Side-Channel Analysis of Multiplications in GF(2 128) Application to AES-GCM

Sonia Belaïd, Pierre-Alain Fouque, Benoit Gérard

Side-Channel Analysis of Multiplications in $\mathsf{GF}(2^{128})$

Application to AES-GCM

Sonia Belaïd[1], Pierre-Alain Fouque[2], and Benoît Gérard[3]

[1] École normale supérieure and Thales Communications & Security,
[2] Université de Rennes 1 and Institut Universitaire de France
[3] DGA–MI and IRISA,
Sonia.Belaid@ens.fr
{fouque,benoit.gerard}@irisa.fr

Abstract. In this paper, we study the side-channel security of the field multiplication in $\mathsf{GF}(2^n)$. We particularly focus on $\mathsf{GF}(2^{128})$ multiplication which is the one used in the authentication part of AES-GCM but the proposed attack also applies to other binary extensions. In a hardware implementation using a 128-bit multiplier, the full 128-bit secret is manipulated at once. In this context, classical DPA attacks based on the divide and conquer strategy cannot be applied. In this work, the algebraic structure of the multiplication is leveraged to recover bits of information about the secret multiplicand without having to perform any key-guess. To do so, the leakage corresponding to the writing of the multiplication output into a register is considered. It is assumed to follow a Hamming weight/distance leakage model. Under these particular, yet easily met, assumption we exhibit a nice connection between the key recovery problem and some classical coding and Learning Parities with Noise problems with certain instance parameters. In our case, the noise is very high, but the length of the secret is rather short. In this work we investigate different solving techniques corresponding to different attacker models and eventually refine the attack when considering particular implementations of the multiplication.
Keywords: Field Multiplication, Authenticated Encryption, AES-GCM, Side-Channel.

1 Introduction

The multiplication in $\mathsf{GF}(2^{128})$ is used in several cryptographic algorithms to diffuse a secret parameter. Two widely deployed examples are the authentication encryption mode AES-GCM and the mode of operation OCB. While it is important to guarantee the security of such algorithms against *black-box* attacks, e.g. using the knowledge of the inputs and outputs, it becomes mandatory to thwart *side-channel attacks* for an industrial use.

The main motivation of this work is to show that such multiplication, although manipulating huge part of the secret at once, can be attacked by a

side-channel adversary. Hence, in a major part of this work we consider that the multiplication is an atomic operation (that is performed using a 128-bit multiplier) what is the worst case for an attacker. In an additional part we show that, as one may expect, considering designs having intermediate results indeed provides more leakages and thus lead to more powerful attacks.

As already mentioned, we focus on the application to AES-GCM. Proposed by McGrew and Viega in [25] and standardized by NIST since 2007, this authenticated encryption algorithm aims to provide both confidentiality and integrity. It combines an encryption based on the widely used AES algorithm in counter mode and an authentication based on the GHASH function involving multiplications in $\mathsf{GF}(2^{128})$. This latter one mixes ciphertexts, potential additional data and a secret parameter derived from the encryption key to produce a tag. The security of the algorithm has been analyzed by many authors but despite significant progress in these attacks [28, 16, 30] there is currently no real attack on this mode. The most efficient attacks are the ones described by Ferguson when the tag is very short (32 bits) [14] and by Joux when the nonces are reused [18].

In the particular case of AES-GCM, attacking the multiplier will provide to the attacker the knowledge of the authentication key H. Due to the malleability of the counter mode, the knowledge of the authentication key induces a huge security breach and thus protecting the multiplier is of real importance in contexts where a side-channel attacker is considered. Notice that if the multiplication is protected, then the simple additional countermeasure that consists in masking the tag register is enough to thwart the proposed attack.

Related Work. Some of the algorithms that we consider here come from the coding theory and we think it is a nice view to cast many side-channel attacks. Indeed, a secret value H for instance is encoded as the different leakage values obtained by the adversary. Usually, these leakages allow to recover H, but here for 128 bits, the Hamming weight does not give enough information. Moreover, we only get noisy versions of the leakage values and these values form the codeword with errors. The errors are independent from each other and the noise level is rather high as in many stream cipher cryptanalysis. Given these values, the goal of the adversary is to recover the original message H and the adversary faces a classical decoding problem.

As for the AES-GCM, Jaffe describes in [17] a very efficient Differential Power Analysis attack on its encryption counter mode. Basically, the main idea is to use a DPA attack on the two first rounds of the AES block cipher. Then, as most of the plaintext is the same between two evaluations, it is possible to recover the secret key by guessing parts of the first and second round subkeys. This attack is particularly efficient since it also allows to recover the counter if it is hidden. However, the implementations of AES are now well protected using masking and many papers proposed such countermeasures [8, 29, 9, 15], so that we can assume that it is not possible to recover the secret key on the encryption part.

Our Contributions. In this paper, we consider a particular leakage model where only the storage of values leaks information to the adversary. We assume that each time a value is written in the large register, a noisy version of the Hamming distance or the Hamming weight of this value can be known by the adversary. For instance, in the context of the AES-GCM authentication, the first time the register is written, we can learn the Hamming weight of the multiplication result between the authentication key H and some known value M. Our key point is that the least significant bit of the Hamming weight can be expressed as a linear function of the bits of H. If we are able to find 128 such equations, then it is easy to recover H. However, in side-channel attacks, we only access to noisy versions of the Hamming weight and then, the problem becomes more difficult. Classically, this problem has been known as the Learning Parities with Noise (LPN) [7] and it is famous to have many applications in cryptography. We then consider many attacker models, according to whether the inputs M are known, can be chosen and repeated. If we consider only the tag generation algorithm, additional authentication data can be input to the encryption and these values are first authenticated. We think that this model is powerful and allows us to consider many attacks on different implementations. For instance, since we only consider the writing in the accumulator of the polynomial evaluation, we do not take into account the way the multiplication is implemented and our attack also works even though the multiplication is protected against side-channel attacks.

In the first part of this paper, we consider inputs that are non controlled by the adversary. This is the case for instance in AES-GCM with the authentication of encrypted messages. We show through practical experiments that the proposed attacks may even be successful for reasonable levels of noise if averaging traces is allowed. Then, we consider methods to choose the input values for instance for the additional data or for the tag verification algorithm (e.g., with ciphertexts whose tag is incorrect), so that to improve the basic attacks. In the final part, we discuss three examples. The first one is the mode of operation OCB in which the same multiplication is used and lead to the same attacks. The specificity in this algorithm comes from the uncontrolled messages which are actually advantageously structured for our needs. The second one is a multiplication used in a context of re-keying, which is an alternative of masking. A new secret key is computed for each encryption through a multiplication which is performed differently. The previous attacks do not work in this model. The last example consider classical implementations of the field multiplication in GF(2^{128}) to show how the inner steps can improve the complexities of the aforementioned attacks.

2 Backgrounds, Leakage and Attacker Models

2.1 AES-GCM description

AES-GCM is an authenticated encryption algorithm which aims to provide both confidentiality and integrity. It combines an encryption based on the widely used AES algorithm in counter mode and an authentication with the Galois

mode. The so-called *hash key* H used for the authentication is derived from the *encryption key* K as $H = \mathsf{AES}_K(0^{128})$. The *ciphertext* of the encryption is denoted as $C_1, \ldots, C_n$ where the blocks C_i have length 128 bits except C_n which is of size u ($u \leqslant 128$). Similarly, the additional *authenticated data* is composed of 128-bit blocks $A_1, \ldots, A_m$ where the last one has size ν ($\nu \leqslant 128$). Eventually, we denote by $(X_i)_{0 \leqslant i \leqslant m+n+1}$ the intermediate results of function GHASH with X_{m+n+1} being the output exclusively ored with an encryption of the initial counter to form the tag. Figure 1 illustrates the procedure with the previously defined notations. For the sake of simplicity we will use a single letter M for

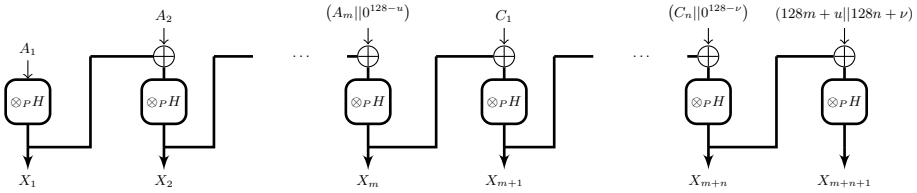

Fig. 1. AES-GCM authentication

both kinds of outputs (A_i or C_i). Then, the definition of the GHASH function can be simply described by the following recursion

$$X_{i+1} = (X_i \oplus M_{i+1}) \otimes_P H, \tag{1}$$

where $\otimes_P$ is the Galois Field multiplication described below.

Galois Field Multiplication. The multiplication $\otimes_P H$ is performed in the field $\mathsf{GF}(2^{128})$ between 128-bit data. For AES-GCM, the Galois' extension is defined by the primitive polynomial $P(Y) = Y^{128} + Y^7 + Y^2 + Y + 1$. We denote by α a root of this polynomial P. An element Q in $\mathsf{GF}(2^{128})$ can be represented by a vector of coefficient $(q_0, q_1, \ldots, q_{127})$ where $Q = \sum_{0 \leqslant i < 128} q_i \alpha^i$. In the following we denote by Q either the element of $\mathsf{GF}(2^{128})$ or the corresponding vector. To avoid ambiguities we will differentiate field multiplication ($\otimes_P$) from matrix/vector multiplication ($\cdot$). Since attacks we present in the paper heavily rely on the linearity of multiplication in $\mathsf{GF}(2^{128})$, we will use a matrix representation of the multiplication. Namely, let Q and R be two elements of $\mathsf{GF}(2^{128})$, the result of the multiplication $Q \otimes_P R$ can be seen as a matrix/vector product $Q_P \cdot R$ where Q_P is a matrix obtained by concatenating columns representing coefficients of $Q \otimes_P \alpha^i$:

$$Q_P = \begin{pmatrix} q_0 & q_{127} & \cdots & q_1 \oplus q_{127} \oplus q_{126} \\ q_1 & q_0 \oplus q_{127} & \cdots & q_2 \oplus q_{123} \oplus q_1 \oplus q_{127} \oplus q_{122} \\ \vdots & \vdots & \ddots & \vdots \\ q_{127} & q_{126} & \cdots & q_0 \oplus q_{127} \oplus q_{126} \oplus q_{121} \end{pmatrix}.$$

2.2 Attacker Context

Leakage Model. A usual assumption (see for instance [24]) when there is no information on the implementation is to consider that all the variables V_i written in the registers of a cryptographic computation leak the sum of their *Hamming weight* (HW) and a *independent noise* ε_σ which follows a Gaussian distribution with a null mean and standard deviation σ (denoted by $\mathcal{N}(0,\sigma)$):

$$L_i^{(\mathsf{HW})} = \mathsf{HW}(V_i) + \varepsilon_\sigma, \quad \varepsilon_\sigma \sim \mathcal{N}(0,\sigma).$$

A common generalization of this leakage model when the attacker is given the successive stored variables is to consider the *Hamming distance* (HD) between two consecutive data V_{i-1} and V_i:

$$L_i^{(\mathsf{HD})} = \mathsf{HD}(V_i, V_{i-1}) + \varepsilon_\sigma = \mathsf{HW}(V_i \oplus V_{i-1}) + \varepsilon_\sigma.$$

This generalization depends on the implementation. If a register is initialized to zero before storing a variable V_i, the Hamming distance between both stored data is exactly the Hamming weight of V_i. However, in the case of a sum for instance, we can reasonably assume that the new computed variable overwrites the stored one (intermediate result), leaking the Hamming distance between them. In the following and in order to cover most embedded devices, we consider both models.

Attacker Model. Now we defined the models for information leakage, we discuss the attacker capabilities. From the axiom "Only Computation Leaks" of Micali and Reyzin [27], we only give the attacker the leakage of the manipulated data. Furthermore, in the most part of this paper, we restrict the leaking data to the multiplication's output to cover all the implementations. We now discuss the three characteristics that define the attacker model.

Known/Chosen Inputs. For the known operands of the Galois field multiplication, we consider the two classical attacker models namely the known message model (e.g., ciphertexts) and the chosen message model (e.g., additional data to authenticate). These two models will be respectively considered in Section 3 and Section 4.

Limited/Unlimited Queries. The attacker may face limitation in the number of queries. Such limitation may be due to time constraints but we may also consider an attacker querying for forged tag verifications in which case an error-counter may limit the number of invalid tag verifications.

Enabled/Disabled Averaging. Eventually, the attacker may be able to average traces obtained for the same computation. This is the case in the chosen messages setting but it may also be the case in the known messages setting when for instance the first blocks to authenticate have a specific format. If such feature is available, then the attacker may execute λ times each computation and average the corresponding traces. Since the leakage model considers an additive Gaussian noise, this decreases the standard deviation of the noise from σ to $\sigma/\sqrt{\lambda}$.

Attack Paths. We present hereafter the key idea of this paper and some preliminary results.

Main Observation. The cornerstone of the attacks presented in Section 3 and Section 4 is the fact that the less significant bit (further referred to as LSB) of the Hamming weight of a variable (equivalently distance between two variables) is a linear function of its bits. While a side-channel attacker generally uses a divide-and-conquer strategy to recover small parts of the key by making guesses, it is not possible anymore as the size of chunks gets large. This prevents attackers from targeting whole 128-bit variables. Nevertheless, in the particular case where the intermediate variable is the output of a linear function involving a public input and the key, then it means that the LSB of the Hamming weight is a linear function of this input and the key. If we denote by $\mathsf{lsb}_0\left(\mathsf{HW}(M \otimes_P H)\right)$ (or also b_0) the bit 0 of the Hamming weight of the product $M \otimes_P H$, we get

$$\mathsf{lsb}_0\left(\mathsf{HW}(M \otimes_P H)\right) = \bigoplus_{0 \leqslant i \leqslant 127} \left(\bigoplus_{0 \leqslant j \leqslant 127} (M \otimes_P \alpha^i)_j \right) h_i. \tag{2}$$

This is precisely what is exploited in the attacks we present. Obviously this work can also be applied to any algorithm in which such multiplication appears and is not restricted to AES-GCM.

First Block. Observing Equation 1, we see that we only know the input of the multiplication with H for the first block of data X_1 since the input of further blocks will depend on H. Moreover, since X_0 is zero, we are in a context where Hamming distance and Hamming weight leakages are equivalent and we thus only refer to the Hamming weight in the following. While the linearity of its parity bit is a very good thing from an attacker point of view, the drawback is that this value is highly influenced by the measurement noise. Assume that we use the following decision procedure to guess the bit from a leakage value,

$$\widetilde{b_0} \stackrel{\text{def}}{=} \mathsf{lsb}_0\left(\lceil \mathsf{HW}(M \otimes_P H) + \varepsilon_\sigma \rfloor\right),$$

where $\lceil \cdot \rfloor$ is the rounding operator. Then, we obtain a noisy bit of information that we can model as follows

$$\widetilde{b_0} = \mathsf{lsb}_0\left(\mathsf{HW}(M \otimes_P H)\right) \oplus b_{\mathcal{N}}. \tag{3}$$

where the error-bit $b_{\mathcal{N}}$ is the potential error due to the Gaussian noise. This error-bit follows a Bernoulli distribution with a parameter p such that the probability of no error is

$$1 - p = \sum_{i=-\infty}^{\infty} \int_{2i-0.5}^{2i+0.5} \phi_{0,\sigma}(t)dt \tag{4}$$

with $\phi_{0,\sigma}(x) = e^{-\frac{x^2}{2\sigma^2}}/(\sigma\sqrt{2\pi})$, $\forall x \in \mathbb{R}$ the probability density function of the Gaussian law with null mean and standard deviation σ. In Table 1 we provide

a few values of this Bernoulli parameter for several standard deviations. Note that we generally evaluate the complexity of an attack according to the Signal-to-Noise Ratio which is the ratio between the signal variance and the noise variance. For 8-bit implementations[4], we consider this SNR around 0.2 [23, 4] which is a typical value both for hardware [19] and software implementations [11]. It corresponds to a signal variance of 8 (with the chosen leakage model) and a noise variance of 10 (standard deviation around 3). While we do not have reference measurements for 128-bit implementations, we can assume that the noise standard deviation is close, that is around 3.

Other Blocks. The generation of traces is expensive for an attacker and in some models it may also be limited. Therefore the number of traces is generally the main criteria when evaluating the complexity of an attack. In the context of the AES-GCM, the authentication is performed through a chained sequence of multiplications. This is quite frustrating for an attacker to only consider the first block when so much information is available. We will discuss in Section 3 and Section 4 how to exploit some of the following multiplications to obtain more bits of information from a single trace.

Other Leakage Bits. As mentioned above, we only exploit the LSB of the leakage for the attacks because it directly depends on a linear combination of the key bits. However, it is also strongly impacted by the noise which involves multiple errors in the system. In this paragraph, we discuss the complexities of considering further bits of leakage. We first focus on the impact of noise on each of them. In this purpose, Table 1 gives the values of the parameters of the Bernoulli law followed by each one of them. They are computed using (4) with i varying from $\lfloor -7\sigma \rfloor$ to $\lceil 7\sigma \rceil$ to capture at least $\left(1f00 - 2.56 \cdot 10^{-10}\right)$ % of the values. The

Table 1. Bernoulli parameter p for different levels of noise with all $\varepsilon \ll 10^{-9}$

std dev σ	Bernoulli parameter p						
	1^{st} bit	2^{nd} bit	3^{rd} bit	4^{th} bit	5^{th} bit	6^{th} bit	7^{th} bit
0.5	$3.1\ 10^{-1}$	$2.7\ 10^{-3}$	ε	ε	ε	ε	ε
1	$0.5 - 4.6\ 10^{-3}$	$1.3\ 10^{-1}$	$4.7\ 10^{-4}$	ε	ε	ε	ε
2	$0.5 - 1.7\ 10^{-9}$	$3.8\ 10^{-1}$	$8.0\ 10^{-2}$	$1.8\ 10^{-4}$	ε	ε	ε
3	$0.5 - \varepsilon$	$4.3\ 10^{-1}$	$2.3\ 10^{-1}$	$1.2\ 10^{-2}$	$2.0\ 10^{-7}$	ε	ε
4	$0.5 - \varepsilon$	$4.5\ 10^{-1}$	$3.2\ 10^{-1}$	$6.1\ 10^{-2}$	$1.1\ 10^{-4}$	ε	ε
5	$0.5 - \varepsilon$	$4.6\ 10^{-1}$	$3.7\ 10^{-1}$	$1.3\ 10^{-1}$	$1.9\ 10^{-3}$	ε	ε

results are directly related to the number of errors in the system which decrease together with the increase of the bits indices. However, the resulting systems are

[4] Notice that our attacks also work on 8-bit implementations where they are more efficient since the attacker can capture intermediate leakage on 8-bit values.

made of equations of higher degrees (exponential with the index):

$$b_i = \bigoplus_{0 \leqslant j_1 < \cdots < j_{2^i} \leqslant 127} \left(\prod_{1 \leqslant \ell \leqslant 2^i} \bigoplus_{0 \leqslant k \leqslant 127} (M \otimes_P \alpha^k)_{j_\ell} \; h_k \right), \quad \forall \; 0 \leqslant i \leqslant 7.$$

and thus are more complicated to solve. In particular, the methods capturing the errors removal like LPN and linear decoding unfortunately do not apply on non-linear systems[5]. We thus have to consider first a solver on the error-free system of equations and then complete its complexity with the errors removal. To the best of our knowledge, one of the most efficient solver is the algorithm F5 [13] provided by Faugere and based on the Gröbner bases. While the solving complexity of the (error-free) quadratic system may be reasonable, it gets computationally impractical when considering the most significant bits[6].

3 Known Inputs

As described in the previous section, for each observed first multiplication, an attacker obtains a noisy Hamming weight value of the output. The LSB of the Hamming weight being linearly dependent on the key (see Equation 2), the attacker can gather many measurements to form a linear system having the authentication key H as solution. In this section we discuss different techniques to solve this noisy linear system. First we propose a simple procedure that allows the attacker to recover the key. Then, we investigate enhancements and other techniques that help decreasing the attack complexity in presence of higher noise.

3.1 Naive Attack

From Equation 2, the (noisy) linear system formed from t messages $(M^{(\ell)})_{0 \leqslant \ell \leqslant t}$ can be written as follows:

$$\mathcal{S} = \begin{cases} \bigoplus_{0 \leqslant i \leqslant 127} \left(\bigoplus_{0 \leqslant j \leqslant 127} \; (M^{(0)} \otimes_P \alpha^i)_j \right) \; h_i = \widetilde{b_0}^{(0)} \\ \qquad \cdots \\ \bigoplus_{0 \leqslant i \leqslant 127} \left(\bigoplus_{0 \leqslant j \leqslant 127} \; (M^{(t-1)} \otimes_P \alpha^i)_j \right) \; h_i = \widetilde{b_0}^{(t-1)}. \end{cases} \tag{5}$$

The values $\widetilde{b_0}^{(\ell)}$ are obtained as in Equation 3 that is we simply round the leakage observations to the closest integer and extract the least significant bits. Once the system $\mathcal{S}$ is correctly defined in $\mathsf{GF}(2)$, we can efficiently and directly solve it (e.g., calling `mzd_solve_left` of the library `m4ri` [1]) if these two conditions are fulfilled:

i) $\mathcal{S}$ contains at least as many linearly independent equations as the number of unknown variables (128),

[5] A linearised system would be involve too many variables to be efficiently solved.
[6] The complexities related to each bit can be computed with the package [6].

ii) there is no error in the bits $\widetilde{b_0}^{(\ell)}$ (i.e., $\widetilde{b_0}^{(\ell)} = b_0^{(\ell)}$).

First, the probability of obtaining a full-rank matrix from k k-bit messages, is $\prod_{0 \leqslant i \leqslant k-1} \left(1 - 2^{i-k}\right)$. In our context (that is $k = 128$), this probability is close to 0.3. To obtain a full-ranked matrix with high probability (say 0.9) the number of additional messages m must satisfy $1 - \prod_{1 \leqslant j \leqslant 2^m} \left(1 - \prod_{0 \leqslant i \leqslant k-1} \left(1 - 2^{i-k}\right)\right) \geqslant 0.9$. For $k = 128$, a single additional message ($m = 1$) makes the equation holds. Note however that the full rank condition does not need to be fulfilled to recover the key. If it is not the attacker should first recover a solution of the system and the kernel of the linear application then test all the solutions to eventually recover the key.

Second, we consider the negative impact of the measurement noise. The latter introduces errors in the system which thus cannot be solved with classical techniques. A simple (naive) solution is to consider that one of the $\widetilde{b_0}^{(i)}$'s is erroneous and to solve k times the system with the k possible vectors $\widetilde{b_0} \oplus \alpha^i$. If the key is not found we can incrementally test all other numbers of errors until the correct key is found. Notice that the inversion is only done once: solving the system with a different vector $\widetilde{b_0}$ only requires a matrix/vector multiplication. If e errors are made among the k messages, then the correct key will be found in at most $C_k^{(e)}$ matrix/vector products:

$$C_k^{(e)} = \sum_{i=0}^{e} \binom{k}{i}. \tag{6}$$

When the number of errors grows, it quickly becomes computationally hard. For instance, for $e = 6$ and $k = 128$, $C_k^{(e)} \approx 2^{32}$. In the next section we investigate techniques to decrease the number of errors in $\mathcal{S}$.

3.2 Improved Attack

In this section, we propose two improvements for the attack. The first one consists in an optimal decision to guess the Hamming weight LSB. This criterium can also be used to advantageously select 128 traces among many more to limit the errors. The second improvement is to show that an attacker can actually use the leakage obtained from the two first multiplications and not only from the first one.

Reducing the Noise.

An Optimal Decision Rule. We propose here to use the LLR statistics (for Log Likelihood Ratio) to derive a bit value $\widehat{b_0}$ *in average* closer to b_0 than $\widetilde{b_0}$ from a leaked Hamming weight. This statistics is extensively used in classical cryptanalysis (an application to linear cryptanalysis can be found in [2]) since the

Neyman-Pearson lemma states that for a binary hypothesis test the optimal[7] decision rule is to compare the LLR to a threshold. The LLR of a leakage ℓ is given by:

$$\mathsf{LLR}(\ell) = \log(\mathbb{P}[b_0 = 0|\ell]) - \log(\mathbb{P}[b_0 = 1|\ell]).$$

The bit b_0 is equally likely equal to 0 or 1 since we have an *a priori* uniform distribution for the secret. Thus, using Bayes relation we obtain that

$$\begin{aligned} \mathsf{LLR}(\ell) &= \log\left(\mathbb{P}[\ell|b_0 = 0]\right) - \log\left(\mathbb{P}[\ell|b_0 = 1]\right) \\ \text{with} \qquad \mathbb{P}[\ell|b_0 = i] &= \sum_{w=0}^{128} \mathbb{P}[\ell|b_0 = i, \mathsf{HW}(\ell) = w]\mathbb{P}[\mathsf{HW}(\ell) = w]. \end{aligned}$$

If the result of $\mathsf{LLR}(\ell)$ is positive it means that the parity bit is likely to be equal to 0. Otherwise, we should assume it is equal to 1. We thus define:

$$\widehat{b_0} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \mathsf{LLR}(\ell) \geqslant 0, \\ 1 & \text{otherwise.} \end{cases}$$

Such technique will decrease the error rate since it boils down to select the most probable value for b_0. Unfortunately it turns out that it has a small impact on the number of errors made, but its combination with the following technique will be useful as illustrated in Figure 2.

Selecting Traces. Nevertheless, when more than k traces are available, it would be of interest to only select the k most reliable ones to decrease the number of errors in the system. Basically, we would like to take into account the confidence we have in a given bit. For instance, assuming a 0 parity bit from a leakage 64.01 seems more reliable than for a leakage equal to 64.49. Interestingly, the higher the absolute value of the LLR is for a given trace, the more confident we are in the choice. Therefore, an attacker should select the n samples with the highest LLR values to form the system. The point is that those k samples may not be linearly independent. Two solutions can then be used:

i) one may only consider a subset of these k samples, solve the system and brute force the remaining bits,
ii) or one may choose k linearly independent samples from the highest LLR values.

Finding the set of k linearly independent samples maximizing this sum is a combinatorial optimization problem which may be quite hard, thus we use a simple "first come/first selected" algorithm that provides a set of k samples. The algorithm iteratively looks for the sample with the highest absolute LLR value that increases the system rank. Figure 2 represents the averaged experimental values (on 10.000 samples) of the Bernoulli parameter p for 500 messages in different scenarios. The black curve represents the use of function *round* to fix

[7] For more precisions about this lemma and the meaning of optimal refer to [10].

$\widetilde{b_0}$. The green one represents the LLR without selection of the best traces while the blue one integers this selection among the first linearly independent traces. Eventually, the red curve models the optimal with the 128 best traces (having the highest LLR values but not necessarily linearly independent). As mentioned,

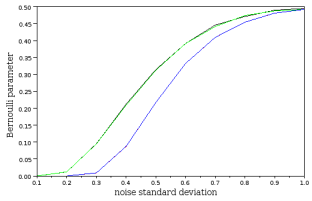

Fig. 2. Bernoulli parameter with rounding (black), LLR (green), traces selection (blue) and best LLR traces (red)

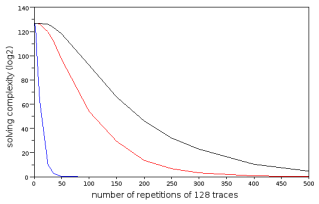

Fig. 3. Solving complexities for several repetitions numbers with $\sigma = 1$ (blue), $\sigma = 3$ (red) and $\sigma = 4$ (black)

we can observe that the use of the LLR does not significantly improve the attack (black and green curves very close). However, the chosen selection of the best LLR allows the attack to resist higher levels of noise: 0.4 (resp. 0.5) instead of 0.3 (resp. 0.4) to achieve the same Bernoulli parameter. Notice that we cannot distinguish the red curve from the blue one. This proximity means that while the "first come/first selected" approach is not optimal it is not worth working on a refined algorithm since the improvement will be bounded by the distance between both curves.

Averaging Traces. In the context where the attacker can monitor few multiplications with the same input, we can also consider another commonly used method which consists in averaging the traces. As claimed in Section 2 and experimentally confirmed in Section 3.3, repeating the traces m times allows to divide the noise standard deviation by $\sqrt{m}$. Figure 3 gives the complexity of removing the errors (averaged from 10,000 tests computed from Eq. 6) according to the number of repetitions of 128 traces for several levels of noise. Note that the full complexity of the attack also includes the system solving (a single inversion in k^3 and $C_k^{(e)} - 1$ matrix/vector products in k^2). Considering it, we can claim that with less than 2^{16} traces (i.e., 500 repetitions), the attacker can practicaly recover the key for σ^2 up to 10 ($\sigma \approx 3$).

Saving Traces with Further Blocks. Up to now we only considered the first multiplication since not knowing H implies not knowing the input of the second multiplication (indeed $X_2 = (M_1 \otimes_P H \oplus M_2) \otimes_P H$). Nonetheless, re-writting this equality as

$$X_2 = M_1 \otimes_P H^2 \oplus M_2 \otimes_P H,$$

we observe that X_2 also is a linear function of H since squaring is linear over $\mathsf{GF}(2)$. Denoting by S the matrix corresponding to the squaring operation, then

$$X_2 = (M_1 \cdot S \oplus M_2) \otimes_P H.$$

Thus a linear relation can also be obtained from the second multiplication substituting $M_1 \cdot S \oplus M_2$ to M in Equation 2. And this is also true in the Hamming distance model since $X_1 \oplus X_2 = (M_1 \cdot S \oplus M_1 \oplus M_2) \otimes_P H$. This observation is of great importance since it significantly improves the complexity of the attacks with a number of required traces divided by a factor two.

3.3 Experimental Results

We illustrate here the necessity of averaging and we confirm the corresponding decreasing of the noise by giving the results obtained on real experiments.

Settings. We implemented the GHASH function on the Virtex-5 FPGA of a SASEBO board and acquired traces from an EM probe. We obtained 10^5 traces that we separated in two $5 \cdot 10^4$ trace sets (Set 1 and Set 2). We then built templates using Set 1 and a projection obtained using the same technique as in [12]. Afterwards we performed the first part of the attack (that is guessing parity bits of Hamming weights with the LLR technique) using this template. We then attacked both sets of traces.

Results. In Table 2 we provide the results we obtained. For a given number of averaging (*av.*) we report (i) the noise standard deviation σ, (ii) the simulated error rate obtained from this standard deviation (10^7 simulations) and (iii) the error rates obtained when applying the template to Set 1 and Set 2.

Table 2. Noise levels and error rates obtained from EM traces.

av.	σ	error rates			av.	σ	error rates		
		simul	Set 1	Set 2			simul	Set 1	Set 2
1	1.958	n/c	n/c	n/c	6	0.770	0.466	0.454	0.467
2	1.287	n/c	n/c	n/c	8	0.637	0.414	0.407	0.457
3	1.063	0.5 - 2.26 10^{-3}	n/c	n/c	10	0.579	0.378	0.370	0.422
4	0.882	0.486	0.483	0.495	12	0.520	0.333	0.338	0.404

First, we see that doubling the number of averaging roughly leads to a reduction of noise standard deviation by a factor $\sqrt{2}$ as we would expect. Second, the attack performs better on the first set since it is the one that have been used to build templates. For Set 1, the error rates actually correspond to theoretical approximations based on the noise standard deviation. We also see that the error rates obtained for Set 2, while obviously deviating from expected values, are significantly decreasing with the number of averaging. Indeed, when averaging is

possible, the obtained features show that an attack can be mount easily. As one can see, Table 2 does not contain data for error rates corresponding to less than 4 averagings. This is due to the fact that the deviation from 0.5 is too small to be estimated using 50,000 traces. We did not managed to get more traces since experiments with higher levels of averaging confirm our predictions.

3.4 Solving the System with more Errors and Advanced algorithms

There are many algorithms to recover the authentication key from noisy Hamming weight LSBs. In the case where more than n multiplications are observed, the attacker will obtain an overdefined linear system. In other words, the attacker will get redundant linear relations involving bits of the key H. Guessed LSBs extracted from leaking multiplication can thus be seen as forming a noisy codeword that encodes the authentication key H using the code defined by the linear relations of the form of Equation 2. Recovering the key is then equivalent to decoding the noisy codeword.

Learning Parities with Noise Algorithms. The Learning Parities with Noise (LPN) problem is the problem of recovering $\boldsymbol{x} \in \mathsf{GF}(2)^k$ given many pairs $(\boldsymbol{a}, (\boldsymbol{a}, \boldsymbol{s}) \oplus e)$ with $\boldsymbol{a} \in \mathsf{GF}(2)^k$ samples randomly chosen, $(\boldsymbol{a}, \boldsymbol{s})$ denotes the scalar product and e is a Bernoulli distribution of parameter p. The most efficient algorithms to solve the LPN problem are based on the Blum-Kalai-Wasserman algorithm [7]. This algorithm tries to perform Gaussian elimination in a smart way but cancelling many bits with one single xor. The idea is to use many samples and xoring those that have many bits in common. However, this algorithm is exponential in the number of samples, time and memory of order $2^{O(k/\log k)}$ where k is the size of the secret values. This algorithm has been improved by Fouque and Levieil in [21] but it allows to reduce the constant in the exponent. In practice, it requires a huge number of samples but since here the size is relatively short $k = 128$, we could use such algorithms. However, since the noise involves a Bernoulli parameter p getting closer to $1/2$, it expects 2^{40} bytes of memory and 2^{34} queries when the standard deviation σ equals 0.5, while it grows to 2^{241} bytes of memory and 2^{334} queries when σ equals 2. Lyubashevsky gave in [22] a variant of BKW with running time $2^{O(k/\log\log k)}$ for $k^{1+\varepsilon}$ samples. A further modification proposed more recently by Kirchner in [20] achieved better runtimes for small values of p. This algorithm runs in time $O(2^{\sqrt{k}})$ with $O(k)$ samples when $p = O(1/\sqrt{k})$.

Linear Decoding. Since inputs are not controlled by the attacker, the corresponding linear code is random. Decoding over random linear codes is known to be a hard problem (NP problem). The currently best algorithm that solves this problem is the one presented by Becker *et al.* in [3] which has complexity $O(2^{0.0494n})$ (where n is the code length). Nevertheless, using such algorithm only makes sense if the noise is low enough to ensure that the actual key-codeword is the closest to the noisy word obtained by the attacker. Indeed if the noise is too

high, then the channel capacity will decrease below the code rate and thus the closest codeword to the obtained noisy one may not be the one the attacker looks for. Using the binary symmetric channel model[8] we obtain that for a standard deviation σ of 0.5 the code length should be at least $\frac{128}{0.107} \approx 1200$ which would yield a complexity $2^{59.28}$ using [3]. Obviously the attacker has better using less than 1200 relations and test more than a single key candidate. To do so she will need a list-decoding algorithm. For cryptographic parameters (that is a key that can be very badly ranked), the only known solution is to see the linear code as a punctured Reed-Muller code and to use a Fast Walsh transform to obtain probabilities for each possible codeword. Since this technique has complexity $O(k2^k)$ with k the code dimension, it is not straightforwardly applicable here. We discuss in Section 4 how we can take profit of controlling inputs to use such decoding algorithm.

3.5 Complexities Evaluation

In this section, we built a system of equations from a new trick, that is the use of a single leakage bit. Then, we discussed methods to solve it involving step by step decoding (Eq. 6) and LLR statistics and the existing tools: LPN and linear decoding. We now propose a comparison of the methods complexities through Table 3 both in terms of number of samples C_s and of computation time C_t. As for the LLR method combined with the step by step error removal (Equation 6) the time complexity C_t includes not only the errors removal $C_k^{(e)}$ but also the the linear system solving (a single inversion in k^3 and $C_k^{(e)} - 1$ matrix/vector products in k^2). As for the number of samples C_s, it is divided by two in all methods thanks to the smart use of the second GCM block X_2. We remark that

Table 3. Complexities of recovering the key with LLR and Eq. 6, LPN and linear decoding according to the level of noise

σ / *Method*	0.1 C_s/C_t	0.2 C_s/C_t	0.3 C_s/C_t	0.4 C_s/C_t	0.5 C_s/C_t
LLR and Eq. 6	$2^8/2^{21}$	$2^8/2^{21}$	$2^8/2^{22}$	$2^8/2^{65}$	$2^8/2^{107}$
LPN (LF Algo)	$2^7/2^{21}$	$2^7/2^{23}$	$2^{26}/2^{28}$	$2^{32}/2^{34}$	$2^{48}/2^{50}$
Linear decoding	$2^6/2^6$	$2^6/2^7$	$2^7/2^{11}$	$2^8/2^{25}$	$2^9/2^{62}$

for low levels of noise (at least until $\sigma = 0.4$), linear decoding is the best method to choose both in terms of number of samples and time complexity. Afterwards, it depends on the number of available samples. Concretely, the more traces we have the less time we need.

[8] That is using the aforementioned Bernoulli parameter as error probability.

4 Chosen Inputs

Let us now consider techniques that may be used to recover the key in the model where the attacker is able to control multiplication inputs. A first idea is that in such context averaging should be considered as obviously enabled[9] and thus measurement noise could be decreased by repeating inputs. Two other ideas are:

i) structuring the messages to make the system easier to solve,
ii) choosing messages to be able to exploit more than two multiplications.

The following is dedicated to the discussion of these two ideas.

4.1 Structured Messages

In Section 3 we saw that recovering the key could be seen as a decoding problem. The difficulty arose from the fact that the linear code corresponding to our attack is random and have a high dimension (128). Assuming the attacker is now able to control inputs of the multiplication, she may choose the underlying code.

Choice of the Code. The question is now *which code should we use?* As a cryptanalyst the requirements for a linear code may be different from the one found in coding theory.

List Decoding. First, an attacker aims at recovering the key. She has computing power and can enumerate many key candidates before finding the correct one. Such a feature means that a list decoding algorithm should be available for the chosen code. Moreover, the list size is not of the same order of magnitude that can be found in coding theory. Ideally, we would like to obtain a list of all key candidates ordered by probabilities of being the correct one. Obviously such a list cannot be created since its size would be 2^{128}. Nevertheless, using the key enumeration algorithm of [31], an attacker can enumerate keys from ordered lists of key chunks. If the linear code underlying the attack is a concatenated code then such algorithm can be used. Indeed, the corresponding matrix of the system would be a block diagonal matrix. Each block corresponds to a smaller linear code that may be fully decoded, that is the attacker obtains a list of all possible keys with the corresponding probabilities.

Soft Information. Second, since the noise may be high, we would like to take profit of the whole available information and not only consider obtained bits $\widetilde{b_0}$ or $\widehat{b_0}$. We illustrated in Section 3.2 the gain obtained when considering LLR statistics to take into account the relations reliabilities. Here, we would like a code which decoding algorithm can exploit such soft information.

[9] Except maybe in pathological cases.

Taking into account the aforementioned constraints, we opted for a concatenating code of smaller random linear codes. The latters can efficiently be decoded using a Fast Walsh Transform (FWT) as mentioned in Section 2. We thus aim at obtaining a matrix corresponding to the system $\mathcal{S}$ of the form

$$\begin{pmatrix} \boxed{\mathcal{S}_0} & & \\ & \boxed{\mathcal{S}_1} & \\ & & \ddots \end{pmatrix} \cdot \begin{pmatrix} \\ H \\ \\ \end{pmatrix} = \begin{pmatrix} \widehat{b_0} \\ \vdots \\ \widehat{b_t} \end{pmatrix}. \tag{7}$$

Generating Structured Inputs. To generate the inputs that yield a matrix similar to the one in (7), the attacker has to consider the application

$$\varphi : M \mapsto \left(\bigoplus_{0 \leqslant j \leqslant 127} \mathsf{lsb}_j \left(M \otimes_P \alpha^0 \right), \ldots, \bigoplus_{0 \leqslant j \leqslant 127} \mathsf{lsb}_j \left(M \otimes_P \alpha^{127} \right) \right)$$

that maps an input M to the corresponding vector of coefficients for the system $\mathcal{S}$. To generate the bloc $\mathcal{S}_c$, she chooses inputs in the kernel of $\varphi_{|\mathcal{I}_c}$ where indices in $\mathcal{I}_c$ correspond to columns outside block $\mathcal{S}_c$. A basis of these kernels are efficiently computed using Gauss eliminations.

Simulations. To illustrate the method results, we give two graphs. The left one presents the averaged rank of the correct key among all the 2^{128} possible ones from the key chunks probabilities according to the noise standard deviations for 256 samples (blue) and 1024 samples (red). The right one is a security graph [32] which draws the evolution of the bounds of the correct key rank according to the number of samples for $\sigma = 0.5$.

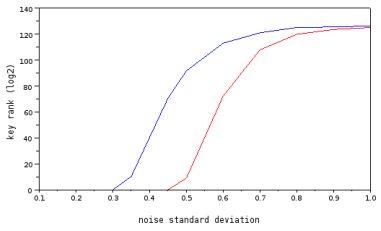

Fig. 4. Key rank for 256 (blue) and 1024 (red) samples

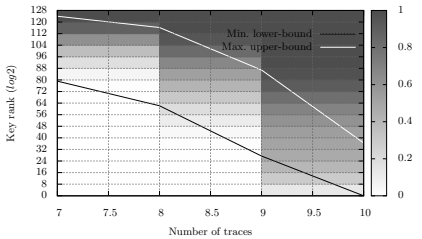

Fig. 5. Security graph for $\sigma = 0.5$

4.2 Saving Traces

A second way, for the attacker, to take profit of the control she has on inputs is to leverage on Ferguson observation [14]. During the specification process of AES-GCM, Ferguson observed that it was possible to obtain a tag that is linearly dependent on the authentication key H in the particular case where the polynomial corresponding to the tag only has non-zero coefficients in positions where the exponent of H is a power of two. This observation relies on the linearity of the squaring operation as mentioned in Section 3.

We saw that this observation allows to exploit the two first multiplications but if the attacker has the control on the inputs she can choose them to do more. Again, this trick can be used either in case of Hamming weight or Hamming distance. The only limitation is that the number of blocks to authenticate grows exponentially in the number of exploitable multiplications. The trade-off will depend on the available time for getting traces and on a potential limitation in the number of queries. To illustrate this we show how an attacker can exploit 3 multiplications in a single trace. From Eq. 1, we obtain the expression of the four first X_i's when M_2 is set to 0:

$$X_1 = M_1 \otimes_P H, \qquad X_3 = M_1 \otimes_P H^3 \oplus M_3 \otimes_P H,$$
$$X_2 = M_1 \otimes_P H^2, \qquad X_4 = M_1 \otimes_P H^4 \oplus M_3 \otimes_P H^2 \oplus M_4 \otimes_P H.$$

We see that relations obtained from X_1, X_2 and X_4 only involve power-of-two of H which means that the relation is a linear function of H. For instance $X_4 = (M_1 \cdot S \cdot S \oplus M_3 \cdot S \oplus M_4) \otimes_P H$, and because she knows S and can choose M_4, the attacker can obtain the input of its choice for the fourth multiplication.

5 Other Applications

In this section, we discuss different applications of the presented attacks. We first consider the OCB mode of operation on which the proposed attacks allow to recover the masks. Then, we look at another multiplication on which our attacks unpractical. Finally, we give some hints on the complexity of our attacks if the attacker has access to the inner parts of the multiplication.

5.1 OCB Mode of Operation

In OCB mode of operation, the masks added before and after the encryption of each block are computed with a multiplication in GF(2^{128}). As in AES-GCM, the process uses a secret constant computed by encrypting the message zero with the secret encryption key. Despite some small differences between the versions OCB1 and OCB2, in both cases, a secret value ($E_K(0^{128})$) is multiplied by a power of two. Thus, the scenario is easier than for the uncontrolled setting since the messages are very sparse. Plus, their shape is close to the one considered in the controlled setting. As a consequence, the secret constant can be recovered in OCB1 and OCB2 at least as well as (but generally easier than) in AES-GCM and allows to recover the masks which are supposed to protect the encryption.

5.2 Re-keying

In [26], Medwed et al. propose to multiply known uniformly distributed 128-bit messages r with a 128-bit secret master key k to generate session keys in the context of re-keying. Each resulting session key is then used for a single encryption of a plaintext block. Doing so, only the generation of the session keys is required to resist Differential Power Analysis attacks. Therefore, the authors only mask this operation but the resulting session key can still leak its (noisy) Hamming weight. The context is thus the same than in AES-GCM but the multiplication is different, the variables being defined on $\mathsf{GF}(2^8)[y]/y^{16}+1$. Re-using the matrix/vector modelization, we can represent the message r being multiplied with the key k by a (16×16) matrix R_p as follows:

$$R_p = \begin{pmatrix} r_0 & r_{m-1} & \cdots & r_1 \\ r_1 & r_0 & \cdots & r_2 \\ \vdots & \vdots & \ddots & \vdots \\ r_{m-1} & r_{m-2} & \cdots & r_0 \end{pmatrix}$$

with the r_i in $\mathsf{GF}(2^8)$. From this matrix, we can easily write the equation involving the LSB b_0 of the Hamming weight of the result:

$$\mathsf{lsb}_0 \left(\mathsf{HW} \left[\left(\bigoplus_{0 \leqslant i \leqslant m-1} r_i \right) \cdot \left(\bigoplus_{0 \leqslant j \leqslant m-1} k_j \right) \right] \right) = b_0$$

with $\cdot$ the multiplication in $\mathsf{GF}(2^8)$. As we can see, only the sum of all the key bytes can be recovered if the attack is successful. However, no individual key bit can be determined. If we extended the attack to more leakage bits, we could (at most) successively recover all the bits of the Hamming weight of the key. It is worth noting that the non-applicability of the attack directly comes from the multiplication's polynomial. Any polynomial with an even number of monomials makes the attack fail when considering only the LSB of the Hamming weight.

5.3 Specific Implementations

Previously, we considered a secure multiplication on $\mathsf{GF}(2^{128})$ for which we just had access to the result. Doing so, we covered all the multiplier implementations like [5] including the protected ones (e.g., with masking). We now show that making assumptions on the multiplier implementation improves the efficiency of our attack. As explained in [25, 33], a usual method to implement a multiplier is to split one of the two operands in smaller blocks (a (128×128)-bit multiplier does not generally fit the area requirements) and perform intermediate multiplications which are progressively accumulated. In the full version of this paper, we focus on two such multipliers with one or the other operand being split. When the secret key is split, the attacker can follow a divide-and-conqueer strategy to recover each block. Nevertheless, when the message is split, the attacker cannot practically enumerate all the possible secret keys. In this case the scenario is still easier than the generic one since we focus on sparse messages of at most n bits.

References

1. Martin Albrecht and Gregory Bard. The M4RI Library – Version 20130416, 2009.
2. Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 432–450. Springer, December 2004.
3. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, April 2012.
4. Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? Cryptology ePrint Archive, Report 2014/053, 2014. http://eprint.iacr.org/2014/053.
5. Daniel J. Bernstein. Faster binary-field multiplication and faster binary-field macs. In *SAC*, Lecture Notes in Computer Science. Springer, 2014.
6. Luk Bettale. Magma Package: Hybrid Approach for Solving Multivariate Polynomial Systems over Finite Fields.
7. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.
8. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, August 1999.
9. Jean-Sébastien Coron. Higher order masking of look-up tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 441–458. Springer, May 2014.
10. Thomas M. Cover and Joy A. Thomas. *Information theory*. Wiley series in communications. Wiley, 1991.
11. François Durvaux, Mathieu Renauld, François-Xavier Standaert, Loïc van Oldeneel tot Oldenzeel, and Nicolas Veyrat-Charvillon. Efficient Removal of Random Delays from Embedded Software Implementations Using Hidden Markov Models. In Stefan Mangard, editor, *CARDIS*, volume 7771 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2012.
12. François Durvaux, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Jean-Baptiste Mairy, and Yves Deville. Efficient selection of time samples for higher-order DPA with projection pursuits. *IACR Cryptology ePrint Archive*, 2014:412, 2014.
13. Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero F5. In *International Symposium on Symbolic and Algebraic Computation Symposium - ISSAC*, 2002.
14. Niels Ferguson. Authentication weaknesses in GCM, 2005. http://csrc.nist.gov/groups/ST/toolkit/BCM/.
15. Vincent Grosso, Emmanuel Prouff, and François-Xavier Standaert. Efficient masked S-boxes processing - A step forward -. In David Pointcheval and Damien Vergnaud, editors, *AFRICACRYPT 14*, volume 8469 of *LNCS*, pages 251–266. Springer, May 2014.
16. Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based MAC algorithms. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 144–161. Springer, August 2008.
17. Joshua Jaffe. A first-order DPA attack against AES in counter mode with unknown initial counter. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 1–13. Springer, September 2007.

18. Antoine Joux. Authentication Failures in NIST version of GCM, 2006. http://csrc.nist.gov/CryptoToolkit/modes/.
19. Toshihiro Katashita, Akashi Satoh, Katsuya Kikuchi, Hiroshi Nakagawa, and Masahiro Aoyagi. Evaluation of DPA Characteristics of SASEBO for Board Level Simulation, 2010.
20. Paul Kirchner. Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377, 2011. http://eprint.iacr.org/2011/377.
21. Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In Roberto De Prisco and Moti Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pages 348–359. Springer, September 2006.
22. Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *APPROX-RANDOM*, volume 3624 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2005.
23. Stefan Mangard. Hardware countermeasures against DPA? a statistical analysis of their effectiveness. In Tatsuaki Okamoto, editor, *CT-RSA 2004*, volume 2964 of *LNCS*, pages 222–235. Springer, February 2004.
24. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
25. D. A. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM), May 2005.
26. Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 279–296. Springer, May 2010.
27. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 278–296. Springer, February 2004.
28. Gordon Procter and Carlos Cid. On weak keys and forgery attacks against polynomial-based MAC schemes. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 287–304. Springer, March 2013.
29. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, August 2010.
30. Markku-Juhani Olavi Saarinen. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 216–225. Springer, March 2012.
31. Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 390–406. Springer, August 2012.
32. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 126–141. Springer, May 2013.
33. Bo Yang, Sambit Mishra, and Ramesh Karri. A high speed architecture for galois/counter mode of operation (GCM). Cryptology ePrint Archive, Report 2005/146, 2005. http://eprint.iacr.org/2005/146.