One trace is all it takes: Machine Learning-based Side-channel Attack on EdDSA

Léo Weissbart[1,2], Stjepan Picek[1], and Lejla Batina[2]

[1] Delft University of Technology, The Netherlands
[2] Digital Security Group, Radboud University, The Netherlands

Abstract. Profiling attacks, especially those based on machine learning proved as very successful techniques in recent years when considering side-channel analysis of block ciphers implementations. At the same time, the results for implementations of public-key cryptosystems are very sparse. In this paper, we consider several machine learning techniques in order to mount a power analysis attack on EdDSA using the curve Curve25519 as implemented in WolfSSL. The results show all considered techniques to be viable and powerful options. Especially convolutional neural networks (CNNs) are effective as we can break the implementation with only a single measurement in the attack phase while requiring less than 500 measurements in the training phase. Interestingly, that same convolutional neural network was recently shown to perform extremely well for attacking the implementation of the AES cipher. Our results show that some common grounds can be established when using deep learning for profiling attacks on distinct cryptographic algorithms and their corresponding implementations.

1 Introduction

Cryptographic algorithms ensure the security of a system (e.g., communication on a network or payment with a smartcard), by providing security features (e.g., authenticity and non-repudiation). However, implementations of those algorithms can fail during the engineering process and present flaws, leaking secret information over side-channels, even for the strongest protocols. Side-channel analysis (SCA) designates a set of signal processing techniques targeting the execution of cryptographic implementations, evaluating a system's security.

Since Differential Power Analysis by Kocher et al. [16], many other powerful SCAs have been successfully used to break all cryptographic algorithms, including recent machine learning approaches, on both symmetric key cryptography [9,14,15,18,20,27,28,31] and public-key cryptography [21,30]. Among all SCAs, profiling attacks are the most powerful provided that the attacker has access to a clone device with full control that can be profiled offline, to later use this knowledge on another device during the attack phase. Template attack [9] has been the most popular instance of profiling attacks, but in recent years, new techniques based on machine learning were able to outperform template attack and break implementations protected with countermeasures. However, most of those results are obtained on block ciphers implementations (and more precisely on AES) and there are almost no results considering machine learning (deep learning) on public-key cryptography.

In this paper, we attack the digital signature algorithm Ed25519 as implemented in WolfSSL on an STM32F4 microcontroller and we also compare the results obtained from different profiling attacks. To that end, we consider several machine learning techniques (i.e., Random Forest, Support Vector Machines, and Convolutional Neural Network) that have been proved strong in related work (albeit mostly on block ciphers) and template attack, which we consider the standard technique and a baseline setting.

1.1 Related Work

Template attacks (TAs) have been introduced by Chari et al. in 2003 [9] as the most powerful SCA in the information-theoretic point of view and became a standard tool for profiling SCA. As straightforward implementations of TA can lead to computationally intensive computation, one option for more efficient computation is to use only a single covariance matrix, and is referred as the so-called pooled template attack presented by Choudary and Kuhn [10] where they were able to template a LOAD instruction and recover all 8 bits treated with a guessing entropy of 0. Several works applied machine learning methods to SCA of block ciphers because of their resemblance to general profiling techniques. Two methods stand out particularly in profiling SCA, namely Support Vector Machines (see, e.g., [28,20,33,18]) and Random Forest (see, e.g., [14,27,33]). With the general evolution in the field of deep learning, more and more works deal with neural networks for SCA and often show top performance. There, most of the research concentrated on either multilayer perceptron or convolutional neural networks [20,29,7,11].

There is a large portion of works considering profiling techniques for block ciphers but there is much less for public-key cryptography. Lerman et al. considered template attack and several machine learning techniques to attack RSA. However, the targeted implementation was not secure, which makes the comparison with non-machine learning techniques less favorable [18]. Nascimento et al. applied a horizontal attack on ECC implementation for AVR ATmega microcontroller targeting the side-channel leakage of cmov operation. Their approach to side-channel is similar to ours but they don't use deep learning in the analysis [23]. Poussier et al. used horizontal attacks and linear regression to conduct an attack on ECC implementations but their approach cannot be classified as deep learning [30]. Carbone et al. used deep learning to attack a secure implementation of RSA [8]. Previous work has shown TA to be efficient for attacking SPA-resistant ECDSA with P192 NIST curve on 32-bit microcontroller [21].

1.2 Contributions

There are two main contributions of this paper:

1. We present a comprehensive analysis of several profiling attacks by exploring different sets of hyper-parameters that permit to obtain the best results for each method. This evaluation can be helpful when deciding on an optimal strategy for machine learning and in particular, deep learning attacks on implementations of public-key cryptography.
2. We consider elliptic curve cryptography (actually EdDSA using curve Curve25519) and profiling attacks where we show that such techniques, and especially the convolutional neural networks can be extremely powerful attacks.

Besides those contributions, we also present a publicly available dataset we developed for this work. We aim to make our results more reproducible but also motivate other researchers to publish their datasets for public-key cryptography. Indeed, while the SCA community realizes the lack of publicly available datasets for block ciphers (and tries to improve it), the situation for public-key cryptography seems to attract less attention despite even worse availability of codes, testbeds, and datasets.

The rest of this paper is organized as follows. In Section 2, we give relevant background on elliptic curve scalar multiplication, Ed25519 algorithm, and profiling attack techniques. In Section 3, we explain the way an attacker can exploit this implementation of Ed25519. In Section 4, we present our testbed and data collection strategy. In Section 5, we give the results of the hyper-parameter tuning phase, dimensionality reduction, and profiling results. Finally, in Section 6, we conclude the paper and give some possible future research directions.

2 Background

In this section, we start by introducing the elliptic curve scalar multiplication operation and EdDSA algorithm. Afterward, we discuss profiling attacks that we use in our experiments.

2.1 EdDSA

In the context of public-key cryptography, one important feature is the authentication of a message between two parties. This feature ensures to party B that party A has indeed sent a message M and that this message is original and unaltered. Message authentication can be performed by Digital Signature Algorithms (DSA). DSA creates a signature pair (R, S) for proving that a message M was emitted by the known party A, unaltered and that A cannot repudiate. For security reasons and computational speed, public-key cryptography has turned toward Elliptic Curves based cryptography (ECC) as it tends to become the successor of RSA for public-key cryptography because it can meet higher security levels with smaller key lengths. ECC is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), which states that it is easy and hence efficient to compute $Q = k \cdot P$, but it is difficult to find k knowing Q and P.

EdDSA [3] is a variant of the Schnorr digital signature scheme [34] using Twisted Edward Curves, a subgroup of elliptic curves that uses unified formulas, enabling speed-ups for specific curve parameters. This algorithm proposes a deterministic generation of the ephemeral key, different for every different message, to prevent flaws from a predictable random number generator. The ephemeral key r is made of the hash value of the message M and the auxiliary key b, generating a unique ephemeral public key R for every message.

EdDSA, when using parameters of Curve25519 is referred to as Ed25519 (domain parameters are given in Appendix A) [2]. EdDSA scheme for signature generation and verification is described in Algorithm 1, where the notation $(x, \ldots, y)$ denotes the concatenation of the elements. The notation used in Algorithm 1 is given in Table 1.

After the signature generation, party A sends (M, R, S), i.e., the message along with the signature pair (R, S) to B. The verification of the signature is done by B with Steps 10 to 11. If the last equation is verified, it represents a point on the elliptic curve and the signature is correct, ensuring that the message can be trusted as an authentic message from A.

Table 1: Notation for EdDSA

Name	Symbol
Private key	k
Private scalar	a (first part of $H(k)$).
Auxiliary key	b (last part of $H(k)$).
Ephemeral key	r
Message	M

Algorithm 1 EdDSA Signature generating and verification

```
    **Keypair Generation** $(k, P)$: (Used once, first time private key is used.)
 1: Hash $k$ such that $H(k) = (h_0, h_1, \ldots, h_{2u-1}) = (a, b)$
 2: $a = (h_0, \ldots, h_{u-1})$, interpret as integer in little-endian notation
 3: $b = (h_u, \ldots, h_{2u-1})$
 4: Compute public key: $P = aB$.

    **Signature Generation:**
 5: Compute ephemeral private key $r = H(b, M)$ .
 6: Compute ephemeral public key $R = rB$.
 7: Compute $h = H(R, P, M) \mod l$.
 8: Compute: $S = (r + ha) \mod l$.
 9: Signature pair $(R, S)$

    **Signature Verification:**
10: Compute $h = H(R, P, M)$
11: Verify if $8SB = 8R + 8hP$ holds in $E$
```

2.2 Elliptic Curve Scalar Multiplication

The security of ECC algorithms depends on the ability to compute a point multiplication and the presumed inability to reverse the computation to retrieve the multiplicand given the original and product points. This security is strengthened with a greater prime order of the underlying finite field. In our attack, we aim to extract the ephemeral key r from its scalar multiplication with the Elliptic Curve base point B (see step 5 in Algorithm 1). To understand how this attack works, we decompose this computation as implemented in the case of WolfSSL Ed25519.

The implementation of Ed25519 in WolfSSL is based on the work of Bernstein et al. [3]. The implementation of elliptic curve scalar multiplication is a window-based method with radix-16, making use of a precomputed table containing results of the scalar multiplication of $16^i|r_i| \cdot B$, where $r_i \in [-8, 7] \cap \mathbb{Z}$ and B is the base point of Curve25519 (see Appendix B). This method is popular because of its trade-off between memory usage and computation speed, but also because the implementation is time-constant and does not feature any branch condition nor array indices and hence is presumably secure against timing attack. Leaking information from the corresponding value loaded from the memory with a function *ge_select* is used here to recover e and hence can be used to easily connect to the ephemeral key r. More details are given in the remainder of this paper.

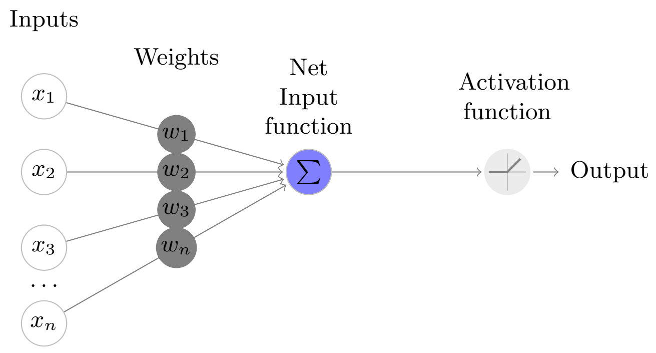
2.3 Profiling Attacks

In this work, we consider several machine learning techniques that showed very good performance when considering side-channel attacks on block ciphers. Besides, we briefly introduce the template attack, which serves as a baseline to compare the performance of algorithms.

Random Forest. Random Forest (RF) is a well-known ensemble learning method consisting of a number of decision trees [6]. Decision trees consist of combinations of Boolean decisions on a different random subset of attributes of input data (called bootstrap sampling). For each node of each tree, the best split is taken among these randomly chosen attributes. RF is a stochastic algorithm because of its two sources of randomness: bootstrap sampling and attribute selection at node splitting. The most important hyper-parameter to tune is the number of trees in the forest (we do not limit the tree size nor use pruning methods.)

Support Vector Machines. Support Vector Machines (denoted SVM) is a kernel-based machine learning family of methods that are used to accurately classify both linearly separable and linearly inseparable data [38]. The idea for linearly inseparable data is to transform them into a higher dimensional space using a kernel function, wherein the data can usually be classified with higher accuracy. The scikit-learn implementation we use considers libsvm's C-SVC classifier [25] that implements SMO-type algorithm [12]. The multi-class support is handled according to a one-vs-one scheme. We investigate two variations of SVM: with a linear kernel and with a radial kernel. Linear kernel-based SVM has the penalty hyper-parameter C of the error term. Radial kernel-based SVM has two significant hyper-parameters to tune: the cost of the margin C and the kernel γ.

Fig. 1: Anatomy of a neuron.



Convolutional Neural Networks. Convolutional Neural Networks (CNNs) are a type of neural networks initially designed to mimic the biological process of animal's cortex to interpret visual data [17]. CNNs show excellent results for classifying images for various applications and have also proved to be a powerful tool to classify time series data such as music or speech [24]. The *VGG-16* architecture introduced in [35] for image recognition was also recently applied to the problem of side-channel analysis with very good results [15].

From the operational perspective, CNNs are similar to ordinary neural networks (e.g., multilayer perceptron): they consist of several layers where each layer is made up of neurons as depicted in Figure 1. Every neuron in a layer computes a weighted combination of an input set by a net input function (e.g., the sum function in neurons of a fully-connected layer) from which a nonlinear activation function produces an output. When the output is different from zero, we say that the neuron activation feeds the next layer as its input. Layers with a convolution function as the Net Input Function are referred to as convolutional layers and are the core building blocks in a CNN.

CNNs use three main types of layers: convolutional layers, pooling layers, and fully-connected layers. Convolution layer computes the output of neurons from locally sparse combinations of initial raw input features, to reduce the space volume of information into smaller regions of interest. Pooling layers are used after a convolution layer to sample down local regions and create spatial regions of interest. The fully-connected layer at the end of a CNN behaves as a classifier for the extracted features from the inputs. The `ReLU` activation function will apply an element-wise activation function, such as the $max(0, x)$ thresholding at zero.

The architecture of CNN we choose in this paper makes use of some additional elements: batch normalization is used to normalize the input layer by applying standard scaling on the activations of the previous layer, using running mean and standard deviation. Flatten layer transforms input data of rank greater than two into a one-dimensional feature vector that is used in the fully-connected layer. Dropout is a regularization technique for reducing overfitting by preventing complex co-adaptations on training data. The term refers to randomly dropping out units (both hidden and visible) in a neural network with a certain probability at each batch.

The architecture of a CNN is dependent on a large number of hyper-parameters making the choice of hyper-parameters for each different application an engineering challenge. The choices made in this paper are discussed in Section 5.

Template Attack. The template attack relies on the Bayes theorem and considers the features as dependent. In the state-of-the-art, TA relies mostly on a normal distribution [9]. Accordingly, TA assumes that each $P(\boldsymbol{X} = \boldsymbol{x}|Y = y)$ follows a (multivariate) Gaussian distribution that is parameterized by its mean and covariance matrix for each class Y. The authors of [10] propose to use only one pooled covariance matrix averaged over all classes Y to cope with statistical difficulties and thus lower efficiency. In our experiments, we use the version of the attack with only one pooled covariance matrix.

3 Attacker Model

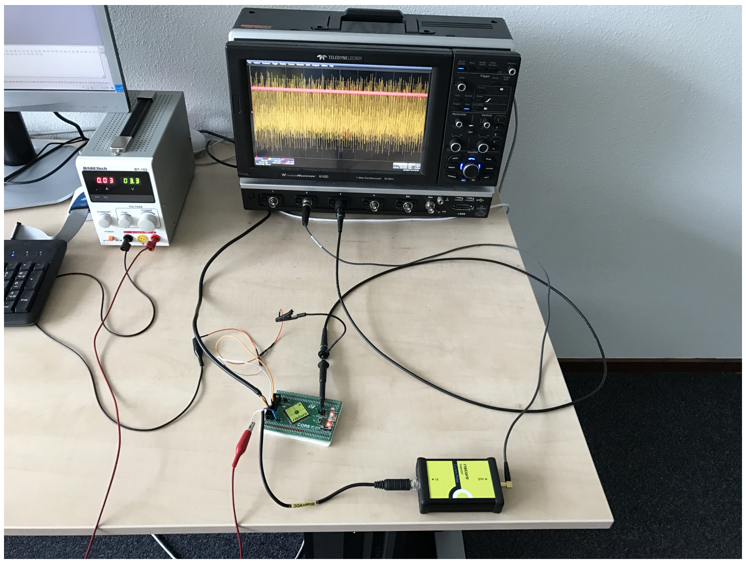
The general warning for implementations of ECDSA is to select different ephemeral private keys r for different signature. The flaw of using the same r for different messages happens since the two corresponding signatures would result in two signature pairs (R, S) and (R, S') for messages M and M', respectively. Then, an attacker can use this information to recover r as $r = (z - z')(S - S')^{-1}$ (with z and z', few bits of $H(M)$ and $H(M')$ interpreted as integers). Finally, to recover the private scalar a required to forge signatures, the attacker can trivially compute $a = R^{-1}(Sr - z)$.

Here, the aim of the attacker is the same as for every ECDSA attack: recover the secret scalar a. The difference is that the attacker cannot acquire two signatures with the same random r, but can still recover the secret scalar in two different ways. One method would consist of attacking the implementation of the hash function to recover b from the computation of ephemeral private key [32]. Another method (developed in this paper) attacks the implementation of the scalar multiplication during the computation of the ephemeral public key. With this method, the attacker collects side-channel traces of each computation since r is different in every message. This paper shows that even with a single attack trace, the attacker can recover private scalar with high confidence where we provide a comparison with different state-of-the-art profiling SCA.

4 Dataset Generation

In this section, we first present the measurement setup and explain the methodology for creating a dataset from the power traces obtained with our setup (see Figure 2).

Fig. 2: The measurement setup



4.1 Measurement Setup

The device under attack is a Piñata development board developed by Riscure to perform SCA evaluations [1]. The board is based on a 32-bit STM32F4 microcontroller with an ARM-based architecture, running at the clock frequency of 168 MHz. The board is modified to perform SCA through power consumption. The target is Ed25519 implementation of WolfSSL 3.10.2. As WolfSSL is an open-source library written in C, we have a fully transparent and controllable implementation for the profiling phase.

Power consumption is measured with a current probe [2] placed between the power source and the board's power supply source. Power measurements are

[1] Pinata Board: `https://www.riscure.com/product/pinata-training-target/`
[2] Current Probe: `https://www.riscure.com/product/current-probe/`

obtained with a Lecroy Waverunner z610i oscilloscope. The measures are performed with a sampling frequency of 1.025 GHz and the trigger is implemented with an I/O pin of the board around the *ge_select* function (see Algorithm 2) to retrieve a part of the key $\mathbf{e}$.

4.2 Dataset

To evaluate the attack proposed in this paper and to facilitate reproducible experiments, we present the dataset we built for this purpose [1]. We follow the same format for the dataset as in recently presented ASCAD database [31]. For this attack, we profile the EC scalar multiplication with the ephemeral key with the base point of curve Ed25519 (as explained in Section 3). Regarding the implementation of this operation for our target, we focus on the profiling of one function of the operation as it is more challenging by exploiting less information. We focus on the Lookup Table (LUT) operation used to fetch the precomputed chunks of the result in a table stored in memory. For speed reasons, the 256 bits scalar/ephemeral secret key r is interpreted in slices of 4-bits (nibbles) $e[i], i \in [0, 63]$, and to compute $R = rB$, the field multiplication with the base point B, we would have to compute $\sum_{i=0}^{63} e[i]16^i B$. As multiplication is resource consuming, the implementation stores the results for every nibble number i and nibble value $e[i]$ in a precomputed LUT and loads corresponding chunks when needed.

Table 2: Organization of the database.

DATABASE			
ATTACK_TRACES		PROFILING_TRACES	
TRACES	trace_1[1 000]	TRACES	trace_1[1 000]
	...		...
	trace_n_a[1 000]		trace_n_p[1 000]
LABELS	label_1[1]	LABELS	label_1[1]
	...		...
	label_n_a[1]		label_n_p[1]

Each trace in the database is represented by a tuple composed of one power trace and its corresponding label (class). The database is composed of two groups: the first group is PROFILING_TRACES, which contains n_p tuples. The second group is ATTACK_TRACES, which contains n_a tuples (see Table 2). In total, there are 6 400 labeled traces. We divide the traces in 80/20 ratio for profiling/attacking groups, and consequently, have $n_p = 5\,120$ and $n_a = 1\,280$. The profiling group is additionally divided in 80/20 ratio for training and validation sets.

A group contains two datasets: TRACES and LABELS. The dataset TRACES contains the raw traces recorded from different nibbles during the encryption. Each trace contains 1 000 samples and represents the relevant information of one nibble encryption. The dataset LABELS contains the correct subkey candidate for the corresponding trace. In total, there are 16 classes since we consider all possible nibble values.

To the best of our knowledge, besides the dataset we presented here, there is only one publicly available dataset for SCA on public-key cryptography on elliptic curves. Tuveri et al. conducted a side-channel analysis of SM2 (a digital signature algorithm) public-key cryptography suite where they consider various side channels [37]. Additionally, the authors published EM side-channel measurements of elliptic curve point multiplication[3]. We note that due to the choice of the suite (SM2 is not an international standard), this dataset is difficult to compare with ours.

5 Experimental Setting and Results

To examine the feasibility and performance of our attack, we present different settings for power analysis and use two different metrics. We first compare the performance by using the accuracy metric since it is a standard metric in machine learning. The second metric we use is the success rate as it is an SCA metric that gives a more concrete idea on the power of the attacker [36]. Note that we assume the attacker who can collect as many power traces as she wants and that the profiling phase is nearly-perfect as also suggested by Lerman et al. [19].

5.1 Hyper-parameters Choice

Here we discuss the choice of hyper-parameters for each method we consider in this paper.

TA: Classical Template Attack is applied with pooled covariance [10]. Profiling phase is repeated for a different choice of points of interest (POI).

RF: Hyper-parameter optimization is applied to tune the number of decision trees used in Random Forest. We consider the following number of trees: 50, 100, 500. The best number of decision trees is 100 with no PCA and 500 when PCA is applied for 10 and 656 POI.

SVM: For the linear kernel, the hyper-parameter to optimize is the penalty parameter C. We search for the best C among a range of $[1, 10^5]$ in logarithmic space. In the case of the radial basis function (RBF) kernel, we have two hyper-parameters to tune: the penalty C and the kernel coefficient γ. The search for best hyper-parameters is done within $C = [1, 10^5]$ and $\gamma = [-5, 2]$ in logarithmic spaces. We consider only those hyper-parameters that give the best scores for each choice of POI (see Table 3).

[3] available at `https://zenodo.org/record/1436828#.XRhmfY-xWrw`

Table 3: Chosen hyper-parameters for SVM

Number of features	Kernel	C	γ
1 000	linear	1 000	–
	rbf	1 000	1
656	linear	1 000	–
	rbf	1 000	1
10	linear	1 333	–
	rbf	1 000	1.23

Table 4: Architecture of the CNN

Hyper-parameter	Value
Input shape	$(1000, 1)$
Convolution layers	$(8, 16, 32, 64, 128, 256, 512, 512, 512)$
Pooling type	Max
Fully-connected layers	512
Dropout rate	0.5

CNN: The chosen hyper-parameters for *VGG-16* follows several rules that have been adapted for SCA in [15] or [31] and that we describe here:

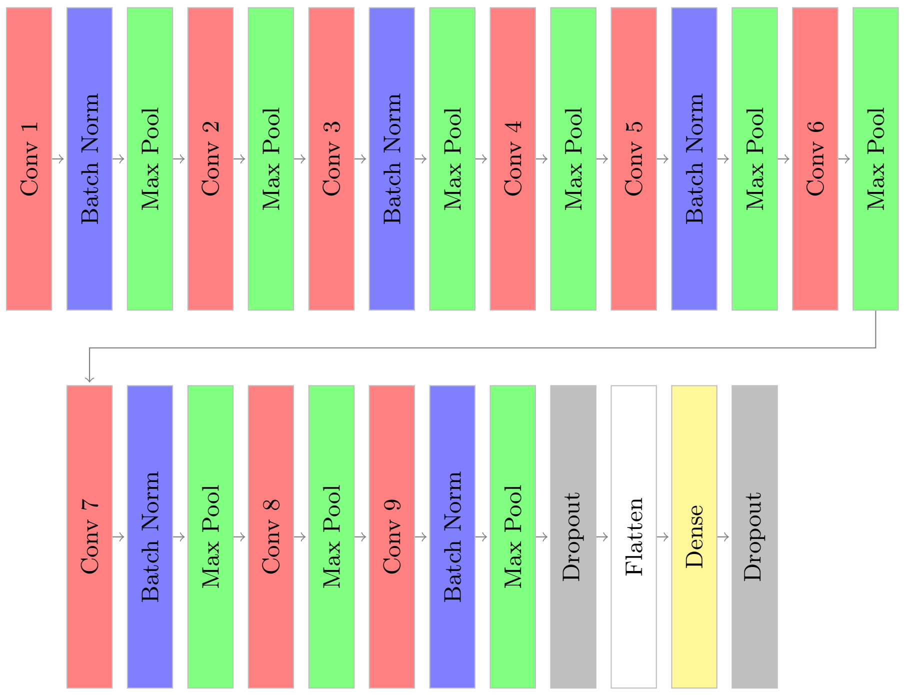
1. The model is composed of several convolution blocks and ends with a dropout layer followed by a fully connected layer and an output layer with the Softmax activation function.
2. Convolutional and fully-connected layers use the ReLU activation function.
3. A convolution block is composed of one convolution layer followed by a pooling layer.
4. An additional batch normalization layer is applied for every odd-numbered convolution block and is preceding the pooling layer.
5. The chosen filter size for convolution layers is fixed on size 3.
6. The number of filters $n_{\mathit{filters},i}$ in a convolution block i keeps increasing according to the following rule: $n_{\mathit{filters},i} = max(2^i \cdot n_{\mathit{filters},1}, 512)$ for every layer $i \geq 0$ and we choose $n_{\mathit{filters},1} = 8$
7. The stride of the pooling layers is of size 2 and halves the input data for each block.
8. Convolution blocks follow each other until the size of the input data is reduce to 1.

The resulting architecture is represented in Table 4 and Figure 3.

5.2 Dimensionality Reduction

For computational reasons, one may want to select points of interest (POI) and consequently, we explore several different setting where we either use all the features in a trace or we conduct dimensionality reduction. Here, for dimensionality

Fig. 3: CNN architecture as implemented in Keras. This architecture consists of 9 convolutional layers followed by max pooling layers. For each odd convolutional layer, there is a batch normalization layer before the pooling layer. At the end of the network, there is one fully connected layer.



reduction, we use Principal Component Analysis (PCA) [5]. Principal component analysis (PCA) is a well-known linear dimensionality reduction method that may use Singular Value Decomposition (SVD) of the data matrix to project it to a lower dimensional space. PCA creates a new set of features (called principal components) that are linearly uncorrelated, orthogonal, and form a new coordinate system. The number of components equals the number of original features. The components are arranged in a way that the first component covers the largest variance by a projection of the original data and the subsequent components cover less and less of the remaining data variance. The projection contains (weighted) contributions from all the original features. Not all principal components need to be kept in the transformed dataset. Since the components are sorted by the variance covered, the number of kept components, designated with L, maximizes the variance in the original data and minimizes the reconstruction error of the data transformation.

Note, while PCA is meant to select the principal information from a data, there is no guarantee that the reduced data form will give better results for profiling attacks than its complete form. We apply PCA to have the least possible number of points of interest that maximize the score from TA (10 points of interest) and the number of POI using a Bayesian model selection that estimates the dimensionality of the data based on a heuristics (see [22]). After an automatic selection of the number of components to use, we have 656 points of interest.

5.3 Results

In Table 5, we give results for different profiling methods when considering recovery of a single nibble of the key. We can see that all profiling techniques reach very good performance with all accuracy scores above 95%. Still, some differences can be noted. When considering all available features (1 000), CNN performs the best and has the accuracy of 100%. Both linear and rbf SVM and RF have the same accuracy. The performance of SVM is interesting since the same value for linear and rbf kernel indicates that there is no advantage of going into higher dimensional space, which means that the classes are linearly separable. Finally, TA performs the worst of all considered techniques.

Applying PCA to the dataset results in lower accuracy scores. More precisely, when considering the results with PCA that uses an optimal number of components (656), we see that the results for TA slightly improve while the results for RF and CNN decrease. While the drop in the performance for RF is small, CNN has a significant performance drop and becomes the worst performing technique. SVM with both kernels retains the same accuracy level as for the full number of features. Finally, when considering the scenario where we take only 10 most important components from PCA, all the results deteriorate when compared with the results with 1 000 features. Interestingly, CNN performs better with only 10 most important components than with 656 components but is still the worst performing technique from all the considered ones.

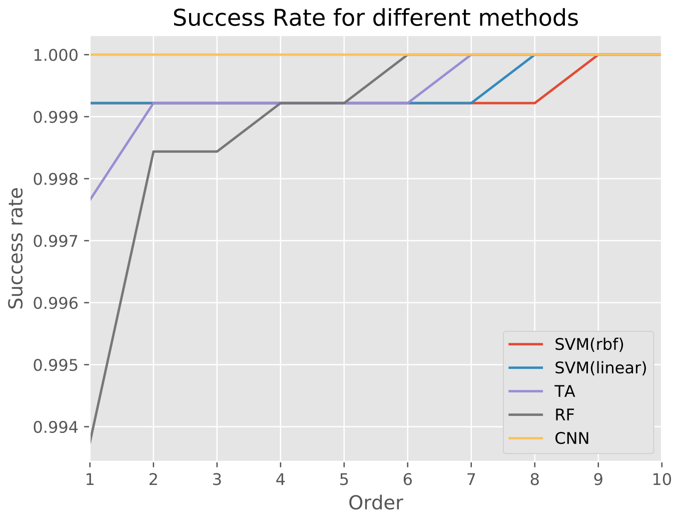
To conclude, all techniques exhibit very good performance but CNN is the best if no dimensionality reduction is done. There, the maximum accuracy is obtained after only a few epochs (see Figures 5 and 6). If there is dimensionality reduction, CNN shows a quick performance deterioration. This behavior should not come as a surprise since CNNs are usually used with the raw features (i.e., no pre-processing). In fact, applying such techniques could reduce the performance due to a loss of information and changes in the spatial representation of features. Interestingly, TA is never the best technique while SVM and RF show good and stable behavior for all feature set sizes.

In Figure 4, we give the success rate with orders up to 10 for all profiling methods on the dataset without applying PCA. Note, a success rate of order o is the probability that the correct subkey is ranked among the firsts o candidates of the guessing vector. While CNN has a hundred percent success rate of order 1, other methods achieve the perfect score only for orders greater than 6.

Table 5: Accuracy for the different methods obtained on the attacking dataset.

Algorithm	1 000 features	656 PCA components	10 PCA components
TA	0.9977	0.9984	0.9830
RF	0.9992	0.9914	0.9937
SVM (linear)	0.9992	0.9992	0.995
SVM (rbf)	0.9992	0.9992	0.995
CNN	1.00	0.95	0.96

Fig. 4: Success rate results.

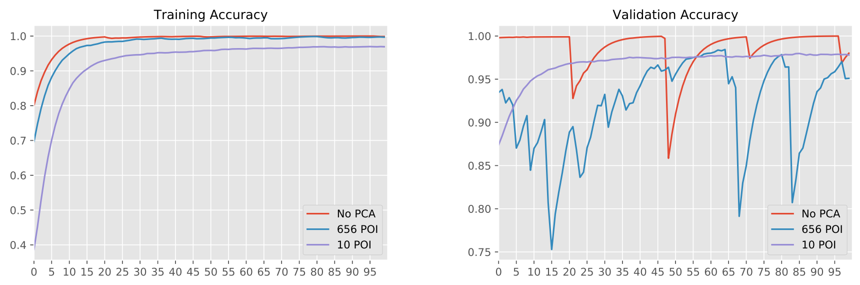


The results for all methods are similar in the recovery of a single nibble from the key. If we want to have an idea of how good these methods are for the recovery of a full 256-bit key, we must apply the classification on the successive 64 nibbles. We can have an intuitive glimpse of the resulting accuracy P_c with the cumulative probability of the probability of one nibble $P_s : P_c = \Pi_{64} P_s$ (see Table 6). The cumulative accuracy obtained in such a way can be interpreted as the predictive first-order success rate of a full key for the different methods in terms of a security metric.

From these results, we can observe that the best result is obtained with CNN when there is no dimensionality reduction. Other machine learning methods and TA are nonetheless powerful profiling attacks with up to 95 and 90% performance to recover the full key on the first guess with the best choice of hyper-parameters and dimensionality reduction. Note the low accuracy value for CNN when using 656 PCA components: this result is obtained as the accuracy of CNN for a single nibble raised to the power of 64 (since now we consider 64 nibbles). When considering the results after dimensionality reduction, we see that SVM is the best performing technique, which is especially apparent when using only 10 PCA components. Finally, we observe again that TA is never the best performing technique.

Table 6: Cumulative probabilities of the profiling methods.

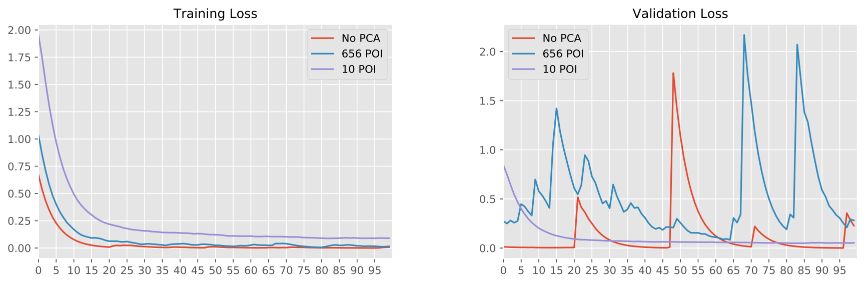
Algorithm	1 000 features	656 PCA components	10 PCA components
TA	0.86	0.90	0.33
RF	0.95	0.57	0.66
SVM (linear)	0.95	0.95	0.72
SVM (rbf)	0.95	0.95	0.72
CNN	1.00	0.03	0.07



(a) Training Accuracy (b) Validation Accuracy

Fig. 5: Accuracy of the CNN method over 100 epochs

As it can be observed from Figures 5 and 6, both scenarios without dimensionality reduction and dimensionality reduction to 656 components, reach the maximal performance very fast. On the other hand, the scenario with 10 PCA components does not seem to reach the maximal performance within 100 epochs since we see that the validation accuracy does not start to decrease. Still, even longer experiments do not show further improvement in the performance, which indicates that the network simply learned all that is possible and that there is no more information that can be used to further increase the performance.



(a) Training Loss (b) Validation Loss

Fig. 6: Loss of the CNN method over 100 epochs.

Choosing the Minimum Number of Traces for Training on CNN. As it is possible to obtain a perfect profiling phase on our dataset using CNN, we focus here on finding the smallest training set that gives a success rate of 1. More precisely, we evaluate the attacker in a more restricted setting [26]. To do so, we first reduce the size of the training set to k number of traces per class (to always have a balanced distribution of the traces) and then we gradually increase it to find out when the success rate reaches 1. In Table 7, we give the results obtained after one hundred epochs.

Table 7: Validation and test accuracy of CNN with an increasing number of training traces.

Number of traces per class k	10	20	30	50	100	300
Validation accuracy	0.937	1.0	1.0	1.0	1.0	1.0
Testing accuracy	0.992	0.992	1.0	1.0	1.0	1.0

Interestingly, it turns out that 30 traces per class for training the CNN is enough to reach the perfect profiling of this dataset. At the same time, the additional experiments did not show good enough behavior with a lower number of traces per class. Note the scenario with only 10 traces per class where the validation accuracy is lower than the testing accuracy. This happens since we use only 20% of the training set for the validation, which results in an extremely small validation set and consequently, less reliable results.

6 Conclusions and Future Work

In this paper, we consider a number of profiling techniques to attack the Ed25519 implementation in WolfSSL. The results show that although several techniques perform well, convolutional neural networks are the best if no dimensionality reduction is done. In fact, in such a scenario, we can obtain the accuracy of 100%, which means that the attack is perfect in the sense that we obtain the full information with only a single trace in the attack phase. What is especially interesting is the fact that CNN used here is taken from related work (more precisely, CNN used for profiling SCA on AES) and is not further adapted to the scenario here. This indicates that CNNs can perform well over various scenarios in SCA. Finally, to obtain such results, we require only 30 measurements per class, which results in less than 500 measurements to reach a success rate of 1 with CNN.

The implementation of Ed25519 we attack in this work does not feature any countermeasure for SCA (that is, beyond constant-time implementation). In future work, we plan to evaluate CNN for SCA on Ed25519 with different countermeasures to test the limits of CNN in the side-channel analysis.

References

1. Database for EdDSA. URL: `https://github.com/leoweissbart/MachineLearningBasedSideChannelAttackonEdDSA`
2. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records (2006). URL: `http://cr.yp.to/papers.html#curve25519`. Citations in this document **1**(5) (2016)
3. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering **2**(2), 77–89 (2012)
4. Blake, I., Seroussi, G., Smart, N.: Elliptic curves in cryptography, vol. 265. Cambridge university press (1999)
5. Bohy, L., Neve, M., Samyde, D., Quisquater, J.J.: Principal and independent component analysis for crypto-systems with hardware unmasked units. In: Proceedings of e-Smart 2003 (January 2003), cannes, France
6. Breiman, L.: Random Forests. Machine Learning **45**(1), 5–32 (2001)
7. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. pp. 45–68 (2017)

8. Carbone, M., Conin, V., Cornlie, M.A., Dassance, F., Dufresne, G., Dumas, C., Prouff, E., Venelli, A.: Deep learning to evaluate secure RSA implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(2), 132–161 (Feb 2019). https://doi.org/10.13154/tches.v2019.i2.132-161, `https://tches.iacr.org/index.php/TCHES/article/view/7388`
9. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 13–28. Springer (2002)
10. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. LNCS, vol. 8419, pp. 253–270. Springer (2013)
11. Cid, C., Jr., M.J.J. (eds.): Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers, Lecture Notes in Computer Science, vol. 11349. Springer (2019). https://doi.org/10.1007/978-3-030-10970-7
12. Fan, R.E., Chen, P.H., Lin, C.J.: Working Set Selection Using Second Order Information for Training Support Vector Machines. J. Mach. Learn. Res. **6**, 1889–1918 (Dec 2005), `http://dl.acm.org/citation.cfm?id=1046920.1194907`
13. FIPS, P.: 180-4. Secure hash standard (SHS), March (2012)
14. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers **PP**(99), 1–1 (2017). https://doi.org/10.1109/TC.2017.2757921
15. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(3), 148–179 (May 2019). https://doi.org/10.13154/tches.v2019.i3.148-179, `https://tches.iacr.org/index.php/TCHES/article/view/8292`
16. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual International Cryptology Conference. pp. 388–397. Springer (1999)
17. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10) (1995)
18. Lerman, L., Bontempi, G., Markowitch, O.: Power analysis attack: An approach based on machine learning. Int. J. Appl. Cryptol. **3**(2), 97–115 (Jun 2014). https://doi.org/10.1504/IJACT.2014.062722, `http://dx.doi.org/10.1504/IJACT.2014.062722`
19. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.: Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). In: COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers. pp. 20–33 (2015)
20. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. pp. 3–26 (2016)
21. Medwed, M., Oswald, E.: Template attacks on ECDSA. In: International Workshop on Information Security Applications. pp. 14–27. Springer (2008)
22. Minka, T.P.: Automatic choice of dimensionality for PCA. In: Advances in neural information processing systems. pp. 598–604 (2001)
23. Nascimento, E., Chmielewski, Ł., Oswald, D., Schwabe, P.: Attacking embedded ecc implementations through cmov side channels. In: Avanzi, R., Heys, H. (eds.) Selected Areas in Cryptography – SAC 2016. pp. 99–119. Springer International Publishing, Cham (2017)

24. Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499 (2016)
25. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
26. Picek, S., Heuser, A., Guilley, S.: Profiling side-channel analysis in the restricted attacker framework. Cryptology ePrint Archive, Report 2019/168 (2019), `https://eprint.iacr.org/2019/168`
27. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 209–237 (2019). https://doi.org/10.13154/tches.v2019.i1.209-237, `https://doi.org/10.13154/tches.v2019.i1.209-237`
28. Picek, S., Heuser, A., Jovic, A., Ludwig, S.A., Guilley, S., Jakobovic, D., Mentens, N.: Side-channel analysis and machine learning: A practical perspective. In: 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017. pp. 4095–4102 (2017)
29. Picek, S., Samiotis, I.P., Kim, J., Heuser, A., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. In: Chattopadhyay, A., Rebeiro, C., Yarom, Y. (eds.) Security, Privacy, and Applied Cryptography Engineering. pp. 157–176. Springer International Publishing, Cham (2018)
30. Poussier, R., Zhou, Y., Standaert, F.X.: A systematic approach to the side-channel analysis of ECC implementations with worst-case horizontal attacks. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 534–554. Springer International Publishing, Cham (2017)
31. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptology ePrint Archive **2018**, 53 (2018)
32. Samwel, N., Batina, L., Bertoni, G., Daemen, J., Susella, R.: Breaking ed25519 in WolfSSL. In: Cryptographers Track at the RSA Conference. pp. 1–20. Springer (2018)
33. Schindler, W., Huss, S.A. (eds.): Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings, LNCS, vol. 7275. Springer (2012)
34. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
35. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
36. Standaert, F.X., Malkin, T., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: EUROCRYPT. LNCS, vol. 5479, pp. 443–461. Springer (April 26-30 2009), Cologne, Germany
37. Tuveri, N., Hassan, S.u., Garcia, C.P., Brumley, B.B.: Side-channel analysis of sm2: A late-stage featurization case study. In: Proceedings of the 34th Annual Computer Security Applications Conference. pp. 147–160. ACSAC '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3274694.3274725, `http://doi.acm.org/10.1145/3274694.3274725`
38. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer-Verlag New York, Inc., New York, NY, USA (1995)

Appendix

A Ed25519 Domain Parameters

Ed25519 domain parameters:

– Finite field F_q, where $q = 2^{255} - 19$ is the prime.
– Elliptic curve $E(F_q)$, Curve25519
– Base point B
– Order of the point B, l
– Hash function H, SHA-512 [13]
– Key length $u = 256$ (also length of the prime)

For more details on other parameters of Curve25519 and the corresponding curve equations we refer to Bernstein [2].

B EC Scalar Multiplication

Algorithm 2 Elliptic curve scalar multiplication with base point [4]

```
Input: $R, a$ with $a = a[0] + 256 * a[1] + ... + 256^{31}a[31]$
Output: $H(a, s, m)$
 1: for $i = 0; i < 32; ++i$ do
 2:     $e[2i + 0] = (a[i] >> 0\&15);$
 3:     $e[2i + 1] = (a[i] >> 4)\&15;$
 4: end for
 5: $carry = 0;$
 6: for $i = 0; i < 63; ++i$ do
 7:     $e[i] += carry;$
 8:     $carry = (e[i] + 8);$
 9:     $carry >>= 4;$
10:     $e[i] -= carry << 4;$
11: end for
12: $e[63] += carry;$                                   ▷ $\forall i < 64,\ -8 \le e[i] \le 8$
13: $ge\_p3\_0(h);$
14: for $i = 1; i < 64; i += 2$ do
15:     $ge\_select(\&t, i/2, e[i]);$      ▷ load from precomputed table $(e[i] \cdot 16^i) \cdot B$ in $E$.
16:     $ge\_madd(\&r, R, \&t);$ $ge\_p1p1\_to\_p3(R, \&r);$
17: end for
18: $ge\_p3\_dbl(\&r, R);$ $ge\_p1p1\_to\_p2(\&s, \&r);$
19: $ge\_p2\_dbl(\&r, \&s);$ $ge\_p1p1\_to\_p2(\&s, \&r);$
20: $ge\_p2\_dbl(\&r, \&s);$ $ge\_p1p1\_to\_p2(\&s, \&r);$
21: $ge\_p2\_dbl(\&r, \&s);$ $ge\_p1p1\_to\_p3(R, \&r);$
22: for $i = 0; i < 64; i += 2$ do
23:     $ge\_select(\&t, i/2, e[i]);$      ▷ load from precomputed table $(e[i] \cdot 16^i) \cdot B$ in $E$.
24:     $ge\_madd(\&r, R, \&t);$ $ge\_p1p1\_to\_p3(R, \&r);$
25: end for
```