A Systematic Approach to the Side-Channel Analysis of ECC Implementations with Worst-Case Horizontal Attacks

Romain Poussier[1],Yuanyuan Zhou[1,2], François-Xavier Standaert[1].

[1] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.
[2] Brightsight BV, The Netherlands.

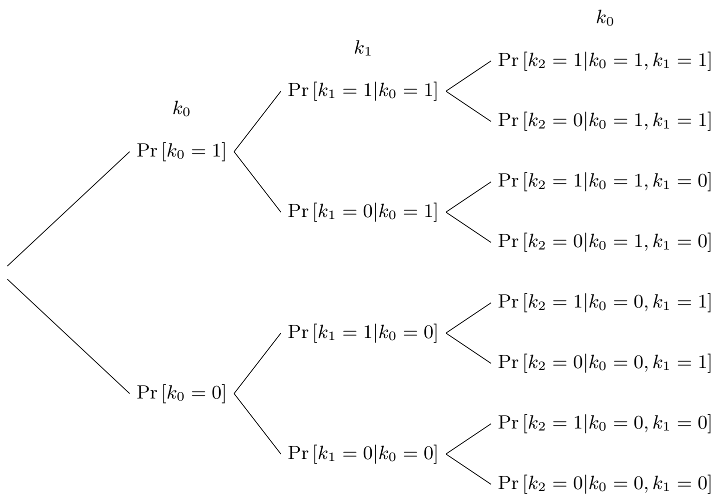
Abstract. The wide number and variety of side-channel attacks against scalar multiplication algorithms makes their security evaluations complex, in particular in case of time constraints making exhaustive analyses impossible. In this paper, we present a systematic way to evaluate the security of such implementations against horizontal attacks. As horizontal attacks allow extracting most of the information in the leakage traces of scalar multiplications, they are suitable to avoid risks of overestimated security levels. For this purpose, we additionally propose to use linear regression in order to accurately characterize the leakage function and therefore approach worst-case security evaluations. We then show how to apply our tools in the contexts of ECDSA and ECDH implementations, and validate them against two targets: a Cortex-M4 and a Cortex-A8 micro-controllers.

1 Introduction

State of the art. The secure implementation of Elliptic Curve Cryptography (ECC) is an important ingredient in modern information systems. In this paper, we are concerned with side-channel attacks against scalar multiplication implementations which have been the focus of continuous interest over the last 20 years. This literature informally divides these attacks in two main categories: attacks using a *Divide and Conquer* (DC) approach and attacks using an *Extend and Prune* (EP) approach – which we next survey.

Attacks that belong to the first category aim at recovering the scalar bits independently and are therefore simple to analyze. They associate a probability or a score to each scalar bit. The scalar is trivially recovered if all the correct bits have the highest probability. If it is not the case, computational power can be used to mitigate the lack of side-channel information thanks to key enumeration [40, 32, 37]. If the key is beyond computational reach (e.g. beyond 2^{60}), rank estimation (which requires the knowledge of the key and is therefore only accessible to evaluators, not to concrete adversaries) allows estimating the computational security level of a leaking implementation [41, 6, 21, 32].

Attacks using an EP approach recover the scalar bits in a recursive manner. Recovering the i-th bit requires to first recover all the previous ones. For a n-bit scalar, EP attacks can be seen as a probability tree with 2^n leaves where



k_0
k_1
k_0
$\Pr[k_0 = 1]$
$\Pr[k_0 = 0]$
$\Pr[k_1 = 1|k_0 = 1]$
$\Pr[k_1 = 0|k_0 = 1]$
$\Pr[k_1 = 1|k_0 = 0]$
$\Pr[k_1 = 0|k_0 = 0]$
$\Pr[k_2 = 1|k_0 = 1, k_1 = 1]$
$\Pr[k_2 = 0|k_0 = 1, k_1 = 1]$
$\Pr[k_2 = 1|k_0 = 1, k_1 = 0]$
$\Pr[k_2 = 0|k_0 = 1, k_1 = 0]$
$\Pr[k_2 = 1|k_0 = 0, k_1 = 1]$
$\Pr[k_2 = 0|k_0 = 0, k_1 = 1]$
$\Pr[k_2 = 1|k_0 = 0, k_1 = 0]$
$\Pr[k_2 = 0|k_0 = 0, k_1 = 0]$

Fig. 1: Conditional probability tree for a 3 bits scalar $k = (k_0, k_1, k_2)$.

each level corresponds to a different bit. Figure 1 illustrates such a probability tree for $n = 3$, where each node corresponds to the conditional probability of a bit given that the previous ones are correctly recovered. In this context, a first attack strategy is to only look at the most probable tree path. We refer to this method as first-order Success Rate (1-O SR). This strategy fails if not enough side-channel information is available for at least one of the bits. In such a case, the aforementioned enumeration algorithms cannot be applied due to the conditional dependencies. Yet, a recent study [29] describes a method to apply key enumeration/rank estimation on EP attacks.[1]

According to these two classes, we now describe the state of the art on side-channel attacks against scalar multiplications. For each of them, we describe its overall modus operandi, we show its associated class and we finally exhibit a countermeasure (if existing).

The first attacks discovered on scalar multiplication, such as timing and Simple Power Analysis [27, 28] (SPA), aimed at finding different patterns that depend on a scalar bit value. Such differences occur if the scalar multiplication or the elliptic curve operations are irregular, which makes the timing pattern of the leaking implementations dependent on all the scalar bits. As a consequence, an adversary has to recover the scalar in a recursive manner and these attacks

[1] Instead of only looking at the most probable path, it assumes that the tree can be divided into three parts. The first part corresponds to the bits which the adversary is certain to have recovered correctly. The second part contains the (computationally feasible) paths on which the adversary has partial information. The final part is the exhaustive remaining part of the tree on which the adversary has no information. Enumeration is done on this sub-tree using a Pollard-like method.

belong to the EP class. Using a regular scalar multiplication as further described in Section 2 naturally thwarts them.

Later on, (vertical) Differential Power Analysis (DPA) [28] and Correlation Power Analysis (CPA) [7] were applied to scalar multiplications [14, 34]. Such methods aim at recovering the scalar bits iteratively using several side-channel leakages traces of a scalar multiplication with a fixed scalar and several known inputs. Thanks to the inputs knowledge, the scalar bits are recovered by guessing one internal value that depends on the input and the guessed bits. Independently of the distinguisher (e.g. correlation, difference of means,...), we refer to all attacks using this framework as DPA. Here again, guessing the internal value associated with a scalar bit requires that all the previous bits have been correctly recovered. Hence, DPA also belongs to the EP class. Since this method requires leakages on several executions with a fixed scalar, scalar randomization techniques [14, 13] are efficient countermeasures. Protocols using a scalar nonce such as ECDSA [38] are naturally protected against DPA.

Next, Template Attacks (TA) [8] were introduced as a powerful tool to extract all the information available in some leakage traces. In the context of scalar multiplication, it has been first introduced against ECDSA [33]. From the prior knowledge of the input, the attack computes 2^d templates in order to recover the first d bits of the scalar nonce. The actual bit of the nonce is then found with templates matching. The secret key is finally recovered using lattice techniques [36, 5] from several partially known nonces and their associated signatures. Interestingly, since such TA do not recover the whole scalar, they neither belong to the EP nor DC classes.

Alternatively, Online Template Attacks (OTA) [2, 17] were introduced to recover the full scalar. This method interleaves the templates building and the attack phases, thus requiring more online access to the target device. As the iterative template building requires the knowledge of the current internal state which depends on the previous bits, OTA belongs to the EP class. Since both TA and OTA require the knowledge of the input, point randomization techniques [14, 25] are effective. Using the fact that $(X, Y, Z) = (\lambda^2 X, \lambda^3 Y, \lambda Z)$ in Jacobian coordinates (see section 2), an option to randomize the input is to change its coordinates at the beginning of each scalar multiplication using a random λ.

Horizontal Differential Power Attacks (HDPA) [12, 3] are another powerful alternative to TA. From the posterior knowledge of the input, the scalar bits are recovered by guessing several internal values that depend on both the input and the guessed bits (instead of only one in the case of DPA). As guessing the internal values associated with a particular bit requires the knowledge of the previous ones, HDPA belongs to the EP class. As for TA, point randomization is effective against HDPA.

Finally, Horizontal Collision Attacks (HCA) [42, 22, 15, 3, 4, 11] have been proposed in order to bypass the point randomization countermeasures. These attacks require the scalar multiplication to exhibit different operand input/output collisions that depend on the scalar bit values. Being able to detect such collisions from a trace allows recovering the scalar bits. Up to our knowledge, only one collision attack defeats the Montgomery scalar multiplication implemented as further described in Section 2.2 by Algorithm 1 [22]. The collision is able to find whether two consecutive scalar bits are the same or not. Given this information, the scalar is finally recovered using a modified version of the Baby-Step/Giant-Step algorithm. If they target a regular scalar multiplication, HCA belong to the DC class, since the position of the collisions in the leakage traces depends only on the current scalar bit value in this case. In case of irregular algorithms, this position depends on the previous bits and HCA then belong to the EP class. In general, countermeasures aiming at increasing the noise level (e.g. shuffling, random delays, ...) are the only effective ones against the attack presented in [22]. In this respect, one drawback of these attacks (from the adversary's viewpoint) is that they only exploit a very small part of the available information compared to HDPA and TA (which implies that the amount of noise needed to hide the collisions is more limited).

We additionally mention that all the aforementioned attacks (except the TA) ignore the leakages due to the direct manipulation of the scalar bits during the scalar multiplication (such as discussed in [23, 9, 35]). Exploiting these leakages is an orthogonal concern to this study and is therefore out of the scope of our investigations.

Our contribution. Based on this broad state of the art, our goal is to further investigate and systematize the security evaluation of scalar multiplication implementations against HDPA. Our motivations in this context are threefold:

First of all, such attacks can potentially exploit most of the informative samples provided by a leaking implementation, and are therefore a natural candidate for approaching their worst-case security level, which we aim for.

Second, most of the HDPA literature is based on the correlation distinguisher and assumes an a priori leakage model. Yet, given our goal to approach the worst-case security level of scalar multiplication implementations, it is natural to study efficient solutions allowing a better characterization of the leakages. In view of its broad applicability in the context of block cipher implementations, linear regression appears as an excellent candidate for this purpose [39], and we therefore study its applicability in our asymmetric setting.

Third, and quite importantly, only few practical experiments have been reported on the application of HDPA against actual implementations of scalar multiplication algorithms. We contribute to this issue by providing the results of experiments against two (more or less challenging) targets: the first one is a low frequency ARM Cortex M4 micro-controller (without interrupts), the second one runs a Linux operating system in background and runs at high frequency. While successful attacks against this second target have been published for block ciphers [1, 31], no public report discusses their vulnerabilities in the ECC case. We also illustrate the application of framework to both ECDH and ECDSA.

The rest of the paper is organized as followed. Section 2 introduces the notations and the necessary background on elliptic curve cryptosystems and implementations. Section 3 describes the generic view we consider for regular scalar multiplication along with the systematic security evaluation method. Finally, Section 4 and Section 5 respectively show the experimental results for the application of this framework against our two targets.

2 Background

2.1 Notations

We use capital letters for random variables and small caps for their realizations. We use sans serif font for functions (e.g. F) and calligraphic fonts for sets (e.g. $\mathcal{A}$). We use capital bold letters for matrices (e.g. $\mathbf{M}$) and small bold caps for vectors (e.g. $\mathbf{v}$). We denote the conditional probability of a random variable A given B with $\Pr[A|B]$.

2.2 Elliptic Curves Cryptography (ECC)

Let $\mathbb{F}_p$ be a finite field with a characteristic bigger than 3. We define by $\mathcal{E}(\mathbb{F}_p)$ the set of points $(x, y) \in \mathbb{F}_p^2$ (called affine coordinates) that satisfy the Weierstrass equation $y^2 = x^3+ax+b$, $(a, b) \in \mathbb{F}_p^2$ with discriminant $\Delta = -16(4a^3+27b^2) \neq 0$. $\mathcal{E}(\mathbb{F}_p)$ along with a point at infinity which form an Abelian additive group. The addition over $\mathcal{E}(\mathbb{F}_p)$ requires field additions, subtractions, multiplications and one inversion. We denote by $+$ the addition of two points P and Q, and by $[k]P$ the k-times repeated additions $P + P + ... + P$ with $k \in \mathbb{N}$ (called scalar multiplication).

Scalar multiplication. Most elliptic curve cryptosystems require to compute a scalar multiplication $[k]P$ from a number $k \in [1, |{<}P{>}| - 1]$ and a curve point P, where $|{<}P{>}|$ is the order of the subgroup generated by P. In each case, k is a sensitive variable unknown from the attacker which can be a private key (e.g. for ECDH key exchange) or directly linked to it (e.g. for ECDSA). As a result, the scalar multiplication represents an important source of potential side-channel leakages. In order to thwart the most basic side-channel attacks, scalar multiplication algorithms avoid conditional branching and have a regular execution independently of the bits of k. In the following we will consider the (left to right) Montgomery ladder [26] as described by Algorithm 1. We now view the n-bit scalar as a binary vector $\mathbf{k} = (k_0, ..., k_{n-1})$ (where k_0 is the most significant bit).

Jacobian Coordinates In general, field inversions are costly compared to additions, subtractions and multiplications. Moving from affine coordinates to Jacobian coordinates allows avoiding the inversion when performing an addition or a doubling over $\mathcal{E}(\mathbb{F}_p)$. The Jacobian plan $\mathcal{J}$ over $\mathbb{F}_p^3$ is defined as $\{(X, Y, Z) \in \mathbb{F}_p^3$ *s.t.* $\forall \lambda \in \mathbb{F}_p, (X, Y, Z) = (\lambda^2 X, \lambda^3 Y, \lambda Z)\}$. The set of points $\mathcal{E}_{\mathcal{J}}(\mathbb{F}_p)$ defined by the equation $Y^2 = X^3 + aXZ^4 + bZ^6$ defines an elliptic curve over the Jacobian plan. The Jacobian point (X, Y, Z), $Z \neq 0$ corresponds to the affine point

Algorithm 1 Montgomery ladder.

```
**Input:** $P, \mathbf{k} = (k_0, ..., k_{n-1})$
**Output:** $[k]P$
  $R_0 \leftarrow \mathcal{O}$
  $R_1 \leftarrow P$
  **for** $i = 0$ to $n - 1$ **do**
    $R_{1-k_i} \leftarrow R_{1-k_i} + R_{k_i}$
    $R_{k_i} \leftarrow [2]R_{k_i}$
  **end for**
  **return** $R_0$
```

$(X/Z^2, Y/Z^3)$. The point at infinity in affine coordinates corresponds to the point $(\lambda^2, \lambda^3, 0)$ in Jacobian coordinates.

Given two points P and Q in $\mathcal{E}_{\mathcal{J}}(\mathbb{F}_p)$ with $P \neq \pm Q$, the formulas for the addition $P+Q$ and doubling $P+P$ are respectively given by Algorithms 2 and 3. As it is important for the rest of the paper, we stress the fact that an addition over $\mathcal{E}_{\mathcal{J}}(\mathbb{F}_p)$ (resp. a doubling) requires 16 field multiplications (resp. 10).

Algorithm 2 Addition over $\mathcal{E}_{\mathcal{J}}(\mathbb{F}_p)$.

```
**Input:** $P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2), P \neq \pm Q$
**Output:** $P + Q = (X_3, Y_3, Z_3)$
  $\mathsf{Z}_1 \leftarrow Z_1^2, \mathsf{Z}_2 \leftarrow Z_2^2, U_1 \leftarrow X_1\mathsf{Z}_2, U_2 \leftarrow X_2\mathsf{Z}_1, H \leftarrow U_1 - U_2, S_1 \leftarrow Y_1\mathsf{Z}_2Z_2, S_2 \leftarrow$
  $Y_2\mathsf{Z}_1Z_1, R \leftarrow S_1 - S_2, \mathsf{H} \leftarrow H^2, G \leftarrow \mathsf{H}H, V \leftarrow U_1\mathsf{H}$
  $X_3 \leftarrow R^2 + G - 2V$
  $Y_3 \leftarrow R(V - X_3) - S_1G$
  $Z_3 \leftarrow Z_1Z_2H$
  **return** $(X_3, Y_3, Z_3)$
```

Algorithm 3 Doubling over $\mathcal{E}_{\mathcal{J}}(\mathbb{F}_p)$.

```
**Input:** $P = (X_1, Y_1, Z_1)$
**Output:** $P + P = (X_2, Y_2, Z_2)$
  $\mathsf{X} \leftarrow X_1^2, \mathsf{Y} \leftarrow Y_1^2, \mathsf{Z} \leftarrow Z_1^2, M \leftarrow 3\mathsf{X} + a\mathsf{Z}^2, T \leftarrow \mathsf{Y}^2, S \leftarrow 4X_1Y$
  $X_2 \leftarrow M^2 - 2S$
  $Y_2 \leftarrow M(S - X_2) - 8T$
  $Z_2 \leftarrow 2Y_1Z_1$
  **return** $(X_2, Y_2, Z_2)$
```

Note that the Montgomery ladder algorithm is typically reflective of the state of the art implementations of ECC secure against SPA. In the following, we further considered implementations protected with scalar randomization in order to avoid DPA attacks. So our focus is on single-trace attacks which naturally goes with our worst-case information extraction motivation.

3 Systematic approach

In this section we describe a systematic method for the worst-case security analysis of scalar multiplications. We first give an abstract view of regular scalar multiplications. We then use this abstraction to specify the amount of informative points in our leakage traces. We finally show how these informative points can be extracted and combined to attack the scalar bits, and show how this method applies on two ECC primitives, namely ECDH and ECDSA.

3.1 Generic scalar multiplication architecture

As explained in Section 2.2, elliptic curve cryptosystems require to compute a scalar multiplication. Section 1 also discussed the regularity requirements of the scalar multiplication implementations, which implies that they can be described as a fixed and predictable sequence of operations. In this context, all operations that affect the internal state depending on the scalar bit value contain sensitive information. In order to quantify this information, we will next describe the scalar multiplication based on different levels, depicted in Figure 2. At the top level, a regular binary scalar multiplication is an iterative processing of the scalar bits. Each bit handling is itself composed of a fixed number of additions and doublings. Then, each addition (resp. doubling) contains a fixed number of field operations (such as field additions, subtractions and multiplications). Finally, each field operation is composed of a fixed number of register operations (such as register additions, subtractions and multiplications). As a result, and for an n-bit scalar, the sequence of register operations that can be exploited by an adversary can be divided into n parts that depend on the scalar bit index. Independently of the kind of operation, we assume that each part contains N register operations. We therefore have that a regular binary scalar multiplication leads to n sequences of N sensitive operations of which the results are stored in registers. We denote as $\mathbf{r}_i = (r_i^j)$, $j \in [0, N-1]$ the N intermediate computation results occurring during the manipulation of the i-th scalar bit.[2] Eventually, each of these computations will lead to side-channel leakages denoted as $\mathbf{l}_i = (l_i^j)$, $j \in [0, N-1]$.

3.2 Information extraction

From the previous abstract view of the scalar multiplication and its associated leakages, the next step is to extract the information. Given a leakage l on a register r, we compute the probability $\Pr[l|r = x], x \in [0, 2^{|r|} - 1]$ that the observed

[2] Note that this is an abstract view. In practice, an operation can have more than one register input/output. In such case, one can count this operation as corresponding to several registers or only use one of them.

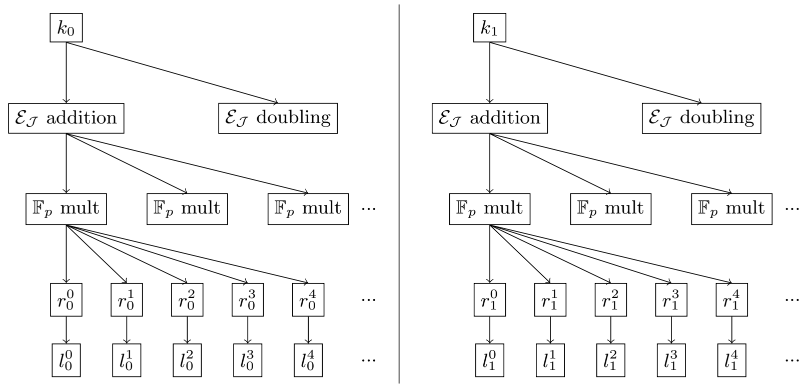

k_0
$\mathcal{E}_{\mathcal{J}}$ addition
$\mathcal{E}_{\mathcal{J}}$ doubling
$\mathbb{F}_p$ mult
$\mathbb{F}_p$ mult
$\mathbb{F}_p$ mult
···
r_0^0 r_0^1 r_0^2 r_0^3 r_0^4 ···
l_0^0 l_0^1 l_0^2 l_0^3 l_0^4 ···
k_1
$\mathcal{E}_{\mathcal{J}}$ addition
$\mathcal{E}_{\mathcal{J}}$ doubling
$\mathbb{F}_p$ mult
$\mathbb{F}_p$ mult
$\mathbb{F}_p$ mult
···
r_1^0 r_1^1 r_1^2 r_1^3 r_1^4 ···
l_1^0 l_1^1 l_1^2 l_1^3 l_1^4 ···

Fig. 2: Leveled view of a regular scalar multiplication. First level (top): scalar bit handling. Second: elliptic curve arithmetic. Third: Field arithmetic. Fourth: register operations. Fifth: leakages on register operations

leakage comes from the manipulation of the value x by r (where $|r|$ denotes the size of the register in bits). While using templates would be optimal, their computational complexity is exponential in $|r|$ as it requires to estimate $2^{|r|}$ Probability Density Functions (PDF) per register (e.g. $2^{|r|}$ means and variances for Gaussian templates). Therefore, and as a more efficient alternative, we use linear regression [39] with a linear basis containing the $|r|$ bits of the registers. The latter decreases the profiling complexity from $\mathrm{O}(2^{|r|})$ to $\mathrm{O}(|r|)$, which is particularly interesting in the case of a $r = 32$-bit architecture (as we will investigate). Yet, this admittedly comes at the cost of a potential information loss in case the leakage function has informative non-linear terms (which we will briefly discuss in Sections 4 and 5).

We denote by $\lceil \mathbf{k} \rceil_i$ the first i bits of $\mathbf{k}$. We denote by $r_i^j(P, \lceil \mathbf{k} \rceil_i)$ the internal value processed by the register operation r_i^j when the input point is P and the first i bits of $\mathbf{k}$ are equal to $\lceil \mathbf{k} \rceil_i$. Similarly, we denote by $l_i^j(P, \lceil \mathbf{k} \rceil_i)$ the leakage corresponding to its manipulation. Using N_{prof} leakages from random inputs (P^q) and random scalars $(\mathbf{k}^q)$, $q \in [0, N_{prof} - 1]$, we compute the $N_{prof} \times 33$ matrix $\mathbf{A}$ where the q-th line is the binary representation of $r_i^j(P^q, \lceil \mathbf{k}^q \rceil_i)$ appended with a constant 1. From $\mathbf{A}$ and the vector $\mathbf{l}_i^j = (l_i^j(P^q, \lceil \mathbf{k}^q \rceil_i))$, $q \in [0, N_{prof} - 1]$ containing the N_{prof} leakages, we compute the linear basis $\mathbf{b}$ that characterizes l_i^j using Equation 1 (where A^T denotes the transpose of $\mathbf{A}$):

$$\mathbf{b} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{l}_i^j. \tag{1}$$

Assuming that the noise follows a Gaussian distribution, the variance σ^2 is computed using a second set of traces (in order to avoid overfitting) as shown by Equation 2. Note that we assume independence between the noise and the value processed by the register [10, 30].

$$\sigma^2 = (\mathbf{l}_i^j - \mathbf{A}\mathbf{b})^T \cdot (\mathbf{l}_i^j - \mathbf{A}\mathbf{b}). \tag{2}$$

From $\mathbf{b}$ and σ^2, the probability that a leakage l_i^j comes from the manipulation of x by r_i^j is then given by Equation 3, where $\mathcal{N}_{\mu,\sigma^2}(\mathrm{l})$ denotes the evaluation in l of the normal density function with mean μ and variance σ^2, and $(x)_2$ denotes the binary decomposition of x appended with 1:

$$\Pr\left[l_i^j \,\middle|\, x\right] = \mathcal{N}_{(x)_2 \cdot b, \sigma^2}(l_i^j). \tag{3}$$

For more detail on the linear regression, we refer to [39].

3.3 Information combination

Now that we are able to identify and extract the information, we aim at combining it to attack the scalar bits. As shown in section 2, HDPA belong to the EP class. In order to exploit the information on a specific bit, an hypothesis on the values of the previous ones must be made.

From the knowledge of input P and the knowledge of the first $i-1$ bits, the likelihood that the bit i is equal to 1 (resp. 0) is computed by guessing all the register values $r_i^j(P, \lceil \mathbf{k} \rceil_{i-1} || 1)$ (resp. $r_i^j(P, \lceil \mathbf{k} \rceil_{i-1} || 0)$ and combining their likelihoods. Given the leakages $\mathbf{l}_i$ on the i-th bit, Equation 4 gives the probability that the leakages come from the first $i-1$ bits being equal to $\lceil \mathbf{k} \rceil_{i-1}$ and the i-th bit being equal to $v \in \{0, 1\}$ (where $||$ denotes the concatenation):

$$\Pr\left[\mathbf{l}_i \,\middle|\, \lceil \mathbf{k} \rceil_i = \lceil \mathbf{k} \rceil_{i-1} || v\right] = \prod_{j=0}^{N-1} \Pr\left[l_i^j \,\middle|\, r_i^j(P, \lceil \mathbf{k} \rceil_{i-1} || v)\right] \cdot \tag{4}$$

This formula assumes that the leakages l_i^j are independent (which simplifies the profiling phase as we only consider univariate samples). While this assumption may not be perfectly correct, we verified experimentally that considering multivariate samples did not improve the efficiency of our attacks in the next section, and therefore assume it to be sufficiently correct for simplicity. Extending this framework to attack k by chunks of d bits is straightforward. Given a d bits hypothesis vector $\mathbf{v}$, Equation 5 extends the previous equation to attack d bits at a time, where $\mathbf{v}_{|\mathbf{j}}$ denotes the j first elements of $\mathbf{v}$:

$$\Pr\left[\mathbf{l}_i, ..., \mathbf{l}_{i+d-1} \,\middle|\, \lceil \mathbf{k} \rceil_{i+d-1} = \lceil \mathbf{k} \rceil_{i-1} || \mathbf{v}\right] = \prod_{j=0}^{d-1} \Pr\left[\mathbf{l}_{i+j} \,\middle|\, \lceil \mathbf{k} \rceil_{i-1} || \mathbf{v}_{|\mathbf{j}}\right] \cdot \tag{5}$$

When there is no ambiguity, we will refer to $\Pr\left[\mathbf{l}_i, ..., \mathbf{l}_{i+d-1} \,\middle|\, \lceil \mathbf{k} \rceil_{i+d-1} = \lceil \mathbf{k} \rceil_{i-1} || \mathbf{v}\right]$ as $\Pr_{\lceil \mathbf{k} \rceil_{i-1} || \mathbf{v}}$ for readability reasons.

3.4 ECDH vs. ECDSA

In the rest of the paper we always assume that $\mathbf{k}$ is attacked by chunks of d bits. For simplicity, we assume that $n_d = \frac{n}{d} \in \mathbb{N}$ and we rewrite $\mathbf{k} = (\mathbf{k}_0, ..., \mathbf{k}_{n_d-1})$ being viewed as binary vectors of d elements.

ECDH. In order to attack ECDH, the attacker has to recover the full scalar in one trace. He starts by attacking the first d bits using Equation 5. Using a leakage from a known input, he computes the likelihoods $\Pr_{\mathbf{v}}$ for all the 2^d scalar hypotheses $\mathbf{v} \in \{0,1\}^d$. He then selects the hypothesis $\mathbf{k}_0^* = \mathsf{argmax}_{\mathbf{v}}(\Pr_{\mathbf{v}})$ that maximizes the likelihood as being the correct guess. The following scalar guesses are selected iteratively according the previous results as $\mathbf{k}_i^* = \mathsf{argmax}_{\mathbf{v}}(\Pr_{\mathbf{k}_0^*||...||\mathbf{k}_{i-1}^*||\mathbf{v}})$, with $\mathbf{v} \in \{0,1\}^d$. The iterative process ends when the adversary has obtained a full scalar hypothesis $\mathbf{k}^* = (\mathbf{k}_0^*, ..., \mathbf{k}_{n_d-1}^*)$.

The attack trivially succeeds if $\mathbf{k}^* = \mathbf{k}$, which corresponds to a 1-O SR. If this is not the case, one can use computational power to mitigate the lack of information by enumerating through the other paths [29]. In this study we only look at the 1-O SR as our focus is on optimal information extraction (rather than exploitation).

Eventually, the chunk size d is a parameter chosen by the adversary. It increases the attack complexity exponentially in d, but allows increasing linearly the amount of leakage samples exploited by the adversary.

ECDSA. As for ECDH, a potential strategy to attack the ECDSA scalar nonce is to recover all its bits. Another option is to use the algebraic relation between the nonce and the secret key. Using lattice techniques, one can attack the secret key by recovering partial on the first d bits of several nonces. This strategy fails if at least one of the nonces' partial information is not recovered properly. As a consequence, the attacker has to make sure that the d-bit partial information of each nonce is correct. As the attack on the first d bits does not suffer from the conditional dependencies, one can turn the previously estimated likelihoods into true probabilities by applying Bayes' theorem, as in Equation 6. From these probabilities, the adversary decides to ignore all the results having a probability lower than a given threshold to maximize the success of the lattice attack.

$$\Pr\left[\mathbf{k}_0 = \mathbf{v} \,\middle|\, \mathbf{l}_0, ..., \mathbf{l}_{d-1}\right] = \frac{\Pr_{\mathbf{v}}}{\sum_{\mathbf{v}^* \in \{0,1\}^d} \Pr_{\mathbf{v}^*}}. \tag{6}$$

To give more insight on the side-channel requirements of a lattice-based attack against ECDSA, Figure 3 shows how many nonces are needed (y axis) in function of the partial information d (x axis). As we can see, the number of required nonces decreases exponentially when d increases. It confirms the need of being able to extract most of the information and to discard the wrong results in a meaningful way. We use the fplll [16] library v4.0.4 with block sizes of 20 and 30 (only for 4 bits leaked case) to perform the experiments.

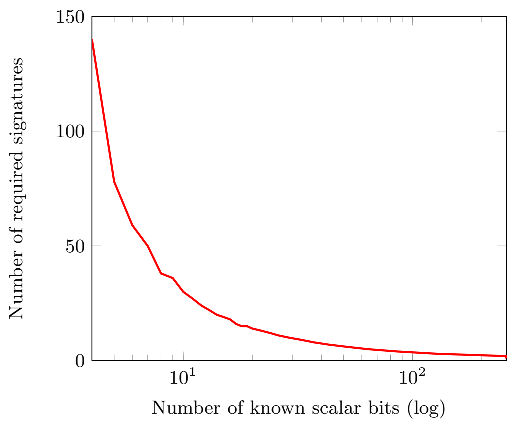


Fig. 3: Complexity of the lattice-based attack against ECDSA. Left: number of nonces (y axis) needed when having d bits of partial information per nonce (x axis).

4 Experimental results on Cortex-M4

In this section we apply previous systematic approach to attack ECC implementations in a 32-bit Cortex-M4 micro-controller. We first describe our implementation, device and the measurement setup. We then follow our evaluation framework step by step and discuss experimental results.

4.1 Target implementation

We implemented the finite field and elliptic curve arithmetic in assembly on both chips. We chose the NIST P-256 curve [38]. Our attack framework is independent of the choice of curve, as its only requirement is the regularity of the implementation. We thus focused on achieving constant time without optimizations. We implemented the Montgomery ladder as described in Section 2.2 using Jacobian coordinates. We used the addition and doubling formulas from Algorithm 2 and 3. The whole scalar multiplication runs in approximately 17,000,000 clock cycles.

Field additions and subtractions are implemented in a straightforward manner using carry additions and subtractions. Field multiplications are done using the Long Integer Multiplication (LIM) followed by a modular reduction. LIM is easily implemented using the 32-bit unsigned multiplication umull and umaal (with accumulate) assembly instructions producing a 64 bits result. Algorithm 4 in Appendix A shows how we perform LIM in assembly from two 256 bits numbers represented as eight 32-bit words $a = (a_0, ..., a_7)$ and $b = (b_0, ..., b_7)$, where a_0 and b_0 denote the least significant words and mod denotes the modular operator. Note that a and b are stored in RAM, and reg_i denotes registers. Using the fact that the field characteristic is a generalized Mersenne prime number, the modular reduction is done as described in [38].

Constant time arithmetic is achieved by always performing a modular reduction for both the addition and the multiplication even if it is not needed. The results before and after reduction are then saved in memory. A Boolean is computed whose value is true or false depending on the need or not of a reduction. According to the Boolean value, the result's pointer is linked to the actual value to be returned. Such a straightforward method to achieve constant time admittedly leads to performance overheads. We emphasize that better performance could have been achieved using algorithmic optimization, while keeping the constant time execution (e.g by using a 2-level Karatsuba multiplication). However, these overheads have no impact on our conclusions.More generally, the proposed attack framework is independent of the implementation as long as it is constant time. In this respect, even optimized implementations still contain a large amount of computations that can be exploited using an HDPA attack. Although they are not directly comparable as they neither use the same micro-controller nor the same curve, different examples of actual implementations can be found in [24].

4.2 Device and setup

Our first target is a 32-bit ARM Cortex-M4 micro-controller from the atmel SAM4C-EK evaluation kit.[3] It proceeds most instructions in constant time and does not have any programs running in parallel that could disturb the signal. Moreover, this micro-controller runs at 100 MHz which makes it a relatively easy target for power acquisition.

We monitored the voltage variation using a 4.7 Ω resistor inserted in the supply circuit of the chip. We performed the trace acquisition using a Lecroy WaveRunner HRO 66 ZI oscilloscope running at 200 megasamples per second. For each scalar multiplication execution, we triggered the measurement at the beginning of the execution and recorded the processing of the first 123 bits of the scalar. Each trace contains 40,000,000 time samples and weighs 320,000,000 mega-bytes (written in double precision format).

Since the device does not suffer from any interruptions, the power traces are directly used without any preprocessing.

4.3 Identifying and extracting the information

Among all the register operations $\mathbf{r}_i$ available for a given bit, we target only the higher 32-bit result of each umull and umaal instructions. This choice allows our attack to remain implementation-independent (since those intermediate results will indeed appear in most implementations). As shown by the doubling and addition formulas in Section 2.2, an addition plus a doubling consist in 25 field multiplications (the curve uses $a = -3$, thus the multiplication by a is done

[3] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf
http://www.atmel.com/tools/SAM4C-EK.aspx

using subtractions). Each field multiplication itself consists in 64 32-bits register multiplications. As a result, we attack the implementation using $N = 25 \times 64 = 1,600$ leakage samples per scalar bit.

In order to efficiently identify the time positions of the corresponding registers r_i^j, we use the unprofiled correlation and partial SNR described below. They exploit a set of $N_{poi} = 8,000$ traces $\mathbf{l}$ acquired using random known inputs (P^q) and scalars $(\mathbf{k}^q)$, $q \in [0, N_{poi} - 1]$. We denote by $\mathbf{l}[t]$ vector of size N_{poi} containing the N_{poi} leakages of the t-th time sample of each trace.

Unprofiled correlation. Given the N_{poi} internal values $\mathsf{r}_i^j = r_i^j(P^q, \lceil \mathbf{k}^q \rceil_i)$ and a leakage model M, we apply the Pearson's correlation ρ on each time sample. That is, we compute $\rho(\mathsf{M}(\mathsf{r}_i^j), \mathbf{l}[t])$, $t \in [0; 40,000,000]$. We used the Hamming weight function for M. The time sample showing the highest correlation is selected as the time sample of r_i^j. The disadvantage of using unprofilied correlation for the POI research is the requirement of a leakage model. However, it allows using the information from the full 32 bits of the internal values r_i^j.

Partial Signal to Noise Ratio (SNR). Computing the standard SNR is not suitable to identify a specific register as any bijective relation between two registers will make them impossible to distinguish. Moreover, a 32-bit SNR would require more than 2^{32} leakages samples in the case of our 32-bit devices. Using partial SNR allows avoiding these issues at the cost of a controlled information lost. For this purpose, the 32-bit values of r_i^j are first truncated to x bits. Each trace is then labeled according to its truncated value and split into 2^x sets $\mathcal{S}_i$. For each time sample, the partial SNR is finally computed as $\frac{\mathsf{var}(\mathsf{mean}(\mathcal{S}_i))}{\mathsf{mean}(\mathsf{var}(\mathcal{S}_i))}$ where var and mean respectively denote the sample variance and mean functions. The time sample showing the highest SNR ratio is selected as the time sample of r_i^j. While partial SNR does not rely on a leakage model, it suffers from algorithmic noise because of the truncation process (which reduces the information exploited). That is, since the remaining $32 - x$ bits are not taken into account when creating the sets $\mathcal{S}_i$, the actual signal is reduced. However, ignoring the value of this remaining bits allows avoiding the bijection issue. In order to illustrate the bijection issue, one can take the AES as an example. Computing the full SNR related to the plaintext m XORed with the key k doesn't allow differentiating whether we are before or after the S-box computation. This means an SNR spike will be exhibited for both $x \oplus k$ and $\mathsf{S}(m \oplus k)$, where S denotes the AES S-box. However, computing a partial SNR on e.g. only 4 bits of $x \oplus k$ instead of the 8 bits will break the bijection between the input and the output of the S-Box. As a result, it will allow discriminating between the input and the output of the S-box in the partial SNR spike.

Optimizations. Applying one of these two methods on the full trace for all r_i^j's is very time consuming as we have $123 \times 1,600$ registers to characterize over 40,000,000 time samples. However, we know that the time order of the register is $(r_0^0, ..., r_0^{N-1}, r_1^0, ...r_i^j, ...r_{n-1}^{N-1})$. Using that knowledge, we can first search r_0^0 among the first W time samples. Using correlation, we select r_0^0's position by computing a p-value with a threshold of 5 [18]. If r_0^0 has not been found, we move the window to the next W time samples and repeat this process until r_0^0 is found. Once this temporal location is found, we search r_0^1 similarly, by setting the initial offset of the window at r_0^0. This process is iterated until all the registers are found. We set the window value W to 20,000, chosen to be slightly higher than $\frac{40{,}000{,}000}{123\times 25}$ (the time samples divided by the number of field multiplications).

Extraction. Once the temporal locations of all the registers are found, we apply linear regression on each of them to extract the information as described in section 3.2, using a set of $N_{prof} = 10,000$ traces. Note that for this simple device, the leakage model was found to be close to Hamming weight and the linear leakage model is not expected to cause a significant information loss. Yet, the formal analysis of this statement with leakage certification tools is an interesting scope for further research [20, 19].

4.4 Information combination

Using the maximum likelihood approach of Section 3.3, we attack the first 123 bits of the Cortex-M4 implementation with the 1-O SR described in Section 3.4. We used a new set of traces that has not been used to identify nor to extract the information for this purpose. As a first experiment, we compute the success rate of recovering the 123 bits using all the information. Secondly, we simulate an implementation with less information by reducing the number N of register per scalar bit. In that case, we study how using computational power by increasing the chunk's size d can mitigate the lack of information. Finally, we study how the number N of informative registers impacts the success rate of the attack.

Our first experiment is to look at the success rate using all the information. In that case we have $N = 1,600$ registers per scalar bit. We achieved a success rate of 0.85 using a chunk's size of 1 bit. It shows that such a device would require much more algorithmic noise to be protected against worst case attacks (e.g. using random delays, shuffling...).

As using all the information allows recovering all the 123 bits with a high success rate, we next simulate a less informative implementation by using $N = 600$. In such a case, increasing the chunk size d is an option to increase success rate. As stated in Section 3.4, the number of points of interest increases linearly with the size of d at the cost of an exponential time complexity increase. Figure 4 shows the impact of the chunk size on the success rate. As we can see, the success rate increases linearly with d. We also see that the slope of the curve is lower than 1. This is explained by the fact that the number of points of interest for the bits of indexes $(0, 1, ..., d-1)$ of each chunk is equal to $(dN, (d-1)N, ..., N)$. That is, only the first chunk's bit fully benefits from increasing d, while the last one does not get any improvement.

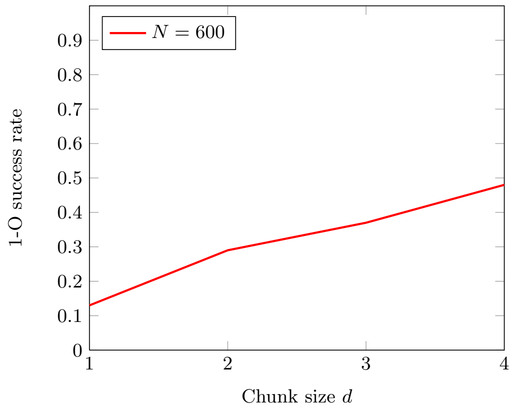


Fig. 4: First-order success rate of the 123-bit recovery on the Cortex-M4 depending on the chunk size d for different $N = 600$.

As a last experiment, we study the impact of the number of target register N on the success rate. Figure 5 shows the evolution of the success rate in function of N for different values of d. Independently of d, the success rate increases exponentially with N.

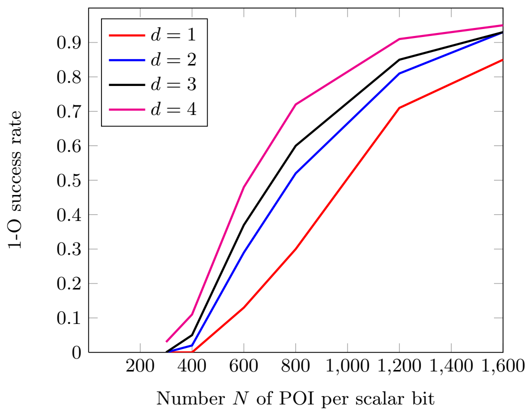


Fig. 5: First-order success rate of the 123-bit recovery on the Cortex-M4 depending on the number of poi N for different values of d.

From these experiments, we conclude that as general in side-channel analysis, the information extraction phase of the attacks is the most critical one, since it is the one causing the exponential security loss. Computational power can then be used as a useful (sometimes necessary) complementary ingredient to mitigate a possible lack of information.

5 Experimental results on Cortex-A8

In this section we show the attack results on the Cortex-A8 micro-controller. As in the previous section, we first describe the device and measurement setup. We then show the application of the framework against this target along with the results. The scalar multiplication is implemented the same way in Jacobian coordinates as described in Section 4.1.

5.1 Device and setup

Our second target is a 32-bit AM335x 1GHz ARM Cortex-A8 linux-based single board computer.[4] As opposed to the previous target, this one is way more challenging [1, 31]. As it is running a full version of Ubuntu 14.04 and more than 100 processes are running in the background while we execute our assembly Montgomery ladder scalar multiplication implementation via SSH from the host PC. The CPU has instruction cache and 13-stage ARM pipeline, all those factors introduce a lot of noise and interruptions. Moreover, the high (1 GHz) frequency also add more obstacles in terms of side channel measurements.

We measured the EM emission using a Langer HV100-27 magnetic near field probe. As in [1, 31], we got the best EM signal when the probe is around the capacitor *C65*. During the measurements, we set the CPU frequency to the highest 1 GHz and the CPU frequency governor to *'Performance'*. We measured the EM traces using a Lecroy WaveRunner 620Zi oscilloscope at a sampling rate of 10 GS/s. For each scalar multiplication execution, we also triggered the measurement at the beginning of the execution and recorded the processing of the first 4 bits of the scalar, so that each trace contains 2,000,000 sample points. As mentioned in [1, 31], the traces contain long interruptions that randomly appear due to the Linux system. We eliminated these interruptions by running the program with the "`nohup`" command via SSH without any elevating techniques. While this technique removes the big interruptions, the traces still contain many smaller ones.

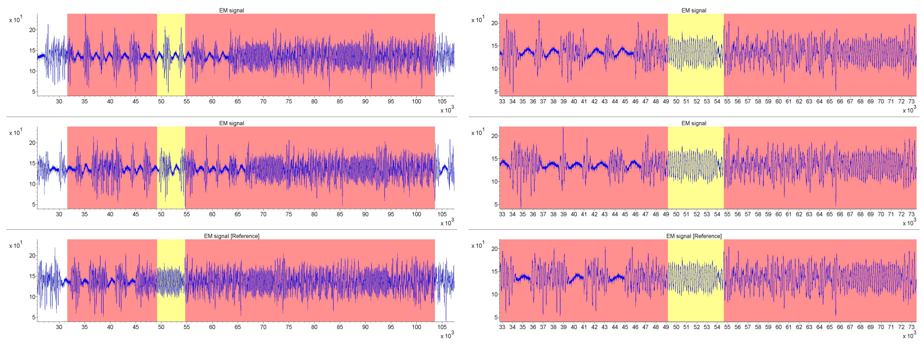
5.2 Preprocessing

In order to deal with the small interruptions, the traces have to be preprocessed. The overall synchronization iterates over three steps. The first one consists in synchronizing the traces around a particular field multiplication. The second step is to cut the traces around the synchronized area into slices. Finally, each slice is added to the set of preprocessed traces by concatenation. These three steps are repeated for each field multiplication.

[4] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344k/DDI0344K_cortex_a8_r3p2_trm.pdf
https://beagleboard.org/black

Synchronization. While the last two steps of the preprocessing are straightforward, the synchronization part deserves more insight. We use a correlation-based alignment method to synchronize the EM traces per field multiplication operation. This method works in three steps that are depicted in Figure 6. The left (resp. right) part of the figure shows the traces before (resp. after) synchronization.

- Firstly, a searching interval A that contains the operation to be synchronized is selected among all the traces. This is shown by the red window.
- Secondly, a second smaller reference interval B_q specific to each trace q is chosen, shown by the yellow window on the three traces in Figure 6(a).
- For each trace, we finally find the portion to be synchronized by using the second window B_q to search over the whole interval A. The right portion is selected as the one having the maximum correlation with the reference interval. If the correlation is lower than a given threshold (arbitrarily chosen by the attacker), the trace is assumed not good enough and is thus discarded.



(a) Before alignment (b) After alignment

Fig. 6: Three EM traces before and after alignment

Once the traces are synchronized, we identify and extract the information the same way as in the previous section on the Cortex-M4 with a different number of traces. We used $N_{poi} = N_{prof} = 100,000$ traces to both identify and extract the information. As will be clear next, a linear leakage model was sufficient to obtain positive results even for this more challenging device. However, as in the previous section, analyzing the quality of this model with leakage certification tools would certainly be worth further investigations.

5.3 Information combination

As for the experiment on the Cortex-M4, we use the maximum likelihood approach of Section 3.3 to attack the first 4 bits of the Cortex-A8 implementation.

This time, we assume we are attacking an ECDSA secret key and use the probabilistic approach of Section 3.4 on ECDSA. We used a new set of 2,200 traces that has not been used to identify nor to extract the information.

Our first experiment simply looks at the 1-O SR. In that case, we recovered the 4 bits with a success rate of 0.8155. As shown by Figure 3, we know that 140 ECDSA nonces are required to recover the secret key with 4 bits of partial information. As no error on the partial information is tolerated, the success rate of the key recovery is equal to $0.8155^{140} \approx 3.9 \cdot 10^{-13}$. This confirms the strong need of a sound way to discriminate the wrong results.

Motivated by this first experiment, we next studied how we can automatically remove the wrong attack results. This is achieved by setting a probability threshold under which some attack traces will be discarded. That is, an attack trace is considered as invalid if the probability given by formula 6 for the most likely partial nonce after the attack is below a given threshold. Intuitively, the higher the threshold is, the more confident we are in having a successful partial nonce recovery, which comes at the cost of increasing the number of discarded attack traces. Table 1 shows how the success rate evolves in function of the probability threshold over the 2,200 attack traces. As we can see, using a threshold of 0.5 does not discard much results and thus does not increase the probability to recover the ECDSA secret key. However, the first-order success rate increases when setting a higher probability threshold. We finally achieve a perfect success rate using a threshold of 0.9999999999. Using this value, 1,958 of the attack results were discarded, thus keeping 242 of them. As only 140 correct scalars are needed to recover the ECDSA secret key, we achieve a success rate of 1.

Threshold	Scalar 1-O SR	Key 1-O SR	# discarded result	# remaining result
0.5	0.8174349612	$3.9 \cdot 10^{-13}$	9	2,191
0.75	0.8462296698	$5.5 \cdot 10^{-13}$	171	2,029
0.9	0.8680042239	$2.5 \cdot 10^{-9}$	306	1,894
0.99	0.9025578563	$5.8 \cdot 10^{-7}$	558	1,642
0.9999	0.956596	0.002	1,025	1,175
0.99999	0.980366	0.062	1,235	965
0.9999999	0.991708	0.312	1,597	613
0.999999999	0.9942363112	0.445	1,853	347
0.9999999999	1	1	1,958	242

Table 1: Evolution of the ECDSA scalar and key recovery success rate in function of the threshold defining “invalid traces”.

6 Conclusion

This paper provides a generic evaluation framework for HDPA against scalar multiplication algorithms, instantiates it with state of the art tools for preprocessing, POI detection, profiling and information extraction, and applies it to concrete implementations reflective of the variety of targets that can be used for embedded cryptographic applications. In view of the limited experimental reports available on the topic, we hope these implementation and systematization efforts can be used to clarify the practicality of such advanced attacks in practice, even against challenging targets running at high clock frequencies, and therefore argue for their integration as a part of current certification practices.

From a designer's point of view, our results also highlight that implementations of scalar multiplications on commercial platforms with scalar randomization activated are generally at risk, in view of the huge amount of informative samples such implementations provide. Straightforward solutions to improve this situation include performance optimizations (since implementations with less cycles inevitably leak less to the horizontal adversary) and the addition of noise. Yet, our analysis of an ARM Cortex-A8 running at 1 GHz (a presumably noisy target) suggests that this may not be enough. In this respect, the systematic implementation of point randomization seems strictly necessary to reach high security levels, the evaluation of which (with a systematic evaluation framework as we described in this paper) is an interesting scope for further research, in order to better assess its worst-case security.

Acknowledgements. François-Xavier Standaert is a research associate of the Belgian Fund for Scientific Research. This work has been funded in parts by the European Commission through the H2020 project 731591 (acronym REASSURE) and the ERC project 724725 (acronym SWORD). Yuanyuan Zhou would like to thank Brightsight management board for the support and his colleagues for fruitful discussions.

References

1. Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. Dpa, bitslicing and masking at 1 ghz. In *CHES 2015*, pages 599–619, 2015.
2. Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 21–36, 2014.
3. Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and vertical side-channel attacks against secure RSA implementations. In *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, pages 1–17, 2013.
4. Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal collision correlation attack on elliptic curves. In *Selected Areas in Cryptography -*

SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers, pages 553–570, 2013.
5. Naomi Benger, Joop van de Pol, Nigel P. Smart, and Yuval Yarom. "ooh aah... just a little bit" : A small amount of side channel can go a long way. In *CHES 2014*, pages 75–92, 2014.
6. Daniel J. Bernstein, Tanja Lange, and Christine van Vredendaal. Tighter, faster, simpler side-channel security evaluations beyond computing power. *IACR Cryptology ePrint Archive*, 2015:221, 2015.
7. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *CHES 2002*, pages 16–29, 2004.
8. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES 2002*, pages 13–28, 2002.
9. Chien-Ning Chen. Memory address side-channel analysis on exponentiation. In *Information Security and Cryptology - ICISC 2014 - 17th International Conference, Seoul, Korea, December 3-5, 2014, Revised Selected Papers*, pages 421–432, 2014.
10. Omar Choudary and Markus G. Kuhn. Efficient template attacks. In *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, pages 253–270, 2013.
11. Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylène Roussellet, and Vincent Verneuil. ROSETTA for single trace analysis. In *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, pages 140–155, 2012.
12. Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In *Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings*, pages 46–61, 2010.
13. Christophe Clavier and Marc Joye. Universal exponentiation algorithm. In *CHES 2001*, number Generators, pages 300–308, 2001.
14. Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES 1999*, pages 292–302, 1999.
15. Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. Improving the big mac attack on elliptic curve cryptography. In *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, pages 374–386, 2016.
16. The FPLLL development team. fplll, a lattice reduction library. Available at `https://github.com/fplll/fplll`, 2016.
17. Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, and Sylvain Guilley. Dismantling real-world ECC with horizontal and vertical template attacks. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*, pages 88–108, 2016.
18. François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In *EUROCRYPT 2016*, pages 240–262, 2016.
19. François Durvaux, François-Xavier Standaert, and Santos Merino Del Pozo. Towards easy leakage certification: extended version. *J. Cryptographic Engineering*, 7(2):129–147, 2017.
20. François Durvaux, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. How to certify the leakage of a chip? In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 459–476. Springer, 2014.

21. Cezary Glowacz, Vincent Grosso, Romain Poussier, Joachim Schüth, and François-Xavier Standaert. Simpler and more efficient rank estimation for side-channel security assessment. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 117–129, 2015.
22. Neil Hanley, HeeSeok Kim, and Michael Tunstall. Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, pages 431–448, 2015.
23. Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, and Georg Sigl. Localized electromagnetic analysis of cryptographic implementations. In *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, pages 231–244, 2012.
24. Michael Hutter and Peter Schwabe. NaCl on 8-bit AVR microcontrollers. In *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*, pages 156–172, 2013.
25. Marc Joye and Christophe Tymen. Protections against differential analysis for elliptic curve cryptography. In *CHES 2001*, number Generators, pages 377–390, 2001.
26. Marc Joye and Sung-Ming Yen. The montgomery powering ladder. In *CHES 2002*, pages 291–302, 2002.
27. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996.
28. Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.
29. Tanja Lange, Christine van Vredendaal, and Marnix Wakker. Kangaroos in side-channel attacks. In *CARDIS 2014*, pages 104–121, 2014.
30. Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, pages 20–33, 2015.
31. Jake Longo, Elke De Mulder, Dan Page, and Michael Tunstall. Soc it to EM: electromagnetic side-channel attacks on a complex system-on-chip. In *CHES 2015*, pages 620–640, 2015.
32. Daniel P. Martin, Jonathan F. O'Connell, Elisabeth Oswald, and Martijn Stam. Counting keys in parallel after a side channel attack. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 313–337, 2015.
33. Marcel Medwed and Elisabeth Oswald. Template attacks on ECDSA. In *Information Security Applications, 9th International Workshop, WISA 2008, Jeju Island, Korea, September 23-25, 2008, Revised Selected Papers*, pages 14–27, 2008.
34. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In *CHES 1999*, pages 144–157, 1999.

35. Erick Nascimento, Lukasz Chmielewski, David Oswald, and Peter Schwabe. Attacking embedded ECC implementations through cmov side channels. *IACR Cryptology ePrint Archive*, 2016:923, 2016.
36. Phong Q. Nguyen and Igor E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Des. Codes Cryptography*, 30(2):201–217, 2003.
37. Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *CHES 2016*, pages 61–81, 2016.
38. NIST FIPS PUB. 186-2: Digital signature standard (dss). *National Institute for Standards and Technology*, 2000.
39. Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *CHES 2005*, pages 30–46, 2005.
40. Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, pages 390–406, 2012.
41. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 126–141. Springer, 2013.
42. Colin D. Walter. Sliding windows succumbs to big mac attack. In *CHES 2001*, number Generators, pages 286–299, 2001.

A Assembly long integer multiplication

Algorithm 4 Assembly long integer multiplication.

```
Input: $a = (a_0, ..., a_7), b = (b_0, ..., b_7)$
Output: $c = a \times b = (c_0, ..., c_{15})$
  $c \leftarrow 0$
  $\mathrm{reg}_2 \leftarrow \mathsf{load}(a_0)$
  $\mathrm{reg}_3 \leftarrow \mathsf{load}(b_0)$
  $\mathrm{reg}_0, \mathrm{reg}_1 \leftarrow \mathsf{umull}(\mathrm{reg}_2, \mathrm{reg}_3)$
  $c_0 \leftarrow \mathsf{store}(\mathrm{reg}_0)$
  for $i = 1$ to $7$ do
    $\mathrm{reg}_3 \leftarrow \mathsf{load}(b_i)$
    $\mathrm{reg}_{1-i\mathsf{mod2}} \leftarrow 0$
    $\mathrm{reg}_{i\mathsf{mod2}}, \mathrm{reg}_{1-i\mathsf{mod2}} \leftarrow \mathsf{umaal}(r_2, r_3)$
    $c_i \leftarrow \mathsf{store}(\mathrm{reg}_{i\mathsf{mod2}})$
  end for
  $c_8 \leftarrow \mathsf{store}(\mathrm{reg}_0)$
  for $i = 1$ to $7$ do
    $\mathrm{reg}_2 \leftarrow \mathsf{load}(a_i)$
    $\mathrm{reg}_3 \leftarrow \mathsf{load}(b_0)$
    $\mathrm{reg}_0 \leftarrow 0$
    $\mathrm{reg}_1 \leftarrow 0$
    $\mathrm{reg}_0, \mathrm{reg}_1 \leftarrow \mathsf{umull}(\mathrm{reg}_2, \mathrm{reg}_3)$
    $\mathrm{reg}_3 \leftarrow \mathsf{load}(c_i)$
    $\mathrm{reg}_0 \leftarrow \mathsf{adds}(\mathrm{reg}_0, r_3)$
    $c_i \leftarrow \mathrm{store}(\mathrm{reg}_0)$
    for $j = 1$ to $7$ do
      $\mathrm{reg}_3 \leftarrow \mathsf{load}(b_j)$
      $\mathrm{reg}_{1-j\mathsf{mod2}} \leftarrow 0$
      $\mathrm{reg}_{j\mathsf{mod2}}, \mathrm{reg}_{1-j\mathsf{mod2}} \leftarrow \mathsf{umaal}(r_2, r_3)$
      $\mathrm{reg}_3 \leftarrow \mathsf{load}(c_{i+j})$
      $\mathrm{reg}_{j\mathsf{mod2}} \leftarrow \mathsf{adcs}(\mathrm{reg}_{j\mathsf{mod2}}, \mathrm{reg}_3)$
      $c_{i+j} \leftarrow \mathrm{store}(\mathrm{reg}_{j\mathsf{mod2}})$
    end for
    $\mathrm{reg}_0 \leftarrow \mathsf{adcs}(\mathrm{reg}_0, 0)$
    $c_{8+i} \leftarrow \mathsf{store}(\mathrm{reg}_0)$
  end for
  return $c$
```