Enhancing Side-Channel Analysis of Binary-Field Multiplication with Bit Reliability

Peter Pessl, Stefan Mangard

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
peter.pessl@iaik.tugraz.at, stefan.mangard@iaik.tugraz.at

Abstract. At Africacrypt 2010, Medwed et al. presented Fresh Re-Keying as a countermeasure to protect low-cost devices against side-channel analysis. They propose to use binary-field multiplication as a re-keying function. In this paper, we present a new side-channel attack on this construction (and multiplication in general). By using template attacks and the simple algebraic structure of multiplication, the problem of key recovery can be casted to the well known Learning Parity with Noise problem (LPN). However, instead of using standard LPN solving algorithms, we present a method which makes extensive use of bit reliabilities derived from side-channel information. It allows us to decrease the attack runtime in cases with low-to-medium error probabilities. In a practical experiment, we can successfully attack a protected 8-bit Fresh Re-Keying implementation by Medwed et al. using only 512 traces.
Keywords: Side-Channel Analysis, Multiplication, LPN, Linear Decoding

1 Introduction

Binary-field multiplication, while not cryptographically strong in itself, offers nice properties for the design of cryptographic systems. For instance, it provides good diffusion and is, due to its linearity, very easy to mask and thus to protect against side-channel analysis (SCA) attacks.

These properties led Medwed et al. [16] to use it as a re-keying function in their *Fresh Re-Keying* scheme. The basic idea of Fresh Re-Keying is to combine an encryption function f, such as the AES, with a re-keying function g - namely said multiplication. For every invocation of f, first a *fresh* session key is derived by using the re-keying function g with a master key and a public random nonce.

Since no such session key is used twice in the encryption function f, Medwed et al. argue that it suffices to protect it against Simple Power Analysis (SPA) type attacks. However, the definition of *SPA security* is relatively loose. Usually it implies that a secret, in this case the session key, cannot be recovered when using a single trace. Yet, due to the simple algebraic structure of the re-keying function, very limited information on this secret might be sufficient for master-key recovery.

In fact, leakage of a single session-key bit over multiple invocations allows to trivially reveal the master key. The remaining security of each session key however is still 127 bit, which can still be considered SPA secure in the classic definition. Thus, the required side-channel resistance of f still guaranteeing security of the master key is unclear.

The Attack of Belaïd et al. The above problem relates to the work of Belaïd et al. [1,2], where they present a side-channel attack on binary-field multiplications. Their attack applies to settings where a constant secret, i.e., a key, is multiplied with several known values. This is the case with Fresh Re-Keying and the AES-GCM, an authenticated encryption mode.

Belaïd et al. observe that a binary-field multiplication can be written as a matrix-vector product, or system of linear equations, over bits. If one can recover the right-hand side of this system, e.g., by using side-channel analysis, then it can be trivially solved for the key. Belaïd et al. assume that the side-channel adversary is able to observe a noisy Hamming weight of the n-bit multiplication result, but does not have access to leakage of intermediate or partial results. When observing a low or high Hamming weight, i.e., smaller or greater than $n/2$, they assume that all bits of the multiplication result are 0 or 1, respectively. The introduced errors in the system of linear equations do not allow solving by Gaussian elimination.

The problem of solving such erroneous systems is known as *Learning Parity with Noise* (LPN). The algorithms used to solve this problem tend to require a high amount of samples, which is a scarce resource in the side-channel context. In order to decrease this quantity, Belaïd et al. first discard observations with Hamming weight near $n/2$ to decrease the error probability. Then, they use a new LPN algorithm based on LF2 by Levieil and Fouque [13] to recover the key.

Our Contribution. In this work, we present a new side-channel attack on Fresh Re-Keying. Similarly though, our attack also applies to other scenarios using binary-field multiplication, such as AES-GCM.

Our attack makes use of side-channel templates, i.e., device profiling. These templates are used to derive reliability information on each of the session-key bits. We then present a new algorithm aimed at recovering the key when given an erroneous system of equations. This algorithm makes extensive use of the fact that, unlike in standard LPN, we possess said reliability information.

Our analysis suggests that, when compared the previous work, the presented attack can decrease runtime in practical settings. Namely, it performs well if the adversary is able to gather enough LPN samples exhibiting a low-to-medium error probability (e.g., up to 0.2). This makes it particularly well suited for adversaries having access to leakage of partial multiplication results, such as individual bytes.

We use our algorithm to mount an attack on an 8-bit software implementation of Fresh Re-Keying presented by Medwed et al. [15]. Their implementation uses shuffling as means to protect the AES against SPA and algebraic side-channel attacks. We use templates to circumvent this countermeasure without making any major assumptions on the implemented shuffling algorithm. We can

successfully attack this implementation while only requiring a very small number of traces.

Outline. In Section 2, we describe Fresh Re-Keying and side-channel template attacks. Then, in Section 3 we define the Learning Parity with Noise problem and discuss algorithms aimed at solving this problem. After having introduced the groundwork, we give an attack outline and discuss the first steps in Section 4. In Section 5, we present our attack algorithm. Finally, in Section 6 we show an analysis of the attack performance by presenting outcomes of real and simulated experiments.

Notation. We now introduce some notation that is used throughout this paper. We denote bit vectors of length n as $\mathrm{GF}(2^n)$, $\langle \cdot, \cdot \rangle$ denotes the binary inner product of the two such vectors.

We denote the probability of an event e as $P(e)$. For a random variable X, we use $\mathbb{E}(X)$ to denote its mean. In this paper, we use side-channel information to derive the probability that a bit b is set to 1. We write $p_b = P(b = 1)$. We use τ_b as the respective bias, i.e., $\tau_b = |p_b - 1/2|$. When performing a classification, we set $b = \lfloor p_b \rceil$, with $\lfloor \cdot \rceil$ the rounding operator. This classification has an error probability $\epsilon_i = 1/2 - \tau_i$.

The above is exactly the Bernoulli distribution with parameter p_b, we denote it as $\mathrm{Ber}(p_b)$. We also make use of the so-called Poisson binomial distribution. This distribution describes the sum of N independent Bernoulli trials, where each trial has a possibly different Bernoulli parameter p_k. Given the vector $(p_1, \ldots, p_N)$, the respective density function can be computed by using the closed-form expression by Fernandez and Williams [10].

2 Fresh Re-Keying and Template Attacks

We now describe Fresh Re-Keying in more detail. Additionally, we recall side-channel template attacks. They will later allow us to derive probabilities for bits of the session key.

2.1 Fresh Re-Keying

Medwed et al. [16] introduced Fresh Re-Keying as a method to protect low-cost devices, such as RFID tags, against side-channel and fault attacks. The idea is to combine an encryption function f, e.g., the AES, with a re-keying function g. Every plaintext is encrypted using a fresh session key k^*. This k^* is obtained by invoking the re-keying function g with a fixed master key k and an on-tag generated public nonce r. This basic principle is shown in Figure 1. Medwed et al. claim that this approach is particularly suited for challenge-response authentication protocols.

Medwed et al. claim that, as session keys k^* are never reused, it suffices to protect the encryption function f against Simple Power Analysis (SPA) type attacks. The re-keying function g still requires protection against more powerful

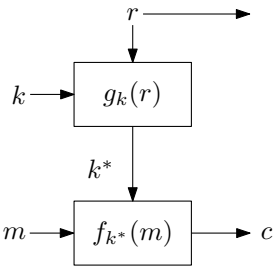


Fig. 1. Schematic of Fresh Re-Keying

DPA-like attacks, but it does not need to be a cryptographically strong function. Instead, it should provide good diffusion and it should be easy to protect against DPA. They propose to use the following modular polynomial multiplication over $\mathrm{GF}(2^8)$:

$$g : (\mathrm{GF}(2^8)[y]/p(y))^2 \to \mathrm{GF}(2^8)[y]/p(y) : (k, r) \to k * r \tag{1}$$

The polynomial $p(y) = y^d+1$, with $d \in \{4, 8, 16\}$. We solely use $p(y) = y^{16}+1$, as it is the most difficult to attack. For multiplication in $\mathrm{GF}(2^8)$, we use the AES polynomial, i.e., $\mathrm{GF}(2^8) = \mathrm{GF}(2)[x]/(x^8 + x^4 + x^3 + x + 1)$.

The function g can be written as a matrix-vector product over $\mathrm{GF}(2^8)$. With r_i and k_i $(0 \leq i < 16)$ the bytes of the nonce and master key, respectively, the bytes of the session key k^* can be computed according to (2). Analogously, multiplications in $\mathrm{GF}(2^8)$ can be written as matrix-vector products over $\mathrm{GF}(2)$. Thus, g can be restated as a linear system over bits.

$$\begin{pmatrix} r_0 & r_{15} & r_{14} & \cdots & r_1 \\ r_1 & r_0 & r_{15} & \cdots & r_2 \\ r_2 & r_1 & r_0 & \cdots & r_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{15} & r_{14} & r_{13} & \cdots & r_0 \end{pmatrix} \cdot \begin{pmatrix} k_0 \\ k_1 \\ k_2 \\ \vdots \\ k_{15} \end{pmatrix} = \begin{pmatrix} k_0^* \\ k_1^* \\ k_2^* \\ \vdots \\ k_{15}^* \end{pmatrix} \tag{2}$$

While this simple and regular structure makes implementation and masking easy, it is a potential risk when considering algebraic side-channel analysis. In fact, Medwed et al. were well aware of this threat, and they claim that cheap countermeasures are sufficient for protection. Concretely, they present a protected implementation of fresh re-keying running on an 8-bit microcontroller [15]. The re-keying function g is protected by means of masking and shuffling, whereas the encryption function f uses just the latter.

Fresh Re-Keying and birthday-bound security. Dobraunig et al. [8] showed that the Fresh Re-Keying scheme by Medwed et al. offers only birthday-bound security. They present a chosen-plaintext key-recovery attack having a time complexity of only $2 \cdot 2^{n/2}$ (instead of 2^n) for an n-bit key. However, their attack requires to pre-compute and store $2^{n/2}$ (key, ciphertext) pairs. Additionally, the

attacked device, e.g, a low-cost tag featuring high execution times, needs to be queried $2^{n/2}$ times. These drawbacks make the attack impractical if, like in this paper, 128-bit AES is used. However, for weaker primitives, such as 80-bit PRESENT, the attack might be feasible.

In a follow-up work, Dobraunig et al. [9] proposed ways to provide higher security levels with Fresh Re-Keying. Yet, when using the same re-keying function g our attack still works. Thus, we omit the details of their work and focus on the original construction of Medwed et al.

2.2 Template Attacks and Leakage Model

Throughout this paper, we make extensive use of side-channel template attacks [6]. Instead of assuming a predetermined power model, such as Hamming-weight leakage, these attacks first profile the side-channel information of the attacked device. This requires possession of an identical device which is used for this profiling and whose key is already known.

When using an 8-bit device running the AES, an exemplary template attacks work as follows. One first profiles the side-channel information (on the profiling device) for each of the 256 possible inputs of the S-box. Then, templates typically following a multivariate Gaussian distribution are built. In the attack phase, each template is matched with the attack trace l coming from the attacked device. This results in a vector of conditional probabilities $p(s = v|l), 0 \leq v < 2^8$, with s the S-box input. Note that side-channel leakage can vary between otherwise identical devices. This might lead to inaccurate templates and conditional probabilities. We briefly discuss the impact on our attack later on.

For simulation of leakage, we use the common assumption that the device leaks a noisy Hamming weight of the processed data. The noise is assumed to be additive Gaussian with zero mean and variance σ_{N}^2. Thus, for a bit vector $\mathbf{z} \in \mathrm{GF}(2^n)$, we assume leakage $\mathcal{L}(\mathbf{z}) = \mathrm{HW}(\mathbf{z}) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma_{\mathrm{N}})$. A common metric for the quality of the traces is the signal-to-noise ratio $\mathrm{SNR} = \sigma_{\mathrm{S}}^2/\sigma_{\mathrm{N}}^2$ [14]. In the Hamming-weight leakage model, the signal variance $\sigma_{\mathrm{S}}^2 = n/4$.

3 LPN and Solving Algorithms

Side-channel attacks on multiplication relate to the well known LPN problem. In this section, we restate its definition and give a brief overview of algorithms aimed at solving this problem. We then give a more in-depth explanation of LPN algorithms originating from coding theory.

3.1 Learning Parity with Noise

We now recall the formal definition of LPN (or more correctly of LPN's search version).

Definition 1 (Learning Parity with Noise). *Let* $\mathbf{k} \in GF(2^n)$ *and* $\epsilon \in (0, 0.5)$ *be a constant noise rate. Then, given* ν *vectors* $\mathbf{a}_i \in GF(2^n)$ *and noisy observations* $b_i = \langle \mathbf{a}_i, \mathbf{k} \rangle + e_i$*, the* $\mathbf{a}_i$ *sampled uniformly, and the* e_i *sampled from* $Ber(\epsilon)$*, find* $\mathbf{k}$*.*

The first algorithm to solve this problem in sub-exponential time was presented by Blum, Kalai, and Wassermann (BKW) [4]. Their algorithm was later improved by, e.g., Levieil and Fouque [13] and by Guo et al. [12]. A major drawback of BKW and its variants is the high number of required LPN samples ν. Especially in a side-channel context, this resource is somewhat scarce. This is even more so the case for Fresh Re-Keying, as it is targeted at low-resource devices typically featuring high execution times.

Connection to random linear codes. The above LPN problem can be restated as decoding a random linear code over GF(2) [17]. Let $\mathbf{A} = [\mathbf{a}_i]_{0 \leq i < \nu}$ be the matrix whose rows are the $\mathbf{a}_i$. Further, let $\mathbf{b}$ and $\mathbf{e}$ be row vectors of the b_i and e_i, respectively, Then, one can think of $\mathbf{A}$ as generator matrix of a random linear code. Decoding requires to find the message $\mathbf{k}$ given a noisy word $\mathbf{b} = \mathbf{kA} + \mathbf{e}$, which is exactly search LPN.

Linear codes are characterized by the three main parameters $[n, k, d]$, with n the code length, k the code dimension, and d the minimum Hamming distance between any two valid codewords. In the case of LPN, the dimension k is equal to the size of the secret. For random linear codes, the obtained code rate $R = k/n$ is, with very high probability, close to the Gilbert-Varshamov bound [7]. That is, $R \approx 1 - \mathrm{H}(d/n)$, with H the binary entropy function. The code length n is chosen according to this bound, with $d/n \approx \epsilon$.

Decoding random linear codes is an NP-hard problem, and decoding algorithms feature a runtime exponential in the code length. However, the sample requirements are much smaller when compared to BKW-style algorithms. We now give a brief introduction.

3.2 Algorithms for Decoding Random Linear Codes

The fastest algorithms for decoding random linear codes rely on *Information-Set Decoding* (ISD). First proposed by Prange in 1962 [18], these algorithms have quite a long history, with probably the most notable version being Stern's algorithm [19].

Syndrome decoding. Before discussing decoding algorithms in detail, we briefly describe syndrome decoding. For a $(k \times n)$ generator matrix $\mathbf{G}$ in standard form, i.e., $\mathbf{G} = (I_k|Q)$, the so-called parity-check matrix $\mathbf{H}$ is given as $\mathbf{H} = (-Q^T|I_{n-k})$, with Q a $(k \times (n-k))$ matrix. The set of valid codewords C forms the kernel of the check matrix, i.e., $\mathbf{Hc} = 0, \forall \mathbf{c} \in C$. For a noisy word $\mathbf{y} = \mathbf{c} + \mathbf{e}$, we have $\mathbf{Hy} = \mathbf{He} = \mathbf{s}$. $\mathbf{s}$ is called the syndrome, it only depends on the error $\mathbf{e}$.

For decoding, we now want to find an error term $\mathbf{e}$ with some maximum weight w such that $\mathbf{He} = \mathbf{s}$. In other words, we are searching for at most w columns of $\mathbf{H}$ summing up to $\mathbf{s}$.

Stern's attack (and improvements). We now review Stern's algorithm. This algorithm takes as input a check matrix $\mathbf{H}$, a syndrome $\mathbf{s}$, and a maximum error weight w.

In a first step, Stern partitions the n columns of the parity-check matrix $\mathbf{H}$ into two distinct sets $\mathcal{I}, \mathcal{Q}$. $\mathcal{I}$ is made up of $(n-k)$ randomly selected columns which must form an invertible subset. $\mathcal{Q}$ is comprised of the remaining k columns. For simplicity, we assume that the columns of the check matrix are permuted such that $\mathbf{H}' = (\mathcal{Q}|\mathcal{I})$. Note that such a permutation also affects the syndrome and the position of error bits.

Next, he selects a size ℓ subset $\mathcal{Z}$ of $\mathcal{I}$, where ℓ is an algorithm parameter. $\mathcal{Q}$ is randomly split into two size $k/2$ subsets $\mathcal{X}, \mathcal{Y}$. The second part of $\mathbf{H}'$ is then transformed into identity form by applying elementary row operations. Stern then searches for (permuted) error terms $\mathbf{e}$ with a maximum weight w having exactly p nonzero bits in $\mathcal{X}$, p nonzero bits in $\mathcal{Y}$, no nonzero bits in $\mathcal{Z}$, and at most $w-2p$ in the remaining columns. This is visualized in (3). This search uses a collision technique. If it fails, then the algorithm is restarted by selecting new $\mathcal{Q}$ and $\mathcal{I}$. For more details on the algorithm we refer to [3].

$$\mathbf{H}' = (\mathcal{Q}|\mathcal{I}) = \left(\begin{array}{cccc|ccc|c} \overbrace{1\ 0\ 0\cdots}^{k/2\text{: } p \text{ err.}} & & & & \overbrace{\cdots 0\ 1\ 0}^{k/2\text{: } p \text{ err.}} & & & \overbrace{1}^{\ell\text{: 0 err.}} \\ 1\ 1\ 0\cdots & & & & \cdots 0\ 0\ 0 & & & \ \ 1 \\ 0\ 1\ 1\cdots & & & & \cdots 1\ 1\ 1 & & & \ \ \ \ 1 \\ \ \ \vdots & & & & \vdots & & & \ \ \ \ \ \ \ddots \\ 0\ 1\ 1\cdots & & & & \cdots 1\ 0\ 1 & & & \ \ \ \ \ \ \ \ 1 \end{array}\right) \tag{3}$$

Canteaut and Chabaud [5] proposed an improvement of this algorithm, which was later refined by Bernstein et al. [3]. Instead of choosing $\mathcal{Q}, \mathcal{I}$ randomly at each iteration and spending considerable time to transform $\mathbf{H}'$ to the desired form, one can use a simple column swapping. In each iteration, c elements of $\mathcal{Q}$ are exchanged with c from $\mathcal{I}$, where c is an algorithm parameter.

Stern and reliability. The possibility of enhancing the performance of Stern's algorithm by using reliability information was briefly mentioned by Valembois [20]. However, thus far it was not used in a cryptographic or side-channel context. Also, it lacks proper description and an in-depth runtime analysis.

4 Attack Outline and Setup

In this section, we give a brief outline of our attack. Then, we describe in more detail the attack setup, i.e., how we compute bit probabilities from side-channel information. This is done for both the 8-bit leakage case and for the 128-bit case.

4.1 Outline

Before diving into the details, we now give a brief outline of our attack on Fresh Re-Keying. Here we focus on attacking the 8-bit software implementation by Medwed et al. [15].

In the very first step, we perform a template attack on the multiplication output k^*. Considering that the re-keying function g is (supposedly) implemented in a DPA-secure fashion, e.g., features masking and shuffling, attacking the multiplication result k^* directly seems unnecessarily difficult. Instead, we make use of the fact that k^* is used as (session) key in the following invocation of the AES. Thus, we use templates on the first-round S-box.

We then use the outcome of the template matching to derive a probability p_b for each session-key bit b. These probabilities are then used as input to our attack algorithm, which starts off by performing a filtering on the samples. Only those bits with low error probability are kept, while all others are discarded. The remaining equations are finally fed to a decoding algorithm, which solves the LPN problem and thus, allows master-key recovery. This algorithm is tweaked to use the reliability of the samples.

Why Decoding? Decoding is not the fastest way of solving LPN. However, in contrast to BKW-style algorithms it has very low sample requirements. In conjunction with filtering, this allows us to keep only the few best samples. Thus, we can reach a low average error rate. With BKW on the other hand, the higher number of required samples results in an increased expected error rate.

4.2 Fresh Re-Keying and 8-bit Leakage

For the 8-bit implementation of Fresh Re-Keying, we perform a template attack on the first round of the AES. For each session key byte k_i^*, the vector of conditional probabilities $P(k_i^* = v|l)_{0 \leq v < 2^8}$ is converted to bit-wise probabilities $p_{i,j} = \sum_{v:v[j]=1} P(k_i^* = v|l)$, with $v[j]$ the j-th bit of v.

Circumventing the shuffling countermeasure. As pointed out in Section 2.1, Medwed et al. [15] propose to use shuffling as a simple protection mechanism against algebraic attacks.

We circumvent this countermeasure by using a particular chosen constant plaintext $m = (00)||(\text{FF})^{15}$, i.e., the first byte is set to 0 and all other bits are set to 1. Assuming a Hamming weight (or distance) leakage characteristic of the device, this particular choice maximizes the difference in power consumption during the initial AES key addition. By using templates on said key addition, it is possible to reveal the shuffled position of the 0 byte with probability significantly better than guessing.[1] In our attack, we use only the most likely position and thus can use 8 linear equations per trace. The bias of the corresponding S-box bits is multiplied with the probability of the classified shuffling position to obtain the final bias τ.[2]

The assumption of chosen plaintexts is reasonable in this context. The main proposed use case of Fresh Re-Keying is challenge-response authentication. In this setting, the attacked device chooses the nonce r, and the reader/attacker

[1] This method is only of limited use in a standard DPA, where the device uses a fixed key, as it strictly limits the number of observable plaintexts per key byte.

[2] This assumes that there is no reshuffling between key addition and S-box processing.

selects the challenge m. Observe that this attack technique does not make any assumptions on the implemented shuffling permutation-generation algorithm.

4.3 128-bit Leakage

In the setting considered by Belaïd et al. [1], the attacker does not have access to partial results and can only observe the Hamming weight of the multiplication output. In this case, deriving bit probabilities is trivial. For an observed leakage $\mathrm{HW}(\mathbf{z})$, with $\mathbf{z} \in \mathrm{GF}(2^n)$, we have for each of the n bits $p_{i,0 \leq i < n} = \mathrm{HW}(\mathbf{z})/n$.

5 Using Reliability to Increase Attack Performance

In this section, we explain our new attack algorithm and thus show how reliability information can be leveraged to reduce the computation time for the attack. First however, we introduce a new version of LPN which better describes the problem at hand.

5.1 LPVN: A new LPN Variant

In standard LPN (Definition 1), the error probability ϵ is constant for all samples. This, however, does not reflect the reality of the side-channel information, where every LPN sample can be assigned a possibly different error probability. We formalize this by introducing a new problem dubbed *Learning Parity with Variable Noise* (LPVN).

Definition 2 (Learning Parity with Variable Noise). *Let $\mathbf{k} \in GF(2^n)$ and ψ be a probability distribution over $[0, 0.5]$. Then, given ν vectors $\mathbf{a}_i \in GF(2^n)$, v error probabilities ϵ_i, and noisy observations $b_i = \langle \mathbf{a}_i, \mathbf{k} \rangle + e_i$, the $\mathbf{a}_i$ sampled uniformly, the ϵ_i sampled from ψ[3], and the e_i sampled from $Ber(\epsilon_i)$, find $\mathbf{k}$.*

Casting LPVN to LPN is possible by simply setting $\epsilon = \mathbb{E}(\epsilon_i)$. However, the additional information in form of the ϵ_i allows to design more efficient algorithms. Also, it is easy to see that with a non-zero meta-probability distribution ψ in close vicinity of 0, the problem becomes trivial given enough samples.

5.2 Filtering

In the context of side-channel analysis, the overall average error rate $\mathbb{E}(\epsilon_i)$ can be expected to be high, i.e., beyond 0.25. The resulting large code length n (cf. Section 3.1) might lead to excruciating decoding runtimes.

In order to cut this time down drastically, we perform a filtering of the samples. When given a certain number ν of LPVN samples, only the n with the lowest error probability are kept. All other samples are simply discarded. This

[3] This is not entirely correct for the attack of [1], in which each sampled error rate is applied to n samples instead of a single one. We neglect this minor difference.

approach differs from the filtering proposed by Belaïd et al. in that we can, at least in the 8-bit setting, filter individual bits. For 128-bit leakage however, the filtering methods are equivalent.

The number of available samples ν plays a crucial role in the expected attack runtime. By increasing ν, the quality of the best samples is also expected to rise. This in turn decreases the required code length n and the runtime of the decoding algorithm. Hence, a trade-off between the number of samples ν and computational complexity is possible. This is in stark contrast to standard LPN, where the decoding runtime is mostly independent of the number of samples.

Choosing the code length n. Thus far, we did not address the problem of selecting the code length n. In a heuristic approach, we choose the smallest n such that $R = k/n \geq 1 - H(d/n)$. We set d to the 75 % quantile of the error distribution function (computed using the Poisson binomial distribution) for the current n.

5.3 Using Reliability in Stern's Attack

After filtering, the n remaining samples are used as input for Stern's algorithm. More concretely, we use the improved version described by Bernstein et al. [3]. This algorithm does not directly cope with reliability information. Hence, by setting $b_i = \lfloor p_i \rceil$ a classification is performed.

Instead of discarding the reliability information at this point, we use it to further speed up the decoding process. Recall that the attack described in Section 3.2 involves a column-swapping step. We now tweak the algorithm by replacing the uniform selection of the swapped columns with a reliability-guided one. Goal is to minimize the expected error in $\mathcal{Q}$, while still assuring a high randomness in the chosen columns.

Column-swapping procedure. The probability that a column $t \in \mathcal{Q}$ is deselected in the next step is set to be directly proportional to its error probability ϵ_t, i.e., $P(t) = \epsilon_t / \sum_{t^* \in \mathcal{Q}} \epsilon_{t^*}$. Analogously, we use the *squared* bias to select the new column, i.e., for every $u \in \mathcal{I}$, $P(u) = \tau_u^2 / \sum_{u^* \in \mathcal{I}} \tau_{u^*}^2$. Experimentally we found that this combination gives the best performance.

We use rejection sampling in order to sample from these continuously changing probability density functions. Rejection sampling is a basic method to generate samples from a target probability distribution $f(x)$ when given samples from a different distribution $g(x)$. Concretely, we sample a $t \in \mathcal{Q}$ and a $u \in [0, \max_{0 \leq i < n}(\epsilon_i)]$ uniformly and accept t if $u < \epsilon_t$. Note that computation of the normalized probabilities $P(t)$ is not required for this method.

A detailed runtime analysis of this algorithm is given in Appendix A. The memory requirements of our decoding algorithm are negligible. They are limited to a single copy of the algorithm input per thread.

The impact of inaccurate templates. As mentioned in Section 2.2, the leakage characteristic of the profiling device can slightly differ from that of the

attacked device. This might lead to inaccurate templates and matching probabilities. As long as the profiling error is not too large, the attack will still work. However, the algorithm runtime and its analysis might suffer from inaccuracies.

6 Simulation and Practical Experiments

In order to show the real-world performance of our attack, we now present the outcome of our practical experiments. The focus is put upon the attack on 8-bit leakage, and more concretely on the software implementation of fresh re-keying proposed by Medwed et al. [15]. For completeness, the complexities for attacks on 128-bit leakage are also given.

6.1 Fresh Re-Keying on an 8-bit platform

We now report the outcome of both simulated and real attacks on the fresh re-keying implementation of [15]. We use the strategy described in Section 4.2, i.e., chosen plaintexts, to counter the proposed shuffling.

It is worth mentioning that both the simulated and real attack target only the block cipher f_{k^*}. Thus, it is independent of any countermeasures used to protect the re-keying function g. Also, we do not make any assumption on the generation of the permutation used for shuffling.

Simulation. For the simulated attack, traces according the Hamming-weight leakage model and some chosen SNR is generated. For each leaking S-box 3 samples (corresponding to the plaintext, the S-box input, and the S-box output) are generated. Due to the chosen plaintexts, the key is equivalent to the S-box input. Thus, it does not reveal any further information and was not included in simulation.

The $\mathrm{SNR}_{\mathrm{PT}}$ for plaintext leakage was chosen to be smaller than $\mathrm{SNR}_{\mathrm{SB}}$ used for S-box simulation. This was done in order to match the characteristic of the real device. Leakage and attack simulation was performed for two such SNR sets, namely for $(\mathrm{SNR}_{\mathrm{SB}} = 1, \mathrm{SNR}_{\mathrm{PT}} = 0.2)$ and for $(\mathrm{SNR}_{\mathrm{SB}} = 0.5, \mathrm{SNR}_{\mathrm{PT}} = 0.2)$. An estimation of the meta-probability distribution $\psi(\epsilon)$ is shown in Figure 2.

Figure 3 depicts the expected attack runtime as a function of the available traces. Solid lines denote performed experiments, whereas dots show estimates. We also compare the runtime of our tweaked decoding algorithm described with an untweaked version.[4] When using the parameters of Figure 3b and 2^{12} traces, the attack requires approximately 2^{50} bit operations. Our attack implementation required, on average, 4 hours to recover the key, using 6 out of 8 virtual cores on a recent Intel Core i7 CPU.

Real traces. We measured the power consumption of a shuffled AES software implementation running on an AVR ATxmega256A3. Using a separate set of

[4] Beware that due to the strong dependency on the quality of the samples and the exponential complexity, the runtime can still vary greatly for a certain trace count.

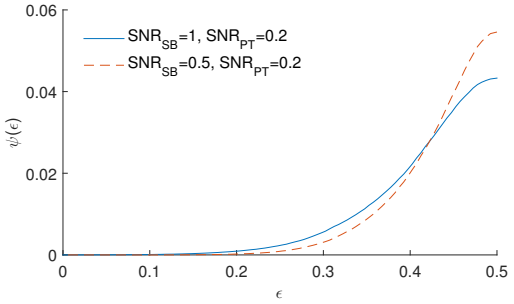
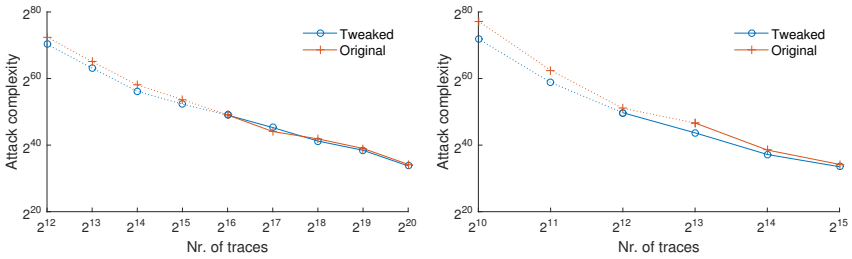


Fig. 2. Meta-probability ψ for simulated traces



(a) $\mathrm{SNR_{SB}} = 0.5, \mathrm{SNR_{PT}} = 0.2$ (b) $\mathrm{SNR_{SB}} = 1, \mathrm{SNR_{PT}} = 0.2$

Fig. 3. Runtime complexity of the attack

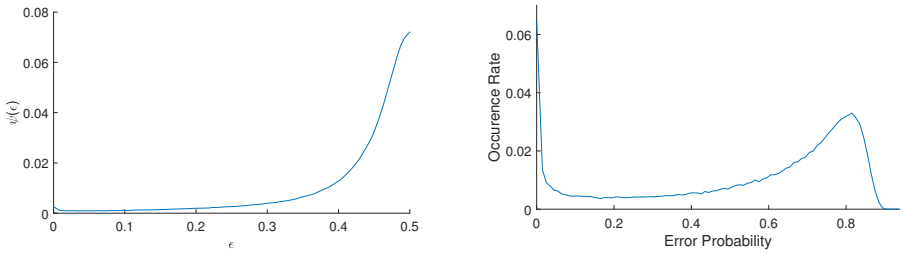
200.000 profiling traces, we built Gaussian templates for each of the 256 possible input values of the S-box and for the two chosen plaintext bytes.[5] The points of interest were chosen according to a student t-test, as proposed by Gierlichs et al. [11]. The attack was then performed on the same device.

Figure 4 depicts the outcome of the template attack. 4b shows the estimated distribution of the confidence in the attack on the shuffling. The peak at 0 shows that there is an acceptable number of traces with high confidence in the identified shuffling position. The overall success rate is roughly 44 %.

Figure 4a shows the resulting $\psi(\epsilon)$. As it turns out, the density near 0 is relatively high. With a reasonable amount of traces, one can expect the 128 best equations to be error free. Thus, the system can be solved by using straightforward Gaussian elimination.[6] Still, the number of traces can be further reduced by using our attack. As shown in Figure 5, 512 traces are sufficient to recover the master key in reasonable time.

[5] For the plaintext, we only consider leakage during the key addition. The initial operand fetching was ignored, as this can be implemented without leaking the shuffling position.

[6] In fact, Belaïd et al. [15] present an attack on an 8-bit implementation using this approach. However, they do not consider the shuffling countermeasure and use Hamming-weight filtering instead of S-box templates.



(a) $\psi(\epsilon)$ (b) Shuffling position

Fig. 4. Template attack on real traces

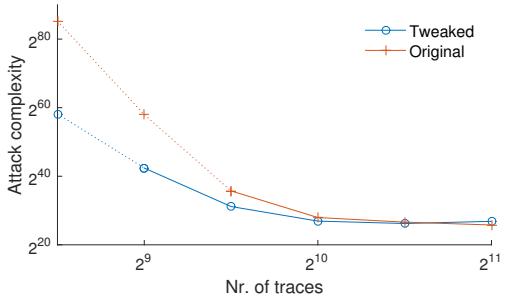


Fig. 5. Attack runtime for real traces

6.2 128-bit Leakage

For completeness, we now give some runtime estimates for the setting of Belaïd et al. In their first example, they recover a 96-bit secret with the lowest error probability being 0.26. These numbers translate to a complexity of approximately 2^{55} for our attack. When comparing it to the above stated runtimes, this is still well within the realms of feasibility. When attacking a 128-bit secret (with best error probability 0.28), the complexity rises to 2^{75}. Feasibility cannot be claimed anymore in this case.

6.3 Comparison of Algorithms

We now discuss the performance increase by using reliabilities in Stern's algorithm. As can be seen above, this speed-up varies greatly. In the attack on the real device it is substantial, whereas it becomes smaller in the simulations. All cases share the property that the speed-up is excepted to rise for rising attack complexity.

The most likely explanation lies in the distribution of the used ϵ_i. Simply speaking, a higher variance in these probabilities allows a more effective selection of the swapped columns. We can expect such a high variance if the number of filtered samples n gets close to the number of available samples ν. This is, e.g., the

case in the attack on the real device. If, on the other hand, the probabilities are all within a narrow region, then the tweaked algorithm is essentially equivalent to its base version.

Comparison to Belaïd et al. We did not implement the algorithm of Belaïd et al., thus making a detailed and fair comparison difficult. Nonetheless, we now try to provide a point of reference. Belaïd et al. report that the attack on a 96-bit secret took 6.5 hours on a 32-core machine with 200 GB RAM. When using these parameters in their runtime analysis, one gets a complexity of roughly 2^{44}.

The same complexity is achieved when using their runtime analysis with the parameters of Figure 3b and 2^{12} traces.[7] For this same attack, we require 4 hours for 2^{50} operations using 6 cores. Thus, the differing time constants in the exponential notation cannot be neglected. This example suggests that our new attack outperforms the algorithm by Belaïd et al. in this case. We performed further such evaluations. They suggest that our attack performs better for cases where the error rate of the filtered samples is low, e.g., up to 0.2, yet not low enough to allow a trivial solution. For high error rates, such as in the 128-bit leakage scenario, their algorithm performs clearly better.

7 Conclusion and Future Work

The results from the previous section clearly show that the simple structure of the re-keying function makes algebraic side-channel attacks a real threat. Also, it seems that shifting the task of DPA security to a dedicated re-keying function is not trivial. Leakage of its output must be considered in all subsequent operations and simple protection mechanisms, such as shuffling, might not be sufficient for protection.

There exist multiple thinkable ways of protecting Fresh Re-Keying against the presented attacks. An obvious one is to add further countermeasures to the AES, which however increases protection overhead. Alternatively, one could change the re-keying function g, e.g., to polynomial multiplication over a prime field instead of $\mathrm{GF}(2^8)$.

In future work, we intend to apply the idea of using reliability to BKW-style algorithms, such as LF1 [13]. Also, we would like to present an in-depth runtime comparison of solving algorithms.

Acknowledgements

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR).
Furthermore, this work has been supported by the Austrian Research Promotion Agency (FFG) under grant number 845589 (SCALAS).

[7] Note that we already used our S-box templates and bit-wise filtering for this estimation. When using the extreme Hamming weight method proposed in [1] (on 8-bit data), then the expected error and thus runtime increases.

We would also like to thank Benoît Gérard and Jean-Gabriel Kammerer for answering questions regarding their work and for providing the source code used for their runtime estimation.

References

1. S. Belaïd, J. Coron, P. Fouque, B. Gérard, J. Kammerer, and E. Prouff. Improved Side-Channel Analysis of Finite-Field Multiplication. *IACR Cryptology ePrint Archive*, 2015:542, 2015. note: to appear at CHES 2015.
2. S. Belaïd, P. Fouque, and B. Gérard. Side-Channel Analysis of Multiplications in GF(2^{128}) - Application to AES-GCM. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 306–325. Springer, 2014.
3. D. J. Bernstein, T. Lange, and C. Peters. Attacking and Defending the McEliece Cryptosystem. In J. A. Buchmann and J. Ding, editors, *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008*, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008.
4. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.
5. A. Canteaut and F. Chabaud. A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.
6. S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In B. S. K. Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
7. J. Coffey and R. Goodman. Any code of which we cannot think is good. *Information Theory, IEEE Transactions on*, 36(6):1453–1461, Nov 1990.
8. C. Dobraunig, M. Eichlseder, S. Mangard, and F. Mendel. On the Security of Fresh Re-keying to Counteract Side-Channel and Fault Attacks. In M. Joye and A. Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014*, volume 8968 of *Lecture Notes in Computer Science*, pages 233–244. Springer, 2014.
9. C. Dobraunig, F. Koeune, S. Mangard, F. Mendel, and F. Standaert. Towards Fresh and Hybrid Re-Keying Schemes with Beyond Birthday Security. In *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015*. Note: to appear.
10. M. Fernandez and S. Williams. Closed-Form Expression for the Poisson-Binomial Probability Density Function. *Aerospace and Electronic Systems, IEEE Transactions on*, 46(2):803–817, April 2010.
11. B. Gierlichs, K. Lemke-Rust, and C. Paar. Templates vs. Stochastic Methods. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.
12. Q. Guo, T. Johansson, and C. Löndahl. Solving LPN Using Covering Codes. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014*, volume 8873 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2014.

13. É. Levieil and P. Fouque. An Improved LPN Algorithm. In R. D. Prisco and M. Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2006.
14. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks – Revealing the Secrets of Smart Cards*. Springer, 2007. ISBN 978-0-387-30857-9.
15. M. Medwed, C. Petit, F. Regazzoni, M. Renauld, and F. Standaert. Fresh Rekeying II: Securing Multiple Parties against Side-Channel and Fault Attacks. In E. Prouff, editor, *Smart Card Research and Advanced Applications - 10th IFIP WG 8.8/11.2 International Conference, CARDIS 2011*, volume 7079 of *Lecture Notes in Computer Science*, pages 115–132. Springer, 2011.
16. M. Medwed, F. Standaert, J. Großschädl, and F. Regazzoni. Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices. In D. J. Bernstein and T. Lange, editors, *Progress in Cryptology - AFRICACRYPT 2010*, volume 6055 of *Lecture Notes in Computer Science*, pages 279–296. Springer, 2010.
17. K. Pietrzak. Cryptography from Learning Parity with Noise. In M. Bielikova, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turan, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer Berlin Heidelberg, 2012.
18. E. Prange. The use of information sets in decoding cyclic codes. *Information Theory, IRE Transactions on*, 8(5):5–9, September 1962.
19. J. Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.
20. A. Valembois. Fast soft-decision decoding of linear codes, stochastic resonance in algorithms. In *Information Theory, 2000. Proceedings. IEEE International Symposium on*, pages 91–, 2000.

A Runtime Analysis of the Tweaked Algorithm

In this Appendix, we present the method used for runtime estimation of our tweaked information-set decoding algorithm. This analysis follows along the lines of the work of Bernstein et al. [3].

The runtime of ISD algorithms is typically measured in the number of required bit operations. As the amount of bit operations per iteration of Stern's algorithm is essentially unchanged when using our tweak, we refer to [3] for the calculation of this quantity. The number of required iterations however is expected to decrease. We now detail the estimation of this quantity.

Probability that a column is part of $\mathcal{Q}$. In a first step, for each column t of the initial parity-check matrix we retrieve $P(t \in \mathcal{Q})$, i.e., the average probability of t being part of $\mathcal{Q}$ when using the replacement rules given in Section 5.3. This is done by using a method similar to a Markov-chain analysis.

We define $\mathbf{s} = (s_0, s_1, \ldots, s_{n-1})$ as the vector containing these probabilities (the state). This vector is initialized uniformly, i.e., $s_i = 128/n, 0 \leq i \leq n$ (assuming a 128-bit key). Then we construct a transition matrix $\mathbf{A}$ which depends on the current state. A column in this matrix corresponds to the event that the

respective column of the check matrix is drawn in the uniform-sampling step of the rejection-sampling procedure. The main diagonal of $\mathbf{A}$ contains the probability that this selection is rejected, which is 2τ in our case. The remaining entries contain the swapping probabilities, i.e., the probability that column $j \in \mathcal{Q}$ is exchanged with column $i \notin \mathcal{Q}$. Note that these depend on the current state, as columns already part of $\mathcal{Q}$ can not be swapped into it. Thus, we have

$$\mathbf{A}_{i,j,i\neq j} = (1-2\tau_j)\frac{(1-s_i)\tau_i^2}{\sum_{j^*\neq j}(1-s_j^*)\tau_{j^*}^2} \text{ and } \mathbf{A}_{i,i} = 2\tau_i$$

Finally, we update $\mathbf{s} = \mathbf{A}\mathbf{s}$. The recalculation of $\mathbf{A}$ and updating is repeated until the probability vector $\mathbf{s}$ reaches a steady state.

Error-count density function. In the subsequent step, we compute the probability density function for the error count for both $\mathcal{Q}$ and $\mathcal{I}$. For that we acquire the Poisson binomial PDF with Bernoulli probabilities $\epsilon_t \cdot P(t \in \mathcal{Q})$ and $\epsilon_t \cdot (1 - P(t \in \mathcal{Q}))$, respectively. We use the closed-form expression of the Poisson binomial PDF as provided by Fernandez and Williams [10].

The resulting quantities are then used to compute the conditional probabilities of selecting or deselecting an erroneous column into $\mathcal{Q}$, depending on the current number of errors in this set. This then directly gives us the probability to increase or decrease the number of errors in $\mathcal{Q}$ for each swapping step.

Final Markov-chain analysis. Finally, a Markov-chain analysis akin to [3] is used to estimate the number of expected iterations. Here we simply use our increase/decrease probabilities from above instead of the one given in [3].