Applying Horizontal Clustering Side-Channel Attacks on Embedded ECC Implementations (Extended Version)

Erick Nascimento[1]⋆ and Łukasz Chmielewski[2]

[1] Institute of Computing, University of Campinas, Campinas, Brazil
`enascimento.pub@gmail.com`

[2] Riscure BV, Delft, The Netherlands
`chmielewski@riscure.com`

Abstract. Side-channel attacks are a threat to cryptographic algorithms running on embedded devices. Public-key cryptosystems, including elliptic curve cryptography (ECC), are particularly vulnerable because their private keys are usually long-term. Well known countermeasures like regularity, projective coordinates and scalar randomization, among others, are used to harden implementations against common side-channel attacks like DPA.

Horizontal clustering attacks can theoretically overcome these countermeasures by attacking individual side-channel traces. In practice horizontal attacks have been applied to overcome protected ECC implementations on FPGAs. However, it has not been known yet whether such attacks can be applied to protected implementations working on embedded devices, especially in a non-profiled setting.

In this paper we mount non-profiled horizontal clustering attacks on two protected implementations of the Montgomery Ladder on Curve25519 available in the μNaCl library targeting electromagnetic (EM) emanations. The first implementation performs the conditional swap (cswap) operation through arithmetic of field elements (cswap-arith), while the second does so by swapping the pointers (cswap-pointer). They run on a 32-bit ARM Cortex-M4F core.

Our best attack has success rates of 97.64% and 99.60% for cswap-arith and cswap-pointer, respectively. This means that at most 6 and 2 bits are incorrectly recovered, and therefore, a subsequent brute-force can fix them in reasonable time. Furthermore, our horizontal clustering framework used for the aforementioned attacks can be applied against other protected implementations.

Keywords: ECC, EM analysis, ARM, horizontal clustering

1 Introduction

Public-key cryptosystems based on ECC [28,23] are frequently used in a wide range of applications, such as: credit card, e-commerce and cryptocurrency. Running on embedded systems, they are a common target of side-channel attacks.

⋆ This work was partially done by the author in a research internship at Riscure BV.

The main goal of these attacks is to recover the private key, which is typically the scalar in a scalar multiplication – the main ECC operation in most protocols.

Horizontal attack (HA) is a methodology for side-channel attacks against basic cryptographic operations in protocols based on RSA or ECC, the modular exponentiation and the scalar multiplication (ECSM), respectively. In theory, a horizontal attack against ECC allows the recovery of secret scalar bits through the analysis of individual traces, i.e., a single trace from the actual target is sufficient; thus, they are effective against implementations protected by classic and popular countermeasures such as scalar randomization (SR), coordinate randomization (CR), point blinding and scalar splitting. A fundamental requirement for an attacker to apply HA is the knowledge of the scalar multiplication algorithm; implementation details, however, are not required. In addition, HA requires to have a good comparison tool, thereafter referred to as a *distinguisher*, to efficiently extract parts of the keys. The following methods can be applied, among others: *correlation, collision-correlation, cross-correlation* and *cluster analysis.*

The correlation analysis method [5] follows the same principle as correlation power analysis (CPA) applied to a set of traces arranged vertically. The difference in the horizontal context is that a single trace is divided in several segments and a hypothetical intermediate value is assigned to each segment, based on a guess about the key value. The correlation between the segment samples and hypothetical values is computed in the same way as in CPA. This method works against implementations protected only with scalar randomization, or when coordinate randomization is applied with a short random parameter. The method of collision-correlation analysis [2,1,4,41,39] computes the correlation or Euclidean distance between segments of a trace. The goal is to identify the occurrence of the same intermediate data in different parts of the trace, and by doing so derive the secret bits. In theory, this method is feasible against the classic countermeasures.

Many side-channel attacks do not work when a stronger version of coordinate randomization is used, the so-called coordinate re-randomization (CRR) [30]. This countermeasure randomizes the working points coordinates at every ECSM iteration. Therefore, the correlation or collision-correlation attacks are prevented, because they rely on the fact that output points of an iteration are equal to the input points of the next iteration. On the other hand, attacks that target iterations independently, like the attacks presented in this paper, are not influenced by this countermeasure. The implementations attacked in this paper are protected with all the aforementioned countermeasures.

Most horizontal attacks require advanced preprocessing of traces, characterization and leakage assessment before applying distinguishers. The main challenge of the horizontal approach revolves around extracting meaningful leakage from a single trace, which usually has strong noise. In this paper we consider the non-profiled scenario (also called unsupervised) in which the adversary does not know and cannot change the private key in any test device. Moreover, she is not allowed to turn off countermeasures[3]. Therefore, the second major challenge

[3] Turning off the countermeasures is not always possible in the ICC EMVCo smart card evaluations [9], for example.

is caused by the unavailability of labeled samples. Note that leakage assessment methods, like TVLA [12], require labeled samples and this is not possible when scalar randomization is enforced.

Related work. Unsupervised learning methods, especially those based on clustering, have been applied to solve the aforementioned limitations and they have been shown to be able to work in practice.

Heyszl et al [14] apply multi-dimensional K-Means [11,25] clustering to successfully attack an FPGA-based ECC implementation by correctly classifying the scalar bits. Sprecht et al. [37] later improved this attack by using Expectation-Maximization clustering [7], Principal Component Analysis (PCA) [21] and multiple EM probes. Both methods target ECC implementations for FPGAs and work well for low noise measurements. In this paper we do not employ dimensionality reduction techniques such as PCA (unsupervised) or LDA [10] (supervised), but instead we apply a points of interest selection method.

Perin et al [35], consider a heuristic approach based on unidimensional difference of means for points of interest selection. This method uses a single trace for the leakage assessment, which is likely affected by noise. Perin and Chmielewski [34] propose a methodology for clustering attacks to amend the aforementioned deficiency by using multiple unlabeled traces for leakage assessment and improving attack robustness in high noise scenarios. Similarly to the above works we use unidimensional clustering for points of interest selection; however, for the attack, we evaluate various clustering methods including the multi-dimensional one.

Jarvinen et al [20] present an unsupervised clustering attack on m-ary ECSM with precomputations[4]. The proposed attack is evaluated using a low noise 8-bit AVR device. While our attacks target binary ECSM, they can be straightforwardly extended to the m-ary case: instead of using binary clustering, the attack would need to employ 2^m clusters. Most clustering algorithms support an arbitrary number of clusters, e.g. K-Means.

Another related work concerns error correction. In [14], to derive the error locations the authors use a probability for cluster belonging derived from the K-Means results. Essentially, scalar indexes with the lowest probability are brute-forced. Similarly, the papers [34,35] use probability density function for various clustering algorithms to perform error correction.

We have applied the approach from [34] to detect the errors in the recovered scalar. Unfortunately, this approach does not work for our experiments because of a presence of strong noise pulses in the EM traces. Essentially, some bit errors occur even if their probabilities of being correct are high. Therefore, we have abandoned this approach and applied a brute-force method sped up by the time-memory trade-off algorithm from [30].

Contributions. The main contributions of this paper are summarized below. First of all, using EM we perform a horizontal clustering attack (HCA) against

[4] In an m-ary method, m bits of the scalar are processed in one iteration of ECSM while in a standard ECSM a single bit is processed per iteration.

the arithmetic-based cswap.[5] Curve25519 μNaCl Montgomery Ladder running on a 32-bit ARM Cortex-M4F. The implementation is additionally protected with projective coordinate re-randomization and scalar randomization. We compare a wide range of leakage assessment methods and statistical classifiers to find out the best settings and we achieve the best success rate of 97.64%[6]. This means there are at most 6 erroneous bits, which can be brute-forced in a reasonable time even without knowing error locations.

Secondly, we attack the pointer-based cswap μNaCl Montgomery Ladder implementation that is protected the same as for the first one. Our best attack on this implementation has a success rate of 99.60%, i.e., only 2 errors.

Note that in our non-profiled approach, choosing the best settings implies running the attack for all the considered parameters. This would significantly increase the attack time. To partially mitigate this issue, we use the same settings for both implementations. Moreover, the attack can be easily parallelized, for example, by using cloud computing; significantly improved results justify this.

Thirdly, we improve the unsupervised RSA HCA framework from [34]. This framework implements the leakage assessment by combining multiple RSA traces protected with exponent blinding. We extend that framework to ECC by attacking a randomized scalar instead of a blinded exponent. In addition, we propose the usage of: (i) multiple dimensions clustering; (ii) methods for outlier detection; and (iii) intrinsic quality evaluation of clusters.

Finally, we generalize the method from [30] to tolerate a certain number of incorrectly recovered scalar bits without relying on confidence probabilities.

Our attacks demonstrate the feasibility of scalar recovery from the μNaCl-based ECSM. Breaking ECSM implies that an attacker can compromise the key exchange protocols: Elliptic Curve Diffie-Hellman (ECDH) and its ephemeral version (ECDHE). Examples of current publicly known applications using μNaCl on ARM Cortex-M devices, and thus potentially vulnerable, include: [8,40,36].

Paper organization. The remainder of this paper is structured as follows. In Sec. 2, we describe the setup of the attacks. Subsequently, Sec. 3 covers preliminaries and Sec. 4 presents our horizontal cluster framework. The experimental results are shown in Sec. 5. We describe how to efficiently correct errors in Sec. 6. Finally, Sec. 7 discusses countermeasures and future work.

2 Attack setup

2.1 Target software implementations

We target μNaCl[7], a cryptographic library for ARM Cortex-M that provides implementations of Curve25519, an elliptic curve at the 128-bit security level

[5] Cswap means conditional swap. In a Montgomery ladder ECSM, the cswap condition value tells whether or not to swap and it depends on the secret scalar bit. Thus, it should ideally be constant time and not leak through other side channels.

[6] We use the term *success rate* to refer to the percentage of correctly recovered bits.

[7] http://munacl.cryptojedi.org/curve25519-cortexm0.shtml

Algorithm 1 Montgomery ladder with cswap and coordinate re-randomization.

```
// ... initialization omitted ..
$bprev \leftarrow 0$
for $i = 254 \ldots 0$ do
    RE_RANDOMIZE_COORDS($work\_state$)
    $b \leftarrow$ bit $i$ of scalar
    $s \leftarrow b \oplus bprev$
    $bprev \leftarrow b$
    CSWAP($work\_state$, $s$)
    LADDERSTEP($work\_state$)
end for
// ... return ommited ..
```

and its associated X25519 key exchange protocol based on Diffie-Hellman. This library provides two ECSM implementations, both based on the Montgomery ladder algorithm (cf. Appendix A for details). They differ on how the conditional swap (cswap) operation, fundamental to implement it in constant time, is performed: either by arithmetic means (cswap-arith) or pointers swapping (cswap-pointer).

At each algorithm iteration, the cswap condition depends on the secret scalar bit processed at that iteration and thus its value should not leak. We argue that in both implementations the cswap condition value leaks. We investigate and confirm that the leakage is strong enough to be exploited by our proposed attacks.

In the cswap-arith implementation, the if/else branch is replaced by conditional swaps of the respective coordinate values of the working points, $P_1 = (X_1, Z_1)$ and $P_2 = (X_2, Z_2)$, to achieve constant time. A high level description of such strategy is described in Alg. 1. Another cswap implementation performs a conditional swap of pointers to the field elements instead (cswap-pointer)[8]. In the latter implementation, during each ECSM iteration, the mask is touched far fewer times by the `AND` (`&`) instruction (3 times) than in the cswap-arith (16 times); thus, in theory, a weaker side-channel leakage is expected.

The ECSM implementations in μNaCl do not provide countermeasures against power/EM analysis, besides a regular and constant-time implementation. To evaluate our proposed attacks against properly protected targets, we added coordinate re-randomization to both implementations [9]. The re-randomization countermeasure multiplies a randomly generated $\lambda \in \mathbb{F}_p$ with the coordinates of P_1 and P_2 at the beginning of every ECSM iteration (Alg. 1).

2.2 Target device and measurement setup

The target software runs on the STM32F4 microcontroller chip on the board, with a 32-bit ARM-M4 CPU core, clocked at 168 MHz. We acquired electromagnetic (EM) traces from the ECSM execution by the target device, using a single

[8] Selected by preprocessor definition `DH_SWAP_BY_POINTERS`.

[9] Our attack also works against implementations protected with scalar randomization. We have not implemented this countermeasure, but instead we set a random scalar for each ECSM execution.

EM probe. The setup consisted of a Lecroy Waverunner 8254M oscilloscope, a Langer RF-U 2.5-2 H-field probe, an amplifier and analog low pass filter (250 MHz).

For the acquisition of each trace, the host PC sends to the target device a pair of scalar (k) and input point (P), both randomly generated. The device receives the pair and executes the scalar multiplication, returning the output point ($R = [k]P$) to the host PC. We have acquired the traces with the following settings: 2.5 GS/s sample rate, 16 mV amplitude and 70 million samples. We have also used a low pass BNC analog filter: BLP-250+ from Mini-Circuits. We acquired and analyzed traces using Riscure's Inspector software package. [10]

3 Preliminaries

3.1 Traces Characterization

The n-th measured side-channel trace, which represents the electromagnetic emanation (EM) of a device over the time domain, is denoted by the uni-dimensional ($1 \times aL$) vector $\mathbf{t}^n = \{O_1^n, O_2^n, ..., O_{aL}^n\}$. Here, we consider a trace $\mathbf{t}^n$ as being the side-channel information of an ECSM composed by a fixed number aL of iterations. The factor a depends on the ECSM algorithm and L is the bit-length of the scalar; for Montgomery Ladder on Curve25519, $a = 1$ and $L = 255$. The trace $\mathbf{t}^n$ can be described by a set of ℓ-sized sub-vectors:

$$\mathbf{t}^n = \{O_1^n, O_2^n, ..., O_{a.L}^n\} = \left\{(t_{1,1}^n, ..., t_{1,\ell}^n), (t_{2,1}^n, ..., t_{2,\ell}^n), ..., (t_{a.L,1}^n, ..., t_{a.L,\ell}^n)\right\}$$

where $t_{i,j}^n$ is the j-th element of each sub-vector O_i^n and ℓ is the number of samples. The element $t_{i,j}^n$ can be viewed as a sample in time from the side-channel trace $\mathbf{t}^n$. The set $\{t_{i,j}^n\}$, $i = 1..aL$, refers to a set of samples where each element $t_{i,j}^n$ is extracted from one ECSM iteration O_i^n for a fixed j. For example, $\{t_{i,10}^n\}$ contains aL samples, each element $t_{i,10}^n$ is selected from the 10^{th} sample of each sub-trace O_i).

A *traceset* is defined as the set of trace segments of one or more ECSM runs, and each trace segment consists of the samples from a single ECSM iteration. A *full traceset* is a set of traces of multiple ECSM runs, where each trace in the set consists of the samples from a full ECSM run, i.e., it is a contiguous trace containing all the samples from all iterations of that ECSM run. The tracesets are assumed to be unlabeled, except in those traces used for known-key analysis.

Note that to analyze an ECSM trace we need to cut it into pieces that represent single ECSM iterations. Subsequently, we need to align these traces to be able to efficiently identify and exploit the leakage. The process is done in a similar way to [34] and is explained in more detail in Appendix B.

[10] http://www.riscure.com/

3.2 Clustering

Clustering algorithms. The clustering algorithms successfully employed so far in the context of horizontal attacks are K-Means (KM), Fuzzy K-Means (FKM) and Expectation-Maximization (EM) [14,35,34,20]. K-Means is a *rigid* clustering algorithm, meaning that each instance (a sample in the context of HCA) is assigned (labeled) to a single cluster. On the other hand, Fuzzy K-Means and EM are *soft* clustering algorithms, because their output includes an association probability matrix, where each instance is associated with its degree of linkage to each cluster.

Intrinsic cluster quality measure. Given a set of clustering outputs from multiple clustering algorithm runs, an intrinsic cluster quality measure can be applied to evaluate the best among them. Such measures are called intrinsic or internal because they consider just the structure of the clusters, and do not take into account any labeled information that might be available and could be used for testing. A clustering result with the best intrinsic cluster quality measure does not guarantee the best results for the application (in this work, for use by cluster leakage assessment and horizontal cluster analysis). But, it is nevertheless useful when clustering results cannot be otherwise tested.

Several intrinsic quality measures have been proposed for unsupervised clustering, among them: Silhouette coefficient [22], Calinski-Harabaz index [3] and Davies-Bouldin (DB) index [6]. We chose to use the DB index, which is based on the ratio of within-cluster and between-cluster distances, and can be defined as:

$$DB = \frac{1}{k}\sum_{i=1}^{k} max_{j \neq i}\{D_{i,j}\} \qquad (1) \qquad\qquad D_{i,j} = \frac{\bar{d_i} + \bar{d_j}}{d_{i,j}} \qquad (2)$$

where $D_{i,j}$ is the within-to-between cluster distance ratio for clusters i and j; $\bar{d_c}$ is the mean distance between the centroid of cluster c and each point in that cluster; and $d_{i,j}$ is the Euclidean distance between the centroids of clusters i and j. The smaller the DB index, the better is the clustering.

The DB index favors clusters that are compact and distant from each other. These are exactly the properties we expect to get at points in time where the clusters provide a good separation of the classes (two classes, one for each possible value of the swap bit b in Alg. 1). We applied the Davies-Bouldin index measure to the clustering outputs of multiple runs of the same randomized clustering algorithm, each run with a different RNG seed, and selected the clustering output with the best (i.e., smallest) index value.

3.3 Outlier detection and handling

According to Hawkings [13], "*An outlier is an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.*" In the HCA context, outliers can appear in the measured samples due to, e.g., measurement errors or unknown device behavior, and have a significant impact on clustering. Most clustering algorithms are not intrinsically robust to outliers, so depending on how large an outlier value deviates from "normal" values, the resulting labels might be negatively influenced by the outlier. Then the resultant clusters might be completely different, and thus wrong, from what would be expected, leading to potentially misleading results. Hence, outlier detection is desirable as a preprocessing step before clustering in HCA.

We implemented and tested the following outlier detection methods for HCA: distance from mean and Tukey's test. A simple outlier detection method, hereafter called *distance from mean*, is given by considering the values that are far from the mean as outliers, i.e., a value x is an outlier if $|x - \mu| \geq \beta\sigma$, for a non-negative parameter β; μ and σ are the mean and standard deviation, respectively. We chose $\beta = 2.0$.[11] Tukey's range test [33] is a method based on order statistics. If Q_1 and Q_3 are the lower and upper quartile, respectively, and $IQ = Q_3 - Q_1$ is the interquartile, any observation outside the closed interval $[Q_1 - k \cdot IQ, Q_3 + k \cdot IQ]$ is considered an outlier, for a non-negative parameter k. We chose $k = 1.5$, the value proposed in [33].

If an outlier detector flags some samples as outliers, they must be dealt with in some way, i.e., the outliers have to be handled. Outlier handling methods are usually heuristic and dependent of the context where they are applied [32]. A simple outlier handling method that could be applied in the context of this work is to simply exclude the data point from consideration. Albeit simple, the implementation of this method is potentially inefficient in the HCA context, due to the need of more complex data structures (e.g., dynamic lists rather than static arrays). To keep the implementation simple and efficient, we replace outliers values by the median of non outliers.

4 Horizontal Cluster Analysis Framework

The horizontal attack described in this paper roughly follows the HCA framework from [34], with contributed analysis methods. Fig. 1 shows the steps in the HCA framework. The first step is to run clustering leakage assessment (CLA). CLA takes as input iteration traces from multiple ECSM runs and finds points in the traces where the leakage most likely is, known as points of interest (POIs). Next, key recovery (KR) is run, yielding an approximate scalar. Then, given the approximate scalar, points-of-interest optimization (POI-OPT) produces a refined list of POIs. Finally, the final KR step outputs the recovered scalar.

Fig. 2 shows the full key recovery process in more detail, including the inputs and outputs at each step. The inputs are a traceset to be used for leakage assessment and the actual target traceset. The output is the correct recovered key/scalar, if it could be found.

The "KR final" step takes as input a traceset with traces from multiple ECSM runs and attacks the sets of segment traces of each ECSM run independently from one another. This step is a probabilistic algorithm that consists of sequentially running KR and error correction (Sec. 6) on each trace in the set, and recovering

[11] Assuming that the sample values at a given index come from a normal distribution, choosing β=2.0 implies that 95% of the values are within the interval $[x - \mu, x + \mu]$.

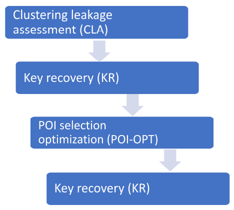
Clustering leakage assessment (CLA)
Key recovery (KR)
POI selection optimization (POI-OPT)
Key recovery (KR)

Fig. 1: HCA framework.

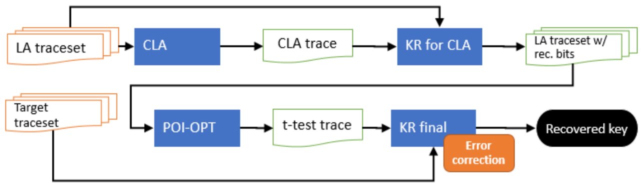

Fig. 2: The full key recovery process.

the correct scalar for at least one of these traces, with a given probability of success. We call this probability the *success rate* of the attack.

In our HCA framework, POIs are chosen from a leakage assessment trace, be it a CLA or t-test trace. They are selected as the time indices of the top m highest peaks in such traces, where m is a parameter. Suitable values for m are derived experimentally (cf. Sec. 5 and Fig. 4).

4.1 Cluster Leakage Assessment (CLA)

Leakage assessment (LA) methods are used to determine whether a cryptographic device leaks information through a side-channel and how strong such leakage is. They are typically employed to find out the points in time (sample indices) where the leakage is strongest, i.e., the points of interest. The sample values at those points are used in later steps, e.g. in the key recovery phase, and they serve two major purposes: (i) for dimensionality reduction, i.e., to use only the samples that provide useful information and thus reduce computation time; (ii) to avoid bringing noisy samples to the attack phase (i.e., key recovery), where they will negatively impact the success rate and potentially turn unfeasible an otherwise successful attack.

In the HCA and non-profiled attack contexts, leakage assessment methods should not require knowledge of the secret key or ephemeral secret data (e.g., numbers randomly generated by the device). Essentially, these methods assume that the adversary does not have control over device's secret information. In particular, the countermeasures like SR and CRR cannot be disabled. Additionally, it is desirable that LA methods be non-parametric and do not require leakage models. That is because in the single-trace HCA attack context the target device is not known a priori. Combined with the fact that real-world modern microcontrollers are complex devices, it means that the estimation of leakage distributions and thus building an accurate leakage model is not trivial.

Distinguishers. Welch's t-test is a parametric statistical test that can be employed to this end; e.g., in methodologies like the TVLA [12]. Certain conditions have to be met for Welch's t-test to be used: normality of the distributions, equal variances and independence. The Mutual Information Analysis (MIA) is a distinguisher that does not require a leakage model. Standaert et al [38] were the first to propose the use of MI as an statistical leakage assessment tool in the SCA context. Meynard et al [27] applied MIA as a method to locate strong leakage in the frequency domain and, consequently, to find the frequency bands in EM traces where the differences between modular squares and multiplications are highest, from a device running RSA modular exponentiation. Mather et al [26] compared the statistical power of t-test and the discrete and continuous versions of MIA for detecting leakage in multiple leakage models. Following [34], we provide results for four different methods for leakage detection: sum-of-squared differences (SOSD), sum-of-squared t-values (SOST) and MIA. Details about these techniques are described in Appendix D.

CLA. The LA method proposed in [34] shows how multiple traces can be combined utilizing clustering, an unsupervised learning method, and demonstrate it through an attack on a RSA software implementation. That method, hereafter called clustering-based leakage assessment (CLA), in principle works even if the device applies any combination of the classic countermeasures for modular exponentiation, i.e., exponent blinding, message or modulus randomization. Additionally, we implemented a supervised LA method in our framework, so called HCA-KKA, to be able to compare the unsupervised method results to the supervised ones. The description of HCA-KKA is in Appendix C.

4.2 Key Recovery (HCA-KR)

The key recovery methods implemented in this work can be classified into two classes, based on the way clustering is applied: single or multi-dimensional. In either case, the number of clusters output by a clustering algorithm is two, one cluster for each possible key bit value.

Single-dimensional clustering method. In the first group, we run clustering on the set of samples at a given single time index (POI), across multiple trace segments. This is the approach used by [34]. After running the clustering for every time index, a set of recovered key candidates is obtained, which are then combined to decide the final key candidate. The following combination methods are used for this purpose: majority rule (MJ) and log-likelihood (LL). We refer the reader to [34] for more details.

Multi-dimensional clustering method. In the second group, clustering is run on multiple attributes or dimensions, i.e., the clustering algorithm is run on all samples, at all points of a set of time indices (POIs), at the same time.

On one hand, the multi-dimensional method has two main advantages over the single-dimensional. First, the combination step is not required. And second, it is capable of exploiting higher order leakage, while the first method exploits only leakage of first order. On the other hand, we verified experimentally that key recovery based on multi-dimensional clustering is more sensitive to noisy samples, because a very noisy sample at a given POI can directly negatively influence the value of the output key candidate. Note that for the first group of attacks the noisy sample effect is contained, i.e., only the key candidate at that POI is affected. Therefore, outlier detection and handling are mandatory in the multi-dimensional method to achieve satisfactory results.

First key-recovery step ("KR for CLA" in Fig. 1). After points of interest have been found by CLA, key recovery is run on the "LA traceset" using the POIs from the CLA trace. The outcome is a list of recovered candidate keys for those traces ("LA traceset w/ recv.bits").

POI optimization step (POI-OPT in Fig. 1). The "LA traceset w/ recv.bits" is used as input for the points of interest selection optimization step. This step refines the POIs found from CLA by applying a t-test with two groups, the first group containing the traces whose corresponding candidate key bit is zero and the second group corresponds to the traces where the candidate key bit is one. The points with the largest t-statistics are considered the refined POIs.

Final key recovery step ("KR final" in Fig. 1). Finally, given the refined list of POIs (i.e., the peaks on "t-test trace"), key recovery is applied sequentially to each trace in the target traceset. For each trace, the key recovery outputs a (possibly incorrect) key/scalar, over which the probabilistic key error correction algorithm in Sec. 6 is applied. If the correct scalar is found, it is returned and the full key recovery process stops. Otherwise, the key recovery is applied to the next trace in the target traceset and the process is repeated.[12]

5 Attack Results

We acquired 300 full Curve25519 ECSM traces for the cswap-arith, and the same number of traces for the cswap-pointer. The traces were preprocessed, resulting in two tracesets of 76,500 ECSM iteration traces each, that are used in all experiments described in this section. Each iteration trace used for the analysis contains 8,000 8-bit samples for cswap-arith. For cswap-pointer, as the time interval where leakage happens is narrower (cf. Sec. 2.1), we trimmed the traces to 1,000 samples for efficiency.[13]

In the evaluation experiments, the recovered scalars are the output of the last KR step, but before error correction. The reported success rates are, unless otherwise noted, the maximum success rates, i.e., if the success rate or percentage of correctly recovered bits for the target traces is $SR_1, \ldots, SR_{n_t}$ (where n_t is the number of traces in the target traceset), then the reported success rate is $\max\{SR_1, \ldots, SR_{n_t}\}$. We use the maximum success rates because, as explained in Sec. 4, our full HCA key recovery attack framework is probabilistic and recovers the correct scalar for at least one of the target traces with a given probability of success, the success rate of the attack. By taking the max success rate (SR^*) in an attack evaluation experiment as the success rate, we guarantee (except

[12] We note that the steps POI-OPT and key recovery can be repeated. Due to the high increase in computational time required to run them more than once, as well as the fact that the results using a single iteration were already feasible for a successful attack, we chose not to further investigate whether that could improve the results.

[13] We knew where and by how much to trim because we knew from the source code and binary the approximate location of the cswap in the iteration traces.

Table 1: Key recovery max success rate (%) for cswap-arith and cswap-pointer, for all combinations of CLA distinguishers (SOSD, MIA and SOST) and HCA statistical combination methods. The best results for each implementation are highlighted.

		cswap-arith			cswap-pointer		
		MJ	LL	MD	MJ	LL	MD
KM	SOSD	92.15	**94.11**	58.03	97.25	60.78	96.47
	MIA	60.78	57.64	58.82	96.47	95.68	57.25
	SOST	**94.11**	92.15	57.64	99.60	96.07	**100.00**
FKM	SOSD	87.84	57.25	58.43	57.64	58.82	98.82
	MIA	60.78	84.31	59.60	99.60	99.21	56.86
	SOST	67.45	59.21	58.03	59.60	98.82	**100.00**
EM	SOSD	60.00	61.56	60.39	97.64	57.64	57.64
	MIA	60.39	68.23	60.39	99.21	95.29	99.60
	SOST	64.31	61.96	57.64	97.64	95.29	57.64

with a negligible chance) that if the full key recovery attack with error correction is applied to a set of target traces, the recovered key for at least one of them will have a ratio of at least SR^* of correctly recovered bits. Therefore, the errors can be successfully corrected by the error correction algorithm in Sec. 6.

5.1 Initial attack evaluation experiment

To evaluate the effect of different distinguishers for CLA and statistical combinators for HCA-KR, we fixed the clustering algorithm (KM, FKM or EM) and ran a full key recovery attack varying the value of such parameters. The evaluation results are shown in Table 1.[14]

In this experiment we used: 100 traces for CLA; 100 traces and 20 POIs for "KR for CLA" and "KR final". We experimented with different numbers of traces for these operations, but from those we tried, 100 was the minimum number of traces that resulted in good enough attack success rates; we did not see any improvement when more traces were used. POI-OPT is enabled. We used Tukey test and replace by median as outlier detection and handling methods, respectively. Intrinsic clustering quality evaluation is disabled.

According to Table 1, the cswap-pointer implementation has a very strong leakage dependent on the cswap bit, in two cases reaching a success rate of 100%, i.e., all scalar bits were correctly recovered. Despite having obtained 100% success rate for two combinations of algorithms KM/SOST/MD and FKM/SOST/MD for the cswap-pointer implementation, similar success rate results for such combinations of algorithms on the cswap-arith implementation do not hold. In fact they were very low for that implementation, with success rates below 60%.

The results obtained in Table 1 do not indicate a single combination of parameters where the success rates are high enough ($\geq 97\%$) for both implementations simultaneously, so as to enable a successful recovery of the correct scalar in feasible time even when error correction (Sec. 6) is applied.

[14] MJ, LL and MD stand for majority rule, log-likelihood and multi-dimensional, resp.

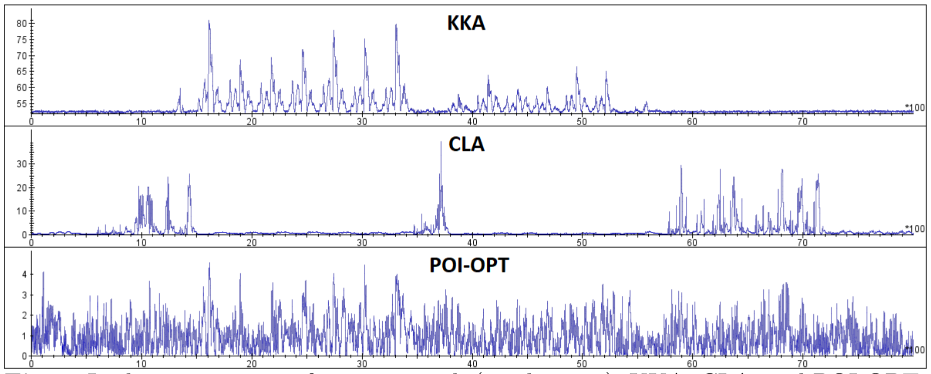

Fig. 3: Leakage assessment for cswap-arith (top-bottom): KKA, CLA, and POI-OPT.

Besides, in a practical non-profiled or single-trace attack scenario, where the attacker do not know details about the implementation targeted, she should fix/choose beforehand the values of all parameters for the full key recovery[15] and run "KR final" for every trace in the target traceset. The motive is the long computation time required to run a full key recovery, where the most expensive step, error correction, can take hours to complete on a common desktop machine.

For the aforementioned reasons, we test our attack with more combinations of parameters values. The results of these experiments are in Appendix E. The values of those parameters that gave the best results are presented in the next subsection.

5.2 Final results

Fig. 3 illustrates, for cswap-arith, the approximated side-channel leakage assessed right after the CLA and POI-OPT steps when compared with a known-key analysis (KKA) trace. The leakage assessment trace after POI-OPT shows peaks that match or are very close to those in the KKA trace. A known-key analysis in essence consists of running the clustering algorithm at each sample index to recover the scalar bit and comparing whether the guessed bit value is equals the known key bit. The output is a trace with the success rate of these guesses, which is indicates the strength of the leakage. Such an analysis is used only for illustrative purposes, we remark that our attacks are completely unsupervised.

Fig. 4 shows the success rate evolution as the number of POIs used by the final HCA-KR step increases, for both cswap implementations. For both implementations, the number of traces used are 100 for CLA is 100, "KR for CLA" and "KR final", the same as in the other experiments. For cswap-arith, the success rate is 97.64% for 100 POIs. The success rate for cswap-pointer is above 90% if the number POIs used is in $[7, 38]$. In particular, it is 99.60% for 38 POIs. Thus, for cswap-pointer a small number of POIs is sufficient to achieve

[15] Among them: number of traces for CLA, "KR for CLA" and "KR final" steps, clustering algorithms, distinguishers and statistical combination methods.

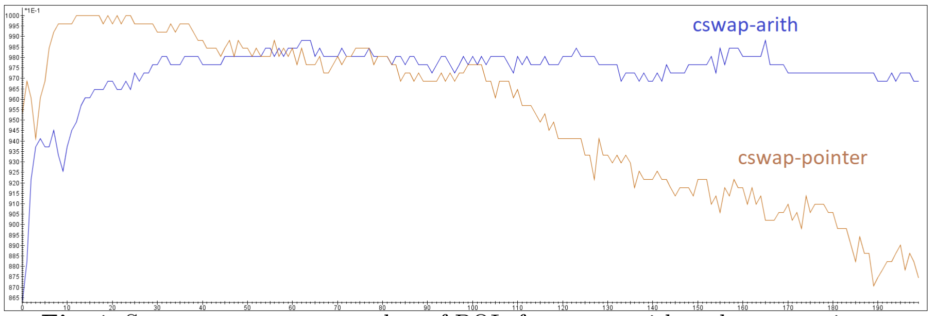

Fig. 4: Success rate versus number of POIs for cswap-arith and cswap-pointer.

a very high success rate. The curves in Fig. 4 have quite different shapes. The cswap-pointer curve shape means that leaks in narrow time intervals, so it is sensitive to the number of POIs used. In this case, using a lot of POIs means adding noise to the analysis, which decreases the success rate. On the contrary, the cswap-arith curve means that it leaks on a wider time interval, so more POIs can be added without dramatically affecting success rate.

Considering these success rate values and taking into account the fact that only 251 bits out of the 255 bits of the scalar are unknown (the first bit is always 1 and the last three are fixed to 100_2), there are at most 6 and 2 errors in the recovered scalar for cswap-arith and cswap-pointer, respectively.

6 Error Correction

Due to noise and other aspects interfering with the side-channel analysis (misalignment for example), the scalar derived by the attack contains errors. A naive brute-force would check all possibilities of 6 and 2 errors in the 251 bits, for each of the 100 recovered scalars. This totals to $100 \cdot \binom{251}{6} \approx 2^{44.9}$ operations for cswap-arith and $100 \cdot \binom{251}{2} \approx 2^{21.6}$ for cswap-pointer. As we can see, the required computation effort is quite feasible, especially for the cswap-pointer case.

Note that confidence probabilities coming from clustering can be used to detect errors, as shown in [14,34,35]. We applied the approach from [34], but unfortunately this method occured unreliable: some errors occured with high confidence probabilities. We suspect that it was caused by strong noise pulses present in our traces. Therefore, we concentrate on improving the naive brute-force.

Efficient error correction based on precomputations. The above naive brute-force can be further by using a modified algorithm from [30]. In [30] the authors use template attack confidence scores to detect errors. Unfortunately, as mentioned above, we cannot use confidence probabilities and therefore, we need to modify the approach in [30], as described below.

First let us assume that the number of errors is at maximum 6 (like for cswap-arith). Now let us divide the scalar in half and assume the errors locations are uniformly distributed across it. Let us denote $R = [k]P$, where R is the resulting point, k the scalar to be recovered, and P is the input point. Then, clearly $R = [k]P = [a \cdot 2^{|k|/2} + b]P = [a]([2^{|k|/2}]P) + [b]P$, where a is the most significant half of k and b is the least significant one[16]. If we denote $[2^{|k|/2}]P$ by H, then the above equation reduces to $R - [b]P = [a]H$.

Consider all different possible guesses for a assuming that there are at most 4 errors in a: that is $\binom{|k|/2}{4}$ guesses. Following [30], for each guess, we compute $[a]H$ and store all pairs $(a, [a]H)$. We then sort all pairs based on the value of $[a]H$ and store them in an ordered table. We make a guess for b assuming it contains at most 4 errors (again $\binom{|k|/2}{4}$ guesses) and compute $z = R - [b]P$. If our guess for b is correct, then z is present in the second column of some row in the table – the first column is the corresponding a. If z is present then we have determined the scalar. Otherwise, we make a new, different guess for b and continue. The complexity of this attack totals to $\binom{126}{4} \cdot 2 \cdot 100 \approx 2^{31}$ operations, because there are 251 unknown bits. The required memory is $\binom{126}{4} \cdot 100 \approx 2^{30}$ points.

The above assumption on uniform distribution of errors can be dropped (cf. Appendix F). We need to estimate the probability that the attack works. The probability that out of 6 errors, 2, 3, or 4 of them are not in a equals: 14/64; for details about computing this probability we refer the reader to Appendix F. Thus, to minimize the error to approximately e.g., 0.0005, it is enough to repeat the algorithm 5 times. Then the overall complexity of the attack would be 2^{36}.

7 Countermeasures and Future Work

In this paper we described horizontal clustering attacks against two Curve25519 Montgomery Ladder ECSM implementations from the μNaCl library. We also showed how to extend the RSA horizontal clustering framework from [34]. Furthermore, we generalized the method from [30] to tolerate a certain number of incorrectly recovered scalar bits without relying on normal exhaustive search.

Now we briefly discuss possible countermeasures against our attack. First let us recall that the following countermeasures do not work against our attack: point re-randomization, scalar blinding and splitting.

The countermeasure of [31] splits scalar into two parts and to randomly interleave two scalar multiplications. We believe that our attack might still be mounted if four clusters are used to recognize which bit is processed and during which ECSM. The idea behind the memory-address countermeasure [16] is to store sensitive variables at addresses that share the same Hamming weight. Although this would decrease the effectiveness of the attack, the addresses leakage may still be identified by clustering. This countermeasure can be improved by randomizing not only the addresses but also the memory accesses [17,18,19].

The countermeasure of [15] protects against localized EM template attacks on Montgomery ladder ECSM by randomly swapping the ladder registers at the end of a ladder iteration. This countermeasure is uniform in its operation sequence what makes our attack infeasible in principle. In addition, several randomization-based protection techniques for the Montgomery ladder are presented in [24].

[16] $|k|$ denotes the length of the base 2 representation of the scalar k.

Similar to [15], these techniques generate operation sequences independent from the scalar and thus, our attack might be ineffective against them.

We consider evaluating and improving our attacks with respect to the two latter countermeasures as future work. We also regard attacking other ECC implementations improving our attacks with PCA as future developments.

Acknowledgements. This work was supported by the European Union's H2020 Programme under grant agreement number ICT-731591 (REASSURE).

References

1. A. Bauer and É. Jaulmes. Correlation analysis against protected SFM implementations of RSA. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8250 LNCS:98–115, 2013.
2. A. Bauer, É. Jaulmes, E. Prouff, and J. Wild. Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations. *CT-RSA*, 1–17. 2013.
3. T. Caliński and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.
4. C. Clavier, B. Feix, G. Gagnerot, C. Giraud, M. Roussellet, and V. Verneuil. ROSETTA for Single Trace Analysis. *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, 140–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
5. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Horizontal Correlation Analysis on Exponentiation. *Information and Communications Security: 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings*, 46–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
6. D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, 1979.
7. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38, 1977.
8. F. Dürr. Key 2.0 is a Bluetooth IoT Door Lock. https://github.com/duerrfk/key20, 2017.
9. EMV. EMVCo Security Evaluation Process, version 5.1, *Security Guidelines*, 2016.
10. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.
11. E. W. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.
12. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. *NIST Workshop 2011*. 2011.
13. D. Hawkings. *Identification of Outliers*. Chapman and Hall, 1980.
14. J. Heyszl, A. Ibing, S. Mangard, F. Santis, and G. Sigl. Clustering Algorithms for Non-profiled Single-Execution Attacks on Exponentiations. *CARDIS*, 79–93. Springer International Publishing, Cham, 2014.
15. J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, and G. Sigl. Localized electromagnetic analysis of cryptographic implementations. *Topics in Cryptology – CT-RSA 2012*, vol. 7178 of *LNCS*, 231–244. Springer, 2012.

16. K. Itoh, T. Izu, and M. Takenaka. Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA. *Cryptographic Hardware and Embedded Systems – CHES 2002*, vol. 2523 of *LNCS*, 129–143. Springer, 2002.
17. K. Itoh, T. Izu, and M. Takenaka. A practical countermeasure against address-bit differential power analysis. *Cryptographic Hardware and Embedded Systems – CHES 2003*, vol. 2779 of *LNCS*, 382–396. Springer, 2003.
18. M. Izumi, J. Ikegami, K. Sakiyama, and K. Ohta. Improved countermeasure against address-bit DPA for ECC scalar multiplication. *2010 Design, Automation & Test in Europe Conference and Exhibition (DATE 2010)*, 981–984. IEEE, 2010.
19. M. Izumi, K. Sakiyama, and K. Ohta. A new approach for implementing the MPL method toward higher SPA resistance. *International Conference on Availability, Reliability and Security, 2009 . ARES '09*, 181–186. IEEE, 2009.
20. K. Jarvinen and J. Balasch. Single-Trace Side-Channel Attacks on Scalar Multiplications with Precomputations. *CARDIS*. 2016.
21. I. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002.
22. L. Kauffman and P. Rousseeuw. Finding groups in data. *An introduction to cluster analysis*, 1990.
23. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.
24. D.-P. Le, C. H. Tan, and M. Tunstall. Randomizing the montgomery powering ladder. *WISTP 2015*, 169–184. Springer International Publishing, 2015.
25. S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
26. L. Mather, E. Oswald, J. Bandenburg, and M. Wójcik. Does My Device Leak Information? An a priori Statistical Power Analysis of Leakage Detection Tests. *Advances in Cryptology-ASIACRYPT 2013*, 486–505. Springer, 2013.
27. O. Meynard, D. Réal, F. Flament, S. Guilley, N. Homma, and J. L. Danger. Enhancement of simple electro-magnetic attacks by pre-characterization in frequency domain and demodulation techniques. *2011 Design, Automation Test in Europe*, 1–6. 2011.
28. V. S. Miller. Use of elliptic curves in cryptography. *Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85*, 417–426. Springer-Verlag New York, Inc., New York, NY, USA, 1986.
29. E. Nascimento and L. Chmielewski. Applying horizontal clustering side-channel attacks on embedded ecc implementations (extended version). Cryptology ePrint Archive, Report 2017/1430. https://eprint.iacr.org/2017/1430, 2017.
30. E. Nascimento, L. Chmielewski, D. Oswald, and P. Schwabe. Attacking embedded ECC implementations through cmov side channels. *Selected Areas in Cryptography, SAC 2016, St Johns, NL, Canada - to appear*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.
31. C. Negre and G. Perin. Trade-off approaches for leak resistant modular arithmetic in RNS. *Information Security and Privacy*, vol. 9144 of *LNCS*, 107–124. Springer, 2015.
32. NIST. NIST/SEMATECH e-Handbook of Statistical Methods. Section 7.1.6. What are outliers in the data?, 2013.
33. NIST. NIST/SEMATECH e-Handbook of Statistical Methods. Section 7.4.7.1. Tukey's method, 2013.
34. G. Perin and L. Chmielewski. A Semi-Parametric Approach for Side-Channel Attacks on Protected RSA Implementations. *CARDIS*. 2015.

35. G. Perin, L. Imbertl, L. Torres, P. Maurine, and R. A. Montpellier. Attacking Randomized Exponentiations Using Unsupervised Learning. *CARDIS*. 2014.
36. T. H. project. picotls - TLS 1.3 implementation in C. https://github.com/h2o/picotls, 2017.
37. R. Specht, J. Heyszl, M. Kleinsteuber, and G. Sigl. Improving non-profiled attacks on exponentiations based on clustering and extracting leakage from multi-channel high-resolution em measurements. *Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design - Volume 9064*, COSADE 2015, 3–19. Springer-Verlag New York, Inc., New York, NY, USA, 2015.
38. F.-X. Standaert, T. G. Malkin, and M. Yung. A formal practice-oriented model for the analysis of side-channel attacks. *IACR e-print archive*, 134(2006):2, 2006.
39. C. D. Walter. Sliding Windows Succumbs to Big Mac Attack. *CHES*, 286–299. 2001.
40. T. Wilkinson. HomeKit for Bluetooth Low Energy (BLE) for Nordic nRF51. https://github.com/aanon4/HomeKit, 2015.
41. M. F. Witteman, J. G. J. van Woudenberg, and F. Menarini. Defeating RSA Multiply-Always and Message Blinding Countermeasures. *CT-RSA 2011*, 77–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

A Montgomery Ladder Implementation Details

In this paper we consider the μNaCl library that provides two ECSM implementations for Curve25519. Both of them are based on the Montgomery ladder algorithm, which is presented in Alg. 2.

Algorithm 2 Montgomery ladder for Curve25519

```
Input: 255-bit scalar $s$, $x$-coordinate $x_P$ of input point P.
Output: $(X_{[s]P}, Z_{[s]P})$, such that $x_{[s]P} = X_{[s]P}/Z_{[s]P}$.
  $P1 \leftarrow (1, 0)$; $P2 \leftarrow (x_P, 1)$.
  for $i \leftarrow 254$ downto 0 do
      if $s_i = 1$ then
          $P1 \leftarrow$ PADD$(P1, P2, x_P)$
          $P2 \leftarrow$ PDBL$(P2)$
      else
          $P2 \leftarrow$ PADD$(P1, P2, x_P)$
          $P1 \leftarrow$ PDBL$(P1)$
      end if
  end for
  return $P1$
```

For concreteness, Listing 1.1 shows the actual C implementation of the arithmetic conditional swap of two field elements in μNaCl (call to `CSWAP` in Alg. 1). This is an arithmetic CSWAP implementation with `XOR` and `AND` instructions, and is known to leak the value of `mask` through side-channels, e.g. on the AVR architecture [30]. Listing 1.2 shows the relevant part of the cswap-pointer implementation that “touches”, i.e., operates on the secret-dependent `condition` (bit s in Alg. 1).

Listing 1.1: Conditional swap based on arithmetic of field operands limbs.

```c
void fe25519_cswap(fe25519* in1, fe25519* in2, int condition)
{
    int32 mask = condition;
    uint32 ctr;
    mask = -mask;
    for (ctr = 0; ctr < 8; ctr++)
    {
        uint32 val1 = in1->as_uint32[ctr];
        uint32 val2 = in2->as_uint32[ctr];
        uint32 temp = val1;
        val1 ^= mask & (val2 ^ val1);
        val2 ^= mask & (val2 ^ temp);
        in1->as_uint32[ctr] = val1;
        in2->as_uint32[ctr] = val2;
    }
}
```

Listing 1.2: Conditional swap of pointers to field operands.

```c
void swapPointersConditionally (void **p1, void **p2, uint8 condition)
{
    uintptr mask = condition;
    uintptr val1 = (uintptr) *p1;
    uintptr val2 = (uintptr) *p2;
    uintptr temp = val2 ^ val1;

    mask = (uintptr)( - (intptr) mask );
    temp ^= mask & (temp ^ val1);
    val1 ^= mask & (val1 ^ val2);
    val2 ^= mask & (val2 ^ temp);

    *p1 = (void *) val1;
    *p2 = (void *) val2;
}
```

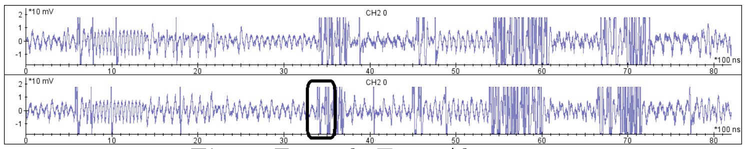

Fig. 5: Example Trace Alignment

B Traces Preprocessing

Trace cutting. First each side-channel trace of a complete ECSM run is cut in trace segments, one per each algorithm iteration. In the context of this work, the ECSM implementation is based on Montgomery Ladder, so each iteration i corresponds to the processing of the *swap* bit (s in Alg. 1), which depends on i-th scalar bit. The trace cutting was done by first applying a low pass filter and then detecting patterns that appeared repeatedly for 254 times using a threshold (i.e., for all ECSM iterations, except the last one). The patterns were detected visually with ease. A simple time-based trace cutting did not work in our case, despite the constant timeness of the target implementation, due to visible time drifts in the measured samples, probably due to clock drifting or measurement imprecisions. The above technique is similar to the one used in [34].

Trace alignment. After the traces are cut they are not precisely aligned. To overcome this issue we align the traces at the location shown in Figure 5. To align the traces exactly on the underlined pattern we used Pearson correlation. We have selected the pattern based on the time of a single Montgomery ladder iteration and source code analysis of the implementation.

C Known-Key Analysis (HCA-KKA)

Known-key analysis (KKA) consists of finding "good" POIs given the knowledge of the key, i.e., to find points (sample indices) where the leakage is strongest.

Due to the need to know the value of the key, KKA is employed only in a profiled attack scenario or when testing an attack against a device similar to the target, to help answer practical questions such as the following: (i) how do we know whether the traces are properly aligned (ii) for a given device, how many traces are necessary for the leakage assessment method (CLA) to find the known-good POIs? This can be done by checking whether the top POIs from CLA (partially) match the known-good ones foreseen by KKA.

Such an analysis is done as follows. We first apply the clustering algorithm to the set of samples at a given point, across the traces in the traceset. Two clusters are obtained: the first corresponds to the samples labeled with the value $b \in \{0, 1\}$, and the second with the opposite value $\bar{b}$. At this point we do not know if b is equal to 0 or 1. Second, we use the key knowledge to figure out how many samples were correctly labeled.

The procedure described above is repeated for all sample points, and the result is a trace of the strength of key-dependent leakage for all sample points

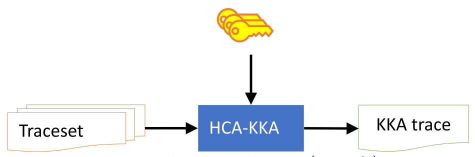

Fig. 6: Known-key analysis (KKA) process.

(Fig. 6). The list of such points, in decreasing order of the number of correctly labeled samples, is considered the known-good POIs.

D Leakage Detection Methods

Let us consider the n-th trace represented as a set of samples. We apply a clustering algorithm to each set of samples $\{t^n_{i:l_0+l_1,j}\}$ at a fixed sampling time j. The clustering returns two centers $c_{0,j}$ and $c_{1,j}$ and two groups of clustered samples $\{g_{0,j}\}$ and $\{g_{1,j}\}$ containing $p_{0,j}$ and $p_{1,j}$ elements, respectively.

For every trace $\mathbf{t}^n$, we have a set of parameters $c^n_{0,j}$, $c^n_{1,j}$, $\{g^n_{0,j}\}$, $\{g^n_{1,j}\}$, $p^n_{0,j}$ and $p^n_{1,j}$ $(j \in \{1,..,\ell\})$ that can be used for the leakage assessment. We provide results for four different techniques: sum-of-squared differences (SOSD), sum-of-squared t-values (SOST) and mutual information analysis (MIA).

The equations for SOSD and SOST are given by:

$$\varsigma_j = \tfrac{1}{N}\textstyle\sum_{n=1}^{N} |c^n_{0,j} - c^n_{1,j}|^2 \quad \text{(SOSD)} \tag{3}$$

$$\tau_j = \tfrac{1}{N}\textstyle\sum_{n=1}^{N} \left(\left(c^n_{0,j} - c^n_{1,j}\right)^2 \Big/ \left(\frac{\sigma^{2(n)}_{0,j}}{p^n_{0,j}} + \frac{\sigma^{2(n)}_{1,j}}{p^n_{1,j}} \right) \right) \quad \text{(SOST)} \tag{4}$$

Considering $\sigma^{2(n)}_j$ as the variance for the sample set $\{g_{0,j}(n)\} \cup \{g_{1,j}(n)\}$, and $\varrho = p^n_{0,j}/(p^n_{0,j} + p^n_{1,j})$, the mutual information value υ_j at each sample index j is computed by:

$$\upsilon_j = \sum_{n=1}^{N} \log \sqrt{\frac{1}{p^n_{0,j} + p^n_{1,j} - 1}\sigma^{2(n)}_j} + \sum_{k=0}^{1} (-1)\varrho^k (1-\varrho)^{1-k} \log \sqrt{\frac{1}{(p^n_{1-k,j}) - 1}\sigma^{2(n)}_{1-k,j}} \tag{5}$$

E Evaluation of the Effects of Attack Parameters

We compute the average of the success rates in Table 1 grouped by parameters clustering algorithm, CLA distinguisher or HCA statistical combinator (Table 2). KM, SOST and MJ give the best results, on average. Thus in the next experiments we fix those parameters to these values and explore the effect of other parameters. [17]

[17] In this section, the values of the parameters used in an experiment are the best ones (i.e., they resulted in the biggest success rate) found in the previous experiment, excluding those parameters that are being evaluated in the experiment in question.

Table 2: Averages of the HCA-KR success rates (%) in Table 1 grouped by parameter.

(a) Clustering algorithm.

KM	FKM	EM
81.39	73.46	72.91

(b) CLA distinguisher.

SOSD	MIA	SOST
72.91	76.12	**78.73**

(c) Combination method.

MJ	LL	MD
80.69	77.45	69.63

Points of interest optimization is an essential step in the key recovery process (Fig. 2), without which the leakage assessment trace (CLA trace) is usually not accurate enough for the KR final step to produce a successful key recovery. For example, the HCA-KR success rates for the attack with KM+SOST+MJ with POI-OPT are 94.11% and 99.60%, while without POI-OPT the results drop to 60.39% and 58.82%, resp. for cswap-arith and cswap-pointer.

The effect of number of traces used for leakage assessment (the CLA step) in the success rate is shown in Table 3, for a few different values of this parameter and for both implementationsAs can be seen, the difference in the success rate is small among them, and 100 traces for CLA gives the best result.

Table 3: Effect of the number of traces for CLA. Max success rate (%).

	50	100	200	300
arith	92.34	94.11	93.04	92.78
pointer	98.27	99.60	98.83	96.25

Table 4: Effect of outlier detection and handling. Results for cswap-arith. Average success rate (%).

	Repl. by median
No outlier detector	72.34
Distance from mean	73.80
Tukey test	72.86

We evaluated the attack without outlier detection and with the methods distance from mean and Tukey test applied. When outlier detection is applied, we replace the outliers values by the median of the other data points. Table 4 shows the average success rate against the cswap-arith implementation. We can see that, on average, outlier detection slightly improves success rate.

The number of points of interest selected from the CLA trace determines the amount of signal (secret-dependent leakage) and noise that will be used in the 1st HCA-KR step (i.e., the first step in POI-OPT) and the 2nd/final HCA-KR step.[18] Table 5 shows the results of our evaluation on how these parameters affect the attack success rate for the cswap-arith traceset. As can be seen, the success rate plummets from above 80% to 60% if fewer than 10, or more than 20, POIs for the 1st HCA-KR step are used, with a peak of 96.07% for 20 POIs. For the 2nd HCA-KR step, the best results were obtained with 100 POIs.

[18] I.e., how many POIs to use from the POI-OPT t-test trace to use in the attack phase.

Table 5: Effect of the number of POIs used in the first and second HCA-KR step. Results for cswap-arith. Max success rate (%).

num. POIs on 1st/2nd step	20	50	100	200	300
5	63.13	59.60	57.25	56.89	56.86
10	91.76	93.72	93.72	89.80	87.45
20	94.11	95.29	96.07	89.01	84.31
50	60.39	59.60	60.39	60.78	60.78
100	61.56	62.35	61.56	61.96	60.39

Table 6 shows the effect of the number of iterations for the Davies-Bouldin clustering intrinsic quality evaluation, when applied only in the CLA step for the cswap-arith traceset. The success rate improves slightly if two iterations are used, but then starts to drop afterwards, reaching 90.19% for 10 iterations.

Table 6: Effect of the number of iterations of Davies-Bouldin clustering intrinsic quality evaluation applied in the CLA step. Results for cswap-arith.

Num. of iterations	1	2	5	10
Max success rate (%)	96.07	97.64	95.68	90.19

F Probability of Successful Efficient Error Correction

We assumed in Sec. 6 that the errors are uniformly distributed. Now we show how to drop this assumption. We create a in the following way: we randomly choose a set A of indices in k such that $|A| = |k|/2$ and we set the corresponding bits to zero. Then we create b by setting the remaining indices of the original k to zero (the set of indices is denoted as B). Now $R = [a]P + [b]P$ holds and if we set $H = P$ then $R - [b]P = [a]H$. The attack can be performed as before assuming that when we guess a and b, we limit the indices to A and B, respectively.

We now compute the probability that the attack from Sec. 6 works correctly, namely, that the 6 errors are corrected. Without loss of generality let us first assume that positions of the 6 errors position are fixed, because the partition to a and b is random. Therefore, the following situations are possible:

- all errors are in a or in b: 2 possibilities;
- one error is in a or b: 12 possibilities;
- two errors are in a or b: 30 possibilities;
- three errors are in both a and b: 20 possibilities.

In the first two cases the numbers of errors in a is 0, 1, 5, or 6. Therefore, the probability that out of 6 errors, 2, 3, or 4 of them are not in a equals:

$$\frac{1+6+6+1}{2+12+30+20} = \frac{14}{64}.$$