An Analysis of the Learning Parity with Noise Assumption Against Fault Attacks

Francesco Berti and François-Xavier Standaert.

ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.

Abstract. We provide a first security evaluation of LPN-based implementations against fault attacks. Our main result is to show that such implementations inherently have good features to resist these attacks. First, some prominent fault models (e.g. where an adversary flips bits in an implementation) are ineffective against LPN. Second, attacks taking advantage of more advanced fault models (e.g. where an adversary sets bits in an implementation) require significantly more samples than against standard symmetric cryptographic primitives such as block ciphers. Furthermore, the sampling complexity of these attacks strongly suffers from inaccurate fault insertion. Combined with the previous observation that the inner products computed in LPN implementations have an interesting algebraic structure for side-channel resistance via masking, these results therefore suggest LPN-based primitives as interesting candidates for physically secure implementations.

1 Introduction

Fault attacks exploit the possibility to force erroneous computations in cryptographic implementations [22]. In the context of symmetric cryptography, such attacks usually give rise to extremely powerful key recoveries. For example, a couple of random faults on the AES bytes are enough to recover its master key [28]. The Learning Parity with Noise (LPN) problem is an emerging cryptographic assumption that has been used to design various primitives over the last years [27]. Typical examples of its applications include identification protocols and MACs [8, 10, 15, 20, 21, 23, 24], but also PRGs, one-way functions [5], secret and public key encryption schemes [11, 16]. However, despite their potential application for low-cost embedded devices, and to the best of our knowledge, the susceptibility of these primitives to fault attacks has not been studied yet.

In this paper, we propose a first investigation of the LPN assumption against fault attacks. In order to keep our conclusions general, we evaluate the resistance of two (serial and parallel) architectures for computing noisy inner products, that reasonably reflect the design principles of real-world implementations. We also study the impact of various types of faults against these architectures (i.e. bit flips vs. set bits, single vs. multiple, with varying accuracies).

Our main conclusion is that LPN-based primitives are surprisingly resistant against fault attacks by default. First, we can easily show that the most usual transient fault model (i.e. where we flip bits in an implementation) does not reveal more information than what would be obtained with standard (non physical) active attacks against LPN-based protocols. Second, even advanced fault models, such as when the adversary is able to set bits to a given value, require a substantial amount of fault to succeed. In the case of serial implementations, we show that attacks based on a maximum likelihood strategy can be mounted – yet succeed with significantly more samples than similar attacks against standard block ciphers such as the AES. Furthermore, these attacks strongly suffer from inaccurate faults. In the case of parallel implementations, the situation is even better (for the designer) as efficient attacks require multiple and accurate faults, and breaking these implementation essentially boils down to analyzing small LPN instances that require a large number of samples to be solved.

Since primitives based on LPN (and inner products) also have good properties for protection against side-channel attacks [1, 12, 13, 26], our results therefore open the way towards more concrete investigations of their implementations, with low-cost security guarantees against both side-channel and fault attacks.

Admittedly, our general investigation of LPN implementations comes at the cost of less specialized conclusions regarding applicability. Yet, and for example, the proposed attacks can target the HB family of protocols [8, 14, 15, 21, 23].

2 Background

Let $(\mathbb{Z}_2, \oplus, \cdot)$ be the field of order 2 and consider the vector space $\mathbb{Z}_2^n$. Let $\mathbf{k} = (k_1, ..., k_n) \in \mathbb{Z}_2^n$ be a secret vector and $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{Z}_2^n$ be a random vector. Let us denote by $\langle \mathbf{x} | \mathbf{k} \rangle$ the inner product of the vectors $\mathbf{x}$ and $\mathbf{k}$ in the vector space $\mathbb{Z}_2^n$, that is $\langle \mathbf{x} | \mathbf{k} \rangle = \bigoplus_{i=1}^{n} (x_i \cdot k_i)$. Let finally Ber_ϵ be the Bernoulli distribution with parameter ϵ (such that if $e \leftarrow \mathsf{Ber}_\epsilon$, then $\Pr[e = 1] = \epsilon$ and $\Pr[e = 0] = 1 - \epsilon$) We use the following definition of the LPN problem.

Definition 1 (LPN problem with parameter ϵ and dimension n). *Consider the distribution $\boldsymbol{D}_{\boldsymbol{k},\epsilon} := \{\boldsymbol{x} \leftarrow \mathbb{Z}_2^n, \nu \leftarrow \mathsf{Ber}(1, \epsilon) : (\boldsymbol{x}, y := \langle \boldsymbol{x} | \boldsymbol{k} \rangle \oplus \nu)\}$. Let $\mathcal{O}_{\boldsymbol{k},\epsilon}$ be an oracle outputting independent samples according to this distribution. The **LPN_ϵ^n problem** is to find the secret vector $\boldsymbol{k}$ having obtained samples from the oracle. The LPN_ϵ^n problem is said to be (q, t, m, θ)-hard to solve if for any algorithm A, the following inequality holds:*

$$\Pr[\boldsymbol{k} \leftarrow \mathbb{Z}_2^n : A^{\mathcal{O}_{\boldsymbol{k},\epsilon}}(1^n) = \boldsymbol{k}] \leq \theta,$$

and A runs in time $< t$, memory $< m$ and makes $< q$ queries to $\mathcal{O}_{k,\epsilon}$.

We introduce the additional notation $\mathcal{Q} = \{(\mathbf{x}_j, y_j)\}_{1 \leq j \leq q}$ to denote a set of q outputs of the LPN oracle $\mathcal{O}_{\mathbf{k},\epsilon}$. In general, the LPN problem is believed to be hard for adversaries interacting only with such an oracle [6, 19, 25].

3 Evaluation settings

Our goal is to analyze the hardness of the LPN problem against fault attacks [22]. In general, such physical attacks do not target the mathematical problems but their implementation. Therefore, this section describes the types of implementation and the types of faults that we consider for this purpose.

3.1 LPN architectures

We consider serial and parallel architectures for the inner product computation that has to be performed by LPN implementations. An example of serial (resp. parallel) inner product computation is given in Figure 1 (resp. Figure 2). We use the notation $\mathcal{S}_{\mathbf{k},\epsilon}$ for serial implementations and the notation $\mathcal{P}_{\mathbf{k},\epsilon}$ for parallel ones. For simplicity, we will further denote the result of the AND and XOR intermediate results involved in these inner product computations by the notations A_i and B_j as represented in the figures. For an n-bit inner product computation, we have n ANDs and $n-1$ XORs in the serial architecture, and n ANDs and $\sum_{i=0}^{\log_2(n)-1} 2^i$ XORs in the parallel ones (the latter equals $n-1$ if n is a power of 2). For serial architectures, the depth of an intermediate XOR is its index, while for parallel ones, the depth of an intermediate XOR is $\lfloor \log_2(j) \rfloor$.

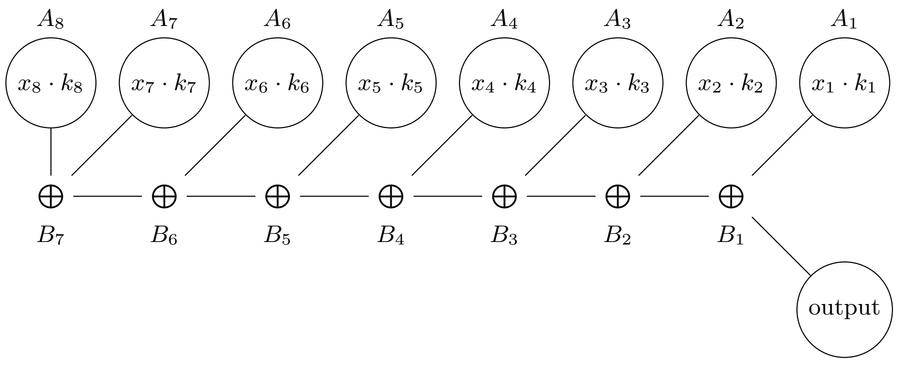


Fig. 1. Serial inner product architecture.

3.2 Fault models

Based on the previous (generic) architectures, we will consider an adversary who does not only observe the outputs of the implementations $\mathcal{S}_{\mathbf{k},\epsilon}$ or $\mathcal{P}_{\mathbf{k},\epsilon}$ (for which the security would be identical as long as their result is XORed with a Bernoulli noise), but is also able to inject faults during their execution.

More precisely, we consider an adversary who is able to manipulate the intermediate results A_i's and B_j's. As generally considered in the literature on fault attacks, we will consider the following features for this adversary:

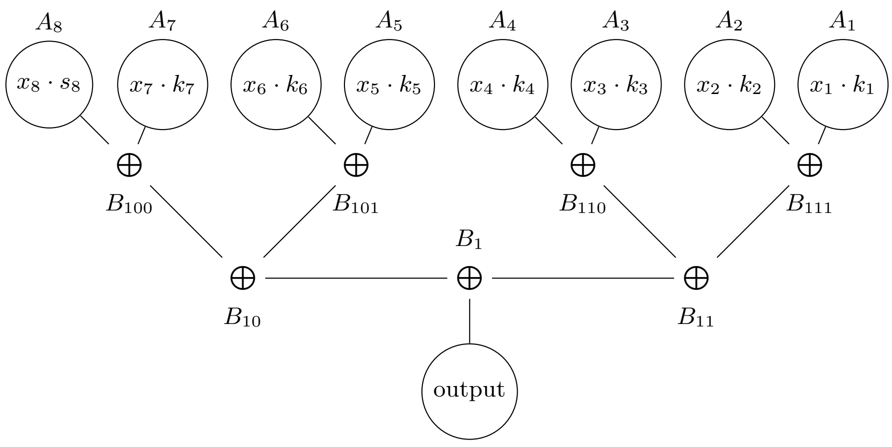


Fig. 2. Parallel inner product architecture.

1. *Fault model:* the adversary can either flip a bit or set it (to 0 or 1).
2. *Fault accuracy:* the adversary can either choose exactly the index of the intermediate ANDs and XORs he targets, or choose a set of indices so that the fault randomly happens on one of them (according to some distribution), which reflect the higher complexity of injecting accurate faults [3].
3. *Fault cardinality:* the adversary can inject a single fault or multiple faults.

4 Flipping bits is (mostly) useless

We first observe that there are two possible options to flip bits in the inner product architectures of Figures 1 and 2, namely targeting the ANDs inputs or targeting the XORs inputs and outputs. When targeting the ANDs inputs, one can affect either the public challenges $\mathbf{x}$'s or the key $\mathbf{k}$. In this respect:

1. *Flipping key bits does not modify the security of the LPN problem.* In standard LPN every challenge $\mathbf{x}$ provides the adversary with a parity equation of the key bits. In this faulty version, one (or several) key bit(s) of this parity equation will simply be XORed with 1 (which is equally informative).
2. *Flipping challenge bits is equivalent to a man-in-the-middle attack* like the one on [14]. So while these attacks have to be considered at the protocol level (e.g. with [15]), they do not target the implementation of the inner product computations and we consider them out of the scope of this paper.

So we are essentially left with the case where we target the XORs inputs and outputs. In this context, we start with the observation that for our two architectures, $\mathcal{S}_{\mathbf{k},\epsilon}$ and $\mathcal{P}_{\mathbf{k},\epsilon}$, flipping a bit at any place of the computation is equivalent to performing $\oplus 1$ at the end of the computation of the inner product. This simply results from the commutativity and associativity of the group operation $\oplus$.

Next, we have that an adversary exploiting this kind of faults can observe outputs of the form $\langle \mathbf{x}|\mathbf{k}\rangle \oplus \nu \oplus 1$ rather than $\langle \mathbf{x}|\mathbf{k}\rangle \oplus \nu$ in a standard LPN problem. In other words, the adversary can observe the distribution $\mathbf{D}_{\mathbf{k},1-\epsilon}$ rather than $\mathbf{D}_{\mathbf{k},\epsilon}$. The complexity of solving the LPN problem is identical in both cases since one can trivially produce the samples of one distribution with the other.

So the important fault model where we flip bits is in fact quite irrelevant to attack LPN implementations, since it does not provide the adversary with better advantage than active (e.g. man-in-the-middle) attacks exploiting careful modifications of the challenges. In particular, flipping bits during the intermediate computations of serial or parallel inner product implementations is useless.

5 Setting bits in serial implementations

We now analyze the security of the serial LPN implementation in case an adversary can take advantage of the (more informative) model where bits are set to zero. Essentially, such faults allow the adversary to simplify the LPN problem to small LPN instances (of size $n' < n$) and to exploit an extend-and-prune strategy similar to the one of [9]. Concretely, there are two possible situations that can occur. Either the resulting instances are so small that one can implement optimal attacks against LPN, where one just applies a maximum likelihood approach to recover key nibbles one by one. In this case, n' is small enough to be enumerated and the attack is in fact reminiscent of the template attacks used in the context of side-channel analysis [9]. Otherwise, the instances are such that n' is still too large to be exhaustively analyzed, in which case efficient algorithms such as [6, 19, 25] have to be exploited. We will see that the optimal strategy is easily applicable against serial implementations of LPN, while the more efficient ones will be required to evaluate parallel implementations in Section 6.

In this section, our goal is to analyze the security of serial LPN implementations against fault attacks in function of their main parameters. This naturally includes the LPN size parameter n and noise parameter ϵ. Additionally, we will consider the computational power of the adversary c (which corresponds to the number of key guesses he is able to make in a maximum likelihood attack), and the accuracy of the faults that we capture with a parameter Δ, which is the the number of positions on which the fault can be inserted according to some distribution: $\Delta = 1$ means that the adversary can exactly select the position of the fault, $\Delta = 2$ means that the support of the distribution is 2, ... Based on these notations, we will consider an extend-and-prune strategy, where the adversary performs $\frac{n}{d}$ attacks against key nibbles of $d = \log_2(c)$ bits (for convenience, we assume c to be a power of 2). Following, we will first evaluate the success rate of an attack against a single key nibble and then its generalization to full keys. For simplicity, we will describe our attacks against single key nibbles using notations corresponding to the first key nibble. In case of accurate attacks (with $\Delta = 1$), this means that the fault is set on bit B_{d+1} of Figure 1. In case of inaccurate attack, the fault will be set on positions ranging from $B_{d-\Delta+2}$ to B_{d+1}.

In practice, performing extend-and-prune attacks essentially requires to maximize the probability of each key nibble k^* after observing q queries of the form $(\mathbf{x}_j, y_j)$. For this purpose, we will start with the simple case of accurate fault attacks without computation and then investigate the more complex cases where the adversary exploits computation and the fault is not perfectly accurate. We will additionally combine different types of results. Namely, we will consider experimental attacks, that we will first explain based on exact formulas and then approximate using simple statistical tools, in order to gain intuition about how our main parameters influence the success rate of the attacks.

Remark 1. We next consider the fault distribution of inaccurate attacks to be uniform over Δ positions, which seems a natural first step to gain intuition about the impact of such inaccuracies. Indeed, the main goal in this paper is to put forward the interesting properties of LPN implementations against fault analysis based on standard models. Conceptually, there is nothing that prevents the following attacks to be applied to any type of distribution if they are given to the adversary. In case of concrete implementations, this would then require some kind of profiling, which we leave as an interesting scope for further research.

5.1 Accurate fault attacks without computation

In this simplest case, we perform extend-and-prune by dividing the key in blocks of a single bit, and repeat this process n times. We first show how to isolate and recover the first key bit k_1, then how to isolate and recover the jth key bit k_j, knowing the key bits $k_1, k_2, \ldots, k_{j-1}$. For this purpose, we set the output of the B_2 XOR to zero, and collect samples corresponding to plaintexts $\mathbf{x}$ with $x_1 = 1$. Setting this bit to zero implies that the output of the faulty computation is $y := x_1 \cdot k_1 \oplus \nu$, and using plaintexts $\mathbf{x}$ whose $x_1 = 1$ implies that $y = k_1 \oplus \nu$ (the other plaintexts do not reveal any information). Since both k_1 and $\nu \in \mathbb{Z}_2$, we obtain k_1 with probability $1 - \epsilon$ and $1 \oplus k_1$ with probability ϵ. Hence, we directly recover the key by performing a majority vote based on the collected samples, where votes for the wrong candidates can be simulated by a binomial random variable of parameters q (number of votes obtained) and ϵ (probability of obtaining that vote). We can compute the success rate of this process. Let $\Phi(s, q, \epsilon)$ be the cumulative function valued at sample s of the binomial distribution $B(q, \epsilon)$ and $\phi(s, q, \epsilon)$ be the corresponding probability mass function. The probability that the good candidate wins is the probability that the wrong candidate obtains less than half votes (and one half if the number of votes is tied), so that:

$$SR_1(q, \epsilon) = \Phi\left(\frac{q}{2}, q, \epsilon\right) - \frac{1}{2} \cdot \phi\left(\frac{q}{2}, q, \epsilon\right) .$$

Figure 3 (1-bit nibble curves) illustrates that this theoretical success rate nicely matches the experimental one obtained based on 1000 independent experiments. By experiments, we mean simulated fault analyses, where the adversary performs template attacks with perfect models $\Pr[k^*|\mathbf{x}_i, y_i]$ obtained via theoretical prediction and confirmed by concrete profiling from $\mathbf{x}_i, y_i$ samples. Note that for

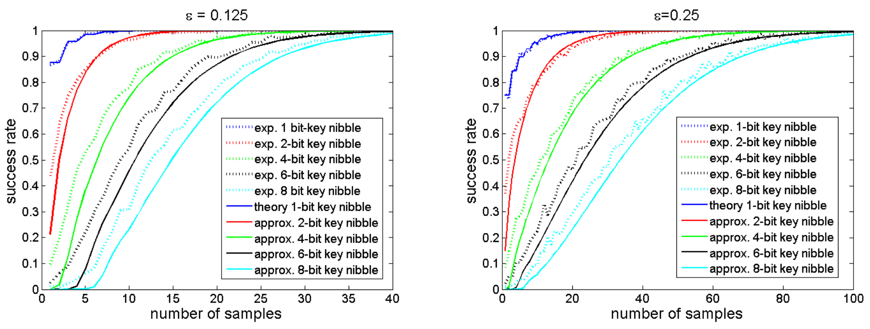


Fig. 3. Accurate fault attacks against a single key nibble.

such small nibbles, the success rate curve has a stepped (i.e. non continuous) shape which is due to the aforementioned possibility of tied votes.

Next, and knowing the first $j-1$ key bits, we can recover the jth one by setting the output of the B_{j+1} XOR to zero, and collecting plaintexts whose $x_j = 1$. As before, from the faulty computation we obtain $y := x_j \cdot k_j \oplus \left(\bigoplus_{i=1}^{j-1} x_i \cdot k_i \right) \oplus \nu$. Since we know the first $j-1$ key bits (and plaintext bits), we can easily subtract them, so that we are able to obtain $z := x_j \cdot k_j \oplus \nu$. Using only plaintexts whose $x_j = 1$ we obtain $z = k_j \oplus \nu$, i.e. the same situation as before, so we can recover the key bit k_j in the same way. We finally compute the success rate $SR_n(q,\epsilon)$ of the full key recovery process. For this purpose, we consider the simple strategy where we aim for a perfect pruning, i.e. we require single-nibble success rates such that no error will harm the full key recovery. In this case, the global success rate equals the probability of not doing errors at every step:

$$SR_n(q,\epsilon) = SR_1(q,\epsilon)^n.$$

If $N_s(n,\epsilon,\theta)$ is the minimum number of samples such that $SR_1(N_s(n,\epsilon,\theta),\epsilon) \geq \sqrt[n]{\theta}$, so that $SR_n(N_s(n,\epsilon,\theta),\epsilon) \geq \theta$, we obtain that we need $2 \cdot N_s(n,\epsilon,\theta) \cdot n$ samples to recover the full key with probability $\geq \theta$. The factor 2 is due to the fact that at each (j-th) step we discard the plaintexts whose $x_j = 0$.

Remark 2. A natural extension of the perfect pruning in this subsection would be to keep a list of the most likely keys for several key nibbles, e.g. thanks to enumeration [31]. The exploitation of computing power by considering larger key nibbles in the next subsection makes a first step in this direction.

5.2 Accurate fault attacks with computation

We now show how to extend the previous attack by taking advantage of computational power. That is, instead of recovering the key bits one by one, we try to recover larger key nibbles one by one. For this purpose, and assuming a computational power of $c = 2^d$, we set the output of the B_{d+1} XOR to zero, thus obtaining $y := \left(\overset{d}{\underset{i=1}{\oplus}} x_i \cdot k_i\right) \oplus \nu$ from the faulty computation. As before we can observe that we have to discard the plaintexts $(x_1, ..., x_d) = (0, ..., 0)$, which appear with probability $\frac{1}{2^d}$, because they do not reveal any information. Furthermore, we observe that for a wrong subkey candidate $\mathbf{k}^*$, the probability that $y = (x_1, ..., x_d) \cdot (k_1^*, ..., k_d^*)$ is $\eta = (1-\epsilon)\frac{2^{n-1}-1}{2^n-1} + \epsilon\frac{2^{n-1}}{2^n-1}$, while this probability is $1-\epsilon$ for the good subkey candidate. Therefore, for each possible subkey $\mathbf{k}^*$ we can count the number of samples for which $y = (x_1, ..., x_d) \cdot (k_1^*, ..., k_d^*)$, which should be maximum for the good subkey candidate when a sufficient number of samples are observed. Unfortunately, although we can assume that number of right answers for a wrong key candidate is a binomial of parameters q (number of samples obtained) and η, whilst this second parameter is $1-\epsilon$ for the good subkey, we cannot assume that these binomial random variables are independent, since they are produced with the same challenges $\mathbf{x}_i$'s. Yet, denoting them as $B_{\mathbf{k}^*}$ and under this independence assumption, we can approximate the success rate of an accurate fault attack against a d-bit nibble as:

$$SR_d(q, \epsilon, d) \simeq \prod_{\substack{\mathbf{k}^* \in \mathbb{Z}_2^d \\ \mathbf{k}^* \neq \mathbf{k}}} \Pr\left[B_{\mathbf{k}^*}(q, \eta) \lneq B_{\mathbf{k}}(q, 1-\epsilon)\right],$$

where we ignore the possibility of ties for simplicity. As illustrated in Figure 3 for the larger nibbles, such a formula indeed provides a good approximation of the experimental success rates. Naturally, the positive impact of exploiting more computing power is not obvious in this figure, since it considers a single nibble recovery. In order to recover the other key nibbles, we then proceed as in the previous section and compute:

$$SR_n(q, \epsilon, d) = SR_d(q, \epsilon)^{\frac{n}{d}}, \qquad (1)$$

since we have $\frac{n}{d}$ blocks of d bits to recover.[1] Again, if $N_s(\frac{n}{d}, \epsilon, \theta, d)$ is the minimum number of samples such that $SR_d(N_s(\frac{n}{d}, \epsilon, \theta), \epsilon) \geq \sqrt[n]{\theta^d}$, so that $SR_n(N_s(\frac{n}{d}, \epsilon, \theta), \epsilon)^{\frac{n}{d}} \geq \theta$, we obtain that we need $\frac{2^d}{2^d-1} \cdot N_s(\frac{n}{d}, \epsilon, \theta) \cdot \frac{n}{d}$ samples to recover the whole key with probability $\geq \theta$. The factor $\frac{2^d}{2^d-1}$ is due to the fact that at each step, we discar the plaintexts of which the bits corresponding to the target key nibble are all zeros (that occur with probability $\frac{1}{2^d}$).

Figure 4 illustrates the evolution of the success rate against a 1024-bit key, for $\epsilon = 0.125$ and $\epsilon = 0.25$ and computing powers ranging from 8-bit (which is instantaneously computed on a single desktop computed) to 32-bit. This time we clearly see the positive impact of exploiting computation from the data complexity point-of-view. We also observe the same "saturation effect" as when

[1] Intuitively, the independence assumption is reasonable since what we require is that for each key candidate $\mathbf{k}^*$ there exists enough plaintexts belonging to $\mathbb{Z}_2^d \setminus V(\langle \mathbf{x} | \mathbf{k} \oplus \mathbf{k}^* \rangle)$, with $V(\langle \mathbf{x} | \mathbf{k} \oplus \mathbf{k}^* \rangle)$ the hyperplane defined by the equation $\langle \mathbf{x} | \mathbf{k} \oplus \mathbf{k}^* \rangle = 0$.

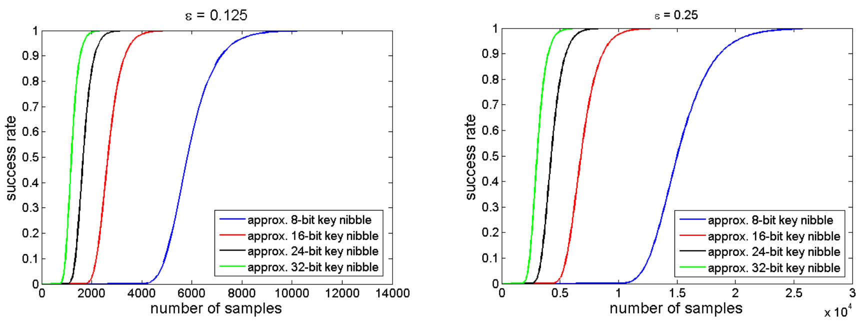


Fig. 4. Accurate fault attacks against a $n = 1024$-bit key.

exploiting enumeration in divide-and-conquer side-channel attacks [31], which typically suggests to limit the nibble size to 32 bits. Eventually, and interestingly, these results highlight that breaking a serial LPN implementation with accurate fault attacks requires a significantly larger number of faulty samples than for standard cryptographic primitives such as block ciphers [4, 28].

5.3 Inaccurate fault attacks without computation

We next extend the previous analyses where the adversary cannot perfectly control the position of his faults. Again, we start with the simple case where we do not use any computation and want to recover the key bit by bit. In order to illustrate the intuitions behind inaccurate fault insertion, we start with an example where $\Delta = 2$ and we generalize it afterwards. Furthermore, and as previously, we only consider plaintexts $\mathbf{x}$ such that $x_1 = 1$.

Example with $\Delta = 2$. Let us suppose that the faulty computation gives $y := k_1 \oplus \nu$ with probability $\frac{1}{2}$ and $y := k_1 \oplus x_2 \cdot k_2 \oplus \nu$ with probability $\frac{1}{2}$. Denoting $W \simeq B(1, \frac{1}{2})$, we can write this outcome in a compact way as:

$$y := k_1 \oplus w\,(x_2 \cdot k_2) \oplus \nu.$$

Since we do not know the value of the bit k_2, we then have 2 possibilities:

- if $k_2 = 0$, $y = k_1 \oplus \nu$,
- if $k_2 = 1$, $y = k_1 \oplus w \cdot x_2 \oplus \nu$.

In the first case ($k_2 = 0$), we directly have $\Pr[y = k_1] = 1 - \epsilon = \frac{1}{2} + \frac{1-2\epsilon}{2}$.
In the second case ($k_2 = 1$), $\Pr[y = k_1]$ becomes:

$$\Pr[w = 0, \nu = 0] + \Pr[w = 1, x_2 = 0, \nu = 0] + \Pr[w = 1, x_2 = 1, \nu = 1] = (*)\,.$$

Since in LPN we obtain uniformly random samples, we have $\Pr[x_2 = 1] = \Pr[x_2 = 0] = \frac{1}{2}$. Further relying on the fact that the noise, the position of the fault and x_2 are independent, we obtain:

$$(*) = \frac{1}{2}(1-\epsilon) + \frac{1}{2}\frac{1}{2}(1-\epsilon) + \frac{1}{2}\frac{1}{2}(\epsilon) = \frac{1-\epsilon}{2} + \frac{1}{4} = \frac{3-2\epsilon}{4} = \frac{1}{2} + \frac{1-2\epsilon}{4}.$$

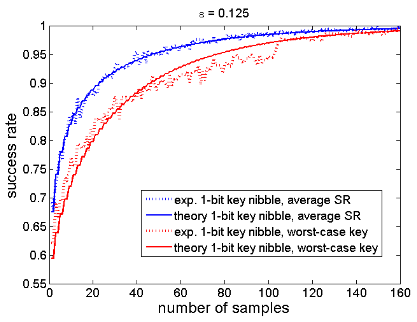
This example leads to the main observation that despite we only target the first key bit k_1, the bias[2] that can be exploited in the attack actually depends on the other key bits that can be "covered" by the inaccurate faults (i.e. of which the index is lower than $\Delta + 1$), namely k_2 in the previous case. This leads to two important consequences. First, the success rate of inaccurate fault attacks against LPN is key-dependent. That is, there are (worst-case) keys that are more difficult to distinguish than others. Hence, we will next consider both average and worst-case success rates. Second, the fact that we target a single key bit while the faults actually cover several key bits inevitably makes our attack suboptimal, because of an imperfect model. This naturally gives incentive for considering attacks with computation as in the next section, since we can build perfect models again when $c > \Delta$. Interestingly, despite suboptimal the attacks without computation are functional, because the impact of the modeled key bit k_1 dominates over the impact of the other key bits in the bias expression.

Note that the success rate to recover the full key can be computed with the same formula as in the accurate case, by adapting the biases. Worst-case partial success rates have to be considered for perfect pruning (since in this case we want to recover each key nibble with probability one), while the average success rate could be exploited in the advanced strategies mentioned in Remark 2.

General case. Let Δ be the number of possible position of the fault. That is the fault can be in the output of the B_i XOR where $i = 2, ..., \Delta+1$. Let $r \leftarrow [2, ..., \Delta]$ be the actual position of a fault for one sample y. Let finally $P := [p_1, ..., p_\Delta]$ be the vector of the probabilities p_i that the fault is at the output of the B_{i+1} XOR (in our uniform errors case, $p_i = \frac{1}{\Delta}$). As previously, we have that if a fault occurs at the output of the B_2 XOR, we obtain $y = k_1 \oplus \nu$, whilst if it is at the output of the B_{r+1} XOR, we obtain $y = k_1 \oplus \left(\overset{r}{\underset{i=2}{\oplus}} x_i \cdot k_i\right) \oplus \nu$. For simplicity, we start by looking at the case where $k_2 = 1$, in which $\Pr\left[\overset{r}{\underset{i=2}{\oplus}} x_i \cdot k_i = 0\right] = \frac{1}{2} = \Pr\left[\overset{r}{\underset{i=2}{\oplus}} x_i \cdot k_i = 1\right]$ for all r's. Using the notation $W = B(1, 1-p_1)$, we can see an output of the faulty computation as $y := k_1 \oplus w\left(\overset{r}{\underset{i=2}{\oplus}} x_i \cdot k_i\right) \oplus \nu$ and compute:

$$\begin{aligned}\Pr[y = k_1] = \Pr[w = 0, \nu = 0] \\ &+ \Pr\left[w = 1, \left(\overset{r}{\underset{i=2}{\oplus}} x_i \cdot k_i\right) = 0, \nu = 0\right] \\ &+ \Pr\left[w = 1, \left(\overset{r}{\underset{i=2}{\oplus}} x_i \cdot k_i\right) = 1, \nu = 1\right], \\ &= p_1(1-\epsilon) + (1-p_1)\frac{1-\epsilon}{2} + \frac{\epsilon}{2}(1-p_1),\end{aligned}$$

[2] Defined as the distance from the uniform probability $\frac{1}{2}$.



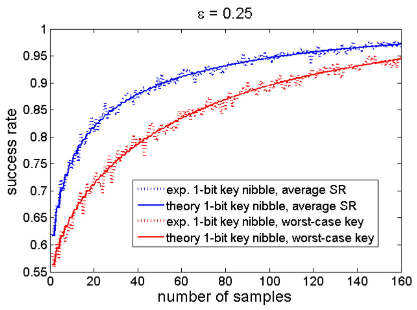


Fig. 5. Inaccurate fault attacks against a single key nibble with $\varDelta = 4$.

$$= p_1(1-\epsilon) + \frac{1-p_1}{2} = \frac{1}{2} + \frac{p_1 - 2p_1\epsilon}{2},$$
$$= \frac{1}{2} + p_1\frac{1-2\epsilon}{2}. \qquad (2)$$

From this equation, we clearly see that the inaccuracy of the fault decreases the bias by a factor $p_1 = \frac{1}{\varDelta}$. Let us now look at the cases where $k_2 = 0$. Then, for any $\delta \in [2, \varDelta]$ we have that if $(k_2, ..., k_\delta) = (0, ..., 0)$ and the fault is at the output of the B_{i+1} XOR with $i = 2, ..., \delta$, the output y equals to $k_1 \oplus \nu$ (i.e. is the same as if the fault is at the output of the B_2 XOR). So this case is captured by replacing p_1 in Equation (2) by $\sum_{i=1}^{\delta} p_i$. Since this sum is larger than p_1, $\Pr[y = k_1]$ will be larger too and we will have a larger bias. More generally, it is easy to see that the increase of the bias depends only on the position of the first 1 in the key bits $(k_2, ..., k_\varDelta)$. That is, the lowest bias is found for keys such that $k_2 = 1$, followed by keys such that $(k_2, k_3) = 01$ which have a slightly larger bias, ...

Based on the previous observations, we are finally able to compute $\Pr[y = k_1]$ for all possible keys. For this purpose, we define a vector $\mathbf{k}_{i,j} := (k_i, ..., k_j)$, such that we have 2 keys with $\Pr[y = k_1] = 1 - \epsilon$ (i.e. the 2 keys such that $\mathbf{k}_{2,\varDelta} = \mathbf{0}$), 2 keys with $\Pr[y = k_1] = 1 - \epsilon - p_n\frac{1-2\epsilon}{2}$ (i.e. the 2 keys such that $\mathbf{k}_{2,\varDelta-1} = \mathbf{0}$ and $k_\varDelta = 1$), 4 keys with $\Pr[y = k_1] = 1 - \epsilon - (p_\varDelta + p_{\varDelta-1})\frac{1-2\epsilon}{2}$ (i.e. the 4 keys such that $\mathbf{k}_{2,\varDelta-2} = \mathbf{0}$ and $k_{\varDelta-1} = 1, \ldots$), until we have $2^{\varDelta-1}$ keys with $\Pr[y = k_1] = 1 - \epsilon - \left(\sum_{i=2}^{\varDelta} p_i\right)\frac{1-2\epsilon}{2} = \frac{1}{2} + p_1\frac{1-2\epsilon}{2}$. Hence, we can compute the average success rate in function of $SR_1(q, \epsilon)$ as defined in Section 5.2:

$$SR_1(q,\epsilon,P) := \frac{1}{2^\varDelta}\left[2 \cdot SR_1(q,\epsilon) + \sum_{i=1}^{\varDelta-1} 2^i \cdot SR_1\left(q, \epsilon + (\sum_{j=\varDelta-i}^{\varDelta} p_j)(1-2\epsilon)\right)\right].$$

We confirm that this prediction of the average success rate, and worst-case success rate $SR_1\left(q, \frac{1}{2} - \frac{p_1}{2}(1-2\epsilon)\right)$, matches experimental results in Figure 5.

Impact on the number of samples. We conclude this section by looking at the impact of a reduced bias due to inaccurate faults on the number of samples needed to perform successful attacks. For this purpose, we first recall that the success rate is the probability that the wrong key nibble (here bit) receives more votes than the good one. (to make this discussion simpler, we omit the case of ties). The number of votes for the wrong key nibble made with q samples is represented by a binomial random variable $X \sim B(q, \eta)$ of parameter $\eta := \frac{1}{2} - \frac{p_1 - 2p_1\epsilon}{2}$. We know that we have a confidence interval of level α for this random variable, that is $\left[\eta q - \frac{k_\alpha}{2}, \eta q + \frac{k_\alpha}{2}\right]$, and we want that all the values in this interval are lower than $\frac{q}{2}$. So we need that:

$$\eta q + \frac{k_\alpha}{2}\sigma(X) \lneqq \frac{q}{2},$$

$$\frac{k_\alpha}{2}\sqrt{q(\eta)(1-\eta)} \lneqq q\left(\frac{1}{2} - \eta\right),$$

$$\frac{k_\alpha}{2}\frac{\sqrt{\eta - \eta^2}}{\frac{1}{2} - \eta} \lneqq \sqrt{q}.$$

Defining $\tau := \frac{1}{2} - \eta$ we have $\eta - \eta^2 = \frac{1}{4} - \tau^2$ and the previous inequality becomes:

$$q^2 \gneqq \frac{k_\alpha^2}{4}\frac{\frac{1}{4} - \tau^2}{\tau^2} \gneqq \frac{k_\alpha^2}{4\tau^2}.$$

So we observe that if we multiply the bias τ by a factor $\frac{1}{\Delta}$ (as caused by Δ-inaccurate faults), we need to multiply the number of samples by a factor Δ^2.

Note that one possible way to mitigate the inaccuracy of the faults would be to filter the challenges so that in case there are Δ possible places for the fault, the adversary only keeps challenges such that the first Δ coordinates are $(1, 0, ..., 0)$. Yet, this filtering increases the data complexity exponentially (in 2^Δ) while the previous treatment of inaccuracies only does it quadratically.

5.4 Inaccurate fault attacks with computation

We finally investigate the practically important case where the adversary can only insert fault with a limited accuracy, but where he has a computational power $c = 2^d$ that can compensate the inaccuracy parameter Δ, meaning that he can insert fault at positions ranging from $B_{d-\Delta+2}$ to B_{d+1} with $d > \Delta$. As previously discussed, this again allows us to mount optimal attacks.

Concretely, and in order to simplify our treatment, we will again apply a strategy similar to the template attack in [9]. For this purpose, a straightforward approach would be to build templates directly from the samples $(\mathbf{x}_i, y_i)_{i=1}^q$ of a “faulty LPN oracle”, which is expensive since for characterizing d-bit partial inner products, we need to build templates for the 2^{2d} combinations of input and key. Luckily, it is actually possible to build such templates more efficiently. For this purpose, let again $P := (p_{d-\Delta+2}, \ldots, p_{d+1})$ be the vector of probabilities p_i that the fault is at the output of the B_{i+1} XOR, and $p^i_{\mathbf{x},\mathbf{k}^*}$ be the probability that putting the fault at the output of the B_{i+1} XOR we obtain a 1. Clearly $p^i_{\mathbf{x},\mathbf{k}^*} = 1-\epsilon$ if $\underset{j=1}{\overset{i}{\oplus}} x_j \cdot k^*_j = 1$ and ϵ if $\underset{j=1}{\overset{i}{\oplus}} x_j \cdot k^*_j = 0$. So by the law of total probability, we have $\Pr\left[y \overset{(f)}{=} 1\right] = \sum_{i=d-\Delta+1}^{d} p_i \cdot p^i_{\mathbf{x},\mathbf{k}^*}$, where the (f) superscript is for faulty inner product outputs, which can be written in a compact way as:

$$\Pr\left[y \overset{(f)}{=} \langle \mathbf{x}|\mathbf{k}^*\rangle = 1\right] = \sum_{i=d-\Delta+1}^{d} p_i \left((1-\epsilon) \underset{j=1}{\overset{i}{\oplus}} x_j \cdot k^*_j + \epsilon(1 \oplus \underset{j=1}{\overset{i}{\oplus}} x_j \cdot k^*_j)\right).$$

Using the previous templates, we can now compute $\prod_{i=1}^{q} \Pr[y_i|\mathbf{k}^*, \mathbf{x}_i]$ for every candidate $\mathbf{k}^*$ and look for the one maximizing the likelihood of the noisy samples y_i. By defining $p_{\mathbf{x},\mathbf{k}^*} := \Pr\left[y \overset{(f)}{=} \langle \mathbf{x}|\mathbf{k}^*\rangle = 1\right]$, we have $\Pr[y_i|\mathbf{k}^*, \mathbf{x}_i] = y_i \cdot p_{\mathbf{x}_i,\mathbf{k}^*} + (1-y_i)\cdot(1-p_{\mathbf{x}_i,\mathbf{k}^*})$. We can observe that $\Pr[0|\mathbf{k}^*, \mathbf{x}_i] = \Pr[1|\mathbf{k}^*, \mathbf{x}_i] = \frac{1}{2}$ iff $p_{\mathbf{x}_i,\mathbf{k}^*} = \frac{1}{2}$. Note that as in the previous section, there are keys that are easier/harder to attack depending on the value of their corresponding bias.

Next, we can define the statistical distance between two key candidates as:

$$d(\mathbf{k}^*, \mathbf{k}^{**}) := \sum_{\mathbf{x}\in\mathbb{Z}_2^d\setminus\mathbf{0}} \left|\Pr\left[y \overset{(f)}{=} \langle \mathbf{x}|\mathbf{k}^*\rangle = 1\right] - \Pr\left[y \overset{(f)}{=} \langle \mathbf{x}|\mathbf{k}^{**}\rangle = 1\right]\right|, \quad (3)$$

and use it to compute the probability to distinguish the good key from an incorrect one. Here, we note that in theory, we should compute the probability to distinguish the correct key from all the incorrect ones. Yet, this would require characterizing the closeness of all the key candidates. For simplicity, we will compute an upper bound on the success rate, where we only compute the probability to distinguish the correct key from its nearest neighbour, i.e. the key candidate $\mathbf{k} \oplus \mathbf{0}1$ for which we have flipped only the last bit of the correct key. As will be confirmed in our following experiments, this provides a good approximation of the actual success rate when the probability of success gets close to one. Note that for this key candidate, the probabilities in Equation (3) are are equal for $2^{d-1}-1$ plaintexts (namely, those for which $\langle \mathbf{x}|\mathbf{0}1\rangle = 0 = \langle \mathbf{x}|\mathbf{k}^{**} \oplus \mathbf{k}^*\rangle$), and their difference is $(1-2\epsilon)\sum_{i=d}^{n} p_i$ for the 2^{d-1} remaining samples.

In order to estimate the success rate, we now use to tools from [9], where the authors solved exactly this problem in the case of a template attack in which they try to distinguish two candidate subkeys $\mathbf{k}^*_1, \mathbf{k}^*_2$, using q leakage samples and assuming a Gaussian noise leading to an error probability:

$$\Pr_{err} := \frac{1}{2} erfc\left(\frac{\Theta}{2\sqrt{2}}\right),$$

with $\Theta^2 := (M_1 - M_2)^T \Sigma_q^{-1}(M_1 - M_2)$ and M_i is the vector containing the average value of q samples for the key $\mathbf{k}_i^*$ and Σ_q is the corresponding covariance matrix, modeling the noise distribution in these q samples. They additionally assume that the noise of every sample is iid for both candidate keys.

We simply apply this formula to the good subkey $\mathbf{k}$ and its nearest subkey $\mathbf{k}^*$ (i.e. $\mathbf{k} \oplus \mathbf{0}1$), by taking the samples $y_1, ..., y_q$ and modeling them as a Bernoulli distribution $\mathrm{Ber}(p_{\mathbf{k},\mathbf{x}_i})$, with $i = 1, ..., q$. Denoting $M_1 := M_{\mathbf{k}}$ and $M_2 := M_{\mathbf{k}^*}$, we have $M_1 = \left(p_{\mathbf{k},\mathbf{x}_1}, ..., p_{\mathbf{k},\mathbf{x}_q}\right)$ and $M_2 = \left(p_{\mathbf{k}^*,\mathbf{x}_1}, ..., p_{\mathbf{k}^*,\mathbf{x}_q}\right)$. Therefore, on average, we find that $\frac{2^{n-1}-1}{2^n-1}q$ coordinates of these vectors are the same, and the others are $\pm p_{d+1}\frac{1-2\epsilon}{2}$. This means that for the vector $[M_1 - M_2]$ we have approximately $\frac{2^{n-1}-1}{2^n-1}q$ 0s and the remaining coordinates are $p_{d+1}\frac{1-2\epsilon}{2}$.

As for the covariance matrix, this is where key dependencies come into play. For simplicity, we only considered the worst-case (which is needed to compute the full key recovery success rate of our extend-and-prune strategy). Hence, we simply set it to a maximum $\Sigma = \mathbb{1} \cdot \frac{1}{4}$. As a result, we directly obtain the following bound of the success rate for worst-case keys and d-bit nibbles:

$$SR_d(q, \epsilon, P) \approx 1 - \frac{1}{2} erfc\left(\frac{\Theta}{2\sqrt{2}}\right),$$

with $\Theta = 2\sqrt{S(q)p_{d+1}\frac{1-2\epsilon}{2}}$ if we define $S(q) := \lfloor q\frac{2^{n-1}-1}{2^n-1}\rfloor$. As previously mentioned, and clear from Figure 6, it starts from $\frac{1}{2}$ since we only distinguish two keys, and gets close to the actual success rate as the number of samples increases. We can then directly use Equation (1) from Section 5.2 to obtain the bounds on the full key success rate of Figure 7. Figure 8 in Appendix A additionally provides results for $\Delta = 8$ which confirms the simple intuition of inaccurate fault attacks without computation, that the data complexity of these attacks is proportional to Δ^2. In all cases, this data complexity is remarkably high.

Remark 3. The case of multiple faults is not very interesting in the context of serial implementation, since it is only the first bit set to zero (starting from the LSB) which matters in this case (as it cancels the effect of other faults).

Remark 4. The intermediate contexts, where the adversary exploits computation but his computational power $c = 2^d$ does not allow him to cover the full range of the possible faults (i.e. when $1 < d < \Delta$) could be analyzed with the same methodology as in this section, by simply adapting the distributions in hand. However, and as in the context of inaccurate fault attacks without computations discussed in Section 5.3, it would then lead to suboptimal attacks.

6 Setting bits in parallel implementations

In this section, we complement the previous results with a discussion of the security of parallel LPN implementations against fault attacks. Interestingly, this discussion can be quite succint, since we can re-use most of the tools in

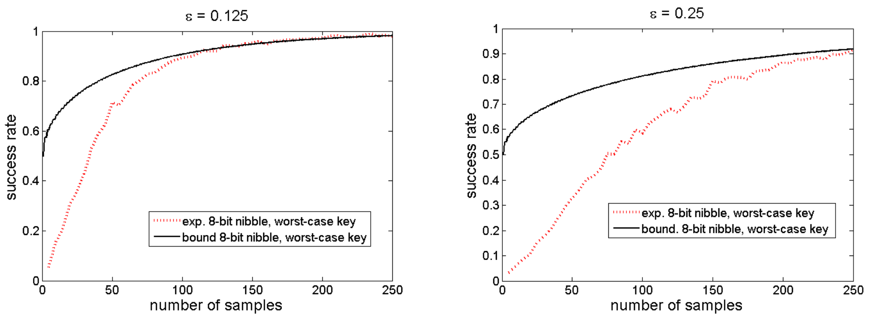


Fig. 6. Inaccurate fault attacks against a single key nibble with $\Delta = 4$.

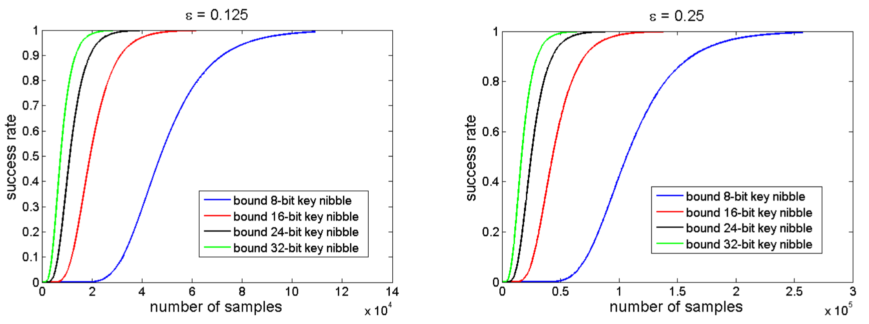


Fig. 7. Inaccurate fault attacks against a $n = 1024$-bit key with $\Delta = 4$.

the previous section. Essentially, the main difference between serial and parallel implementations is that the latter ones can only be attacked efficiently in case adversaries can insert multiple and accurate faults. Hence, we will start our discussion with a description of these attacks. Then, we will argue why accuracy is in fact strictly needed in the case of parallel implementations, by looking at simple examples of (not too) inaccurate fault attacks with small Δ's.

Multiple and accurate faults. Say we consider a n-bit parallel LPN architecture of depth t (i.e. $n = 2^t$). Assuming that an adversary can insert m accurate faults, the best strategy is to cancel the left bit of the top node (i.e. B_{10} in Figure 2), then cancel the left bit of the top node in the remaining right branch (i.e. B_{110} in Figure 2), $\ldots$, so that one obtains samples of a reduced LPN problem of size $n' = 2^{t-m}$. If the fault cardinality is such that n' gets below 32 (which would correspond to a very powerful adversary), then the maximum likelihood attacks in the previous section directly apply. Otherwise, the (more realistic) adversary can only consider these smaller LPN instances and try to solve them with standard algorithms such as BKW [6], LF [25] or based on covering codes [19]. The complexities of some (efficient) attacks, borrowed from the recent work of Bogos et al. [7], can be found in Table 1 (where we report on classical values for the parameter ϵ, namely $\frac{1}{8}$ and $\frac{1}{4}$). Note that various other tradeoffs between data, time and memory complexity could be investigated. We list attacks with optimized time complexities since the main reason to abandon maximum likelihood attacks is that their time complexity becomes untractable for large n' values. In this respect, the only important conclusion of this table is that the overall complexity of fault attacks against LPN anyway becomes larger when leaving the maximum likelihood paradigm, and holds in general. This confirms the excellent properties of parallel LPN implementations against fault attacks.

n'	ϵ	a	b	# of samples	memory	time
64	0.125	4	16	17.65	23.65	25.65
64	0.25	4	16	22.60	28.60	30.60
128	0.125	5	26	28.01	35.01	37.33
128	0.25	4	32	33.59	40.59	42.59
256	0.125	6	43	45.32	53.32	55.91
256	0.25	5	52	54.00	62.00	64.32
512	0.125	7	74	76.59	85.59	88.39
512	0.25	6	86	88.32	97.32	99.91

Table 1. Complexitites to solve LPN with the LF1 algorithm $\log 2$ scale (a and b are technical parameters representing the number of blocks and the block size in LF1).

On the need of accuracy. Let us now consider a similar architecture and an adversary who can only insert a single (slightly) inaccurate fault with $\Delta = 2$. Then, the best strategy will be to hit either the left or the right bit of the top node (i.e. B_{10} or B_{11} in Figure 2). Based on a reasoning similar to the one in Section 5.3 (recall the example with $\Delta = 2$), the adversary will then have to solve a $n' = 512$-bit LPN problem, with a probability $\frac{1}{2} + \frac{1-2\epsilon}{2} \cdot \frac{1}{2}$ (i.e. a halved bias) which is essentially as hard to solve as the original one. Furthermore, if the inaccuracy extends to more stages of the parallel LPN implementation, then any fault that occurs in an "already cancelled" branch of the implementation is lost. Hence, we have that accurate faults are in fact strictly necessary to carry out successful fault attacks against parallel LPN implementations.

7 Fault attacks against the randomness

Before to conclude the paper, we finally note that all the attacks considered so far exploited non-permanent faults targeting inner product computations. Yet, in view of the probabilistic nature of the LPN assumption, a natural question to ask is whether directly targeting the randomness would not be more fruitful for the adversary. For example, and assuming that one can observe noisy samples of the form $y = \langle \mathbf{x} | \mathbf{k} \rangle \oplus \nu$, it is clear that a single permanent fault canceling ν allows breaking the LPN assumption with approximately n samples.

In this respect, we first mention that such permanent faults are generally harder to inject, and most of the literature on fault analysis focuses on non-permanent faults [17]. Hence, the good features of LPN implementations against fault attacks detailed in the previous sections are certainly a meaningful first step in the understanding of their physical security properties. Admittedly, permanent faults need to be prevented, and finding solutions to ensure this condition is an interesting scope for further research. Yet, we also note that this is a quite general issue and a requirement for the security of many cryptographic implementations relying on good randomness. For example, a single permanent fault on the randomness used in masked implementation directly breaks the uniformity property that is need for masking to deliver security guarantees [18].

Besides, the previous attack against the randomness generation can obviously be carried out with non-permanent faults, just by repeating them n times. Yet, here again, the accuracy of the fault insertion has to be high. Indeed, with perfect accuracy, the adversary will observe samples such that $\Pr[y = \langle \mathbf{x}|\mathbf{k}\rangle | \Delta = 1] = 1$. By contrast, as soon as the accuracy decreases, the samples become noisy again and their exploitation requires BKW-like algorithms to break LPN. In general, we have $\Pr[y = \langle \mathbf{x}|\mathbf{k}\rangle | \Delta] = \frac{\Delta+1}{2\Delta}$, meaning that already for $\Delta = 2$, we have $\epsilon = \frac{1}{4}$, therefore confirming the positive observations in the previous sections.

8 Conclusion and open problems

Our results show that fault attacks against LPN implementations (especially parallel ones) are significantly more challenging than similar attacks against standard symmetric cryptographic primitives such as block ciphers. Indeed they can only succeed if accurate fault insertion based on "set bit" models is possible, and even in this case, have quite high sampling requirements. For illustration, we analyzed some of the mainstream fault models. Yet, our evaluations are quite generic and could easily be extended to other fault models, leading to similar intuitions. For example, since it is mainly the position of the last erroneous bit that influences fault attacks against LPN, burst errors could be directly captured. This naturally suggests the further investigation of LPN implementations as an interesting research direction. Open problems include the study of advanced attack paths, e.g. going beyond the simple extend-and-prune strategy that we considered as a first step, or more challenging scenarii, e.g. if the faults and their positions follow an unknown (or imperfectly profiled) distribution. Real world experiments would naturally be interesting too, in order to evaluate the extent to which the conclusions of our generic analyses apply to actual devices. In this respect, it is worth mentioning that in concrete fault attacks, it may also happen that no fault occurs at all. Since in the context of LPN (where the challenges are always different) there is no direct way to verify whether a fault actually occurred, this could make the attack even harder (typically, increase the bias). In view of the algebraic structure of the inner products carried out by LPN implementations, combining them with error detection/correction tools appears as a natural goal as well (to further amplify the good properties of LPN with respect to fault attacks). Eventually, the extension of our work towards other learning problems such as LWE [30] or LWR [2] is certainly worth attention.

Acknowledgments. François-Xavier Standaert is a research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in parts by the ERC project 280141 and by the ARC project NANOSEC.

References

1. Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. Inner product masking revisited. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 486–510. Springer, 2015.
2. Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In Pointcheval and Johansson [29], pages 719–737.
3. Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.
4. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.
5. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993.
6. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 435–440. ACM, 2000.
7. Sonia Bogos, Florian Tramèr, and Serge Vaudenay. On solving lpn using BKW and variants. *IACR Cryptology ePrint Archive*, 2015:49, 2015.
8. Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. Hb^{++}: a lightweight authentication protocol secure against some attacks. In *Second International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SecPerU 2006), 29 June 2006, Lyon, France*, pages 28–33, 2006.
9. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
10. Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In Pointcheval and Johansson [29], pages 355–374.
11. Alexandre Duc and Serge Vaudenay. HELEN: A public-key cryptosystem based on the LPN and the decisional minimal distance problems. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa,*

Cairo, Egypt, June 22-24, 2013. Proceedings, volume 7918 of *Lecture Notes in Computer Science*, pages 107–126. Springer, 2013.

12. Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 702–721. Springer, 2011.
13. Lubos Gaspar, Gaëtan Leurent, and François-Xavier Standaert. Hardware implementation and side-channel analysis of lapin. In Josh Benaloh, editor, *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, volume 8366 of *Lecture Notes in Computer Science*, pages 206–226. Springer, 2014.
14. Henri Gilbert, Matthew Robshaw, and Herve Sibert. Active attack against hb+: a provably secure lightweight authentication protocol. *Electronics Letters*, 41(21):1169–1170, 2005.
15. Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. Hb$^{\#}$: Increasing the security and efficiency of hb^{+}. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 361–378. Springer, 2008.
16. Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. How to encrypt with the LPN problem. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 679–690. Springer, 2008.
17. Christophe Giraud and Hugues Thiebeauld. A survey on fault attacks. In Jean-Jacques Quisquater, Pierre Paradinas, Yves Deswarte, and Anas Abou El Kalam, editors, *Smart Card Research and Advanced Applications VI, IFIP 18th World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), 22-27 August 2004, Toulouse, France*, volume 153 of *IFIP*, pages 159–176. Kluwer/Springer, 2004.
18. Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: how large is the gap for aes? *J. Cryptographic Engineering*, 4(1):47–57, 2014.
19. Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2014.
20. Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An efficient authentication protocol based on ring-lpn. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 346–365. Springer, 2012.

21. Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001.
22. Marc Joye and Michael Tunstall, editors. *Fault Analysis in Cryptography.* Information Security and Cryptography. Springer, 2012.
23. Ari Juels and Stephen A. Weis. Authenticating pervasive devices with human protocols. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.
24. Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient authentication from hard learning problems. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 7–26. Springer, 2011.
25. Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2006.
26. Marcel Medwed and François-Xavier Standaert. Extractors against side-channel attacks: weak or strong? *J. Cryptographic Engineering*, 1(3):231–241, 2011.
27. 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 - 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2012.
28. Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.
29. David Pointcheval and Thomas Johansson, editors. *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*. Springer, 2012.
30. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.
31. Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2012.

A Additional figures

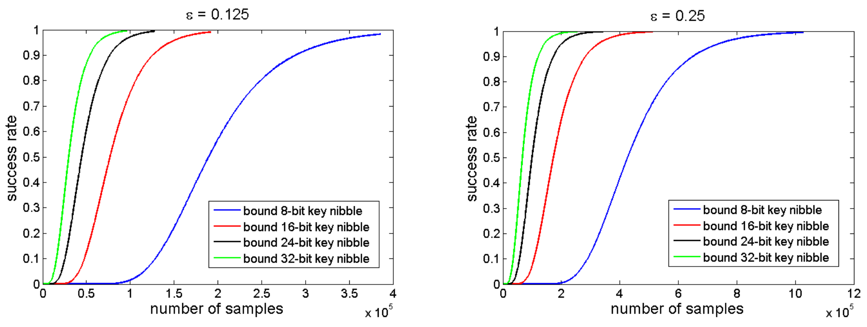


Fig. 8. Inaccurate fault attacks against a $n = 1024$-bit key with $\Delta = 8$.