Keep it Unsupervised: Horizontal Attacks Meet Deep Learning

Guilherme Perin[1,3], Łukasz Chmielewski[2,3], Lejla Batina[2] and Stjepan Picek[1]

[1] Delft University of Technology, The Netherlands
[2] Radboud University Nijmegen, The Netherlands
[3] Riscure BV, The Netherlands

Abstract. To mitigate side-channel attacks, real-world implementations of public-key cryptosystems adopt state-of-the-art countermeasures based on randomization of the private or ephemeral keys. Usually, for each private key operation, a "scalar blinding" is performed using 32 or 64 randomly generated bits. Nevertheless, horizontal attacks based on a single trace still pose serious threats to protected ECC or RSA implementations. If the secrets learned through a single-trace attack contain too many wrong (or noisy) bits, the cryptanalysis methods for recovering remaining bits become impractical due to time and computational constraints. This paper proposes a deep learning-based framework to iteratively correct partially correct secret keys resulting from a clustering-based horizontal attack. By testing the trained network on scalar multiplication (or exponentiation) traces, we demonstrate that a deep neural network can significantly reduce the number of error bits from randomized scalars (or exponents). When a simple horizontal attack can recover around 52% of private key bits, the proposed iterative framework improves the private key correctness to 100%. Our attack model remains fully unsupervised and excludes the need to know where the error or noisy bits are located in each separate randomized private key.

Keywords: Side-channel Analysis · Public-key Algorithms · Horizontal Attacks · Deep Learning

1 Introduction

Modern implementations of public-key cryptosystems in embedded devices are usually protected against non-invasive side-channel attacks. When considering RSA or ECC-based protocols such as key generation or signature verification, the operation to protect is modular exponentiation or scalar multiplication, which are accordingly the most security-critical operations (including the operations they consist of). The private key (scalar or exponent) directly determines the computation length, i.e., the number of iterations in a scalar multiplication or modular exponentiation loop. The side-channel attacks demonstrated to be a threat against even side-channel protected implementations. Typical countermeasures include randomization of the private key (as well as other sensitive parameters for each protocol execution, e.g., projective coordinates for ECC) and the regularity of all the operations involved.

Using the randomization techniques properly deems unsupervised attacks such as DPA [KJJ99], CPA [BCO04], or MIA [GBTP08] unfeasible. On the other hand, Simple Power Analysis (SPA), which is the simplest form of unsupervised attack, can be defeated by regular methods, such as Montgomery ladder [JY02], square-and-multiply always (resp. double-and-add always), or atomicity-based implementations [GV10]. However, profiled attacks still apply, but they require an open device clone with access to the random number generator output to use the profiling phase's random values. This is typically impractical since those values are stored in a protected way and are inaccessible to an adversary or even evaluator. Consequently, the only possible attack in a real-world scenario remains the exploitation of a single trace separately. The secret recovered from a single trace is a blinded private key, which can still be used for decryption or signature verification for RSA or ECC-based protocols.

In the past years, different forms of horizontal attacks that target single traces have been proposed in several research papers [BJPW13, BJP+15]. For a horizontal attack to be successful, the private key bits must be recovered from a single side-channel trace, because only then the blinding procedure has no effect. This makes horizontal attacks a serious threat against private key blinding protections. However, the results provided by horizontal attacks must provide a limited amount of error bits (or noisy bits) in recovered private keys to complete the attack with brute-force or some other cryptanalysis. If the amount of error bits is substantial, the brute-force step is usually impractical. Besides, the attacker usually has unreliable metrics to determine the wrong bits' position inside the recovered private key. These constraints may render horizontal attacks unfeasible when assessing the security of protected implementations of public-key cryptosystems.

This paper proposes a new approach to improving single trace attacks that reduce the number of error bits in a recovered private key. By attacking several single traces, an attacker may recover several partially correct random private keys. This information is then used to label each sub-trace (trace interval representing the processing of a single private key bit) and use them as elements in a training set to train a neural network. Assuming that each recovered private key contains more than 50% of correct bits (just above a random guess), the trained neural network can significantly improve the number of correct bits in a random private key related to a single trace. This way, the attack remains an unsupervised side-channel attack. Our experiments show that even if the partially recovered private keys contain up to 48% of wrong bits, the neural network can return a private key with 100% correct bits. Our new attack framework is based on the fact that deep neural networks can generalize well, even if the training set contains a significant amount of noisy labels [ZS18, HYY+18].

To summarize, the main contributions of this work are as follows:

- We propose a novel framework for horizontal attacks on implementations of public-key cryptosystems. This framework is based on an unsupervised learning paradigm.
- The efficiency of our method reaches 100%, which is a significant improvement from state-of-the-art horizontal attacks.
- We demonstrate our method on two real-world datasets featuring the traces from protected ECC implementations on ARM Cortex-M, i.e., we successfully attack the μNaCl[1] library with our new method.
- As our framework is generic, we argue it is adequate to target other public-key (including post-quantum) crypto implementations.

Structure of the paper The remainder of this work is organized as follows. Section 2 summarizes background information on deep neural networks and the details of the attacked implementations. Subsequently, we describe horizontal attacks and their background, including related work, and the technique used for labeling in Section 3. We propose an iterative deep learning framework in Section 4. In Section 5, we present the results of our practical evaluation of the proposed framework. Finally, in Section 6, we conclude with an overview of what we believe to be the most important directions for future work.

[1] 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
// ... return omitted ...
```

2 Background

This section gives relevant background information about the target, i.e., datasets we used, and deep learning as employed in our framework.

2.1 Target

This work targets protected ECC implementations in software. In particular, we attack the same library as Nascimento and Chmielewski in [NC17], namely, a protected μNaCl. This is a cryptographic library for ARM Cortex-M that provides implementations of Curve25519, an elliptic curve of the 128-bit security level, and its associated X25519 key exchange protocol. The curve Curve25519 [Ber06] is a Montgomery curve defined by the equation $y^2 = x^3 + 486662x^2 + x$ over the prime field defined by the prime number $2^{255} - 19$. It uses $x = 9$ as the base point, which generates a cyclic subgroup of order $2^{252} + 27742317777372353535851937790883648493$ (prime) and cofactor 8.

The μNaCl library provides two elliptic curve scalar multiplication (ECSM) implementations, both based on the Montgomery ladder algorithm They differ in 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). For each algorithm iteration, the cswap condition depends on the scalar bit processed at that iteration. A high-level description of such a strategy is given in Algorithm 1.

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. Another cswap implementation performs a conditional swap of pointers to the field elements instead (cswap-pointer). The main difference between those implementations is that during each ECSM iteration, in the second implementation, the mask is accessed fewer times by the `AND` (`&`) instruction (3 times) than in the cswap-arith (16 times). The implementation of cswap-arith is presented in Appendix A.

Note that the μNaCl ECSM implementations do not provide SCA countermeasures, besides being regular [2] and constant-time. The authors of [NC17] strengthen the SCA resistance by adding coordinate re-randomization [NCOS16] to both implementations.

Although the scalar randomization is not implemented (in μNaCl), we collect the traces with a random scalar for each ECSM execution. Therefore, the techniques presented in this paper also work against implementations protected with scalar randomization. Observe that, since we attack a randomized scalar, this setting is equivalent to attacking Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange or the scalar multiplication with

[2]Regularity means that the scalar multiplication operations, like ECC double and add, are always executed in the same order, regardless the scalar.

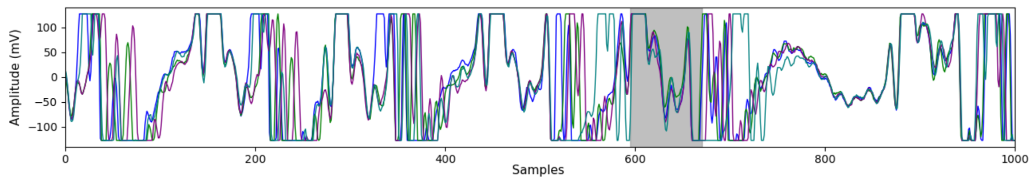


Figure 1: Pointer-Cswap 4 overlapped traces (grey color marks aligned area)

an ephemeral key, as required in some signature generation algorithms, e.g., ECDSA. For details about the targeted implementation, we refer the reader to [NC17].

Leakage from cswap in [NC17] The paper [NC17] shows that both implementations leak the cswap condition value through an elaborate and heavily parameterized horizontal clustering attack. For both implementations, the authors succeed in recovering at least one random scalar when employing an additional brute-force. The attack's success rate reaches more than 95% for a single trace among all 100 attacked traces. In particular, the pointer cswap implementation leaks slightly more than the arithmetic one.

Measurement Setup We use a similar setup to the one from [NC17]. 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 emanation (EM) traces that originate from the ECSM execution on the target device, using a single EM probe. The setup consists of a Lecroy Waverunner 8254M oscilloscope, a Langer RF-U 2.5-2 H-field probe, an amplifier, and an analog low pass filter (250 MHz), where we use a low pass BNC analog filter BLP-250+ from Mini-Circuits. The following settings were used: 2.5 GS/s sample rate, 16 mV amplitude, and 70 million samples. The acquisition and trace pre-processing was done using Riscure's Inspector software package. [3]

2.2 Datasets

Two datasets are considered in this paper. They are collected from an implementation that executes Algorithm 1. The first dataset refers to the implementation protected with conditional *cswap* operation uses pointer swapping. The second implementation uses conditional *cswap* operation with arithmetic means.

Both datasets consist of 300 traces each. The traces are split up to single iterations and aligned as described in Section 5.1. For each iteration, we aim to recover the bit s from Algorithm 1 (i.e., the condition bit to cswap). All the accuracy values in this paper are listed for the values of s.

Dataset Pointer-Cswap The split dataset for pointer cswap consist of $255 * 300 = 76,500$ iteration traces or sub-traces. The 4 overlapped iteration traces are presented in Figure 1. As we can see, the traces are relatively noisy. We mark the area of alignment with a grey color.

Subsequently, we execute a horizontal attack, following a shortened approach from [NC17], as described in detail in Section 3.3. The resulting average accuracy for 300 traces is relatively low: 0.5224. Finally, for the deep learning attack, we label the traces with the recovered noisy scalar bits.

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

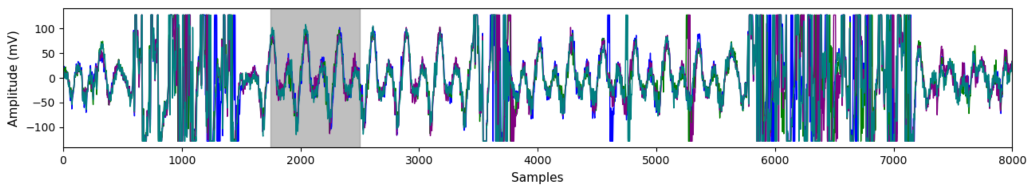


Figure 2: Arith-Cswap 4 overlapped traces (grey color marks aligned area)

Dataset Arith-Cswap The split dataset for arithmetic cswap consists of $76,500$ iteration traces or sub-traces, the same as for the pointer cswap dataset. The four overlapped iteration traces are presented in Figure 2. Like for the other set, the measurements are relatively noisy.

Subsequently, similarly as for Pointer-Cswap, we execute the horizontal attack from Section 3.3. The resulting average accuracy for 300 traces is again relatively low: 0.5244. Finally, we label the traces with the recovered noisy scalar bits.

2.3 Deep Neural Networks

Algorithms like multilayer perceptron (MLP) or convolutional neural networks (CNNs) bring many advantages for SCA as they can break targets protected with countermeasures [CDP17, KPH+19]. Besides reaching top performance, they also do not require pre-processing of leakage traces, making the attack simpler to run. Previous works have applied such architectures to profiled attacks against symmetric [MPP16, CDP17, KPH+19, ZBHV20] and asymmetric crypto algorithms [CCC+19, WPB19].

Convolutional neural networks (CNNs) commonly consist of three types of layers: convolutional layers, pooling layers, and fully-connected layers. Convolution layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. Pooling decrease the number of extracted features by performing a down-sampling operation along the spatial dimensions. The fully-connected layer computes either the hidden activations or the class scores. Additionally, the batch normalization layer normalizes the input layer by adjusting and scaling the activations.

Training deep neural networks with noisy labels is a well-known problem within the machine learning community [HYY+18]. The main challenge relies on the achievement of good model generalization when a significant portion of training data contains noisy or random labels. Deep neural networks have a high capacity for memorizing noisy labels, which hurts generalization performance. A common way to deal with noisy labels is to use (explicit) regularization. For example, in [JNC16], the authors applied dropout to improve learning performance in the presence of noisy labels. Alternatively, data augmentation can serve as an implicit form of regularization. Data augmentation has been successfully applied to deep learning-based attacks on protected AES implementations [CDP17]. In this case, the authors modify the training traces with random shifts and warping to improve the generalization of a profiled attack to side-channel measurements that contain misalignment. We show that our iterative framework also benefits from data augmentation as it drastically improves classification accuracy.

3 Horizontal Attacks

This section provides an overview of horizontal attacks against protected public-key implementations. First, the main types of horizontal attacks are listed and described. Next, we describe what the practical limitations of horizontal attacks in practice are. As this paper considers cluster-based horizontal attacks to which the proposed iterative framework is applied, we also explain how to create initial labels for target traces that are at least above random guessing. Finally, this section discusses what has been published so far in terms of methods to remove noisy bits of secrets obtained from public-key implementation.

There are two kinds of horizontal attacks: profiled and non-profiled. In the first case, attacker use a device under their control to create ‘profiles’ of operations with sensitive data that they can later ‘match’ to measurements taken from the victim’s device. [4]. In the profiled case, the most notable attacks are template attacks on single traces, as demonstrated on ECC [NCOS16], and deep learning attacks on RSA [CCC+19]. In this work, we mean the non-profiled (and therefore unsupervised) setting when referring to horizontal attacks.

3.1 Main Types of Horizontal Attacks

This section lists the main types of unsupervised horizontal attacks, as proposed in the recent literature.

- Horizontal correlation attacks. In [CFG+10], the authors proposed an attack based on predictions of intermediate multiple-precision arithmetic results. Although this method is effective against private key blinding countermeasures, other mitigation methods, e.g., message or point randomization, can prevent this horizontal attack.
- Online template attacks [BCP+17]: These attacks use horizontal techniques to exploit the fact that an internal state of scalar multiplication depends only on the (known) input and the scalar. Advanced types of those attacks need only one leakage trace and can defeat implementations protected with scalar blinding or splitting only. This kind of attack can be prevented by randomizing the internal state using point blinding and projective coordinate randomization [Cor99] or coordinates re-randomization [NCOS16]. Note that online template attacks are actually non-profiled, despite their name.
- Collision-correlation horizontal attacks [BJPW13, BJP+15]. In this type of horizontal attack, an adversary computes statistical correlations between two sub-traces to verify if they share common operands in the modular operations. This method is effective against secret and message/pointer countermeasures, as the statistical distinguisher only needs side-channel leakages representing the processing of input operands to be used in modular operations. Thus, this method does not require knowledge of the intermediate data and can be applied against message randomization techniques.
- Clustering-based horizontal attacks [HIM+13, SHKS15, PITM14, PC15, NC17]. Heyszl et al. proposed to use clustering algorithms in the context of horizontal attacks [HIM+13]. The follow-up work extended the original method to multi-channel and high-resolution electromagnetic measurements [SHKS15]. Perin et al. proposed a heuristic method to attack single exponentiation traces (like in RSA) based on the combination of multi clustering results [PITM14]. The work presented in [PC15] proposed a clustering-based framework to combine multiple points of interest identified through different unsupervised methods. In [NC17], the authors applied the framework proposed in [PC15] to the context of protected ECC implementations.

3.2 Horizontal Attacks in Practice

The application of horizontal attacks requires specific knowledge about the target implementation. This information can be obtained through available documentation or by reverse engineering. The target scalar multiplication or modular exponentiation may be

[4]Here the attacker might know the private key and be even able to control the execution of the profiled device and turn off the countermeasures, for example. The attack can, however, be more complex due to the portability issue.

implemented in many different ways depending on the required performance or additional countermeasures. By observing single side-channel traces, an adversary might recognize the countermeasures and therefore enable a horizontal attack. These cryptographic operations are implemented through a sequence of modular operations. Usually, the minimum required granularity is in distinguishing doubling and adding for scalar multiplication [5] and Montgomery multiplications for modular exponentiation.

If a (post-processed) side-channel trace provides conditions to recognize the sequence of operations, an attacker could deduce some valuable information about countermeasures, such as the bit-length of the blinding factor, or the type of the scalar/exponent blinding countermeasure (e.g., additive, multiplicative, or splitting). Knowing the bit-length of the private scalar or exponent, the attacker can split the trace into sub-traces, each one representing (ideally) the time interval to process one scalar or exponent bit. Even if this process sounds simple from a theoretical point of view, in practice, a single side-channel trace could feature low signal-to-noise ratio (SNR), dummy modular operations, or jitter (also as a countermeasure), increasing the difficulties to split the full trace into sub-traces correctly. While those limitations are not enough to mitigate horizontal attacks, they significantly increase the signal processing requirements.

The results provided by horizontal attacks must reduce error bits (or noisy bits) to fully recover the target key with brute-forcing or even advanced error correction algorithms based on cryptanalysis. As an example, if an attacker can recover 90% of a 2 048-bit RSA exponent, then it means that a trivial brute-force would require a search space of approximately 10^{286} under the assumption that the attacker knows which bits are incorrect. [6] As it turns out, this renders the brute-forcing of wrong bits after a horizontal attack completely impractical.

Therefore, in practice, it is reasonable to assume that, after applying an unsupervised horizontal attack to a single trace, the recovered secret key bits might still contain erroneous bits. Hence, a post-analysis method is still required to reduce the error bits to a quantity that cryptanalysis or brute-force can handle. This is where this work comes in, so basically to improve this process.

In this work, the post-analysis solution is based on an iterative deep-learning framework, as detailed in Section 4. This paper's main contribution is adopting deep learning to correct the error bits while keeping the attack framework in a fully unsupervised setting, as an adversary assumes no knowledge about private key bits.

3.3 Short Horizontal Clustering for Traces Labeling

Again, we emphasize that the datasets we attack with deep learning need to be labeled (a bit) better than random. Consequently, we combine a shortened version of the horizontal attack by Nascimento and Chmielewski [NC17] and the semi-parametric approach by Perin et al. [PC15]. In this section, we recall the attacks and combine them in the horizontal attack framework, as outlined below.

The framework consists roughly of the following phases:

1. 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 located (known as points of interest or POIs).
2. Next, key recovery (KR) is run, yielding an approximate scalar. This scalar is expected to be incorrect (due to some error bits) but better than random.
3. Given the approximate scalar, points-of-interest optimization (POI-OPT) produces a refined list of POIs.

[5] We note that for ECSM, it is sometimes necessary to distinguish even down to field multiplications.

[6] Observe that there exist algorithms that can correct such errors faster than the trivial brute force, for example, [RIL19, HMM10]. However, to execute those algorithms can be very costly (for details, see Section 3.4).

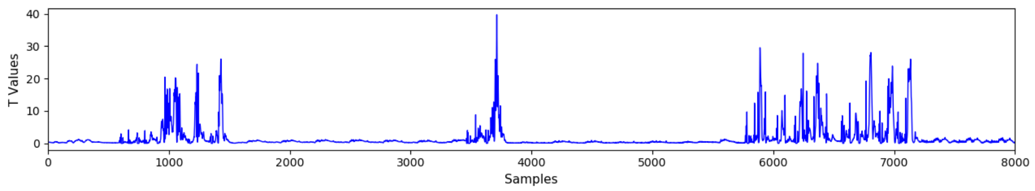


Figure 3: The result of CLA for Cswap-Arith: the leakage metric trace.

4. Finally, the final KR step outputs the recovered scalars for each ECSM.

The above steps are described in detail in [PC15] and further expanded on in [NC17].

This framework is parametrized in [NC17] in the following way.

- **Step 1** can be run with various clustering algorithms: k-means [Alp10], fuzzy k-means [Alp10], or Expectation-Maximization [Bis06].
  Additionally, many outlier detections (for example, Turkey test) and handling methods (e.g., replace the outlier with the median) can be chosen. These methods are used to reduce occasional noise peaks.
  The results of the clustering algorithm are used to make a leakage metric trace using one of the following distinguishers: sum-of-squared differences (SOSD), sum-of-squared t-values (SOST), and mutual information analysis (MIA).
- **Step 2** is parametrized similarly to Step 1 with: clustering algorithms, outliers detectors, and handlers. Additionally, it takes the number of POIs as a parameter (the POIs are chosen based on the leakage metric trace) and an algorithm that combines clustering results. This algorithm can be either majority rule or log-likelihood. Instead of single-dimensional clustering for each POI, it is also possible to use multi-dimensional clustering and then combining algorithm is not necessary.
- **Step 3** is not parametric. It only runs the Welch's t-test on the result of Step 2.
- **Step 4** is parametrized in the same way as Step 2. Note that Steps 4 can be parametrized partially or completely differently than Step 2.

To simplify the proposed iterative framework (Section 4) and limit the number of parameters, we stop the attack just after Step 2. We also try to choose the most straightforward parameters while still obtaining better than random results. We run CLA (Step 1) with k-means, SOST, and Turkey test with a median replacement on 100 traces. The resulting leakage metric trace for Cswap-Arith is presented in Figure 3. Since the highest peaks are not in the same offsets as for the correct t-test trace from Figure 6, we should not expect the attack to work well, but any result better than random should be sufficient; the situation is similar for the Cswap-Pointer implementation. KR (Step 1) with k-means for 20 best POIs (selected from the result of CLA), and majority rule are used to combine 20 k-means results.

The results of this simplified horizontal attack are as follows.

- **Pointer-Cswap:** the resulting accuracy for 300 traces is relatively low: 0.5224. However, it is consistent for both recovered groups of scalar bits: 0.5253 for bit 0 and 0.5195 for bit 1. Furthermore, when we run t-test on the recovered labels, then a few peaks indicate leakage, as presented in Figure 6.
- **Arith-Cswap:** the resulting accuracy for 300 traces is relatively low: 0.5244. For the pointer case, the accuracy is consistent for both recovered groups: 0.5234 for bit 0 and 0.5253 for bit 1. Moreover, when we run t-test on the recovered labels, then several peaks indicating weak leakage are visible, as presented in Figure 6. Observe these peaks seem slightly less pronounced than in the Pointer-Cswap case.

The achieved accuracy is low (approximately 52%), but we find it sufficient for the deep learning method presented in this paper to correct the labels. Moreover, observe that weak leakage can be visible using a t-test, as presented in Figure 6. The resulting datasets and their accuracy values are described in Section 2.2.

Note that the full procedure presented in [NC17] can recover most of the bits for at least one scalar out of 100 attacked ones. Afterward, a single correct scalar can be recovered by brute-forcing using approximately 2^{45} operations for cswap-arith and 2^{22} for cswap-pointer. This brute-force step is not necessary for our approach since we can achieve 100% using a deep learning approach.

3.4 Methods to Remove Noisy Bits from Public-key Implementations

Let us assume that an attacker aims to perform a single trace attack on several scalar multiplications (or modular exponentiations) from a device operating an ECC or RSA protocol. First by measuring the corresponding side-channel leakages, the attacker acquires a set $\{T_N\}$ of single traces. Second they apply a single trace attack, either a profiled or non-profiled, on these traces and obtains a set of blinded scalar (or exponent) values $\{d'_i\}$, for $1 \leq i \leq N$.

Due to noise and other aspects interfering with the side-channel analysis (misalignment, for example), the derived scalar (or exponent) might contain multiple errors. The amount $e_i = HD(d'_i, d_i)$ represents the amount of wrongly recovered bits contained in each d'_i, where HD is the Hamming Distance. In an unsupervised setting, the number of error bits cannot be precisely estimated by an attacker, as well as the indices of the wrong bits in d'_i. An attacker can obtain the probability that a bit is either 0 or 1 from the used SCA attack and use them for faster recovery as presented in [HIM$^+$13], for example. However, this technique does not work in a more noisy environment due to many high-confidence false-positives, as presented in [NC17]. Even for a profiled attack, it is not easy to use these probabilities in a noisy setting [NCOS16].

A standard approach to speed up the trivial brute-force is to use a meet-in-the-middle approach. It is a space-time trade-off cryptographic attack that exploits the fact that multiple encryption operations are performed in sequence. The attack essentially halves the effort necessary for the trivial brute-force, but it requires the amount of memory of exactly that size (i.e., half of the brute-force size). This approach was employed to recover the full private key from its partial knowledge for exponentiation in [GTY07] and scalar multiplication in [NCOS16].

Alternatively, it may be possible to correct the private key with an informed brute force attack from [LvVW15]. Unfortunately, this attack works well if the bits containing errors are adjacent to each other, which is often not the case.

For RSA, there exist an algorithm that can correct multiple errors in a private key (in the exponent and the prime factors) by exploiting RSA mathematical properties [HMM10]. This algorithm utilizes the fact that the public and private key of RSA contain a lot of redundancy and works especially well for RSA-CRT. The attack does not consider exponent blinding or splitting, but there exists a later attack against exponent blinding for RSA-CRT [SW17].

Roche et al. in [RIL19] proposed the most efficient technique to the best of our knowledge. It improves the original solution from [SI11] and recovers a secret scalar by combining information from many noisy blinded scalars (e.g., outputs of horizontal attacks). It even works reasonably efficiently when blinding factors are large (e.g., ≥ 32 bits) and with a significant bit error rate of $10\% - 15\%$. In our setting, this kind of correcting algorithm can be applied not after the horizontal attack but after the deep learning phase. However, this step is not necessary for the results presented in this paper since we reach a 100% recovery for at least some scalars for both implementations.

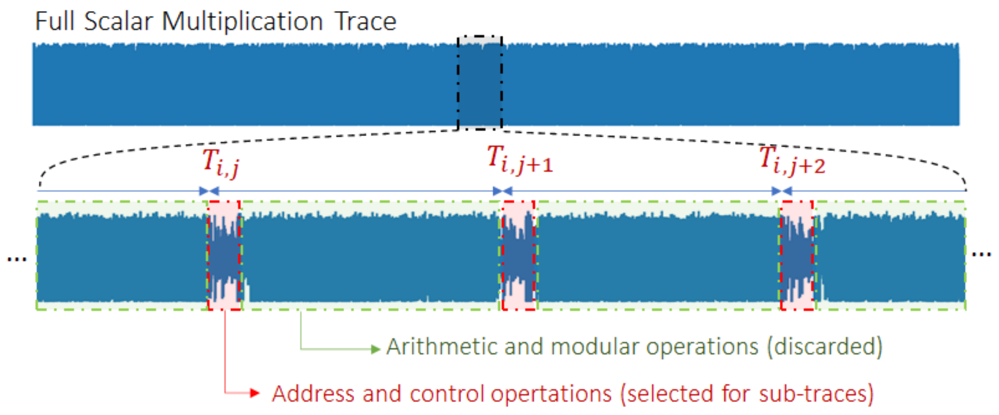

Full Scalar Multiplication Trace
$T_{i,j}$
$T_{i,j+1}$
$T_{i,j+2}$
...
...
Arithmetic and modular operations (discarded)
Address and control opertations (selected for sub-traces)

Figure 4: Trace preprocessing for horizontal attacks. The full scalar multiplication traces are split into sub-intervals, each one framing the processing of one scalar bit. The intervals indicated by $T_{i,j}$ (red area) are selected as sub-traces. Green areas are discarded as these parts present no target leakage.

4 Proposed Iterative Deep Learning Framework

This section proposes an iterative deep learning framework that can correct error bits from private keys derived from a horizontal attack. The proposed iterative framework keeps the attack completely unsupervised as no knowledge about secret bits (i.e., labels) is assumed since the first attack steps. The framework contains an initialization step, where traces are prepared and labeled with a horizontal attack, and the main part, which consists of an iterative process.

4.1 Initialization

The input to the framework is a set of scalar multiplication traces (for RSA applications, it can be replaced by modular exponentiation traces), containing all the sub-operations representing the full secret scalar's processing. Thus, the scalar multiplication traces T_i, $i \in [0, N-1]$, are split into sub-traces $T_{i,j}$, each one representing the time interval of the processing of a single scalar bit. The term j indicates the index of the scalar bit. For each trace T_i, the scalar may be randomized using scalar blinding countermeasure.

After full scalar multiplication traces are split into sub-traces, as illustrated in Figure 4, a horizontal attack is applied to the sub-traces $T_{i,j}$ to define initial labels $Y_{i,j}$ for each sub-trace. The output of a horizontal attack is dataset $D = \{T_{i,j}, Y_{i,j}\}$ labeled according to two possible classes, 0 and 1.

4.2 Main Part of the Framework

Figure 5 illustrates the iterative deep learning framework, where every iteration implements the following four steps:

- **Phase 1– Prepare sets D_1 and D_2:** The full set $D = \{T_{i,j}, Y_{i,j}\}$ is split into an initial training set, D_1, and a subsequent test set, D_2. In our case, the number of sub-traces in each one of these subsets are identical.
- **Phase 2– Train on D_1, predict and relabel D_2:** a deep neural network is trained with D_1, with labels Y_1, and tested or predicted on D_2. The softmax output probabilities obtained by predicting D_2 are then used to re-label Y_2 itself.

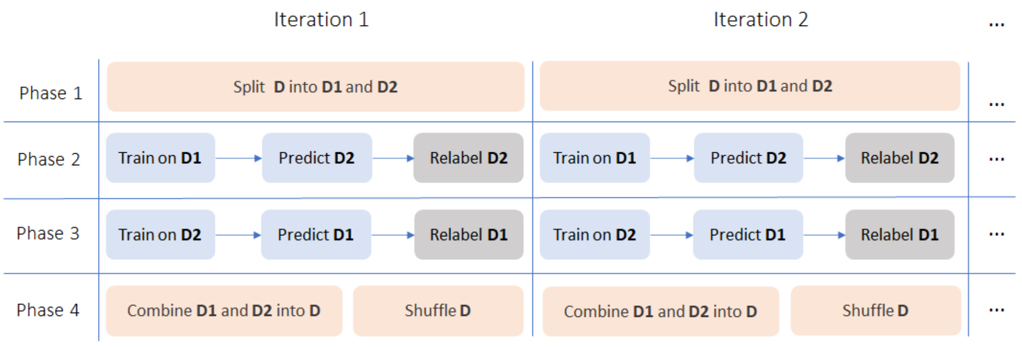


Figure 5: Proposed iterative framework

- **Phase 3– Train on D_2, predict and relabel D_1:** In the next step, the process works in the opposite direction. The deep neural network is trained with the re-labeled D_2, having new labels Y_2, and tested or predicted on D_1. This time, the softmax output probabilities obtained from D_1 are used to re-label Y_1 itself.
- **Phase 4– Joint D_1 and D_2 and shuffle the full set:** this last phase of an iteration combine both relabelled sets D_1 and D_2 into a single dataset $D = \{T_{i,j}, Y_{i,j}\}$. Finally, the combined and shuffled dataset D is again split into D_1 and D_2 as a preparation for Phase 1 of next iteration.

After the deep neural networks are trained on D_1 and D_2 in Phases 2 and 3, respectively, a separated test set is used as target traces. This test set can be composed of scalar multiplication traces that are not used to implement D_1 and D_2.

This procedure continues iteratively until a successful attack is achieved. In every step of this iterative process, it is expected that the amount of noisy labels decreases as a result of deep neural networks learning side-channel leakages from the limited correct labels in the training set. Of course, the higher the error bits in the initial training set, the more iterations we expect to need to reach a successful attack. For example, to determine when the iteration framework should stop, the attacker test N scalar multiplication traces T_i, $i \in [0, N-1]$, with the known public parameters.

5 Results

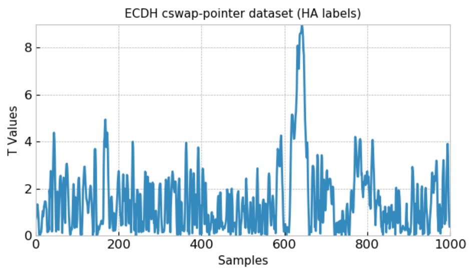
This section provides several results on different applications of the proposed iteration framework. We start by the crucial preliminary step, i.e., to apply a horizontal attack method to the two target datasets. We start with a leakage assessment on the datasets to demonstrate the leakage occurrence, given by t-test peaks, that the framework faces with these datasets. Next, we apply a cluster-based horizontal attack on the two datasets to have initial labels for the framework. Finally, we apply the proposed framework with different variations. The proposed framework is very generic because it does not restrict the type of learning algorithm to be used in each iteration. Therefore, we apply different variations of CNN architectures together with different types of regularization techniques.

5.1 Labeling Traces from Clustering-based Horizontal Attack

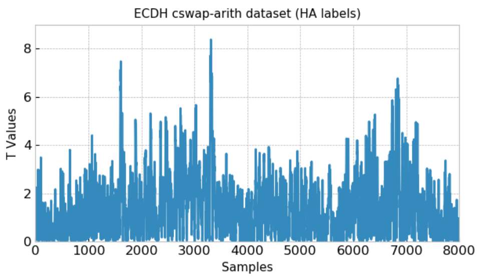
The first phase considers the application of a horizontal attack to the datasets. Here, we adopted clustering-based horizontal attacks [PC15]. In this first phase, the goal is to provide initial labels to the sub-traces. For both datasets, *Cswap-Pointer* and *Cswap-Arith*, we only apply a short version of the horizontal attack, as described in Section 3.3. This

Table 1: Horizontal attack results

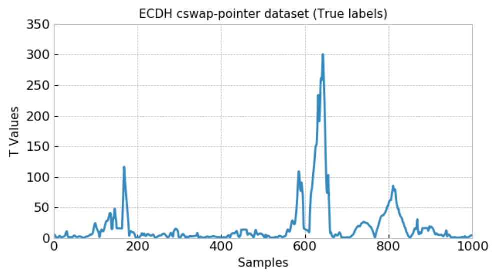
Dataset	Scalar bits	ECSM Traces	K-means Accuracy
Cswap-Arith	255	250	52.24%
Cswap-Pointer	255	250	52.44%



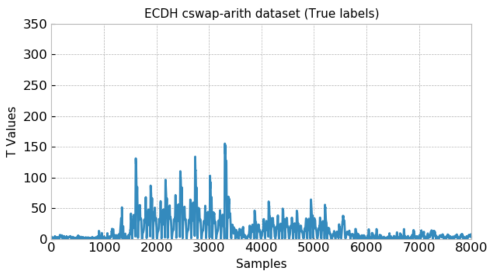
(a) t-test on cswap-pointer dataset with HA labels.



(b) t-test on cswap-arith dataset with HA labels.



(c) t-test on cswap-pointer dataset with true labels.



(d) t-test on cswap-arith dataset with true labels.

Figure 6: t-test on the two considered dataset.

is essentially the attack from [NC17] without optimization. Table 1 details the labeling accuracy obtained from clustering-based horizontal attacks on both datasets.

As we can see in Table 1, the labeling accuracy is close to 52% for both datasets, although it seems that *Cswap-pointer* contains a bit more side-channel leakages compared to *Cswap-Arith*, as visible in Figure 6. This first phase of the subsequent attack is applied to 250 ECSM trace sets. From a total of 300 traces, 50 traces are left for test purposes in Phase 2.

5.2 Leakage Assessment on *cswap-arith* and *cswap-pointer* Datasets

This section presents a leakage assessment on the datasets used in this paper. This analysis's main reason is to demonstrate how much leakage can be captured by standard leakage assessment methods. It is important to emphasize that the proposed framework does not assume any knowledge from this leakage assessment as we want to keep the analysis fully unsupervised. The leakage assessment is based on t-test, and results are shown in Figure 6. This figure shows t-test peaks with labels obtained from the cluster-based horizontal attack and from correct labels. Note that labels from the horizontal attack can already indicate leakage detection, as shown in Figures 6b and 6b, for *cswap-arith* and *cswap-pointer* datasets, respectively.

Table 2: CNN architectures considered for the two datasets described in Section 2.2

Layer	cswap-arith	cswap-pointer
Input	input_size = 8 000	input_size = 1 000
AvgPooling1D	pool size=4, stride=4	-
Conv1D_1	8 filters, ks=20, stride=1	8 filters, ks=40, stride=4
Conv1D_2	8 filters, ks=20, stride=1	8 filters, ks=40, stride=4
Conv1D_3	8 filters, ks=20, stride=1	8 filters, ks=40, stride=4
Dense_1	100 Neurons	100 Neurons
Dense_2	100 Neurons	100 Neurons
Softmax	2 Neurons	2 Neurons

5.3 Iterative Framework Application on Different Cases

This section provides experimental results for the proposed iterative framework. The selected learning algorithm for steps 2 and 3 is a convolutional neural network. First, in subsection 5.3.1, we present results when the CNN hyperparameters are fixed in all the framework iterations. There, we also introduce the four scenarios in which the framework is applied. After, in subsection 5.3.2, we improve the learning algorithm's variability by randomizing the CNN hyperparameters in every framework iteration. In the random hyperparameters case, we also present results for the four application scenarios described in subsection 5.3.1. We decide to use CNNs as a deep neural network instead of, e.g., a multilayer perceptron for the simple reason that CNNs are more robust to jitter-based effects in side-channel traces, as already demonstrated in [CDP17].

Maximum single trace accuracy metric: for the provided results on *cswap-arith* and *cswap-pointer* datasets, we separate 50 scalar multiplication traces, each one containing 255 sub-traces, one for each scalar bit. The metric to estimate the proposed iterative framework's performance is the maximum accuracy obtained by testing the 50 tested scalar multiplication traces. *Note that these 50 traces are never used in training datasets D_1 and D_2, as they are separated traces.*

5.3.1 Fixed CNN Hyperparameters

In the first case, we consider two CNN configuration during all iterations of the framework, with and without dropout layers [SHK+14], as detailed in Tables 2 and 3. A different configuration of convolution layers is defined for each dataset. For all the layers, *ReLU* activation function is chosen. The weights for the two dense layers are always initialized with *random uniform* method, while for the remaining layers, the weight initialization considers *glorot uniform* method. *RMSprop* is used as the stochastic gradient descent optimizer with a learning rate of $1e-5$. Loss function is set to *categorical cross-entropy*. The CNNs are always trained for ten epochs, as we observed that training the networks for more epochs provides overfitting and degrades the results.

As the *cswap-arith* dataset contains sub-traces with 8 000 samples each, we define as the first layer an *AveragePooling1D* layer to implement dimensionality reduction identical to window resampling. This way, we reduce the number of trainable parameters (weights and biases) in the deep neural network and also eliminate part of jitter present in measurements.

Having defined the two CNN architectures in Tables 2 and 3, the iterative framework is applied on both datasets, for 50 iterations, on the following four scenarios:

1. **CNN (Table 2) *without* regularization:** in each framework iteration, the CNN is trained without any type of regularization technique;
2. **CNN (Table 2) *with* data augmentation:** in this case, shifted-based data augmentation is considered. Every time a batch of 100 sub-traces is processed during

Table 3: CNN architectures without *dropout* layers for the two datasets described in Section 2.2.

Layer	**cswap-arith**	**cswap-pointer**
Input	input_size = 8 000	input_size = 1 000
AvgPooling1D	pool size=4, stride=4	-
BN Layer	Batch Normalization	Batch Normalization
Conv1D_1	8 filters, ks=20, stride=1	8 filters, ks=40, stride=4
Conv1D_2	8 filters, ks=20, stride=1	8 filters, ks=40, stride=4
Conv1D_3	8 filters, ks=20, stride=1	8 filters, ks=40, stride=4
Dropout_1	Dropout Rate=0.5	Dropout Rate=0.5
Dense_1	100 Neurons	100 Neurons
Dropout_2	Dropout Rate=0.5	Dropout Rate=0.5
Dense_2	100 Neurons	100 Neurons
Softmax	2 Neurons	2 Neurons

training, we randomly apply shifts of 5 samples to the right or the left in the x-axis. The sub-traces were already aligned before the application of horizontal attacks in order to have initial labels. However, small jitter is still present in the sub-traces, and data augmentation with minimal shifts of 5 samples showed satisfactory results. For each epoch, the CNN processes augmented mini-batch 200 times;

3. **CNN with dropout (Table 3) and *without* data augmentation:** in this case, the only type of regularization against overfitting are the two introduced dropout layers with a dropout rate of 0.5, which is a relatively high rate in order to provide a significant level of regularization;
4. **CNN with dropout (Table 3) and *with* data augmentation:** in this last case, we combine dropout layers with data augmentation using the same definitions as described in Scenario 2;

We expect that dropout and data augmentation regularization methods reduce the overfitting that can happen during iterations. The overfitting likely happens due to training an identical deep neural network after relabeling the datasets based on predictions obtained from this same neural network. Recall that regularization is a widely adopted technique used in the machine learning community to deal with noisy labels.

For each scenario described above, we run the iterative framework ten times and average the results. Figures 7 and 8 show the average, minimum and maximum single trace test accuracy for the two considered datasets. When attacking *cswap-arith* dataset, the iterative framework can reach 100% of maximum single trace accuracy only when dropout layers are considered together with data augmentation. For the other three scenarios, the maximum achieved accuracy values are 92.94%, 78.43%, and 76.07% for dropout only, data augmentation only, and no regularization, respectively.

Results for *cswap-pointer* dataset provided better results for all four scenarios. For this dataset, we obtained 100% of the maximum single test trace accuracy for the three cases with some type of regularization. Only when any type of regularization is considered, the framework achieved a maximum accuracy of 97.64% after 50 iterations. As a highlight, results for *cswap-pointer* case demonstrated that the framework was able to return 100% accuracy for 7 out of 10 framework executions when data augmentation is used as the only regularization method. Table 4 provide the frequency in which the iterative framework achieves 100% in each scenario for both datasets (In Appendix B Figures 17 and 18 show the maximum single test trace accuracy obtained for each framework execution for the two datasets).

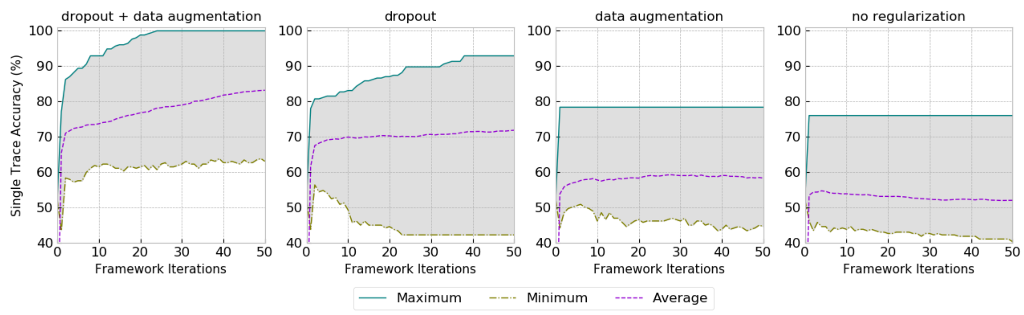


Figure 7: Minimum, maximum and average single trace accuracy with iterative framework on *cswap-arith* dataset.

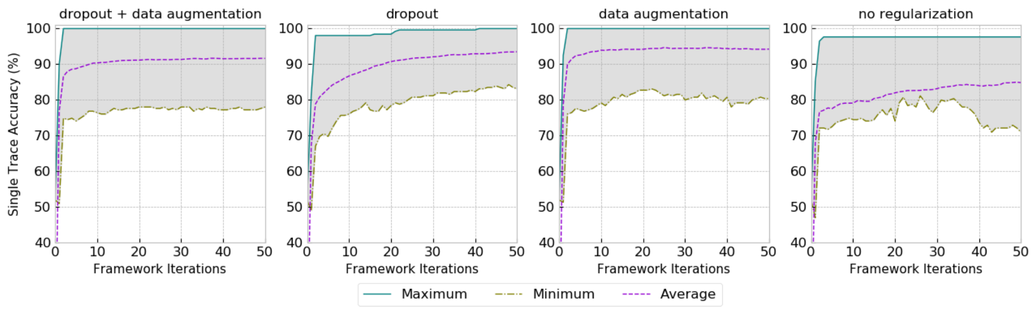


Figure 8: Minimum, maximum and average single trace accuracy with iterative framework on *cswap-pointer* dataset.

The better results for different cases is explained by the high t-test peaks in Figures 6a and 6c, indicating a higher occurrence of address-like leakages in this scalar multiplication implementation. Although the t-test peaks for this implementation are higher compared to *cswap-arith* implementation, a clustering-based horizontal attack was able to return a maximum of 52.44% accuracy for this dataset. Without any method to deal with noisy labels, like regularization, the iterative framework's application could not achieve 100% for one of the 50 tested scalar multiplication traces. However, as the leakage is relatively higher for this dataset compared to *cswap-arith*, we can obtain a successful attack on the *cswap-pointer* dataset in all scenarios with regularization. For the *cswap-arith* dataset, higher successful results are achieved when random CNN hyperparameters and/or the attack interval is optimized with gradient visualization, as detailed in the following sections.

5.3.2 Random CNN Hyperparameters

In this section, the CNN hyperparameters will vary before every framework iteration. For both datasets, the selected CNN architectures have the same structure from the CNNs presented in Tables 2 and 3, except that we vary the following hyperparameters:

- *Number of filters*: in the first convolution layer (Conv1D_1), the number of filters is randomly selected between 4 and 8. The two subsequent convolution layers (Conv1D_2 and (Conv1D_3) will have number filters equal to two and four times, respectively, the number of filters randomly defined in Conv1D_1. This proportion is kept to improve feature extraction from side-channel traces.
- *Kernel size and strides*: for the three convolution layers (Conv1D_1, Conv1D_2 and (Conv1D_3), the kernel size is the same and randomly selected between 10 and 40. Similarly, the three convolution layers' strides are the same and randomly selected

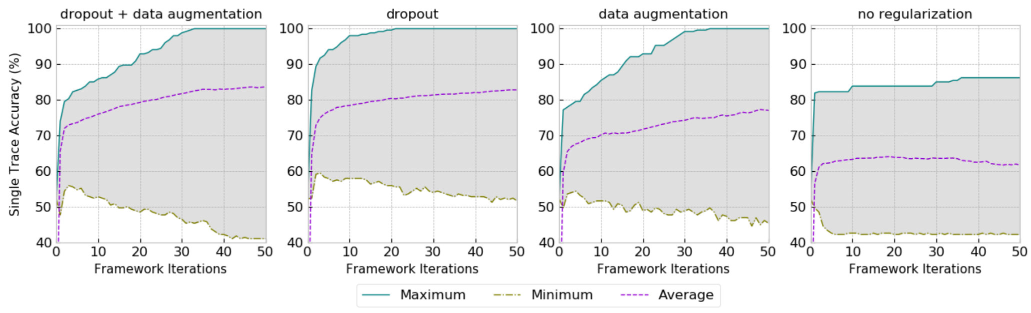


Figure 9: Random CNN hyperparameters: minimum, maximum and average single trace accuracy with iterative framework on *cswap-arith* dataset.

between 1 and 4.

- *Fully-connected Layers*: The number of fully-connected layers is randomly selected between 1 and 5. The number of neurons for all fully-connected layers is randomly defined between 100 and 400.
- *Activation Function*: The activation function for all the layers is randomly selected from 'ReLU', 'Tanh', 'SELU' or 'ELU'.

The main reason to randomize the CNN configuration before every iteration is to reduce the same model's overfitting in a subsequent iteration. Retraining an identical CNN with datasets relabeled in the previous iteration can bias the model in the next iteration, reducing the framework's chances to improve the label's correctness during the framework execution consistently.

Figures 9 and 10 show results for both datasets. These figures show an average of ten framework executions and the maximum and minimum single trace test accuracy. As we can see, without any type of regularization, the maximum achieved single trace accuracies are 86.27% and 96.47% for *cswap-arith* and *cswap-pointer* datasets, respectively. Combining random CNN hyperparameters with regularization techniques improves the results on the *cswap-arith* dataset significantly. We can achieve 100% of single trace test accuracy for all three different scenarios with some type of regularization. Especially the case when dropout is combined with data augmentation, running the framework ten times on *cswap-arith* dataset resulted in 100% of test accuracy in 6 cases.

For the *cswap-pointer* dataset, we were able to achieve 100% of single trace test accuracy when any type of regularization is used. For this last dataset, random CNN hyperparameters presented similar results in comparison to fixed CNN hyperparameters. Results with data augmentation only were more successful when fixed CNN hyperparameters are considered, as detailed in Table 4. On the other hand, the combination of dropout and data augmentation for random CNN hyperparameters achieved superior results than the fixed CNN hyperparameters case. For this specific scenario, we were able to achieve 100% test accuracy in 4 out of 10 framework executions.

5.4 Using Gradient Visualization to Optimize Attack Interval

The proposed iterative framework's attack performance can be significantly increased by narrowing down the samples interval in sub-traces. One alternative could be to split sub-traces into smaller sample intervals and attack each one of them at a time. However, this would increase computational time and render the analysis too slow. Therefore, in this section, we propose using a deep learning tool based on gradient visualization to optimize the attack interval and, as a consequence, reduce iterative framework computational complexity.

Gradient visualization (GV) is conducted by analyzing what input features (given by

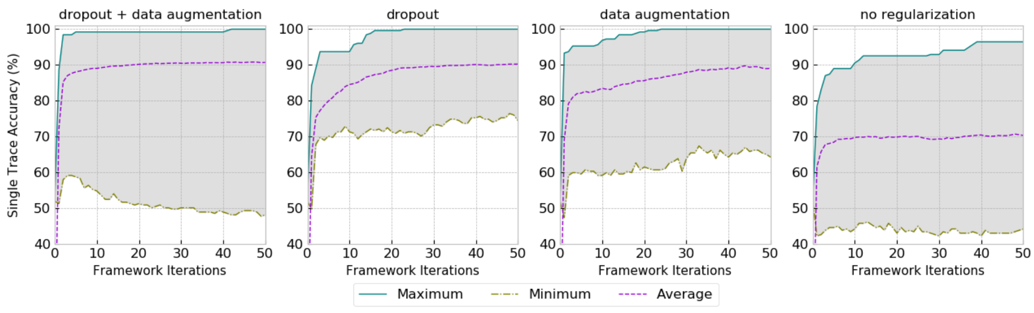


Figure 10: Random CNN hyperparameters: minimum, maximum and average single trace accuracy with iterative framework on *cswap-pointer* dataset for random CNN hyperparameters.

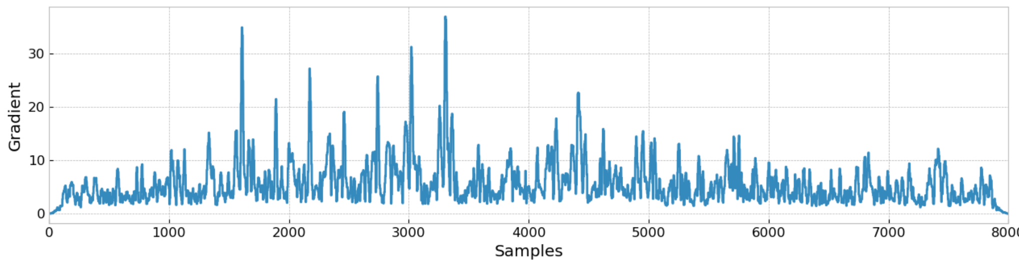


Figure 11: Input gradient visualization from *cswap-arith* dataset

a neuron in the input deep neural network layer as a one-to-one mapping) have more influence in classification during training. It is a technique that computes the values of derivatives in a neural network regarding the input trace. These derivatives are then used to point out what feature needs to be modified the least to affect the loss function the most. In [MDP19], the authors proposed the visualization of input activation gradients as a technique to characterize the automated selection of points of interest by deep neural networks. The result is a vector of gradients computed by the backpropagation algorithm as the derivative of the cost function concerning the input activation.

When applied to our framework, the gradient needs to consider the labels for sub-traces obtained after each framework iteration. For the initial iterations, the label correctness can be very low, and the gradients in these first iterations will most likely indicate unclear results in terms of leakage location. However, by summing up the gradients obtained for all iterations, we can obtain satisfactory and clear results.

Figures 11 and 12 show gradient peaks for both datasets. The gradient values were obtained from CNNs trained on both datasets when fixed CNN hyper-parameters are used in all framework iterations. We can clearly observe that gradient peaks correspond to the leakage occurrence given by t-test results from 6. As a result, an adversary can narrow down the attack interval in sub-traces. For *cwsap-pointer* dataset, Figure 12 indicates high gradient peaks from sample 550 to sample 800. In case of *cwsap-arith* dataset, Figure 11 shows larger gradients from sample 1 500 to 3 500.

Figures 13 and 14 show results for fixed and random CNN hyper-parameters, respectively, on *cswap-arith* dataset. The iterative framework considers only sample interval range from 1 500 to 3 500 of all sub-traces, which is trace interval where the leakage is more significant. The CNN configurations are the same as provided in Tables 2 and 3. Now, we can reach 100% accuracy for all four scenarios, with and without regularization, as detailed in Tables 4 and 5. Without regularization, 1 out of 10 framework executions

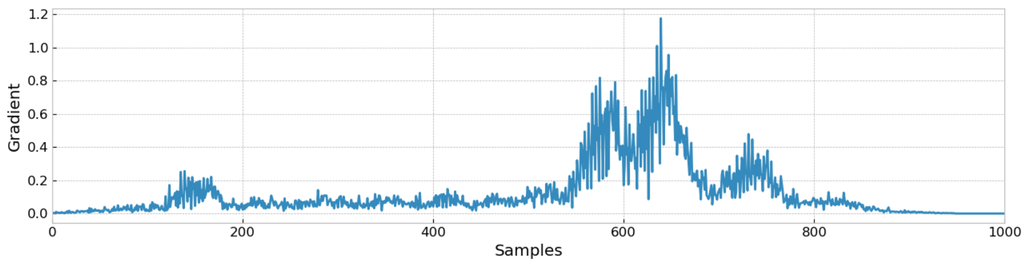


Figure 12: Input gradient visualization from *cswap-pointer* dataset

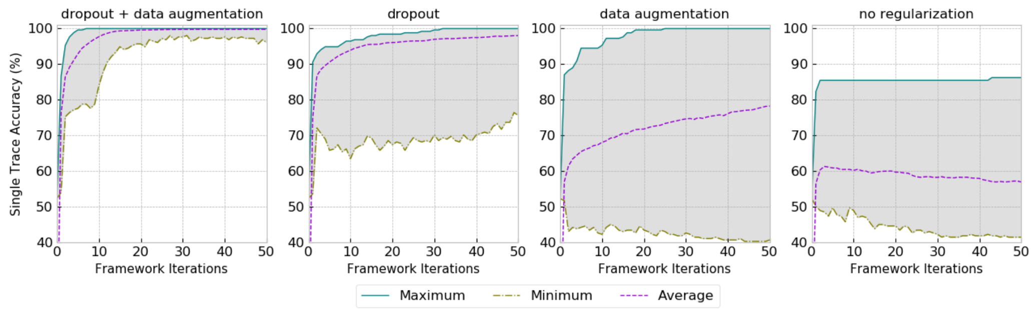


Figure 13: Fixed CNN hyperparameters: minimum, maximum and average single trace accuracy with iterative framework on *cswap-arith* dataset and attacking the sample interval range from 1 500 to 3 500 of all sub-traces.

achieved 100% of single trace test accuracy. When no regularization is in place, we could not achieved 100% for all 10 framework executions.

Figures 15 and 16 show results for fixed and random CNN hyper-parameters, respectively, on *cswap-pointer* dataset. After using gradient visualization to narrow down the interval for cswap-pointer dataset, we achieved 100% of maximum test accuracy for the three scenarios with regularization. Again, only when no regularization method is adopted, the test accuracy reaches 94.50% and 96.47% for the cases when fixed, and random CNN hyperparameters are considered on an optimal interval, respectively. In the end, narrowing down the attack interval for the cswap-pointer dataset did not lead to superior results, indicating that the original interval of 1 000 samples in sub-traces is needed for the CNN to fit the existing leakages. On the other hand, for *cswap-arith* dataset, the improvement after optimizing the interval is very significant.

Table 5 also summarizes the maximum single trace accuracy results obtained before and after considering gradient visualization to optimize the attack interval.

6 Conclusions and Future Works

This paper presented a novel deep learning-based iterative framework to correct the remaining error bits resulting from horizontal attacks. As discussed in this work, horizontal attacks are the only applicable method against protected public-key implementations, and the only alternative for attackers is to recover the full secret from a single trace. As horizontal attacks face many limitations in practice, it is common to return results with a high and unknown number of error bits. We show that deep learning techniques, through an iterative process, can continuously improve the correctness of labels in a dataset with a high number of noisy bits, given by wrong target scalar bits. From a cluster-based horizontal attack, which provided very poor accuracy of 52% for two datasets, our framework was

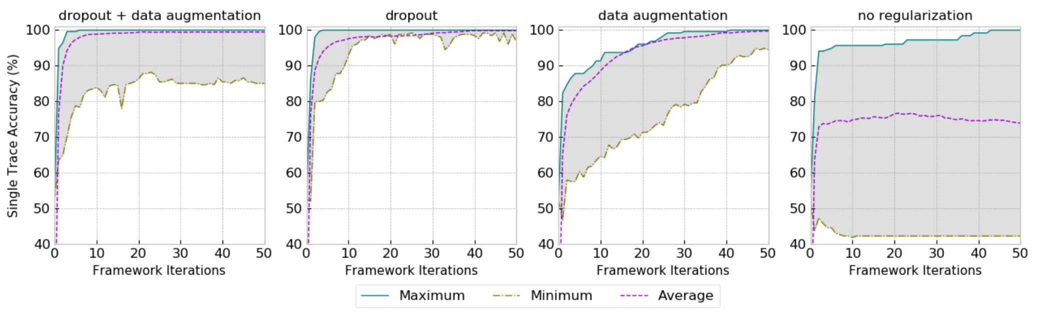


Figure 14: Random CNN hyperparameters: minimum, maximum, and average single trace accuracy with iterative framework on *cswap-arith* dataset and attacking the sample interval range from 1 500 to 3 500 of all sub-traces.

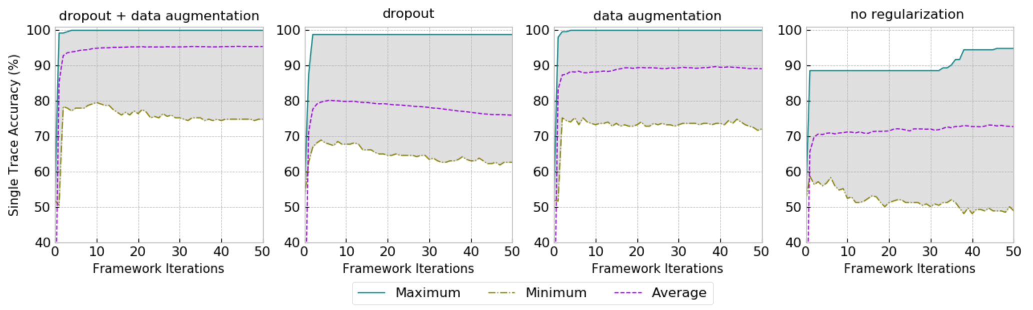


Figure 15: Fixed CNN hyperparameters: minimum, maximum and average single trace accuracy with iterative framework on *cswap-pointer* dataset and attacking the sample interval range from 550 to 800 of all sub-traces.

able to return 100% of correct blind and secret scalars. For that, we made use of deep learning techniques such as dropout, data augmentation, and gradient visualization.

Nevertheless, this work leaves space for several future research directions. As regularization based on dropout layers and data augmentation was highly efficient in our application cases, we suggest upgrading the proposed framework's regularization techniques. For that, we suggest proposing a metric for early stopping while keeping the framework completely unsupervised. Another direction would be to adopt ensembles for the iterative framework to reduce error variability by combining results for several models in each framework iteration [PCP19]. Finally, an interesting future research direction would be to create a noisy transition matrix based on classification probabilities for each bit in target scalar multiplication traces. With this method, it would be possible to identify, with some probability, the location of wrong bits in final results.

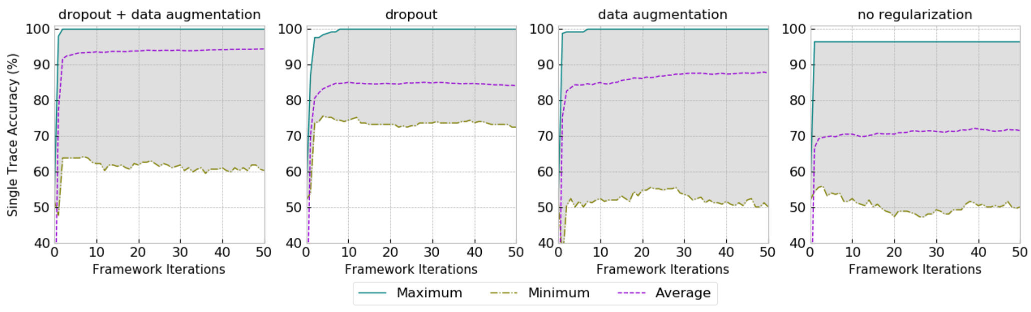


Figure 16: Random CNN hyperparameters: minimum, maximum, and average single trace accuracy with iterative framework on *cswap-pointer* dataset and attacking the sample interval range from 550 to 800 of all sub-traces.

Table 4: Summary of 10 iterative framework executions: frequency that the framework achieves 100% of maximum single trace test accucacy, for fixed and random CNN hyper-parameters. Results are provided for full sample intervals and after selecting an optimal interval gradient visualization (GV) results.

Fixed CNN Hyperparameters				
	cswap-arith		**cswap-pointer**	
Method	Full interval	GV Optimized	Full interval	GV Optimized
CNN No regularization	0/10	0/10	0/10	0/10
CNN Dropout	0/10	8/10	1/10	1/10
CNN Data Augmentation	0/10	2/10	7/10	2/10
CNN Dropout + Data Augmentation	1/10	9/10	2/10	4/10
Random CNN Hyperparameters				
	cswap-arith		**cswap-pointer**	
Method	Full interval	GV Optimized	Full interval	GV Optimized
CNN No regularization	0/10	1/10	0/10	0/10
CNN Dropout	4/10	10/10	1/10	2/10
CNN Data Augmentation	1/10	10/10	2/10	1/10
CNN Dropout + Data Augmentation	6/10	10/10	4/10	3/10

Table 5: Summary of iterative framework results: maximum single trace accuracy achieved from 10 framework executions. Results for *cswap-arith* and *cswap-pointer* datasets for full sample intervals and after selecting an optimal interval gradient visualization (GV) results.

Fixed CNN Hyperparameters				
	cswap-arith		**cswap-pointer**	
Method	Full interval	GV Optimized	Full interval	GV Optimized
CNN No regularization	76.07%	86.27%	97.64%	94.50%
CNN Dropout	92.94%	**100%**	**100%**	98.82%
CNN Data Augmentation	78.43%	**100%**	**100%**	**100%**
CNN Dropout + Data Augmentation	**100%**	**100%**	**100%**	**100%**
Random CNN Hyperparameters				
	cswap-arith		**cswap-pointer**	
Method	Full interval	GV Optimized	Full interval	GV Optimized
CNN No regularization	86.27%	**100%**	96.47%	96.47%
CNN Dropout	**100%**	**100%**	**100%**	**100%**
CNN Data Augmentation	**100%**	**100%**	**100%**	**100%**
CNN Dropout + Data Augmentation	**100%**	**100%**	**100%**	**100%**

References

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning.* The MIT Press, 2nd edition, 2010.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BCP+17] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. *Journal of Cryptographic Engineering*, August 2017.

[Ber06] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BJP+15] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, Jean-René Reinhard, and Justine Wild. Horizontal collision correlation attack on elliptic curves - - extended version -. *Cryptogr. Commun.*, 7(1):91–119, 2015.

[BJPW13] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal collision correlation attack on elliptic curves. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 553–570. Springer, 2013.

[CCC+19] Mathieu Carbone, Vincent Conin, Marie-Angela Cornelie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff, and Alexandre Venelli. Deep learning to evaluate secure RSA implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):132–161, 2019.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings*, volume 6476 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2010.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, pages 292–302, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[GTY07] K. Gopalakrishnan, Nicolas Thériault, and Chui Zhi Yao. Solving discrete logarithms from partial knowledge of the key. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *Progress in Cryptology – INDOCRYPT 2007*, pages 224–237, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[GV10] Christophe Giraud and Vincent Verneuil. Atomicity improvement for elliptic curve scalar multiplication. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 14-16, 2010. Proceedings*, volume 6035 of *Lecture Notes in Computer Science*, pages 80–101. Springer, 2010.

[HIM$^+$13] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering algorithms for non-profiled single-execution attacks on exponentiations. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 79–93. Springer, 2013.

[HMM10] Wilko Henecka, Alexander May, and Alexander Meurer. Correcting errors in RSA private keys. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference*, volume 6223 of *Lecture Notes in Computer Science*, pages 351–369. Springer, 2010.

[HYY$^+$18] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8536–8546, Red Hook, NY, USA, 2018. Curran Associates Inc.

[JNC16] Ishan Jindal, Matthew S. Nokleby, and Xuewen Chen. Learning deep networks from noisy labels with dropout regularization. *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 967–972, 2016.

[JY02] Marc Joye and Sung-Ming Yen. The montgomery powering ladder. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KPH$^+$19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):148–179, 2019.

[LvVW15] Tanja Lange, Christine van Vredendaal, and Marnix Wakker. Kangaroos in side-channel attacks. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications*, pages 104–121, Cham, 2015. Springer International Publishing.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 145–167. Springer, 2019.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016.

[NC17] Erick Nascimento and Lukasz Chmielewski. Applying horizontal clustering side-channel attacks on embedded ECC implementations. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, volume 10728 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2017.

[NCOS16] Erick Nascimento, Lukasz Chmielewski, David Oswald, and Peter Schwabe. Attacking embedded ECC implementations through cmov side channels. In Roberto Avanzi and Howard M. Heys, editors, *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, volume 10532 of *Lecture Notes in Computer Science*, pages 99–119. Springer, 2016.

[PC15] Guilherme Perin and Lukasz Chmielewski. A semi-parametric approach for side-channel attacks on protected RSA implementations. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 34–53. Springer, 2015.

[PCP19] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/978, 2019. https://eprint.iacr.org/2019/978.

[PITM14] Guilherme Perin, Laurent Imbert, Lionel Torres, and Philippe Maurine. Attacking randomized exponentiations using unsupervised learning. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2014.

[RIL19] Thomas Roche, Laurent Imbert, and Victor Lomné. Side-channel attacks on blinded scalar multiplications revisited. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2019.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[SHKS15] Robert Specht, Johann Heyszl, Martin Kleinsteuber, and Georg Sigl. Improving non-profiled attacks on exponentiations based on clustering and extracting leakage from multi-channel high-resolution EM measurements. In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2015.

[SI11] Werner Schindler and Kouichi Itoh. Exponent blinding does not always lift (partial) spa resistance to higher-level security. In Javier López and Gene Tsudik, editors, *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, volume 6715 of *Lecture Notes in Computer Science*, pages 73–90, 2011.

[SW17] Werner Schindler and Andreas Wiemers. Generic power attacks on RSA with CRT and exponent blinding: new results. *J. Cryptogr. Eng.*, 7(4):255–272, 2017.

[WPB19] Leo Weissbart, Stjepan Picek, and Lejla Batina. One trace is all it takes: Machine learning-based side-channel attack on eddsa. In Shivam Bhasin, Avi Mendelson, and Mridul Nandi, editors, *Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings*, volume 11947 of *Lecture Notes in Computer Science*, pages 86–105. Springer, 2019.

[ZBHV20] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):1–36, 2020.

[ZS18] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *CoRR*, abs/1805.07836, 2018.

A Arithmetic Conditional Swap Implementation

Listing 1 shows the actual C implementation of the arithmetic conditional swap of two field elements in μNaCl (call to `CSWAP` in Algorithm 1). This implementation uses `XOR` and `AND` instructions.

B Detailed Iterative Framework Results

This section provides detailed results obtained from executing the proposed framework ten times on each scenario, described in Sections 5.3.1, for fixed and random CNN hyperparameters, before and after optimizing the attack interval from gradient visualization results.

```
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: Conditional swap of 2 field elements based on arithmetic of field operands limbs.

B.1 Fixed Hyperparameters

Figures 17 and 18 show results for 10 iterative framework executions for fixed CNN hyperparameters in all the iterations.

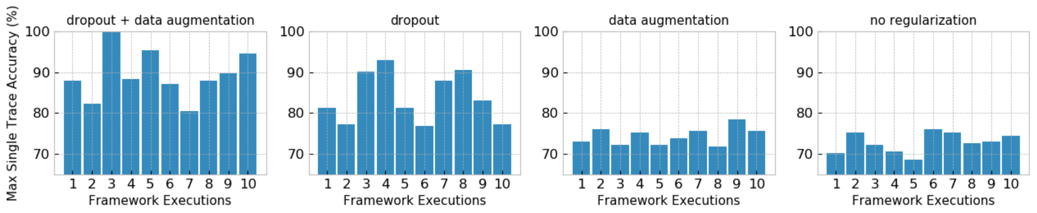


Figure 17: Fixed CNN hyperparameters: maximum single trace accuracy on *cswap-arith* dataset for 10 different framework executions.

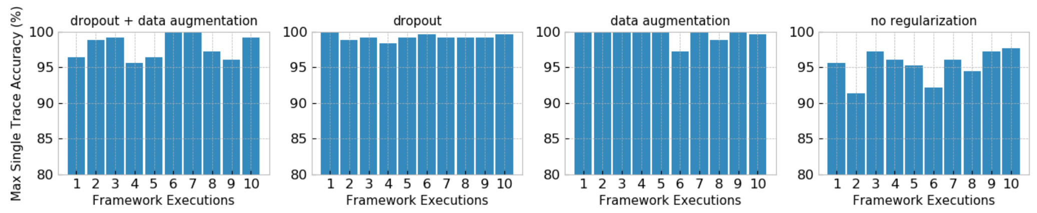


Figure 18: Fixed CNN hyperparameters: maximum single trace accuracy on *cswap-pointer* dataset for 10 different framework executions.

B.2 Random Hyperparameters

Figures 19 and 20 show results for 10 iterative framework executions for random CNN hyperparameters in all the iterations.

B.3 Fixed Hyperparameters after GV Interval Optimization

Figures 21 and 22 show results for 10 iterative framework executions for fixed CNN hyperparameters, in all the iterations, after the attack interval is optimized with gradient visualization tool.

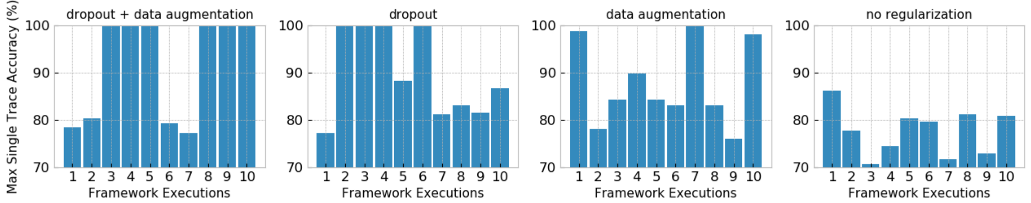


Figure 19: Random CNN hyperparameters: maximum single trace accuracy on *cswap-arith* dataset for 10 different framework executions and for random CNN hyperparameters.

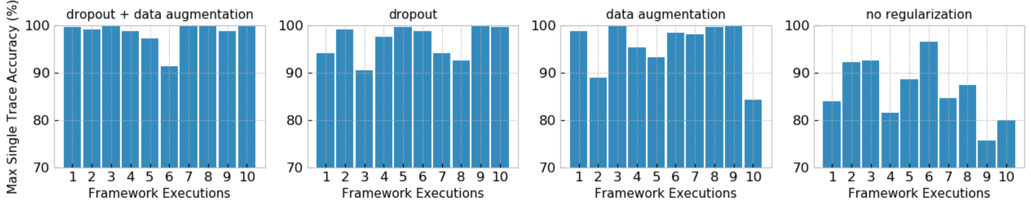


Figure 20: Random CNN hyperparameters: maximum single trace accuracy on *cswap-pointer* dataset for 10 different framework executions and for random CNN hyperparameters.

B.4 Random Hyperparameters after GV Interval Optimization

Figures 23 and 24 show results for 10 iterative framework executions for random CNN hyperparameters, in all the iterations, after the attack interval is optimized with gradient visualization tool.

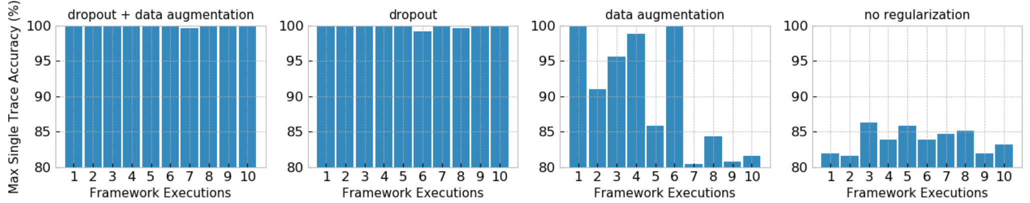


Figure 21: Fixed CNN hyperparameters: maximum single trace accuracy on *cswap-arith* dataset for 10 different framework executions and attacking the sample interval range from 1 500 to 3 500 of all sub-traces.

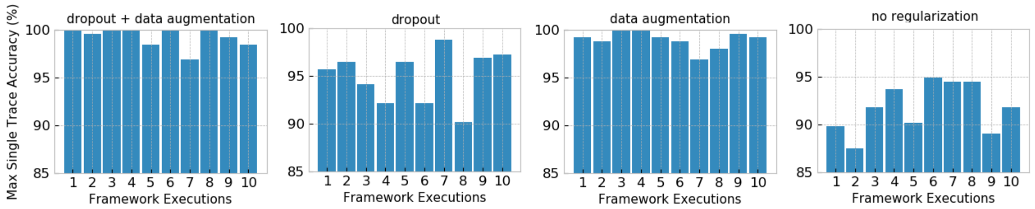


Figure 22: Fixed CNN hyperparameters: maximum single trace accuracy on *cswap-pointer* dataset for 10 different framework executions and attacking the sample interval range from 550 to 800 of all sub-traces.

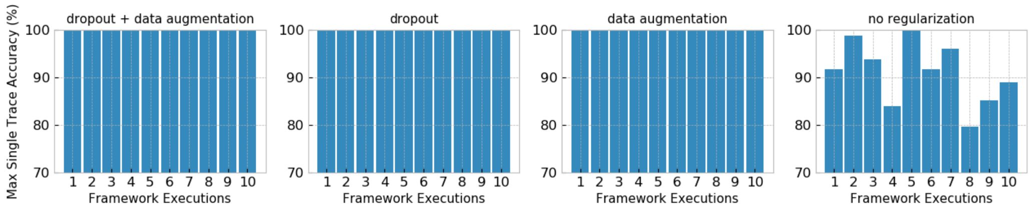


Figure 23: Random CNN hyperparameters: maximum single trace accuracy on *cswap-arith* dataset for 10 different framework executions and attacking the sample interval range from 1 500 to 3 500 of all sub-traces.

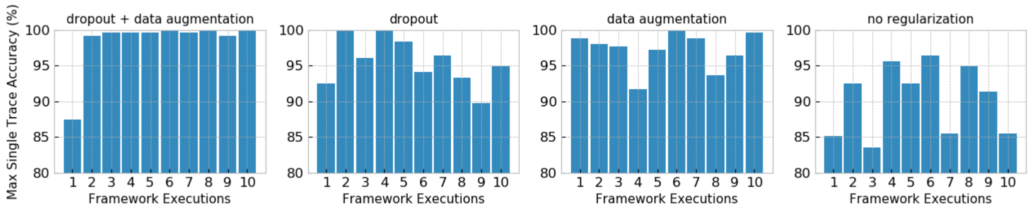


Figure 24: Random CNN hyperparameters: maximum single trace accuracy on *cswap-pointer* dataset for 10 different framework executions and attacking the sample interval range from 550 to 800 of all sub-traces.