. Valette, *SCARE of the DES*, ACNS 2005, LNCS, vol 3531, pp 393-406, New York, USA, June 2005.
10. Y. Dodis, K. Haralambiev, A. Lopez-Alt, D. Wichs, *Cryptography Against Continuous Memory Attacks*, FOCS 2010, Las Vegas, NV, USA, October 2010.
11. G. Fumaroli, A. Martinelli, E. Prouff, M. Rivain, *Affine Masking against Higher-Order Side-Channel Analysis*, SAC 2010, LNCS, vol xxxx, pp yyy-zzz, Waterloo, Ontario, Canada, August 2010.
12. L. Goubin, J. Patarin, *DES and Differential Power Analysis (The "Duplication" Method)*, CHES 1999, LNCS, vol 1717, pp 158-172, Worcester, MA, USA, August 1999.
13. L. Goubin, J.-M. Masereel, M. Quisquater, *Cryptanalysis of White Box DES Implementations*, SAC 2007, LNCS, vol 4876, pp 278-295, Ottawa, Canada, August 2007.
14. S. Goldwasser, Y. Tauman Kalai, G.N. Rothblum, *One-Time Programs*, CRYPTO 2008, LNCS, vol 5157, pp 39-56, Santa Barbara, CA, USA, August 2008.
15. J.D. Golic, C. Tymen, *Multiplicative Masking and Power Analysis of AES*, CHES 2002, LNCS, 2523, pp 198-212, Redwood Shores, CA, USA, August 2002.
16. C. Herbst, E. Oswald, S. Mangard, *An AES Smart Card Implementation Resistant to Power Analysis Attacks*, ACNS 2006, LNCS, vol 3989, pp 239-252, Singapore, June 2006.
17. Y. Ishai, A. Sahai, D. Wagner, *Private Circuits: Securing Hardware against Probing Attacks*, CRYPTO 2003, LNCS, vol 2729, pp 463-481, Santa Barbara, CA, USA, August 2003.
18. K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, T. Schneider, *Garbled Circuits for Leakage-Resilience: Hardware Implementation and Evaluation of One-Time Programs*, CHES 2010, LNCS, vol 6225, pp 383-397, Santa Barbara, CA, USA, August 2010.

19. P.C. Kocher, J. Jaffe, B. Jun, *Differential Power Analysis*, CRYPTO 1999, LNCS, vol 1666, pp 388-397, Santa Barbara, CA, USA, August 1999.
20. S. Mangard, T. Popp, B.M. Gammel, *Side-Channel Leakage of Masked CMOS Gates*, CT-RSA 2005, LNCS, vol 3376, pp 351-365, San Francisco, CA, USA, February 2005.
21. S. Mangard, K. Schramm, *Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations*, CHES 2006, LNCS, vol 4249, pp 76-90, Yokohama, Japan, October 2006.
22. S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks*, Springer, 2007.
23. T.S. Messerges, *Using Second-Order Power Analysis to Attack DPA Resistant Software*, CHES 2000, LNCS, vol 1965, pp 238-251, Worcester, MA, USA, August 2000.
24. S. Nikova, C. Rechberger, V. Rijmen, *Threshold Implementations Against Side-Channel Attacks and Glitches*, ICISC 2006, LNCS, vol 4307, pp 529-545, Seoul, Korea, December 2006.
25. S. Nikova, V. Rijmen, M. Schläffer, *Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches*, ICISC 2008, LNCS, vol 5461, pp 218-234, Seoul, Korea, December 2008.
26. E. Oswald, S. Mangard, N. Pramstaller, V. Rijmen, *A Side-Channel Analysis Resistant Description of the AES S-Box*, FSE 2005, LNCS, vol 3557, pp 413-423, Paris, France, February 2005.
27. E. Oswald, K. Schramm, *An Efficient Masking Scheme for AES Software Implementations*, WISA 2005, LNCS, vol 3786, pp 292-305, Jeju Island, Korea, August 2005.
28. E. Oswald, S. Mangard, *Template Attacks on Masking - Resistance Is Futile*, CT-RSA 2007, LNCS, vol 4377, pp 243-256, San Francisco, CA, USA, February 2007.
29. G. Piret, F.-X. Standaert, *Security Analysis of Higher-Order Boolean Masking Schemes for Block Ciphers (with Conditions of Perfect Masking)*, IET Information Security, vol 2, num 1, pp 1-11, March 2008.
30. E. Prouff, M. Rivain, R. Bevan, *Statistical Analysis of Second Order Differential Power Analysis*, IEEE Transactions on Computers, vol 58, num 6, pp 799-811, 2009.
31. D. Réal, V. Dubois, A.-M. Guilloux, F. Valette, M. Drissi, *SCARE of an Unknown Hardware Feistel Implementation*, CARDIS 2008, LNCS, vol 5189, pp 218-227, London, UK, September 2008.
32. V. Rijmen, *Cryptanalysis and Design of Iterated Block Ciphers*, PhD thesis, Katholieke Universiteit Leuven, Belgium, October 1997.
33. M. Rivain, E. Dottax, E. Prouff, *Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis*, FSE 2008, LNCS, vol 2008, pp 127-143, Lausanne, Switzerland, February 2008.
34. M. Rivain, E. Prouff, *Provably Secure Higher-Order Masking of AES*, CHES 2010, LNCS, vol 6225, pp 413-427, Santa Barbara, CA, USA, August 2010.
35. K. Schramm, C. Paar, *Higher Order Masking of the AES*, CT-RSA 2006, LNCS, vol 3860, pp 208-225, San Jose, CA, USA, February 2006.
36. F.-X. Standaert, G. Piret, N. Gershenfeld, J.-J. Quisquater, *SEA: a Scalable Encryption Algorithm for Small Embedded Applications*, CARDIS 2006, LNCS, vol 3928 , pp 222-236, Tarragona, Catalonia, Aprils 2006.
37. F.-X. Standaert, T.G. Malkin, M. Yung, *A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks*, EUROCRYPT 2009, LNCS, vol 5479, pp 443-461, Cologne, Germany, April 2009.
38. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, S. Mangard, *The World is Not Enough: Another Look on Second-Order DPA*, Asiacrypt 2010, LNCS, vol 6477, pp 112-129, Singapore, December 2010.

39. M. von Willich, *A Technique with an Information-Theoretic Basis for Protecting Secret Data from Differential Power Attacks*, IMA Conference on Cryptography and Coding 2001, LNCS, vol 2260, pp 44-62, Cirencester, UK, December 2001.
40. B. Wyseur, *White-Box Cryptography*, PhD thesis, Katholieke Universiteit Leuven, Belgium, 2009.

A Modified Algorithms 1 & 2

Algorithm 3 - Table refreshing.

- **input:** p_k, a_1.
1. Pick $a_2 \xleftarrow{R} \{0,1\}^n$
2. Pick $a_3 \xleftarrow{R} \{0,1\}^n$;
3. Precompute $\mathsf{r}(I) = \mathsf{p}_k(I) \oplus a_2$;
4. Precompute $\mathsf{g}_2(I,J) = I \oplus J \oplus a_3$;
5. Precompute $\mathsf{c}(I,J) = \mathsf{r}(I) \oplus \mathsf{p}_k(I \oplus J \oplus a_1) \oplus a_3$;
- **output:** $\mathsf{r}, \mathsf{g}_2, \mathsf{c}$.

Algorithm 4 - S-box evaluation on masked input $\mathsf{g}_1(x,m)$ and mask m.

- **input:** r, c.
1. Compute $\mathsf{r}(\mathsf{g}_1(x,m))$;
2. Compute $\mathsf{c}(\mathsf{g}_1(x,m), m)$;
- **output:** $\mathsf{r}(\mathsf{g}_1(x,m))$, $\mathsf{c}(\mathsf{g}_1(x,m), m)$

Note that $\mathsf{g}_1, \mathsf{g}_2$ are not used explicitly during the S-box evaluation but must be kept in memory for unmasking, after a secure computation is completed.