Improved Side-Channel Analysis of Finite-Field Multiplication

Sonia Belaïd[1], Jean-Sébastien Coron[2], Pierre-Alain Fouque[3], Benoît Gérard[4], Jean-Gabriel Kammerer[5], and Emmanuel Prouff[6]

[1] École normale supérieure and Thales Communications & Security, France
[2] University of Luxembourg, Luxembourg
[3] Université de Rennes 1 et IRISA, France
[4] DGA.MI and IRISA, France
[5] DGA.MI and IRMAR, France
[6] ANSSI, France

Abstract. A side-channel analysis of multiplication in $\mathsf{GF}(2^{128})$ has recently been published by Belaïd, Fouque and Gérard at Asiacrypt 2014, with an application to AES-GCM. Using the least significant bit of the Hamming weight of the multiplication result, the authors have shown how to recover the secret multiplier efficiently. However such least significant bit is very sensitive to noise measurement; this implies that without averaging their attack can only work for high signal-to-noise ratios ($\mathsf{SNR} > 128$). In this paper we describe a new side-channel attack against the multiplication in $\mathsf{GF}(2^{128})$ that uses the most significant bits of the Hamming weight. We show that much higher values of noise can be then tolerated. For instance with an SNR equal to 8, the key can be recovered using 2^{20} consumption traces with time and memory complexities respectively equal to $2^{51.68}$ and 2^{36}. We moreover show that the new method can be extended to attack the fresh re-keying countermeasure proposed by Medwed, Standaert, Großschädl and Regazzoni at Africacrypt 2010.

Keywords: Side-Channel Analysis, Galois Field Multiplication, LPN problem.

1 Introduction

Side-Channel Attacks. The cornerstone of side-channel analysis (SCA for short) is that information about some key-dependent variable x leaks through *e.g.* the power consumption or the electromagnetic information of the device manipulating x. A side-channel attack classically follows a *divide-and-conquer* approach and the secret is recovered by exhaustively testing the likelihood of every possible value for every secret piece. This *modus operandi* implicitly assumes that x depends on a short portion of the secret (for example only 8 bits if x corresponds to the output of the AES sbox). It is particularly suited to the context of software implementations where the processing is sequentially split into operations on data whose size depends on the device architecture (*e.g.* 8 bit or even 32 bit for smart cards).

Side-Channel Analysis of Finite-Field Multiplication. At Asiacrypt 2014 [BFG14], Belaïd, Fouque and Gérard consider a different setting which is more realistic for hardware implementations where many operations are performed simultaneously. Following previous works as [MSGR10,MSJ12], they assume that when performing a multiplication $\mathbf{a} \cdot \mathbf{k}$ over $\mathsf{GF}(2^n)$ for some known $\mathbf{a}$, only the Hamming weight of the result $\mathbf{a} \cdot \mathbf{k} \in \mathsf{GF}(2^n)$ is leaking, with some noise; the goal is to recover the secret multiplier $\mathbf{k}$. Formally, after denoting by $\mathcal{N}(0, \sigma)$ the Gaussian distribution with null mean and standard deviation σ and by HW the Hamming weight over $\mathsf{GF}(2^n)$, for a given basis of $\mathsf{GF}(2^n)$, the SCA then amounts to solve the following problem:

Definition 1 (Hidden Multiplier Problem). *Let* $\mathbf{k} \leftarrow \mathsf{GF}(2^n)$. *Let* $\ell \in \mathbb{N}$. *Given a sequence* $(\mathbf{a}_i, \mathcal{L}_i)_{1 \leq i \leq \ell}$ *where* $\mathbf{a}_i \leftarrow \mathsf{GF}(2^n)$ *and* $\mathcal{L}_i = \mathsf{HW}(\mathbf{a}_i \cdot \mathbf{k}) + \varepsilon_i$ *where* $\varepsilon_i \leftarrow \mathcal{N}(0, \sigma)$, *recover* $\mathbf{k}$.

The Belaïd-Fouque-Gérard Attack and the LPN Problem. As noted in [BFG14], for $\sigma = 0$ (no noise) the above problem is easy to solve. Namely the least significant bit of the Hamming weight of x is the xor of the bits of x. Hence for known $\mathbf{a}_i$ the least significant bit of $\mathsf{HW}(\mathbf{a}_i \cdot \mathbf{k})$ is a linear function of the bits of the secret $\mathbf{k}$. Therefore every Hamming weight gives a linear equation over the n bits of $\mathbf{k}$ and, if the system of equations has rank n (which happens with good probability), the secret $\mathbf{k}$ can be recovered by solving a linear system.

However such least significant bit is very sensitive to the observation noise ε_i. Even for relatively high signal-to-noise ratios (*i.e.,* low σ), this induces a significant error probability for the linear equations. This is all the more damageable that a device is never exactly leaking the Hamming weight of manipulated data, and a modelling (aka epistemic) error therefore adds to the observation noise. The problem of solving a system of noisy linear equations over $\mathsf{GF}(2)$ is known as the Learning Parity with Noise (LPN) problem. New algorithms for solving LPN have recently been proposed [GJL14,BTV15]. The previous best method to solve the LPN problem was the Fouque-Levieil algorithm from [LF06], which is a variant of the algorithm BKW proposed by Blum, Kalai and Wasserman in [BKW00]. According to [BFG14] the Fouque-Levieil algorithm can solve the LPN for $n = 128$ bits with error probability $p = 0.31$ (corresponding to $\mathsf{SNR} = 128$) with 2^{48} acquisitions and 2^{50} complexity (it becomes 2^{334} when $\mathsf{SNR} = 8$). Therefore the Belaïd-Fouque-Gérard algorithm for solving the Hidden Multiplier Problem is quite efficient for relatively high signal-to-noise ratios ($\mathsf{SNR} > 128$); however it becomes prohibitively inefficient for smaller values (*e.g.*, larger values of σ).

Our new Attack. In this paper we describe a new algorithm for solving the Hidden Multiplier Problem, in which we use several most significant bits of the Hamming weight instead of the single least significant bit; we show that much smaller values of SNR can then be tolerated ($\mathsf{SNR} \simeq 8$), which increases the practicability of the attack.

Our technique works as follows. We only keep the observations with small Hamming weight or high Hamming weight. Namely if $\mathsf{HW}(\mathbf{a}_i \cdot \mathbf{k})$ is close to 0, this means that most of the bits of $\mathbf{a}_i \cdot \mathbf{k}$ are equal to 0. This can be written as a system of n

equations over the bits of $\mathbf{k}$, all equal to 0, where some of the equations are erroneous. Similarly if the Hamming weight is close to n, we can assume that all n equations are equal to 1, and we obtain again a set of n noisy equations. Hence in both cases we obtain an instance of the LPN problem. For example, if we only keep observations with Hamming weight less than $n/4$ or greater than $3n/4$, we obtain a set of noisy equations with error probability less than $1/4$.

To solve the LPN problem we will use BKW style algorithms [BKW00]. The main drawback of these algorithms is the huge samples requirement that makes them unpractical for side-channel attacks. In this paper we use some improvements to reduce the query complexity using Shamir-Schroeppel [SS79] or the variant proposed by Howgrave-Graham and Joux in [HGJ10]. We also take advantage of secret-error switching lemma [Kir11,ACPS09] to further reduce the time complexity.

Since our attack is based on filtering for abnormally low or high Hamming weights, it is much less sensitive to noise in Hamming weight measurement than the Belaïd-Fouque-Gérard attack, which relies on the least significant bit of the Hamming weight. Namely even for small SNR (*i.e.,* close to 8), our filtering remains essentially correct, whereas the information from the least significant bit of the Hamming weight is buried in noise and becomes useless. However the main drawback of our attack compared to Belaïd-Fouque-Gérard is that for high SNR a much larger amount of observations is required. Therefore in the latter contexts, the Belaïd-Fouque-Gérard attack is always better.

We also describe an attack when the messages $\mathbf{a}_i$ can be chosen. In that case, the attack becomes much more efficient; we present a distinguisher for the most significant bit and we use BKW algorithms to recover the secret. We also attack a fresh re-keying scheme proposed in [MSGR10] to defeat side-channel cryptanalysis. Whereas the latter scheme is not vulnerable to the technique used in [BFG14], we demonstrate that our attack enables to recover the secret key very efficiently.

Organization of the paper. In Section 2, we recall the field multiplication for the AES-GCM, the leakage model, the LPN problem and the BKW algorithm. Then, we present our new attack in Section 3 and the new algorithmic techniques to reduce the number of queries. In Section 4 we describe a new chosen message attack and in Section 5 our attack on the fresh re-keying scheme. Finally, in Section 6 we present the result of our practical experiments.

2 Preliminaries

2.1 Galois Field Multiplication

For any positive integer n, the finite field of 2^n elements is denoted by $\mathsf{GF}(2^n)$ and the n-dimensional vector space over $\mathsf{GF}(2)$ is denoted by $\mathsf{GF}(2)^n$. Choosing a basis of $\mathsf{GF}(2^n)$ over $\mathsf{GF}(2)$ enables to represent elements of $\mathsf{GF}(2^n)$ as elements of $\mathsf{GF}(2)^n$ and *vice versa*. In the following, we assume that the same basis is always used to represent elements of $\mathsf{GF}(2^n)$ over $\mathsf{GF}(2)$.

This paper analyses the multiplication in the field $\mathsf{GF}(2^n)$, with a particular focus on $n = 128$, with the representation $\mathsf{GF}(2)[x]/(x^{128} + x^7 + x^2 + x + 1)$ which is used in the AES-GCM protocol. If $\mathbf{a} = (\mathrm{a}_0, \mathrm{a}_1, \cdots, \mathrm{a}_{127})$ and $\mathbf{k} = (\mathrm{k}_0, \mathrm{k}_1, \cdots, \mathrm{k}_{127})$ are two elements of $\mathsf{GF}(2^{128})$ viewed as 128-bit vectors, the multiplication $\mathbf{a} \cdot \mathbf{k}$ can be represented by a matrix/vector product in the following way:

$$\begin{pmatrix} \mathrm{a}_0 & \mathrm{a}_{127} & \cdots & \mathrm{a}_1 \oplus \mathrm{a}_{127} \oplus \mathrm{a}_{126} \\ \mathrm{a}_1 & \mathrm{a}_0 \oplus \mathrm{a}_{127} & \cdots & \mathrm{a}_2 \oplus \mathrm{a}_{123} \oplus \mathrm{a}_1 \oplus \mathrm{a}_{127} \oplus \mathrm{a}_{122} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{a}_{127} & \mathrm{a}_{126} & \cdots & \mathrm{a}_0 \oplus \mathrm{a}_{127} \oplus \mathrm{a}_{126} \oplus \mathrm{a}_{121} \end{pmatrix} \cdot \begin{pmatrix} \mathrm{k}_0 \\ \mathrm{k}_1 \\ \vdots \\ \mathrm{k}_{127} \end{pmatrix} = \begin{pmatrix} \mathrm{z}_0 \\ \mathrm{z}_1 \\ \vdots \\ \mathrm{z}_{127} \end{pmatrix} \tag{1}$$

where the product $\cdot$ is processed over $\mathsf{GF}(2)$.

2.2 Probabilities

In this paper we shall use an upper-case letter, *e.g.* X, to denote a random variable, while the lower-case letter, x, shall denote a value taken by X. The probability of an event ev is denoted by $\Pr(ev)$. The *mean* and the *variance* of a random variable X are respectively denoted by $\mathbb{E}(X)$ and $\mathrm{Var}(X)$ (the *standard deviation* of X is the square root of the variance). A *continuous* random variable X with mean μ and variance σ^2 is said to follow a *normal* (Gaussian) distribution if, for every $x \in \mathbb{R}$, $\Pr[X \leqslant x] = \int_{-\infty}^{x} \phi_{\mu,\sigma}(x)$ where $\phi_{\mu,\sigma}$ is the normal *probability distribution function* (pdf) defined by

$$\phi_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} .$$

This is denoted by $X \sim \mathcal{N}(\mu, \sigma)$.

The other distributions used in this paper are the *uniform distribution* over $\mathsf{GF}(2)^n$, denoted by $\mathcal{U}(\mathsf{GF}(2)^n)$, and the Bernoulli distribution $\mathsf{Ber}(p)$ over $\mathsf{GF}(2)$, with $\Pr[X = 1] = p$, for some $p \in [0, 1]$. We shall also use the Binomial distribution which is defined over $\{0, \ldots, n\}$ by $\Pr[X = j] = \binom{n}{j} p^j (1-p)^{n-j}$ and is denoted by $\mathrm{B}(n, p)$.

2.3 Leakage Model

A common assumption is to consider that the processing leaks a noisy observation of the Hamming weight of the manipulated values (see *e.g.* [BCO04]). Namely, for such manipulated value $\mathbf{z} \in \mathsf{GF}(2)^n$, it is assumed that the adversary obtains the following observation $\mathcal{L}(\mathbf{z})$:

$$\mathcal{L}(\mathbf{z}) = \mathsf{HW}(\mathbf{z}) + \varepsilon \ , \tag{2}$$

with an independent noise ε satisfying $\varepsilon \sim \mathcal{N}(0, \sigma)$. That is, the adversary obtains the Hamming weight with noise σ.

In practice, each bit of $\mathbf{z}$ can leak differently. Instead of the Hamming weight, the deterministic part of the observation can hence be modeled as a multivariate polynomial in the bits of $\mathbf{z}$, where the coefficients are taken in $\mathbb{R}$, see [SLP05,RKSF11,DDP13]. For most of current microprocessor architectures, the latter polynomial is well approximated by a linear combination of the bits of $\mathbf{z}$, leading to generalize (2) with

$\mathcal{L}(\mathbf{z}) = \sum_{i=0}^{n-1} \beta_i z_i + \varepsilon$. In Section 6, we will show that our attack also works for such generic leakage model. For simplicity we describe our attack under the noisy Hamming Weight leakage model given by (2) in the rest of the paper.

In the rest of the paper, the level of noise in the observations is quantified with the *signal-to-noise ratio* (SNR for short), that we define as the ratio between the signal variance and the noise variance. This value, which equals $n/(4\sigma^2)$ under Assumption (2), is a useful notion to compare different contexts where the variances of both the signal and the noise are different (which will be the case when comparing our simulations in the Hamming weight model and our experiments on a real device).

As in [BFG14], the main purpose of our attack is to show that we can recover the key $\mathbf{k}$ when we only observe $\mathcal{L}(\mathbf{k} \cdot \mathbf{a_i})$ for many known $\mathbf{a_i}$'s. Thus, we assume that we do not have access to the internal leakage of the field multiplication $\mathbf{k} \cdot \mathbf{a_i}$ and we also assume that the n-bit results are stored in n-bit registers, which is the worst case from an attacker point of view.

2.4 Learning Parities with Noise.

As briefly explained in the introduction the problem of recovering a secret $\mathbf{k}$ from noisy observations of $\mathsf{HW}(\mathbf{a} \cdot \mathbf{k})$ relates to the well known LPN problem whose definition is recalled hereafter.

Definition 2 (Learning Parity with Noise (LPN) Problem). *Let* $\mathbf{k} \in \mathsf{GF}(2)^n$ *and* $p \in (0, 1/2)$. *Given a family of* ν *values* $(\mathbf{a}_i)_{0 \leqslant i < \nu}$ *in* $\mathsf{GF}(2)^n$ *and the family of corresponding observations* $(b_i = \langle \mathbf{a}_i, \mathbf{k} \rangle + e_i)_{0 \leqslant i < \nu}$, *where* $\langle \cdot, \cdot \rangle$ *denotes the scalar product* $\in \mathsf{GF}(2)^n$ *and where the* $\mathbf{a}_i$ *are drawn uniformly in* $\mathsf{GF}(2^n)$ *and the* e_i *are generated according to Bernoulli's distribution* $\mathsf{Ber}(p)$ *with parameter* p, *recover* $\mathbf{k}$.

We denote by $\mathsf{LPN}(n, \nu, p)$ an instance of the LPN problem with parameters (n, ν, p). In this paper, the noisy equations $\langle \mathbf{a}_i, \mathbf{k} \rangle + e_i$ will come from the noisy observations of a device performing field (or ring) multiplications in the form $\mathbf{z} = \mathbf{a} \cdot \mathbf{k}$ in $\mathsf{GF}(2^n)$.

2.5 The BKW algorithm and its Variants

Blum, Kalai and Wassermann described in [BKW00] a subexponential algorithm for solving the LPN problem. This algorithm consists in performing a clever Gaussian elimination by using a small number of linear combinations in order to reduce the dimension of the problem. Then, Levieil and Fouque proposed a practical improvement in [LF06] for the second phase of the algorithm and Kirchner proposed to use the secret-error switching lemma [Kir11,ACPS09] to further improve the method. Later Arora and Ge in [AG11] proposed an algebraic approach in a specifically structured noise and recently Guo *et al.* proposed to use error-correcting code [GJL14].

The BKW algorithm. Given as input $b_i = \langle \mathbf{a}_i, \mathbf{k} \rangle + e_i$ for known $\mathbf{a}_i$'s, the goal of the BKW algorithm is to find linear combinations of the $\mathbf{a}_i$'s with ℓ terms such that:

$$\mathbf{a}_{i_1} \oplus \cdots \oplus \mathbf{a}_{i_\ell} = \mathbf{u}_j \quad , \tag{3}$$

where $\mathbf{u}_j$ for $0 \leq j < n$ is the canonical basis, that is $\mathbf{u}_j$ has its j-th coordinate equal to 1 and the other coordinates are 0. Then one gets:

$$\langle \mathbf{u}_j, \mathbf{k} \rangle = \mathrm{k}_j = \bigoplus_{j=1}^{\ell} b_{i_j} \oplus \bigoplus_{j=1}^{\ell} e_{i_j}.$$

It is not difficult to evaluate the new bias of the linear combination of equations using the Piling-Up lemma. Letting $\delta = 1 - 2p$, for ℓ variables $e_1, \ldots, e_\ell$ such that $\Pr[e_i = 1] = p = (1-\delta)/2$, we have $\Pr[e_1 \oplus \cdots \oplus e_\ell = 0] = \frac{1+\delta^\ell}{2}$. This shows that if we sum ℓ error terms e_i with $\Pr[e_i = 1] = (1-\delta)/2$, the resulting error term e is such that $\Pr[e = 1] = (1-\delta')/2$ with $\delta' = \delta^\ell$. If ℓ is not too large, then the bias of the error term $\bigoplus_{j=1}^{\ell} e_{i_j}$ is also not too large and with enough such equations and a majority vote one can recover the j-th coordinate k_j of $\mathbf{k}$.

Finding linear combinations. To find linear combinations satisfying (3), we first split the $\mathbf{a}_i$'s into a blocks of b bits, where $n = a \cdot b$ (*e.g.* for $n = 128$ we can take $a = 8$ and $b = 32$).

Initially we have ν vectors $\mathbf{a}_i$'s. Consider the rightmost b bits of each $\mathbf{a}_i$, and sort the $\mathbf{a}_i$'s according to this value. There are 2^b possible classes. In every class we xor all elements of the class with one element of the class, and we discard this element. Hence we obtain at least $\nu - 2^b$ new vectors $\mathbf{a}_i^{(1)}$, whose rightmost b bits are zero; these new vectors $\mathbf{a}_i^{(1)}$ are the xor of 2 initial vectors $\mathbf{a}_i$.

One can then proceed recursively. For the next block of b bits we obtain at least $\nu - 2 \cdot 2^b$ vectors $\mathbf{a}_i^{(2)}$ whose rightmost $2b$ bits are zero; these new vectors $\mathbf{a}_i^{(2)}$ are the xor of 4 initial vectors $\mathbf{a}_i$. Stopping at the last-but-one block, we obtain at least $\nu - (a-1) \cdot 2^b$ vectors, for which only the first b-bit block is possibly non-zero, and which are the xor of 2^{a-1} initial vectors $\mathbf{a}_i$. Among these $\nu - (a-1) \cdot 2^b$ vectors, we select the ones equal to the basis vectors $\mathbf{u}_j$ and we perform a majority vote. With the xor of $\ell = 2^{a-1}$ vectors, the bias is:

$$\frac{1 - (1-2p)^{2^{a-1}}}{2}$$

Therefore for the majority vote we need roughly $c/(1-2p)^{2^{a-1}}$ such vectors, for some constant c.

A variant of BKW algorithm is described by Levieil and Fouque in [LF06]. The first phase to find linear combination is similar, however at the end, the Walsh Transform is used to recover all the last b bits of $\mathbf{k}$.

3 Our New Attack

3.1 Overview

In this section, we describe our new side-channel attack on the result of the multiplication in $\mathsf{GF}(2^n)$, which benefits from being weakly impacted by the observation noise. As in [BFG14], we aim at recovering the n-bit secret key $\mathbf{k}$ from a sequence of t queries $(\mathbf{a}_i, \mathsf{HW}(\mathbf{k} \cdot \mathbf{a}_i) + \varepsilon_i)_{0 \leqslant i < t}$ where the $\mathbf{a}_i$ are drawn uniformly in $\mathsf{GF}(2^n)$ and the ε_i are drawn from the Gaussian distribution $\mathcal{N}(0, \sigma)$.

The cornerstone of the attack is to filter the collected measurements to keep only the lowest and the highest Hamming weights. Then we assume that for each lower (resp. higher) Hamming weight, the multiplication result is exactly n bits of zeros (resp. ones). As a consequence, each filtered observation of $\mathbf{z}_i = \mathbf{a}_i \cdot \mathbf{k}$ gives n equations each with some error probability p. In our context, the equations correspond to the row-by-column scalar products in (1) and the binary error associated to the ith equation is denoted by e_i, with $\Pr[e_i = 1] = p$.

Therefore given t messages and corresponding measurements, we get an instance of the LPN$(n, n \cdot t, p)$ problem that we can solve using techniques described in Section 3.3. To correctly scale the latter techniques, we need to know the error probability p with good precision. In the next section we show how to compute p from the filtering threshold and the measurement noise σ in (2).

3.2 Filtering

We describe here how we filter the lowest and highest leakage and we compute the error probabilities of our final set of equations. In order to catch the extreme Hamming weight values of the multiplications results, we choose a threshold real value λ and we filter all the observations below $n/2 - \lambda s$ and above $n/2 + \lambda s$, with $s = \sqrt{n}/2$ the standard deviation of the leakage deterministic part (here the Hamming weight). In the first case, we assume that all the bits of the multiplication result are zeros and in the second case we assume that they are all set to one. In both cases, we get n linear equations on the key bits, each having the same error probability p.

We first compute the proportion of filtered acquisitions before focusing on the error probability p. Let $\mathbf{z} = \mathbf{k} \cdot \mathbf{a}$ be the result of a finite field multiplication; since $\mathbf{z} \sim \mathcal{U}(\mathsf{GF}(2)^n)$, we deduce $\mathsf{HW}(\mathbf{z}) \sim \mathrm{B}(n, 1/2)$. Moreover since

$$\mathcal{L}(\mathbf{z}) = \mathsf{HW}(\mathbf{z}) + \varepsilon \ ,$$

with $\varepsilon \sim \mathcal{N}(0, \sigma)$, we obtain that the pdf h of $\mathcal{L}(\mathbf{z})$ is defined over $\mathbb{R}$ by:

$$h(x) = 2^{-n} \sum_{y=0}^{n} \binom{n}{y} \phi_{y,\sigma}(x) \ .$$

where $\phi_{y,\sigma}(x)$ is the normal probability distribution function with standard deviation σ centered on y.

Since our filtering rejects the observations with leakage $\mathcal{L}(\mathbf{z})$ between $n/2-\lambda s$ and $n/2+\lambda s$ for some parameter λ, the proportion of filtered acquisition $F(\lambda)$ is then:

$$\forall \lambda \in \mathbb{R}, \quad F(\lambda) = 1 - 2^{-n} \sum_{y=0}^{n} \binom{n}{y} \int_{n/2-\lambda s}^{n/2+\lambda s} \phi_{y,\sigma}(t)dt \ . \tag{4}$$

After filtering, our attack consists in assuming that the n bits of $\mathbf{z}$ are all zeroes if $\mathcal{L}(\mathbf{z}) < n/2 - \lambda s$, and are all ones if $\mathcal{L}(\mathbf{z}) > n/2 + \lambda s$. Therefore in the first case out of the n equations, $\mathsf{HW}(\mathbf{z})$ equations are erroneous, whereas in the second case $n - \mathsf{HW}(\mathbf{z})$ equations are erroneous. In the first case, this corresponds to an error probability $\mathsf{HW}(\mathbf{z})/n$, while in the second case this corresponds to an error probability $1 - \mathsf{HW}(\mathbf{z})/n$. On average over filtered observations, we obtain an error probability:

$$p(\lambda) = \frac{1}{F(\lambda)} \sum_{y=0}^{n} \frac{\binom{n}{y}}{2^n} \left(\frac{y}{n} \int_{-\infty}^{n/2-\lambda s} \phi_{y,\sigma}(t)dt + \left(1 - \frac{y}{n}\right) \int_{n/2+\lambda s}^{+\infty} \phi_{y,\sigma}(t)dt \right) .$$

This error probability $p(\lambda)$ (or p for short) is a crucial parameter as it gives the error probability in the LPN problem. Our goal is to minimize p in order to minimize the complexity of solving the LPN problem. This can be done by increasing the filtering threshold λ; however a larger λ implies that a larger number of observations must be obtained initially. Therefore a tradeoff must be found between the error probability p in the LPN problem and the proportion $F(\lambda)$ of filtered observations.

The main advantage of our attack is that this error probability p is quite insensitive to the noise σ in the observations, as illustrated in Table 1. For $n = 128$ and for various values of σ, we provide the corresponding filtering threshold λ that leads to a filtering probability $F(\lambda)$, expressed with $\log_2 1/F(\lambda)$; we then give the corresponding error probability p. For example, for $\mathsf{SNR} = 128$, with $\lambda = 6.00$ we get a filtering probability $F(\lambda) = 2^{-30}$, which means that on average 2^{30} observations are required to get $n = 128$ equations for the LPN problem; in that case the error probability for the LPN problem is $p = 0.23$. We see that this error probability does not grow too fast as SNR decreases, as we get $p = 0.25$ for $\mathsf{SNR} = 8$, $p = 0.28$ for $\mathsf{SNR} = 2$, and $p = 0.34$ for $\mathsf{SNR} = 0.5$.

Table 1: Error probability p and λ w.r.t. the filtering proportion $F(\lambda)$ and the SNR

$\log_2(1/F(\lambda))$	30	25	20	15	10	5	30	25	20	15	10	5
	$\mathsf{SNR} = 128, \sigma = 0.5$						$\mathsf{SNR} = 2, \sigma = 4$					
λ	6.00	5.46	4.85	4.15	3.29	2.16	7.42	6.73	5.97	5.09	4.03	2.64
p	0.23	0.25	0.28	0.31	0.34	0.39	0.28	0.30	0.32	0.34	0.37	0.41
	$\mathsf{SNR} = 8, \sigma = 2$						$\mathsf{SNR} = 0.5, \sigma = 8$					
λ	6.37	5.79	5.14	4.39	3.48	2.28	10.57	9.58	8.48	7.21	5.71	3.73
p	0.25	0.27	0.29	0.32	0.35	0.40	0.34	0.36	0.37	0.39	0.41	0.44

Study in the general case. For completeness, we exhibit hereafter the expressions of the probabilities $F(\lambda)$ and $p(\lambda)$ when the leakage satisfies (2) for another function than $\mathsf{HW}(\cdot)$. If we relax the Hamming weight assumption but still assume that the noise is independent, additive and Gaussian, we get the following natural generalization of (2):

$$\mathcal{L}(\mathbf{z}) = \varphi(\mathbf{z}) + \varepsilon \ ,$$

where $\varphi(\mathbf{z}) \doteq \mathbb{E}\left(\mathcal{L}(Z) \mid Z = \mathbf{z}\right)$ and $\varepsilon \sim \mathcal{N}(0, \sigma)$. This leads to the following generalization of (4):

$$\forall \lambda \in \mathbb{R}, \quad F(\lambda) = 1 - \sum_{y \in \mathrm{Im}(\varphi)} \mathbb{P}\left(\varphi(Z) = y\right) \int_{-\lambda s}^{\lambda s} \phi_{y,\sigma}(t + \mu) dt \ ,$$

where μ and s respectively denote the mean and the standard deviation of $\varphi(Z)$. Analogously, we get:

$$p(\lambda) = \frac{1}{F(\lambda)} \sum_{y=0}^{n} \frac{\binom{n}{y}}{2^n} \left(\frac{y}{n} \int_{-\infty}^{\lambda s} g_y(t + \mu) dt + \left(1 - \frac{y}{n}\right) \int_{\lambda s}^{+\infty} g_y(t + \mu) dt \right) \ ,$$

where for every y, the pdf $g_{\mathcal{L} \mid \mathsf{HW}(Z) = y}$ is defined by:

$$g_y(\ell) = \binom{n}{y}^{-1} \sum_{z \in \mathsf{HW}^{-1}(y)} \phi_{\varphi(\mathbf{z}),\sigma}(\ell) \ .$$

In the case $\varphi = \mathsf{HW}$ (*i.e.,* when the device leaks perfectly in the Hamming weight model), it can be checked that g_y is simply the pdf of $\mathcal{N}(\mathsf{HW}(y), \sigma)$, otherwise it is a Gaussian mixture. In Section 6, we will approximate it by a Gaussian pdf with mean $\mathbb{E}\left(\mathcal{L}(Z) \mid \mathsf{HW}(Z) = y\right)$ and standard deviation $\sqrt{\mathrm{Var}(\mathcal{L}(Z) \mid \mathsf{HW}(Z) = y)}$.

3.3 Solving the LPN Problem

Numerous algorithms for solving LPN are known in the literature; a good survey is given by Pietrzak in [Pie12]. They generally require a huge number of LPN equations. However in our context, these equations come from side-channel acquisitions and thus remain in a rather scarce number. A well-known result of Lyubashevsky reduces the sample complexity, but its limitations on the noise render it inapplicable to our problem [Lyu05]. In this section we summarize the ideas we set-up for solving the LPN problem with a reduced number of samples and under reasonable levels of noise.

We take the point of view of an attacker: she has a limited quantity of side-channel information thus a limited number of initial LPN samples n. She also has a limited computing power and (most importantly) memory. She has two goals: firstly she wants to make sure the attack will indeed be feasible in theory (this depends on the final number of reduced equations), thus she must compute it as exactly as possible (she cannot afford to miss one bit of complexity in the computations). Secondly, she has reasonable but limited resources and want to make the attack as efficient as possible.

Algorithm sketch. In order to have enough samples to apply LPN algorithms, we first artificially square the number ν of LPN samples: for all elements $\mathbf{a}_i$ in the initial set, with error probability p and bias $\varepsilon = 1 - 2p$, , we build the set $(\mathbf{a}_{i,j})_{i \neq j} \doteq (\mathbf{a}_i \oplus \mathbf{a}_j)_{i,j}$. Similarly as in the BKW and $\mathsf{LF1}$ algorithms, we then want to reduce the LPN equations, by finding small-weight linear combinations of $\mathbf{a}_{i,j}$ that have their most significant bits cancelled. However, on the one hand, the BKW algorithm recalled in section 2.5 would not find enough reduced LPN equations. On the other hand, exhaustively looking for linear combinations of (say) at most 4 amplified equations would not be very efficient. Consequently, we make a tradeoff by applying two steps of a generalized birthday paradox-like algorithm [Wag02].

Then assume that we obtain w-bits reduced equations. Once enough equations are found (this depends on the final bias of the equations, which is ε^8), we can directly apply a Walsh-Hadamard transform to recover the w LSB of the secret if the attacker memory is greater than 2^w w-bits words. If we obtain only equations reduced to $w' > w$ bits, we can simply guess the $w' - w$ bits of the secret and do a Walsh on the last w bits. In this case, this exhaustive search space can be reduced using the error/secret switching idea at the very beginning of the algorithm.

The algorithm steps as well as its time and space complexities are analyzed in details in Appendix A.1. From a practical perspective, the optimal choice depends on several parameters: number of traces, filtering ratio, noise of the traces, available memory, computing power. Several trade-offs are thus available to the attacker, listed below.

The most obvious one is to trade side-channel measurements against computing needs. Using more traces will either make it possible to reduce the bias of the selected equations, or increase their number, reducing the time needed by birthday paradox to find reduced equations. In a nutshell, the more traces are available, the better.

Given a fixed number of traces (order of magnitude 2^{20} to 2^{24}), the attacker fixes the filtering threshold λ. Increasing λ improves the bias of the selected equations. Thus we need less reduced equations for the Walsh transform to correctly find w bits of the secret. Nonetheless, increasing λ also reduces the number of initial equations and thus makes the birthday paradox part of the algorithm slower.

Concerning the reduction phase, it is well known that balancing the two phases of the generalized birthday paradox is the best way to reduce its complexity.

Finally doubling the memory will make it possible recover one bit more with the Walsh-Hadamard transform, while slightly more than doubling its time complexity. We will fill the table with equations that are 1 bit less reduced, halving the time needed by the birthday paradox phase. Since in practice the Walsh-Hadamard transform is much faster than finding partial collision, this is worth doing until a certain point if one can afford the memory.

3.4 Comparison with state-of-the art attacks

Compared to [BFG14], our new attack performs better except in one single scenario when $\mathsf{SNR} = 128$ and the number of available queries is very limited by the context. Indeed, for $\mathsf{SNR} = 128$ the attack in [BFG14] requires only 128 observations to get 128 equations with error probability 0.31 whereas our attack requires 2^{15} observations to achieve the same error probability. In the other contexts (*i.e.,* for higher levels of noise) the attack in [BFG14] faces strong limitations. Concretely, recovering the secret key becomes very hard if the inputs are not chosen. On the contrary, since our attack benefits from being quite insensitive to noise, it stays successful even for higher noise levels.

4 Extension to Chosen Inputs

In this section, we present a key-recovery techniques which can be applied when the attacker is able to control the public multiplication operands $\mathbf{a}_i$. It is based on comparing the leakage for related inputs.

4.1 Comparing Leaks

In the so-called *chosen message model*, the attacker chooses ν messages $(\mathbf{a}_i)_{0\leqslant i<\nu}$ in $\mathsf{GF}(2^n)$ and gets the corresponding leakages $\mathcal{L}(\mathbf{k}\cdot\mathbf{a}_i)$, where

$$\mathcal{L}(\mathbf{k}\cdot\mathbf{a}_i) = \mathsf{HW}(\mathbf{k}\cdot\mathbf{a}_i) + \varepsilon$$

From the underlying associative property of the field $\mathsf{GF}(2^n)$, we remark[7] that the relation $(2\cdot\mathbf{a}_i)\cdot\mathbf{k} = 2\cdot(\mathbf{a}_i\cdot\mathbf{k})$ stands for every query $\mathbf{a}_i$. If the most significant bit of $\mathbf{a}_i\cdot\mathbf{k}$ is zero, then the latter relation implies that the bits of $\mathbf{a}_i\cdot k$ are simply shifted when computing $2\cdot(\mathbf{a}_i\cdot\mathbf{k})$ which results in $\mathsf{HW}((2\cdot\mathbf{a}_i)\cdot\mathbf{k}) = \mathsf{HW}(\mathbf{a}_i\cdot\mathbf{k})$. However, if the most significant bit of $\mathbf{a}_i\cdot k$ is one, then the bits are also shifted but the result is summed with the constant value 23, which corresponds to the decimal representation of the binary coefficients of the non-leading monomials of the polynomial $x^{128}+x^7+x^2+x+1$ involved in the representation of the field $\mathsf{GF}(2^{128})$ in $\mathsf{AES\text{-}GCM}$. In this case,the two Hamming weight values $\mathsf{HW}((2\cdot\mathbf{a}_i)\cdot\mathbf{k})$ and $\mathsf{HW}(\mathbf{a}_i\cdot\mathbf{k})$ are necessarily different. Indeed, the bits are shifted, the less significant bit is set to one and the bits of $(\mathbf{a}_i\cdot\mathbf{k})$ at positions 0, 1 and 6 are flipped. Thus, the absolute value of the difference between both Hamming Weight values is either equal to 3 with probability $1/4$ or to 1 with probability $3/4$.

Without noise, we can perfectly distinguish whether both Hamming weight values are equal or not, and thus we can get knowledge of the most significant bit of $\mathbf{a}_i\cdot\mathbf{k}$. Repeating the experiment for every power of two until 2^{128}, that is with 128 queries, gives us the knowledge of every bit of the multiplication result and thus the recovery of $\mathbf{k}$. In presence of noise, the recovery is no longer straightforward. To decide whether the noisy Hamming weights are equal or different, we fix a threshold τ depending on the SNR. Namely, if the distance $|\mathcal{L}((2\cdot\mathbf{a}_i)\cdot\mathbf{k}) - \mathcal{L}(\mathbf{a}_i\cdot\mathbf{k})|$ is greater than τs where s is the signal standard deviation (here the standard deviation of $\mathsf{HW}(Z)$, say $\sqrt{n}/2$), then we decide that $\mathsf{HW}((2\cdot\mathbf{a}_i)\cdot\mathbf{k}) \neq \mathsf{HW}(\mathbf{a}_i\cdot\mathbf{k})$ and thus that the most significant bit of $(\mathbf{a}_i\cdot\mathbf{k})$ equals one. The type I error probability p_{I} associated to this decision (namely

[7] We can simply choose $\mathbf{a}_i$ equal to 1.

the probability of deciding that the Hamming weight values are different while they are equal $p_{\text{dif|eq}}$) satisfies:

$$\begin{aligned} p_{\text{I}} &= \mathbb{P}\left[|\mathcal{L}((2 \cdot \mathbf{a}_i) \cdot \mathbf{k}) - \mathcal{L}(\mathbf{a}_i \cdot \mathbf{k})| > \tau s \mid \mathsf{HW}((2 \cdot \mathbf{a}_i) \cdot \mathbf{k}) = \mathsf{HW}(\mathbf{a}_i \cdot \mathbf{k})\right] \\ &= \mathbb{P}\left[|\varepsilon^{i+1} - \varepsilon^i| > \tau s\right] \\ &= 1 - \int_{-\tau s}^{\tau s} \phi_{\sigma\sqrt{2}}(u)du. \end{aligned}$$

Similarly, the type II error probability p_{II} (of deciding that the Hamming weight values are equal when they are different) satisfies:

$$\begin{aligned} p_{\text{II}} = &\frac{3}{8}\left(\int_{-\tau s-1}^{\tau s-1} \phi_{\sigma\sqrt{2}}(u)du + \int_{-\tau s+1}^{\tau s+1} \phi_{\sigma\sqrt{2}}(u)du\right) \\ &+ \frac{1}{8}\left(\int_{-\tau s-3}^{\tau s-3} \phi_{\sigma\sqrt{2}}(u)du + \int_{-\tau s+3}^{\tau s+3} \phi_{\sigma\sqrt{2}}(u)du\right). \end{aligned}$$

Since, the key bits are all assumed to be balanced between one and zero, the probability of error p for each key bit is equal to $\frac{1}{2}(p_{\text{I}} + p_{\text{II}})$. Table 2 gives the thresholds τ which minimizes the error probability for different values of standard deviations.

Table 2: Optimal threshold and probability of deciding correctly w.r.t. the SNR.

SNR	128	8	2	0.5
σ	0.5	2	4	8
τ	0.094	0.171	0.301	0.536
p	0.003	0.27	0.39	0.46

Note that we did not consider so far the bias induced by the recovery of the less significant bits (whose values have been altered by previous squarings) but in practice we checked that it is negligible and does not change the numeric values of p.

Comparing to Table 1, the error probabilities in Table 2 are much more advantageous since we only need 129 queries. If we are not limited in the number of queries, we can even average the traces to decrease the noise standard deviation and thus improve the success rate. Another improvement is to correlate not only two consecutive powers of 2 but also non-consecutive ones (*e.g.*, 2^j and 2^{j+2}). Without noise, we do not get more information but in presence of noise we can improve the probability of deciding correctly.

4.2 Key Recovery

With the method described above, we only get 128 different linear equations in the key bits. Thus, we cannot use a LPN solving algorithm to recover the secret key in presence of errors. However, since we can average the measurements to decrease the number of errors, we can significantly reduce the level of noise in order to remove the errors almost completely. For instance, with an SNR of 128 (which can also be achieved from an SNR of 2 and 64 repetitions), we get an average of $128 * 0.003 = 0.384$ errors. Solving the system without error is straightforward when we use the powers of two since we directly have the key bits. Thus, inverting all the second members of the equations one-by-one to remove a single error leads to a global complexity of 2^7 key verifications. This complexity is easily achievable and is completely reasonable to recover a 128-bit key.

5 Adaptation to Fresh Re-keying

The core idea of the fresh re-keying countermeasure originally proposed in [MSGR10] for block cipher algorithm is to create a new "session" key from a public *nonce* for each new processing of the encryption algorithm. It guaranties that the secret (master) key is never used directly. To allow for the decryption of the ciphertext, the latter one is sent together with the nonce. For soundness, the key randomization (aka fresh re-keying) must satisfy two properties. First, it must be easy to protect against side channel attacks. Secondly, it must have a good *diffusion* so that each bit of the new (session) key depends on a large number of bits of the master key, rendering attacks based on key-hypotheses testing inefficient. To satisfy the first property, [MSGR10] proposes to base the randomization on linear functions. Efficient techniques are indeed known to secure the latter functions against SCA (*e.g.* higher-order masking can be efficiently applied, as it has linear complexity for linear functions [ISW03,CGP$^+$12]). To additionally satisfy the second property, [MSGR10] proposes to define the linear functions from *circulant* matrices deduced from the random nonce.

Let $\mathbf{k} \in \mathsf{GF}(2^8)^n$ denote the master key which must be protected and let $\mathbf{a} \in \mathsf{GF}(2^8)^n$ denote the nonce (generated at random). The square matrix whose lines correspond to all the rotations of the byte-coordinates of $\mathbf{a}$ (*e.g.* the ith row corresponds to the vector $\mathbf{a}$ right-rotated i times) is denoted by $\mathsf{circ}(\mathrm{a}_0, \cdots, \mathrm{a}_{n-1})$. It satisfies:

$$\mathsf{circ}(\mathbf{a}_0, \cdots, \mathbf{a}_{n-1}) = \begin{pmatrix} \mathbf{a}_0 & \mathbf{a}_{n-1} & \mathbf{a}_{n-2} & \cdots & \mathbf{a}_1 \\ \mathbf{a}_1 & \mathbf{a}_0 & \mathbf{a}_{n-1} & \cdots & \mathbf{a}_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_{n-1} & \mathbf{a}_{n-2} & \mathbf{a}_{n-3} & \cdots & \mathbf{a}_0 \end{pmatrix} ,$$

and the session key $\mathbf{k}'$ is deduced from $(\mathbf{k}, \mathbf{a})$ as follows:

$$\mathbf{k}' = \mathsf{circ}(\mathbf{a}_0, \cdots, \mathbf{a}_{n-1}) \cdot \mathbf{k} \ , \tag{5}$$

where $\cdot$ denotes the scalar product in $\mathsf{GF}(2^8)^n$. Equation (5) implies in particular that the ith byte of $\mathbf{k}'$ satisfies:

$$\mathbf{k}'_i = \sum_{i=0}^{n-1} \mathbf{a}_{i+j \bmod n} \otimes \mathbf{k}_j \ ,$$

where $\otimes$ denotes the multiplication on $\mathsf{GF}(2^8)$.

It may be checked that the attack described in Section 3.1 applies against the multiplication specified by (5) similarly as for the multiplication in (1). Indeed, the matrix-vector product defined in (5) over $\mathsf{GF}(2^8)$ can be rewritten over $\mathsf{GF}(2)$ expressing each bit of $\mathbf{k}'$ as a linear combination of the bits of $\mathbf{k}$ with coefficients being themselves linear combinations of the bits of $\mathbf{a} \in \mathsf{GF}(2)^{128}$. Eventually, exactly like in previous section, for ν filtered messages the attack leads to an instance of the $\mathrm{LPN}(128, 128\nu, p)$ problem.[8] Actually, looking further in the fresh re-keying protocol, we can improve the attack by taking advantage of the context in which the fresh re-keying is used.

Until now, we have made the assumption that the multiplication output was stored in a 128-bit register, which essentially corresponds to an hardware implementation and is the worst case from the attacker point of view. If we switch to a software implementation *e.g.* running on a w-bit architecture, then the attacker can now target the manipulation of w-bit sub-parts of the refresh key $\mathbf{k}'$ which puts him in a more favourable context. By moreover assuming that $\mathbf{k}'$ is used as a secret parameter of a block cipher like AES (as proposed in [MSGR10]), then the attacker can exploit exploit information leakage when the byte-coordinates of $\mathbf{k}'$ are manipulated separately, as in the first round of the AES. Observing the manipulation of each of the sixteen 8-bit chunks separately gives, for a same filtering ratio, a much lower error probability on the equations that what was achieved in the previous (hardware) context. This can be explained by the fact that exhibiting extreme Hamming weight values is obviously much more easier on 8 bits than on 128 bits. For instance, filtering one observation over 2^{10} (i.e., $F(\lambda) = 2^{-10}$) with a SNR equal to 2 results in an error probability of $p = 0.28$ for $n = 128$ and $p = 0.065$ for $n = 8$, that is more than four times less. Table 3 gives the error probability p according to the proportion of filtered acquisitions $F(\lambda)$ for $\mathsf{SNR} = 128$, $\mathsf{SNR} = 8$, $\mathsf{SNR} = 2$ and $\mathsf{SNR} = 0.5$ (as in Table 1) and $n = 8$. This confirms on different parame-

Table 3: Error probability p according to the proportion of filtered acquisitions $F(\lambda)$.

$\log_2(1/F(\lambda))$	10	5	4	3	2	1	10	5	4	3	2	1
	$\mathsf{SNR} = 128$, $\sigma = 0.125$						$\mathsf{SNR} = 2$, $\sigma = 1$					
λ	2.93	2.15	2.02	1.47	1.33	0.71	3.88	2.62	2.28	1.89	1.42	0.83
p	$2.8 \cdot 10^{-19}$	0.09	0.11	0.17	0.21	0.28	$6.5 \cdot 10^{-2}$	0.16	0.19	0.22	0.26	0.32
	$\mathsf{SNR} = 8$, $\sigma = 0.5$						$\mathsf{SNR} = 0.5$, $\sigma = 2$					
λ	3.25	2.26	1.97	1.63	1.24	0.74	5.66	3.73	3.22	2.66	1.99	1.17
p	$5.9 \cdot 10^{-3}$	0.10	0.14	0.18	0.23	0.29	0.17	0.25	0.28	0.30	0.33	0.37

ters that with much fewer observations, we have smaller error probabilities. Therefore, even for $F(\lambda) = 0.5$ (i.e., we only filter one observation over two), the system can be solved to recover the 128-bit key. Furthermore, it is worth noting that this new attack on an AES using a one-time key allows to recover the master key without observing any leakage in the fresh re-keying algorithm.

[8] As observed by the authors of [BFG14], the attack in [BFG14] does not apply to the multiplication specified by (5), essentially because of the circulant property of the matrix.

Note that using this trick which consists in observing the leakage of eight bits of the session key in the first round of the AES, we can also mount an attack towards the outlines of the approach proposed in [BFG14] against the AES-GCM multiplication. Since in this case only the first matrix row is involved in the computation, the coefficients of the key bits are finally different and each observation gives a useful linear equation. Plus, since we observe the leakage on 8-bit data, the noise impact on the less significant bit of Hamming weight is reduced, which improves the system solving. However, the resulting attack remains much less efficient than our new attack, even in the number of required observations.

6 Practical Experiments

We showed in previous sections how to mount efficient side-channel attacks on finite-field multiplication over 128-bit data in different scenarios according to the attacker capabilities. In order to verify the truthfullness of our leakage assumptions, we have mounted a few of these attacks in practice and made some simulations. Namely, we

- implemented both the AES-GCM multiplication and the complete fresh re-keying protocol on an ATMega328p and have measured the leakage thanks to the Chip-Whisperer kit [OC14],
- obtained the 100.000 traces of 128-bit multiplication from [BFG14] corresponding to the EM radiations of an FPGA implementation on the Virtex 5 of a SASEBO board,
- compared error probabilities obtained from traces to the theoretical expectations,
- performed attacks on simulated traces for a 96-bit multiplication,
- performed attacks on the fresh-rekeying implementation.

We first illustrate the behaviour of the leakage we obtained on the ATMega328p. Then we present experimental confirmations that the filtering step actually behaves as expected. With this result in mind we report simulated attacks on 96-bit multiplication and expected complexities for 128-bit attacks. We finally show how efficient is the attack on fresh-rekeying when the attacker is able to exploit 8-bit leakages of the first round of AES.

Eventually, the reader may find in Appendix B an experiment corresponding to the chosen-message attack presented in Section 4 for a 128-bit multiplication implemented on the ATMega328p.

6.1 ATMega328p Leakage Behaviour

The aim of these experiments is to be easily reproducible since all the material required to make the measurements is freely accessible. Since we are in software on an 8-bit implementation, we simulate a 128-bit leakage by summing the intermediate leakage on 8-bit parts of the result [9].

[9] We could have chosen to adapt our results to target the 8-bit chunks directly, which would correspond to applying our attack to a software implementation of AES-GCM. Our purpose

We randomly generated $100,000$ vectors $\mathbf{a} \in \mathsf{GF}(2)^{128}$ and, for a fixed key $\mathbf{k}$, we measured the leakage during the processing of $\mathbf{z} = \mathbf{a} \cdot \mathbf{k}$ as specified in AES-GCM (see (1)). Each measurement was composed of $4,992$ points among which we detected 16 points of interest by following a T-test approach as *e.g.* described in [GJJR11]. We afterwards verified that these points corresponded to the manipulation of the byte-coordinates $Z[i]$ of $\mathbf{z}$ after the multiplication processing.

In the top of Figure 1, we plot for each $i \in [1..16]$ the distribution of our estimates of the mean of the leakage $\mathcal{L}(Z[i])$ knowing either $Z[i] = z \in \mathsf{GF}(2)^8$ (left-hand figure) or $\mathsf{HW}(Z[i]) = y \in [0..8]$ (right-hand figure). First, it may be observed that all the byte-coordinates, except the first one, leak quite similarly. For each $i \in [1..16]$, we denote by $g_{\mathrm{Id},i}(z)$ the function $z \mapsto \mathbb{E}\left(\mathcal{L}(Z[i]) \mid Z[i] = z\right)$ and by $g_{\mathrm{HW},i}(y)$ the function $y \mapsto \mathbb{E}\left(\mathcal{L}(Z[i]) \mid \mathsf{HW}(Z[i]) = y\right)$. The average mean and standard deviation of the functions $g_{\mathrm{ID},i}$ are -0.0301 and 0.0051 respectively. They are -$0,0291$ and $0,0092$ for the functions $g_{\mathrm{HW},i}$. While the left-hand figure shows that the distributions of values differ from normal distributions, the right-hand figure exhibits a strong dependency between them and the distribution of the Hamming weight values of $Z[i]$. This shows that our implementation is a good target for our attack which requires that the deterministic part of the leakage monotonously depends on the Hamming weight of the manipulated data. Eventually, we plot in the bottom-left figure an estimate (with kernel methods) of the distribution of the values $\mathbb{E}\left(\mathcal{L}(Z)\right) \mid \mathsf{HW}(Z) = y)$ when y ranges in $[0..128]$ and $\mathcal{L}(Z) \doteq \sum_{i=1}^{16} \mathcal{L}(Z[i])$. Once again, the distribution is a not a perfect binomial one, but the figure shows that the deterministic part of the leakage monotonously depends on the Hamming weight of the manipulated data. The mean and the standard deviation of the plotted distribution are -0.1781 and $0,2392$ respectively. For completeness, we also plot in the bottom-right of Figure 1 the distribution of the leakage values (after combining the 16 point of interest): the distribution looks very close to a Gaussian one, with mean -$0,4823$ and standard deviation 0.0629.

6.2 Experiments on Filtering

In the second step of the attack, we filtered the observations as described in Section 3.1.

ATMega328p (128-bit). In this context, the leakage $\mathcal{L}(Z)$ is built by summing the sixteen leakages $\mathcal{L}(Z[i])$, with $i \in [1..16]$. Theoretically, summing the sixteen intermediate Hamming weight values gives us exactly the Hamming weight value of the multiplication result. And summing the sixteen noise of standard deviation σ_8 results in a Gaussian noise of standard deviation $\sigma_{128} = 4 \cdot \sigma_8$. In practice, we get an SNR of 8.21 on the 128-bit simulated leakage.

In Table 4 we provide the experimental bounds λ_{exp} and error probabilities p_{exp} corresponding to a few levels of filtering. We also indicate the theoretical estimates

was however to test the practical soundness of our theoretical analyses; we hence chose to artificially build a 128-bit leakage. The application of our attack against 8-bit chunks is the purpose of Section 6.4 where it is shown that this situation is much more favourable to the attacker.

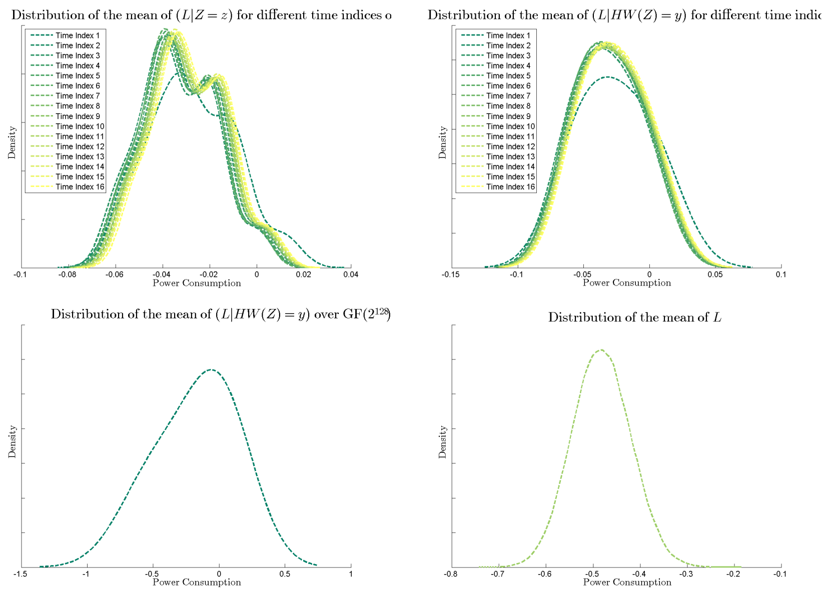

Fig. 1: Behaviour of the leakage w.r.t. the manipulated data Z

λ_{the} and p_{the} obtained by applying formulas (3.2) and (3.2) to the template we obtained using the same set of traces. As it can be observed, the theoretical estimates are very close to the ones obtained experimentally (which validates our theoretical analysis, even for non Hamming weight model).

Table 4: Experimental and theoretical parameters corresponding to filtering proportion $F(\lambda)$ on the ATmega for 128-bit AES-GCM

SNR $= 8.21$, $\sigma = 0.0206$							
$\log_2(1/F(\lambda))$	14	12	10	8	6	4	2
λ_{exp}	4.37	3.96	3.49	3.05	2.54	1.97	1.22
p_{exp}	0.383	0.386	0.393	0.407	0.420	0.434	0.452
λ_{the}	4.27	3.90	3.51	3.08	2.59	2.00	1.24
p_{the}	0.381	0.390	0.399	0.409	0.421	0.435	0.453

Remark 1. It must be noticed that a SNR equal to 8.21 in our experiments (with a noise standard deviation 0.0206) corresponds to a noise with standard deviation $\sigma = \sqrt{32/8.21} = 1.97$ in the theoretical Hamming weight model over 128-bit data.

Virtex 5 (128-bit). We additionally performed filtering on the traces from [BFG14] obtained from an FPGA implementation of GCM. Hereafter we provide theoretical (p_{the}) and experimental (p_{exp}) error probabilities for different avlues of the filtering parameter λ (Table 5).

Table 5: Error probabilities obtained from real traces.

λ	0.906	1.270	1.645	2.022	2.409	2.794	3.165	3.847
p_{the}	0.442	0.431	0.419	0.407	0.395	0.382	0.369	0.357
p_{exp}	0.441	0.430	0.418	0.405	0.392	0.379	0.370	0.361

It must be noticed that experimental results correspond to expectations. The largest deviation (for $\lambda = 3.847$) is due to the fact that only 20 traces were kept.

Remark 2. It must be noticed that, surprisingly, we also obtained an SNR equal to 8.21 in FPGA experiments but corresponding to a noise standard deviation of 7.11.

ATMega328p (96-bit). As in the 128-bit case, the 96-bit leakage is simulated by summing the twelve intermediate 8-bit leakage of the multiplication result. Table 6 gives the bounds q and the error probabilities p corresponding to some levels of filtering.

Table 6: Experimental and theoretical parameters corresponding to filtering proportion $F(\lambda)$ on the ATmega for 96-bit AES-GCM

SNR $= 8.7073$, $\sigma = 0.0173$						
$\log_2(1/F(\lambda))$	12	10	8	6	4	2
λ_{exp}	4.27	3.80	3.29	2.76	2.14	1.31
p_{exp}	0.377	0.387	0.402	0.414	0.429	0.449

Remark 3. A SNR equal to 8.7073 in our experiments (with a noise standard deviation 0.0173) corresponds to a noise with standard deviation $\sqrt{24/8.7073} = 1.66$ in the theoretical Hamming weight model over 96-bit data.

6.3 LPN Experiments

Attack on Simulated Traces (96-bit). We successfully performed our new attack on AES-GCM for a block-size reduced to 96 bits. We generated a secret key $\mathbf{k}$ of 96 bits, then generated 2^{20} uniform randomly generated $\mathbf{a}_i$. We simulate a leakage corresponding to the one obtained on the ATMega328p (*i.e.,* with the same statistics) and chose λ equal to 3.80 (thus filtering with probability 2^{-10} and having an error probability of 0.387).

This amounted to keeping 916 relations, the less noisy one having weight 25 that is an error rate of 0.260. We used this relation for secret/error permutation. All in all, we got $87840 \approx 2^{16,42}$ LPN equations.

After 6 hours of parallelized generalized birthday computation on a 32-core machine using 200 gigabytes of RAM, we had found more than 2^{39} elements reduced down to 36 bits. After a Walsh transform on the 36 bits (approximatively 2000 seconds on the same machine), we recovered the 36 least significant bits of the error that we convert in 36 bits of the secret. This heavy computation corresponds to the most complex part of the attack and validates its success. We can afterwards recover the remaining bits by iterating the attack with the knowledge of the (already) recovered bits. This is a matter of minutes since it corresponds to an attack on a 60-bit key which is much less expensive than the 96-bit case.

Expected Attack Complexities (128-bit). We provide here theoretical complexities for the key-recovery attack on 128-bit secret key. Experiments have been performed on 96-bit secrets and presented in the previous paragraph which confirm the accuracy of our theoretical estimates.

We can see on Figure 2 the evolution of time complexity as a function of the available memory for the attack. Plots are provided for three different data complexities.

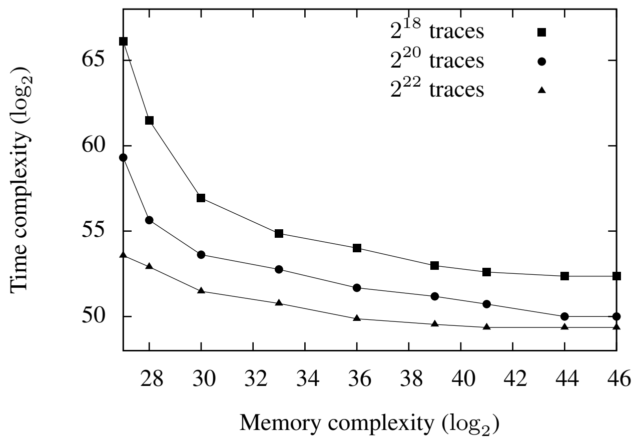

Fig. 2: Estimated complexities of the 128-bit attack ($\mathsf{SNR} = 8.21$).

We notice that the time/memory trade-off is only exploitable up to one point. This is due to the fact that when lots of memory is available, one may perform a larger Walsh-Hadamard transform to obtain more reduced equations. At some point, the time complexity of this transform will be predominant compared to the birthday paradox step and thus there will be no gain in increasing the Walsh size.

Here are few examples of time/memory trade-offs obtained for 2^{20} side-channel acquisitions.

- Time complexity: $2^{59.31}$ and Memory Complexity: $2^{27.00}$
- Time complexity: $2^{51.68}$ and Memory Complexity: $2^{36.00}$
- Time complexity: $2^{50.00}$ and Memory Complexity: $2^{44.00}$

6.4 Attack on Fresh Re-keying

We detail here the attack that aims at recovering the master key from the leakages corresponding to the first round of the AES when the secret key is generated by the fresh re-keying primitive described in Section 5. We present the known-input version of the attack, the chosen-input attack is described in Appendix B

Leakage Acquisition. We randomly generated 15,000 vectors $\mathbf{a} \in \mathsf{GF}(2)^{128}$ and 15,000 vectors $\mathbf{b} \in \mathsf{GF}(2)^8$. We then measured the 8-bit leakage during the processing of $\mathsf{Sbox}(\mathbf{z}_0 \oplus \mathbf{b})$ with $\mathbf{z}_0$ the first byte of the multiplication between $\mathbf{a}$ and $\mathbf{k}$.

Filtering. We filtered the extreme consumption measurements in order to exhibit the extreme Hamming weight values. Table 7 gives the empirical error probabilities according to the proportion of filtering on the 15,000 observations. As explained in Section 5,

Table 7: Error probability p according to the proportion of filtered acquisitions $F(\lambda)$ on the ATMega328p for the fresh re-keying with known inputs

$\log_2(1/F(\lambda))$	9	8	7	6	5	4	3	2	1
$\mathsf{SNR} = 8.6921$, $\sigma = 0.0165$									
λ	0.555	0.514	0.473	0.432	0.391	0.349	0.288	0.226	0.123
p	0.0	0.013	0.056	0.089	0.11	0.15	0.18	0.22	0.29

the error probabilities are naturally much lower than for a 128-bit leakage.

Key Recovery. With a sufficient (but still reasonable) filtering, we can directly recover the key by inverting the linear system of equations. For instance, in our experiments, filtering one observation over 2^9 gives $33 \times 8 = 264$ linear equations on the bits of $\mathbf{k}$ without a single error. Thus, inverting the system directly gives us the correct key.

References

ACPS09. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, August 2009.

AG11. Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415. Springer, July 2011.

BCO04. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, August 2004.

BFG14. Sonia Belaïd, Pierre-Alain Fouque, and Benoît Gérard. Side-channel analysis of multiplications in GF(2128) - application to AES-GCM. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 306–325. Springer, December 2014.

BKW00. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, pages 435–440. ACM Press, May 2000.

BTV15. Sonia Bogos, Florian Tramer, and Serge Vaudenay. On solving LPN using BKW and variants. Cryptology ePrint Archive, Report 2015/049, 2015. `http://eprint.iacr.org/2015/049`.

CGP^{+}12. Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for S-boxes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 366–384. Springer, March 2012.

CJRT05. Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors. *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*. Springer, 2005.

DDP13. Guillaume Dabosville, Julien Doget, and Emmanuel Prouff. A new second-order side channel attack based on linear regression. *IEEE Trans. Computers*, 62(8):1629–1640, 2013.

GJJR11. Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. Workshop NIAT 2011, 2011.

GJL14. Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 1–20. Springer, December 2014.

HGJ10. Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 235–256. Springer, May 2010.

ISW03. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, August 2003.

Kir11. Paul Kirchner. Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377, 2011. `http://eprint.iacr.org/2011/377`.

LF06. 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.

Lyu05. Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In Chekuri et al. [CJRT05], pages 378–389.

MSGR10. 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.

MSJ12. Marcel Medwed, François-Xavier Standaert, and Antoine Joux. Towards super-exponential side-channel security with efficient leakage-resilient PRFs. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 193–212. Springer, September 2012.

OC14. Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. Cryptology ePrint Archive, Report 2014/204, 2014. `http://eprint.iacr.org/`.

Pie12. Krzysztof Pietrzak. Cryptography from learning parity with noise. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer Berlin Heidelberg, 2012.

RKSF11. Mathieu Renauld, Dina Kamel, François-Xavier Standaert, and Denis Flandre. Information theoretic and security analysis of a 65-nanometer DDSLL AES S-box. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 223–239. Springer, September / October 2011.

SLP05. Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 30–46. Springer, August / September 2005.

SS79. Richard Schroeppel and Adi Shamir. A T s^2 = o(2^n) time/space tradeoff for certain np-complete problems. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 328–336. IEEE Computer Society, 1979.

Wag02. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, August 2002.

A Efficient Algorithms with Limited Number of Samples

A.1 Improved Algorithm

In this section, we describe the algorithm that we have implemented to solve the instance of the LPN$(n, n \cdot t, p)$ problem described in Section 3.1. It combines generalized birthday and improvements.

On a practical side, guessing intermediary bits of the secret by making several Walsh-Hadamard transforms (instead of a bigger one) is more efficiently parallelisable and requires much less memory. Whilst this is not an algorithmic improvement *per se*, we leverage on the secret/error switch in order to reduce the expected number of guesses. This leads to a significant improvement of the time complexity.

For instance, assume that the less noisy trace is affected by an error rate equal to $1/4$. Thus, we can use this less noisy trace to switch the secret and the error and obtain a much sparser secret. This characteristic can then be used after the Shamir-Schroeppel or BKW method to perform several Walsh-Hadamard transforms. Concretely, we can fix the size of Walsh-Hadamard transform to 36 bits and we can assume that the sparse secret contains 16 bits set to 1 in its last 64 columns. At this point, we can permute the last columns until all the bits set to one now appear in the last 36 columns. On average, we need $\binom{64}{16}/\binom{36}{16} \approx 2^{16}$ tries which is significantly better than the previous guessing of $64 - 36 = 28$ bits of the secret.

Algorithm 1 describes the attack that we have implemented and which is based on these two improvements.

Algorithm 1: Solving LPN using the General Birthday Paradox.

```
Input: A set $\mathcal{E}$ of filtered equations.
Output: A $\omega$-bit guess for $\omega$ bits of the secret.
Parameters: $\alpha, \gamma, \omega$ and a set $\mathcal{M}$ of $m$-bit values
1  $(\boldsymbol{A}, \mathcal{E}) \leftarrow \mathrm{SecretErrorSwitch}(\mathcal{E})$;
2  $\mathcal{E}' \leftarrow \mathrm{Expand}_\alpha(\mathcal{E})$;
3  $(L_0, L_1, L_2, L_3) \leftarrow \mathrm{Split}(\mathcal{E}')$;
4  $\mathcal{R} \leftarrow \emptyset$;
5  foreach $M \in \mathcal{M}$ do
6      $L_{01} \leftarrow \mathrm{Merge}(L_0, L_1, M)$;
7      $L_{23} \leftarrow \mathrm{Merge}(L_2, L_3, M)$;
8      $L_{0123} \leftarrow \mathrm{Filter}_\gamma(L_{01}, L_{23})$;
9      $\mathcal{R} \leftarrow \mathcal{R} \cup L_{0123}$;
10 $(guess, score) \leftarrow (0, 0)$;
11 foreach $g_{Hi} \in \{0,1\}^{n-\alpha-m-2-\gamma-\omega}$ do
12     $(argmax, max) = \mathrm{WTransform}_\omega(\mathcal{R}, g_{Hi})$ ;
13     if $max > score$ then
14         $(guess, score) \leftarrow (g_{Hi} || argmax, max)$;
15 Iterate(guess);
16 return SecretFromError($\boldsymbol{A}$,guess);
```

- SecretErrorSwitch(). As in [Kir11,ACPS09], we exchange the secret and the first bits of the error as follows. We first write the first n equations as $\boldsymbol{A}\mathbf{k} \oplus \boldsymbol{e_0} = \boldsymbol{b}$ where $\boldsymbol{A}$ is the (n, n) bit matrix of the n equations which have the least error probability. Since it is a multiplication matrix in a finite field, we know that $\boldsymbol{A}$ is invertible. Then in the other equations we replace $\mathbf{k}$ by $\boldsymbol{A}^{-1}\boldsymbol{e_0} \oplus \boldsymbol{A}^{-1}\boldsymbol{b}$. We thus obtain new LPN equations which aim to recover the (new) sparse secret $\boldsymbol{e_0}$. Eventually, we remove the first n equations which cannot be used for LPN resolution.

- Expand$_\alpha$(). Let ν be the number of equations in the input set $\mathcal{E}$ (that is the actual number of queries provided as an input of the LPN instance). The expansion step consists in combining all the pairs of equations by XORing them. The parameter α allows a time-memory trade-off as follows. It corresponds to an additional filtering of equations, namely, we will only consider XORed equations whose α leading bits are equal to 0:

$$\mathcal{E}' = \mathrm{Expand}_\alpha(\mathcal{E}) \triangleq \{e_1 \oplus e_2, e_1, e_2 \in \mathcal{E}, e_1 \neq e_2 \text{ and } (e_1 \oplus e_2) = \underbrace{0\ldots0}_{\alpha}||\ldots\}.$$

After this step, we obtain a set $\mathcal{E}'$ of $\nu^2\, 2^{-\alpha-1}$ equations each with α bits to zero[10]. The time complexity is equal to $\nu^2/2$ while the memory complexity is equal to $\nu + \nu^2\, 2^{-\alpha-1}$.

[10] Note that the equations are not independent any more.

- Split(). This simple procedure splits the obtained expanded set of equations $\mathcal{E}'$ into four smaller lists L_i of equal size $\ell \triangleq \nu^2\, 2^{-\alpha-3}$ (up to one element). By inserting only elements having MSB 00 in L_0 and ones having 01 in L_1, every element in L_{01} will begin with 01. Also, by inserting elements having MSB 10 and 11 in the lists L_2 and L_3 respectively, elements in L_{23} will also begin with 01. Thus we will have 2 bits "for free", which corresponds to the logarithm of the number of initial lists. The time complexity of this step is 4ℓ and the memory complexity is 4ℓ.

- Merge(). For a given m-bit value M, the merging step combines equations from two lists L_i and L_j as follows:

$$L_{ij} \triangleq \left\{ e_i \oplus e_j, e_i \in L_i, e_j \in L_j, (e_i \oplus e_j) = \underbrace{0\dots0}_{\alpha} ||01|| \underbrace{M}_{m} || \underbrace{\dots}_{n-\alpha-2-m} \right\}.$$

We end up with two lists of average size $\ell' \triangleq l^2/2^m = \nu^4\, 2^{-2\alpha-m-6}$. Algorithmically, we compute $L'_0 = \{e_i \oplus \underbrace{0\dots0}_{\alpha+2} ||M|| \underbrace{0\dots0}_{n-\alpha-2-m} \; |e_i \in L_0\}$, then sort L'_0, then merge L'_0 with L_1 (which is already sorted). Same with L_2 and L_3. Shamir and Shroeppel [SS79] trick would be useful here, since it would provide us with L'_0 already sorted directly from L_0. But unlike integers addition, XORing data does not cope well with preserving order, so we need to sort anyway. Note however that we need only to sort data according to their $\alpha + m + 2$ first bits. The time complexity is $2 \times (\ell \log(\ell) + \max(2\ell, \ell'))$ and the memory complexity is $4\ell + 2\ell'$.

- Filter$_\gamma$(). The filtering step combines equations from two lists L_{01} and L_{23} by XORing them. Resulting equations will have their $\alpha + m + 2$ most significant bits to zero by construction of the lists. The parameter γ indicates the number of additional zeroes that we force that is we only keep in the output list L_{0123} equations having $\alpha + m + 2 + \gamma$ most significant bits to zero. On average, the resulting list is of size $2^{8q-4\alpha-2m-\gamma-12}$. The time complexity is $2\ell' \log(\ell') + \max(2\ell', 2^{8q-4\alpha-2m-\gamma-12})$ and the memory complexity is $4\ell + 2\ell' + \nu^8\, 2^{-4\alpha-2m-\gamma-12}$.

- WTransform$_\omega$(). At this step, we get equations reduced to $n - \alpha - 2 - m - \gamma$ bits and we try to recover that much lsb of the (secret) initial error using Walsh Transforms on ω bits[11]. The reduced equations obtained as an input of this procedure are biased. From this bias one may recover those bits of the secret. Assume first that each equation is reduced to exactly ω bits[12]. We first fill a table t of size 2^ω with 0, then for each reduced noisy equation $\langle \mathbf{a}'_i, \mathbf{e_0} \rangle + e'_i = b_i$, with $\mathbf{a}'_i$ having at most ω bits, we add $(-1)^{b_i}$ to $t[\mathbf{a}'_i]$. Then we apply a Walsh transform to t, whose peak gives the most probable ω lsb of the (secret) initial error.

When equations are reduced to more than ω bits (say ω'), then we have to guess the $\omega' - \omega$ additional bits (since we cannot afford a larger Walsh transform). While the guess g_{Hi} may take $2^{\omega'-\omega}$ different values, the initial secret/error exchange implies that these bits have the same sparsity as the initial error (which is lesser than 0.5). Thus, the expected number of trials for g_{Hi} before finding the correct one will be significantly less

[11] We should have $\omega \leq n - \alpha - 2 - m - \gamma$, otherwise it means that we have reduced equations too much.

[12] That is $\omega = n - \alpha - 2 - m - \gamma$.

than the aforementioned upper-bound. Indeed, rather than guessing bits of the secret at specific positions, we may randomly shuffling the ω' columns of the matrix of the $\mathbf{a}'_i$. This also shuffles the lsb of the secret. Then we suppose that the first $\omega' - \omega$ of the (secret) error are zeroes. This is equivalent as being able to truncate the $\mathbf{a}'_i$ to the ω remaining bits where the (secret) error is supposed to be non-zero. Due to the sparsity of the (secret) error, the expected number of trials before such a situation occurs will be significantly smaller than $2^{\omega'-\omega}$. More precisely, assuming we performed secret-error switching using n equations having error probability $p_{\min}$, then the expected number of trials is $\binom{\omega'}{p_{\min}\omega'}/\binom{\omega}{p_{\min}\omega'}$.
- Iterate(guess). At this point, we have recovered $n - \alpha - m - 2 - \gamma$ of the (permuted) initial error. We just add this information to the equations and restart the attack from the beginning to get the next chunk of bits of the initial error, except this time we reduce much less: we only stop ω bits before.

- SecretFromError(). Once we recovered the full (permuted) initial error, we recover the secret using the same error/secret permutation as above, using the matrix $\boldsymbol{A}$.

B Attack on the AES-GCM Multiplication's Output with Chosen Inputs

We have mounted an attack on the AES-GCM multiplication's result with chosen inputs as described in Section 4.

Leakage Acquisition. Instead of generating random vectors, we measured this time the leakage during the processing of $\mathbf{z} = \mathbf{a} \cdot \mathbf{k}$ for 129 vectors $\mathbf{a}$ corresponding to all the powers of two between 0 and 128. We simulated the leakage on 128 bits by summing the intermediate 8-bit leakage as for the attack on known inputs.

Comparison. In the case of chosen inputs, we do not need to filter. We compare two consecutive measurements and if the absolute value of their difference overpasses the optimal threshold, we assume that the targeted bit was set to one. Otherwise, we assume that it was zero. We thus obtain 128 equations of the 128 key bits with a certain error probability. We can directly use them or we can repeat the measurements on the same set of 129 vectors in order to reduce the level of the noise (i.e., repeating α times the same computation results in a standard deviation divided by $\sqrt{\alpha}$). Table 8 gives the empirical thresholds and error probabilities according to the number of repetitions of our set of 129 measurements.

Key Recovery. We can recover the key very efficiently under the context of the last column of Table 8, that is with 2^{18} measurements and more precisely 2^{11} repetitions of 129 traces. In this case, the error probability is zero which means that we directly get all the 128 key bits.

Table 8: Experimental parameters for the attack on 128-bit AES-GCM (chosen inputs).

number of repetitions α number of traces	1 129	2^5 $\approx 2^{12}$	2^{10} $\approx 2^{17}$	2^{11} $\approx 2^{18}$
$\mathsf{SNR} = 8.21$, $\sigma = 0.0206$				
τ	1.440	0.265	0.080	0.052
p	0.49	0.39	0.016	0.0