Profiled Attacks against the Elliptic Curve Scalar Point Multiplication using Neural Networks

Alessandro Barenghi$^{[0000-0003-0840-6358]1}$, Diego Carrera$^{[0000-1111-2222-3333]2}$, Silvia Mella$^{[0000-0002-4664-3541]2,3}$, Andrea Pace^{2}, Gerardo Pelosi$^{[0000-0002-3812-5429]1}$, and Ruggero Susella$^{[0000-1111-2222-3333]2}$

^{1} Politecnico di Milano, Milano, Italy, `name.surname@polimi.it`
^{2} STMicroelectronics, Agrate Brianza, Italy, `name.surname@st.com`
^{3} RadBoud University, Nijmegen, The Netherlands

Abstract. In recent years, machine learning techniques have been successfully applied to improve side-channel attacks against different cryptographic algorithms. In this work, we deal with the use of neural networks to attack elliptic curve-based cryptosystems. In particular, we propose a deep learning based strategy to retrieve the scalar from a double-and-add scalar-point multiplication. As a proof of concept, we conduct an effective attack against the scalar-point multiplication on NIST standard curve P-256 implemented in BearSSL, a timing side-channel hardened public library. The experimental results show that our attack strategy allows to recover the secret scalar value with a single trace from the attacked device and an exhaustive search over a set containing a few hundreds of the sought secret.

Keywords: Computer security · Applied cryptography · Profiled side channel attacks · Neural Networks · Elliptic curve cryptography

1 Introduction

Side channel attacks (SCAs) aim at deriving the data being processed during a computation, by modeling the data-dependent behavior of the computing device itself. So called passive side channel attacks exploit the physical behaviors that can be measured without disrupting the computation, such as computation time [6,19], power consumption [24,20], or radiated electromagnetic emissions [1,13,29]. The time series obtained by the measurement of such quantities are also known as *trace*s. To retrieve a secret value used during a computation, an attacker tries to map the collected traces onto multiple instances of a key-dependent model of the behavior of the device under attack. By observing which key-dependent model is most fitting, the actual value of the secret is inferred.

A powerful class of SCA is the class of profiled side channel attacks. In this scenario, the attacker has full control of a copy of the device under attack (e.g., a sample of a microcontroller similar to the target one). She first builds a good leakage model of the duplicate and then uses this leakage model to exploit the actual leakage of the target device and retrieve the secret information. One of the most used approach in profiled SCA is the one of *template attacks* [9], where a set of multivariate Gaussian models are built to describe the measurements recorded under fixed value of (a portion of) the secret data. The attacker may derive which value is used during the target computation, by applying a maximum likelihood estimation approach on the built models. In the best case, a single trace from the device under attack can be sufficient to mount the attack. Other approaches, that received significant attention in recent years, are those based on machine learning techniques [22,25,37]. In particular, attacks based on neural networks proved to be effective against both symmetric and asymmetric cryptosystems, also in defeating some side-channel countermeasures [7,8,16,23,26,33,34,36].

Given the nature of the cryptographic algorithm under attack, the amount of secret-dependent data points contained in a measurement may be small with respect to the total amount of collected information. For this reason, it is common practice to process the traces to extract the time instants where most of the secret key-related information is expected to be present. Such instants are known as *Points-of-Interest* (PoI). Effective PoI selection algorithms include the selection of trace samples exhibiting maximum variance, or the selection of trace samples corresponding to time instants when the processing of data is highly correlated with the hypothetical model for the sensitive operation (e.g., the Hamming weight of an operation result or the Hamming distance between a result and an operand) [24].

Contributions. In this work, we present a profiled attack against the double-and-add-always algorithm employed in Elliptic Curve Cryptography (ECC). We use machine learning techniques to model the power consumption behavior of the device under attack. Specifically, we aim at identifying when some arithmetic operations underlying the double-and-add-always algorithm share a common operand. We then use such information to distinguish the portion of the secret scalar value related to such operations. To this end, we build and train a set of Multilayer Perceptrons (MLPs) to extract information on the processed operands (specifically on the Hamming weight (HW) of their words), and propose a simple metric to decide on the equality of the operands themselves based on such information. This approach is similar to the one presented in [8], where information on specific bits of some operands is extracted to infer the secret exponent used in a RSA implementation. As our testbed, we attack a 2-bit windowing-based double-and-add-always implementation of the elliptic curve scalar-point multiplication. The implementation is one of those contained in the open-source BearSSL software library, which is hardened against timing side channel attacks [27]. On this experimental testbed, we are able to correctly guess all-but-two bits of the secret scalar on average, with a single trace from the attacked device.

Paper organization. In Section 2, we summarize related works and the difference with our study. In Section 3, we first provide background information on profiled SCA using neural networks and then detail the BearSSL windowing-based implementation of the elliptic curve scalar-point multiplication algorithm optimized for the NIST curve P-256. In Section 4 we describe our attack methodology, while in Section 5 we report the results of our experimental validation. Finally, Section 6 reports our concluding remarks.

2 Related Work

Attacks based on *collision technique* rely on the capability of distinguishing whether two distinct operations share a common input operand or not. They were applied to extract secret key values from a single execution trace of RSA [35] and ECC [17,31] algorithms, as well as from their implementations exhibiting some SCA countermeasures [4,10]. In these works, the authors employ either Euclidean distance or statistical correlation as quantitative metrics to detect whether two operations share a common input or not.

In [8] the authors describe a profiled strategy based on neural network classifiers to detect such property in an RSA implementation. They take advantage of a peculiar architectural property of the device under attack. In particular, they exploit the fact that the target chip leaks significant information on the value of the twelfth bit of each limb of a multiple-precision arithmetic operation. They use such information to understand whether two operands in subsequent iterations of the main loop of the algorithm have the same value. From this information they can then distinguish the secret exponent bits used in such iterations.

In this work, we use a similar approach and extract information on the HW of each limb, instead of a specific bit, and exploit it to attack ECC implementations. Moving to the HW leakage model is justified by the fact that the HW model carries more information about the whole operand than single bits. Moreover, the HW leakage model, together with the Hamming distance model, has been proven to be the most effective in different SCA scenarios. To manage this new leakage model, we employ neural networks as regressors, instead of (binary) classifiers as in [8].

In [34], the authors present a successful attack on a windowing-based implementation of the Edwards-curve Digital Signature Algorithm (EdDSA) in the WolfSSL software library [2]. They exploit the presence of table-lookup operations in the windowing based implementation and profile their power consumption. They then use the built models to infer which ones of the precomputed values are used during the target execution. In our work, we also try to understand the precomputed values used during the computation, but we profile the multiple-precision multiplication operation instead of table-lookup operations. The reason behind this choice is that our attack may in principle be applied also to non windowing-based implementations, which do not rely on table-lookup but only on multiple-precision arithmetic operations.

Other interesting applications of machine learning applied to profiled attacks against ECC implementations exploit different attack paths. In [36], the authors target an ECDSA implementation where the first four bits of the scalar are recovered via profiled SCA and employed to derive the private key via a computational lattice based attack, improving the effectiveness of the horizontal attack introduced in [28]. In [33] the authors attack two implementations of EdDSA with Curve25519. The first is the implementation provided by the WolfSSL library [2], where the full scalar is profiled by nibbles in a horizontal fashion. The second one is the protected implementation provided by the Libsodium library [11], based on the Montgomery Ladder scalar multiplication. The authors exploit the presence of an arithmetic-based conditional swap to profile each iteration of the main loop. In this work, the authors do not consider only neural network-based attacks, but compare the effectiveness of other profiling techniques, such as Template Attacks, Random Forests, and Support Vector Machines. In [26], the authors propose a deep learning-based method to incrementally correct secret keys recovered via horizontal attack. This is especially helpful in the presence of randomized scalars or exponents.

3 Background

In this section we first provide an overview on profiled attacks and neural networks. We then describe the double-and-add algorithm which will be the target of our attacks, detailing the implementation of the open source library BearSSL that we will use in our experimental validation.

3.1 Profiling Techniques

Profiled side channel attacks are performed in two stages. The first stage is known as *profiling stage* and consists in building a statistical model of the side channel behavior of a copy of the device under attack, for each possible value of a chosen portion of the secret key to be recovered. To build an accurate model, an appropriate leakage model for the device must be chosen. Common choices are the Hamming weight (HW) and the Hamming distance (HD) of a sensitive intermediate value that is computed by the algorithm and depends on the target secret key portion.

The second stage is known as *key-extraction stage.* In this phase, the attacker aims at classifying one (or a few) measurement(s) obtained from the target device by computing the probability that it matches (they match) each of the models built in the profiling phase. Since each model is associated with a guess on the sensitive variable value, the highest probability indicates the most likely one.

Deep Learning Attacks. A class of machine learning algorithms frequently used in profiled SCA is deep learning. In this scenario, the profiling stage maps onto the *learning* phase, where one or more neural networks are trained, while the key-extraction stage is mapped onto the *prediction* phase, where the networks are used to guess the target data. A neural network can be represented through a graph, where the nodes represent the neurons (i.e. the elementary processing elements), and the arcs represent the dataflow across the neurons. The neurons are organized in *layers*, where each layer groups together all the neurons acting in parallel on the same set of inputs. The neurons acting on the input values provided to the network compose the *input layer* of the network, while neurons acting on the outputs of other neurons compose the *hidden layers*. The neurons computing the outputs of the neural network belong to the *output layer*.

In this work, we employ a neural network architecture known as Multilayer Perceptron (MLP). In an MLP, also known as *feed-forward neural network* [5], each layer is fully connected to the next layer. Each node of an MLP computes a weighted sum of its inputs $(x_0, \ldots, x_{m-1})$ and applies to the result a nonlinear *activation function* $\phi(\cdot)$, yielding as output a value $z = \phi\left(\sum_{i=0}^{m-1} w_i\, x_i\right)$. The choice of the activation function depends on the features of the multivariate function to be approximated (learned) by the MLP. Common choices for the activation function include the *sigmoid* or a *softmax* function, when a real valued output in the $[0, 1]$ range is desired (e.g., to model a probability); the hyperbolic tangent, a *Rectified Linear Unit* (ReLU), or a *LeakyReLU function*, otherwise. The weights of the inputs, $w_0, \ldots, w_{m-1}$ are specific to each neuron and they are computed during the learning phase of the MLP, starting from an initial random choice.

The learning phase of an MLP is split in three sub-phases: training, validation and test. During the training phase, a set of inputs for which the desired output is known, is fed to the network. The weights of the neurons are updated in order to steer the results of the network towards the desired outputs. This is done by minimizing the difference between the outputs of the neural network and the expected results, according to a chosen error function (e.g., least squares), which is known as *loss function* [12,18]. Since the learning process of the MLP requires the values for the desired outputs, this is known as *supervised learning*, as opposed to *unsupervised learning* that does not require the desired output values. In the validation phase, a first assessment of the quality of the prediction capability of the neural network is measured. This is done by feeding to the network a fresh set of inputs, known as validation inputs, for which also the desired output is known. The difference between the expected and actual output of the network is known as validation error. Training and validation sub-phases are combined in what is known as *epoch*.

While in principle the learning phase may include an unbounded number of epochs, training over a very large amount of epochs may cause a phenomenon known as *overfitting*. In this case, the neural network is remarkably accurate in producing the desired results on the sets of inputs employed during the training, but poorly tolerates small variations on the said input sets which results in bad performance on unseen data [30]. To preserve the desirable *generalization* property of the network (i.e. tolerance to input changes), the learning phase should be stopped whenever the prediction error on the validation inputs (which are different from the training inputs) has a minimum. In fact, the validation error exhibits a concave trend, with a steep descent in the initial phase, followed by a slow increase due to the start of the overfitting phenomenon. The number of epochs to be computed is thus chosen minimizing the validation error of the network. To mitigate the overfitting phenomenon, a special neuron layer, known as *dropout layer*, can be used. In such layer, each neuron takes as input the output of exactly one neuron from the previous layer, not shared with anyone else, and it outputs either the input received or zero with a chosen probability. Such probability is known as *dropout rate* [32]. Finally, the final sub-phase of the network learning, i.e., the testing sub-phase, is performed. During the testing sub-phase, a further independent set of inputs is fed to the network to assess the performance of the fully trained network. The role of the testing phase is to assess whether the minimum in the validation error that was achieved in the training and validation sub-phases is indeed a satisfactory result for the application scenario at hand.

Feature selection When neural networks are employed in a machine learning based SCA, the inputs to the network are the data point composing the side channel trace. Since the operations involving sensitive data are typically few and far apart in a trace, providing all the trace samples acquired as inputs to the network may provide more noise than actual information to the network itself. Consequentially, it is commonplace to adopt a *feature selection procedure* to improve the efficiency and effectiveness of the network learning phase as well as the effectiveness of the network as a classifier/regressor. This procedure allows either to select a subset of the trace samples based on their sensitive leakage information content or to filter out redundant and uncorrelated information, by combining the trace samples to obtain a smaller set of features. Different methods may be applied for feature selection. Some effective techniques are *Difference Of Means* (DOM) [9], *Sum Of Squared Differences* (SOSD) [15], , *Sum Of Squared pairwise T-differences* (SOST) [15], *Signal-to-Noise Ratios* (SNR) [24], *Mutual Information Analysis* (MIA) [14], and *Principal Component Analysis* [3].

3.2 Elliptic Curve Scalar-Point Multiplication in BearSSL

We now describe a generic double-and-add-always algorithm and provide details of the version which will be used as our case study, i.e., the optimized implementation for P-256 provided in the BearSSL software library [27]. BearSSL is commonly used to implement TLS connectivity on inexpensive embedded systems. One of the main goals of BearSSL is the immunity to timing attacks, achieved through constant-time implementation of cryptographic primitives.

In the following, we denote the binary representation of a value in $\mathbb{F}_p$ stored in a variable v as $(v_{n-1}, \dots, v_0)_2$, while we highlight the physical storage of the same value as a sequence of W-bit architectural words/limbs as $(\mathrm{v}_{m-1}, \dots, \mathrm{v}_0)_{2^W}$. We denote the j-th bit $0 \leq j < W$ of the i-th word as $\mathrm{v}_{i,j}$, whist $v^{(t)}$ denotes the value of the variable v during the t-th iteration of the main loop of a double-and-add(-always) algorithm. We denote with capital letters, e.g., $\mathtt{P}$, the points of an elliptic curve with coordinates over the finite field $\mathbb{F}_p$.

Algorithm 1 reports the general structure of a double-and-add-always algorithm to perform scalar point multiplication. A naive double-and-add (non-always) algorithm scans the natural binary representation of the scalar (e.g., from left to right), performing a doubling at each scanned bit and an addition at each set bit of the scalar. While this approach produces a correct result, it is also vulnerable to timing attacks and simple power analysis, as it performs an

Algorithm 1: Left-to-right double-and-add-always

```
Input: $\mathtt{P} \in \mathbb{E}(\mathbb{F}_p)$: generator of a
       large subgroup of $\mathbb{E}$,
       $k = (k_{t-1}k_{t-2}\dots k_0)_2$:
       scalar value
Output: $\mathtt{Q} = [k]\mathtt{P}$: scalar-point
        multiplication result
1 $\mathtt{Q} \leftarrow \mathcal{O}$, $\mathtt{D} \leftarrow \mathcal{O}$
2 for $i \leftarrow t-1$ to $0$ do
3     $\mathtt{Q} \leftarrow [2]\mathtt{Q}$
4     if $k_i = 1$ then
5         $\mathtt{Q} \leftarrow \mathtt{P} + \mathtt{Q}$
6     else
7         $\mathtt{D} \leftarrow \mathtt{P} + \mathtt{Q}$
8 return $\mathtt{Q}$
```

Algorithm 2: Schoolbook multi-precision multiplication

```
Input: $x = (\mathrm{X}_{m-1}\mathrm{X}_{m-2}\dots\mathrm{X}_0)_{2^W}$
       $y = (\mathrm{Y}_{n-1}\mathrm{Y}_{n-2}\dots\mathrm{Y}_0)_{2^W}$
Output: $z = x \cdot y$
Data: $c = (\mathrm{C}_1\mathrm{C}_0)_{2^W}$: double
      precision temporary value
1 $z \leftarrow 0$
2 for $i \leftarrow 0$ to $m-1$ do
3     for $j \leftarrow 0$ to $n-1$ do
4         $(\mathrm{C}_1\mathrm{C}_0) \leftarrow \mathrm{Z}_{i+j} + \mathrm{X}_i \cdot \mathrm{Y}_j + \mathrm{C}_1$
5         $\mathrm{Z}_{i+j} \leftarrow \mathrm{C}_0$
6     $\mathrm{Z}_{i+n+1} \leftarrow \mathrm{C}_1$
7 return $z$
```

operation (the addition) only if a condition on the secret scalar is met. To avoid this information leakage, the double-and-add-always variant employs a *dummy* accumulator variable (curve point D, line 7), which stores the result of a dummy addition operation that is performed when the scalar bit being scanned is not set. The double-and-add-always algorithm thus always computes both a doubling and an addition for each scalar bit, however the additions performed when the scalar bit is not set do not contribute to the final result.

In the BearSSL optimized implementation for the NIST standard curve P-256, the field operations are realized computing integer addition or multiplication followed by a modulo reduction. We note that for other curves, an alternative approach relying on the Montgomery multiplication is applied in the library, although it does not hinder the nature of our attack. In the multiply and reduce approach applied in BearSSL, multiple precision multiplications and squarings are executed applying the schoolbook approach reported in Algorithm 2, as it is the optimal one given the size of the operands and the fact that the modulo operation can be performed quite efficiently making use of the special binary form of the field characteristic recommended for the NIST standard curve P-256.

A notable detail in the word-by-word processing described in Algorithm 2 is the use of a single-word multiplier (line 4) that is able to compute the multiplication of two W-bit words yielding the corresponding $2W$-bit result. Finally, a notable implementation choice for curve P-256 in the BearSSL library, concerns the encoding of multi-precision integer values as a sequence of W=30 bit words (each one fit into the least significant positions of a 32-bit architecture word) to manage the propagation of carries during additions and multiplications more efficiently.

Algorithm 3: BearSSL Scalar-point Multiplication for $\mathbb{E}(\mathbb{F}_p)$ NIST Standard Elliptic Curves

```
Input: P ∈ $\mathbb{E}(\mathbb{F}_p)$: generator of a large subgroup of $\mathbb{E}(\mathbb{F}_p)$,
       $k = (k_{t-1}k_{t-2}\dots k_0)_2$: scalar value
Output: $Q = [k]P$: scalar-point multiplication result
Data: CMove(BooleanExpression, SecondParam, ThirdParam): a conditional
      move copies the value of the third parameter into the second one, when
      the Boolean condition specified as the first parameter is true
1  $Q \leftarrow \mathcal{O}$
2  $P_2 \leftarrow 2P$
3  $P_3 \leftarrow P_2 + P$
4  for $i \leftarrow t-1$ to $1$ by $-2$ do
5      $w \leftarrow (k_i k_{i-1})_2$            // extract a 2-bit window from the scalar
6      $Q \leftarrow [2]Q$
7      $Q \leftarrow [2]Q$
8      $T \leftarrow P$, $U \leftarrow Q$
9      CMove($w = (10)_2, T, P_2$)
10     CMove($w = (11)_2, T, P_3$)
11     $U \leftarrow U + T$
12     CMove($w \neq (00)_2 \wedge Q = \mathcal{O}, Q, T$)
13     CMove($w \neq (00)_2 \wedge Q \neq \mathcal{O}, Q, U$)
14 return Q
```

Algorithm 3 describes the actual scalar-point multiplication algorithm adopted in the BearSSL cryptographic implementation of ECC based on the NIST standard elliptic curves with prime characteristic, $\mathbb{E}(\mathbb{F}_p)$, in Jacobian coordinates. It shows a left-to-right double-and-add-always approach with the left-to-right scanning of the binary representation of the secret scalar that considers two bits for each iteration of the main loop of the algorithm. The algorithm starts by setting the accumulator variable Q to the point at infinity (line 1), and precomputes the values of the double and triple of the base point P (lines 2–3). The main loop of the Algorithm (lines 4–13) scans the scalar k from left to right, considering two bits of the scalar, $(k_i k_{i-1})_2$ per iteration (line 5), and computes the quadruple of the accumulator Q via two doubling operations (lines 6–7). The remainder of the loop body performs a sequence of conditional assignments with the purpose of adding to Q one out of: nothing, P, [2]P, or [3]P, depending on whether the values of $(k_i k_{i-1})_2$ are $(00)_2$, $(01)_2$, $(10)_2$, or $(11)_2$, respectively. This is accomplished saving the quadruple of Q in a temporary variable, U, and performing a sequence of conditional assignments employing a constant time conditional move function (Cmove), which overwrites its destination operand (fed into the second parameter) with the source one (fed into the third parameter) only if the Boolean condition provided as the first parameter of the Cmove function is true. The first two conditional assignments, (lines 9 and 10) change the value of the point to be added, T, from the P value taken in the assignment in line 8 to [2]P or [3]P, depending on whether the pair of scalar bits is equal to $(10)_2$ or $(11)_2$.

Table 1: Summary of the pairs of variables, employed as operands in the point doubling and addition operations, of which the value collision are exploited, together with the corresponding value of the two scalar bits being revealed

First value	Line in Alg. 3	Second value	Line in Alg. 3	Values are	Inferred (k_i, k_{i-1}) with i from $t-1$ downto 1
$\mathtt{P}^{(0)}$	2	$\mathtt{T}^{(i)}$	11	$=$	$(0,0)$
$\mathtt{Q}^{(i)}$	11	$\mathtt{Q}^{(i-2)}$	6	$=$	
$\mathtt{P}^{(0)}$	2	$\mathtt{T}^{(i)}$	11	$=$	$(0,1)$
$\mathtt{Q}^{(i)}$	11	$\mathtt{Q}^{(i-2)}$	6	$\neq$	
$\mathtt{P}_2^{(0)}$	2	$\mathtt{T}^{(i)}$	11	$=$	$(1,0)$
$\mathtt{P}_3^{(0)}$	2	$\mathtt{T}^{(i)}$	11	$=$	$(1,1)$

The value T is then added to the copy of the quadrupled point value Q (line 11). Finally, the last two conditional assignments (line 12 and line 13) take care of replacing the value of the quadrupled point Q with its sum with one out of P, [2]P or [3]P, which is stored in T. This action needs to be performed only if the two scalar bits are not null (as this in turn implies that an addition is needed in this iteration). The reason for having two conditional assignments is to handle the fact that the accumulator curve point Q is set to the point at infinity at the beginning of the algorithm. This in turn would cause the point addition at line 11 to mis-compute the result of the sum of the accumulator with a multiple of the base point. As a consequence, a simple copy of the base point multiple is made if $\mathtt{Q} = \mathcal{O}$.

4 Proposed Attack Workflow

In the following, we describe our attack strategy against Algorithm 3. We start by describing which algorithmic feature is exploited in our attack, and how we determine a criterion to deduce the value of the scalar bits from the information leaked by the side channel measurements. Subsequently, we detail what is the workflow of the power trace processing in the learning and prediction phase of a neural network based classifier which will allow us to detect the value collisions in the algorithm.

4.1 Inferring the Secret Scalar Value via Collisions

Our attack exploits the fact that, in each iteration of the main loop of Algorithm 3, the conditional assignments driven by the values of the scalar bits will change the value of the accumulator Q only if the scalar bits are not equal to $(0, 0)$, and will select different values to be added to the temporary curve point U. Exploiting the side channel leakage of the device to detect whenever operands of the base field arithmetic operations match among the operations will allow us to deduce the value of the scalar bit pair.

A summary of the values for which we test the equality via side channel attack, and the corresponding inferred values for the scalar bits are reported in Table 1. The two bottom rows in Table 1 describe the two straightforward cases when the derivation of $(k_i, k_{i-1}) = (1, 0)$ or $(k_i, k_{i-1}) = (1, 1)$, is performed by testing whether the value of the addend $\mathtt{T}$ in the point addition at line 11, during the current i-th iteration, equals two or three times the base point, respectively. The values of the double and triple of the base point, $\mathtt{P}_2^{(0)}$ and $\mathtt{P}_3^{(0)}$, are computed outside the loop body (conventionally, at the zero-th iteration), and are denoted at lines 2–3 of Algorithm 3 as $\mathtt{P}_2$ and $\mathtt{P}_3$. The remaining cases when the value of the operand $\mathtt{T}$ of the point addition at line 11 matches the base point itself needs additional information to infer whether $(k_i, k_{i-1}) = (0, 1)$ or $(k_i, k_{i-1}) = (0, 0)$, i.e., if the point addition result is being actually used or not in this iteration. We derive this additional information from the fact that, if $(k_i, k_{i-1}) = (0, 0)$ nothing should be added to the accumulator value $\mathtt{Q}$ during the iteration where the loop index equals i. Indeed, in case $(k_i, k_{i-1}) = (0, 0)$ the conditional move operations at lines 12–13 do not change the value of the accumulator point $\mathtt{Q}^{(i)}$. As a consequence, we will have that, on the next loop body iteration (i.e., the one with loop index equal to $i-2$), the value of $\mathtt{Q}^{(i-2)}$ employed as operand to the point doubling at line 6 of Algorithm 3 will match the value of $\mathtt{Q}^{(i)}$ (i.e., the value at the previous iteration) as first operand in the point addition operation at line 11.

Through the collision analysis of the values in Table 1 we are thus able to distinguish the value of the two bits of the scalar for each loop iteration in all the cases but the single corner case of them being either $(0, 0)$ or $(0, 1)$ in the last iteration of the double-and-add algorithm. Indeed, since we would need information from a subsequent loop iteration to tell these case apart, and there is no further iteration, the last bit of the scalar must be guessed in this case.

Our approach to distinguishing whether the values taken by two curve point variables are the same or different relies on a leakage of the Hamming weight of the multiple precision arithmetic operands, i.e., the curve point coordinates, which we experimentally verified to be present. Under the assumption of this leakage being present, we analyzed the point doubling and point addition formulas, reported in Algorithm 4 and Algorithm 5 in the Appendix. Both algorithms report the formulas considering the points as expressed in Jacobian coordinates, as per the BearSSL implementation. We highlighted in blue all the multiple precision arithmetic operations involving as operands the coordinates of any of the operand points (the addends for the addition, the point to be doubled for the doubling). We note that, save for two instructions in the point doubling (lines 2 and 3, Algorithm 5), all the sensitive multiple precision operations are either multiplications or squares.

4.2 Testing for equality via side channel data extraction

We now describe how we employed a neural-network based regressor to detect whenever two curve point variables have the same values. In particular, we describe our preprocessing of the traces, aimed at reducing their number of samples, keeping only the ones which are expected to contain relevant information. Subsequently, we describe how we trained our neural network based regressors, and how their results are combined to yield the value match estimation.

For the sake of clarity, we will consider the power trace of the entire double-and-add algorithm as split into a sequence of slices of contiguous samples, corresponding to an iteration of the loop body in Algorithm 3, and denote them *single-iteration sub-traces.* Each single-iteration sub-trace contains two sequences of samples which correspond to the execution of the point doubling at line 6 of Algorithm 3 and to the execution of the the point addition at line 11. These two sequences are the ones containing the samples of interest for our attack, and will be denoted as *addition sub-trace* and *doubling sub-trace.*

Since the addition and doubling sub-traces are composed by a number of samples in the tens of thousands, we decided to perform a selection of the *points of interest*, i.e., derive a set of time instants corresponding to the samples most relevant for our analysis. To this end, we collected a set of traces where all the operands were known, and computed the sample Pearson correlation coefficient between the Hamming weight of each operand word and the power consumption itself. We determined a first set of points of interest, S^1_{PoI} considering all the time intervals where the correlation with the Hamming weight of any of the operand words exceeded a given threshold h, and adding a sample preceding and a sample following each time interval. A second set of points of interest S^2_{PoI} was determined as the sequence of time instants between the first and last time instants in which the Pearson correlation coefficient exceeds the threshold h. The threshold is selected to be above the value of the non significant correlation for a given number of traces n_{traces}, i.e., $\frac{4}{\sqrt{n_{traces}}}$ [24].

Once the sets of points of interest were determined, only the projection of the traces on the time instants contained in the said sets were employed as input to our regressor learning, validation and testing phases. We employed as a regressor, an MLP neural network, which was trained with the projected traces labelled with the sequence of Hamming weights of the multiple precision arithmetic operand, therefore learning to regress the sequence of the Hamming weights of the operand words. The PoI selection and learning (training) procedure was repeated for each multiple precision operand which is involved in curve point variable comparison, obtaining a dedicated MLP for each one of them, i.e., for each coordinate of the points reported in Table 1. Our choice employing a set of MLP neural networks for each multiple precision operand (i.e., each coordinate of a curve point) is motivated by the manipulations of the said operands taking place in different time instants during the addition and doubling steps.

In order to exploit the results of our trained classifiers on a power trace sampled from a device with an unknown scalar value, we partition the power trace at hand in the two addition and doubling sub-traces, apply the projection on the PoIs and feed the resulting decimated sub-traces onto the 9 regressors (one for each Jacobian coordinate X, Y, Z of the three elliptic curve point variable $\mathtt{Q}$, $\mathtt{T}$, $\mathtt{U}$). Since a regressor outputs a sequence of values corresponding to its best estimate of the Hamming weights of the operand at hand, we need to compare two sequence of values to infer if the operands match or not. We note that, in the case of the comparison between the three precomputed window points, i.e., $\mathtt{P} = \mathtt{P}^{(0)}$, $\mathtt{P}_2 = [2]\mathtt{P}^{(0)}$, $\mathtt{P}_3^{(0)} = [3]\mathtt{P}^{(0)}$ and the runtime value taken by $\mathtt{T}$, only the Hamming weights of $\mathtt{T}$ are unknown, while the others are fixed and known. This, in turn, allows us to compare the regressed estimate of the Hamming weight of the words of the coordinates of $\mathtt{T}$ with the known values of the Hamming weight of the words of the coordinates of $\mathtt{P}$, $\mathtt{P}_2$, $\mathtt{P}_3$.

The comparison between two curve points is performed taking the three vectors of 9 estimates of the Hamming weights for each coordinate of the points (e.g., $\mathtt{Q}$, $\mathtt{T}$) on the curve P-256, concatenating the vectors to obtain a 27 element vector for each point, and computing the Manhattan distance between the two 27 elements vectors. We recall that the Manhattan distance is defined as the sum of the absolute values of the coordinate-wise differences of two vectors, which are in our case the two sequences of Hamming weights. We deem two curve points equal if the Manhattan distance is below a given threshold. We employed traces from the learning (training) set to determine the threshold on the Manhattan distance allowing the best success rate in comparisons.

5 Experimental Evaluation

The device under test was a 32-bit ARM® Cortex®-M4 microcontroller with 256 kiB of Flash memory, 40 KiB of SRAM, supporting a maximum frequency of 72 MHz, while the recording of the power consumption traces was performed via a ChipWhisperer 1200 Pro digital sampling kit. The sampling is performed synchronously with the target microcontroller clock signal, providing one sample per clock cycle. This clock-locking approach removes artifacts coming from eventual clock jitter, and was possible with the Chipwhisperer Pro acquisition board locking the microcontroller clock to an external crystal oscillating at 7.37 MHz. To focus our analysis on the learning aspects of the profiled phase of attack, we employed triggers signals to single out point addition and doubling subtraces out of the entire execution of the elliptic curve scalar-point multiplication.

Our experiments were done running the BearSSL ver. 0.6 library [27], compiled with `gcc` ver. 9.2.1, and compilation options `-Os`. We recall that BearSSL encodes the multiple precision integer operands employing 30-bit limbs, which are embedded in the least significant bits of 32-bit architectural words. As a consequence, the binary values of curve point coordinates are represented employing a number of 32-bit machine words which is not a power of two; e.g., point coordinates of the NIST standard curve P-256 are defined over a prime finite field, where each element is encoded as a sequence of 256 bits which in turn corresponds to 9, 32-bit machine words.

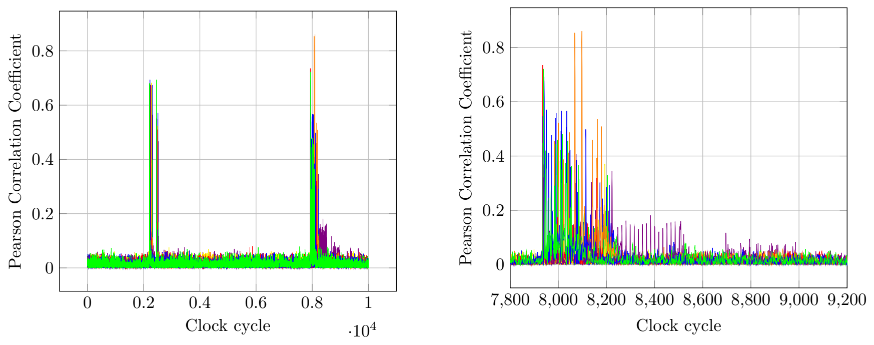

Fig. 1: Pearson correlation coefficient between the values of the Hamming weight of each one of the nine words of the x-coordinate of `Q` (the point being doubled in a point doubling operation) and the power consumption measurements from the device, as a function of time. Each one of the nine colours represents the correlation of a given coordinate word. (Left) Time interval corresponding to the leading part of a point doubling sub-trace – no significant correlation is observed outside the shown samples; (Right) zoom on the second set of peaks reported also in the left sub-figure, highlighting the different time instants where the coordinate words are being processed

The MLPs were generated and tested using TensorFlow v1.12.0 and Keras v2.1.6-tf. Experimental tests have been run on an Intel(R) Xeon(R) Gold 6254 processor running at 3.1 GHz. Training all the required MLPs (i.e., nine neural networks, three for each curve point), requires a total time of seven hours. Extracting the key values once the models are available takes a couple of seconds.

As a first step, we tested if the assumption of Hamming weight leakage on the curve operands was practically verified. To this end, we collected $3,000$ traces of the curve point doubling and addition with known random input and computed the Pearson correlation coefficient among the sampled values and the Hamming weight of the words composing the coordinates of the operands.

Figure 1 reports the result of the computation of the Pearson correlation coefficient between the Hamming weight of each one of the nine words of the x-coordinate of `Q` during a point doubling operation and the power consumption trace (line 6 or line 7 in Algorithm 3). Each correlation with a given word out of nine is represented with a different colour. As it can be seen, there are time instants where a non-negligible correlation is present for each one of the nine coordinate words. Similar results are obtained with the words of the remaining coordinates, and with the words of the curve addition operands, and are omitted for space reasons. Given the obtained results, we set our threshold h for the Pearson correlation coefficient to build the sets of PoIs to 0.3, as there is at least one time instant where a significant correlation is present for each word of each one of the curve coordinates.

MLP architecture

Layer	Type	Neurons	Function
Input	Dense	371	Identity
1	Dense	80	LeakyReLU $\alpha = 0.3$
2	Dropout	80	Dropout, rate 0.2
3	Dense	60	LeakyReLU $\alpha = 0.3$
4	Dense	50	LeakyReLU $\alpha = 0.3$
5	Dense	30	LeakyReLU $\alpha = 0.3$
6	Dense	20	LeakyReLU $\alpha = 0.3$
Output	Dense	9	ReLU

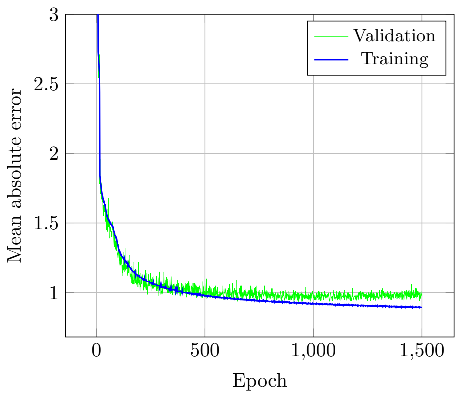

Fig. 2: (Left) Architecture of the MLP employed to regress the value of the Hamming weights of the multiple precision coordinates of curve points. (Right) Training and validation accuracy of the MLP for the prediction of the x-coordinate of the point `Q`. Each epoch employs 24k traces

We built nine regressors, one for each coordinate of the three points `Q`, `U`, and `T`, using the same MLP model. To choose our MLP regressor, we tested different neural network architectures and compared their performance with respect to their topology and hyper-parameters, using the accuracy and loss metrics. The number of layers and the number of neurons per layers were determined by subsequent refinement passes which reduced them until the accuracy results started worsening. We tested different number of epochs and batch sizes. As optimizer algorithms, we tested the ADAM optimizer and the RmsProp optimizer, with different learning rates.

We eventually designed our MLP regressor as an 8-layer MLP; a description of the complete architecture is reported in the left part of Fig. 2. We employed 7 dense neuron layers, and 1 random dropout layer (the third) with dropout rate 0.2 to prevent overfitting. We chose to employ the Leaky Rectified Linear Unit (LeakyReLU) [21] activation function, which is defined as

$$f(x) = \begin{cases} \alpha x, & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}$$

employing a leaking parameter $\alpha = 0.3$, for all the dense layers of the MLP save for the last. The choice of the LeakyReLU activation function was empirically substantiated by the fact that the LeakyReLU performed better than the original ReLU counterpart (i.e., LeakyReLU with $\alpha = 0$). We employed the mean squared error loss function and employed the Root Mean Square propagation (RMSprop) learning algorithm, with a batch size of 32.

For the learning phase of our attack, we collected a set of 75k sub-traces with known inputs. We used 30k of such sub-traces for the training/validation phase, with an 80/20 split between training and validation sets. Each epoch thus employed 24k sub-traces for the training and 6k for the validation. The remaining 45k sub-traces were used for the testing phase. The label associated to the dataset is the sequence of 9 Hamming weights of the operand words.

Figure 2, on the right, reports the Mean Absolute Error (MAE) trends in both the training and validation phase, for the regressor that tries to infer the x-coordinate of the curve point $\mathtt{Q}$. The MLPs for the other points and coordinates gave similar results. We saved the model when no improvement could be observed in the MAE. The test set confirmed a MAE under 1.

The training for each model required 45 minutes on the aforementioned CPU, resulting in about 7 hours for all models. The testing phase took less than a second for each of the nine models. Figure 2, on the right, reports the results of the training performed employing the time instants contained in the S^1_{PoI} sets of PoIs, determined with the criterion described in Section 4.2. No performance improvement have been observed using S^2_{PoI} in our setup, in turn showing that enlarging the set of PoIs (we recall that $S^2_{\mathrm{PoI}} \supseteq S^1_{\mathrm{PoI}}$) introduces no advantage.

Moving to the prediction phase, we employed the aforementioned nine models to extract information from a fresh set of 15k sub-traces to perform an attack. We were able to correctly match the value of $\mathtt{T}$ against the possible matching values, i.e., $\mathtt{P}$, $[2]\mathtt{P}$, and $[3]\mathtt{P}$ with perfect accuracy. We were therefore able to classify correctly all the occurrence of the scalar bit pairs where $w = (10)_2$ (i.e., $\mathtt{T} = [2]\mathtt{P}$) or $w = (11)_2$ (i.e., $\mathtt{T} = [2]\mathtt{P}$). Distinguishing the case where $w = (00)_2$ from $w = (01)_2$, i.e., distinguishing when the accumulator point $\mathtt{Q}$ has the same value was not possible with perfect accuracy.

To maximize the accuracy of the comparison of the values of the doubled point $\mathtt{Q}$, we optimized the threshold above which the Manhattan distance between the two vectors of Hamming weight estimates are deemed to represent the Hamming weights of two different values. Figure 3 reports the number of misclassified bits of the secret scalar (256-bit wide), averaged over 50 scalar multiplications, as a function of the Manhattan distance threshold employed to decide if a pair of regressed Hamming weight sets correspond to the same value. We note that, for a rather large interval of possible thresholds, our approach misclassifies less than a single bit out of the entire scalar, on average. Finally, we note that, in case the least significant bit pair in the scalar is either $(0, 1)$ or $(0, 0)$, we are not able to recover the value of the least significant bit. Indeed, telling the aforementioned cases apart requires to extract the information on the doubled point $\mathtt{Q}$, which should retain its value between the last iteration and the (non-existing) following one of the scalar-point multiplication. We therefore need to guess its value in the aforementioned case,which can be detected as the next-to-least significant bit is extracted to be 0.

Summing up, we are able to retrieve the entire value of the scalar of a scalar-point multiplication on the $\mathtt{P}$-256 curve with a single power trace and an exhaustive search over at most 128 possible values for the scalar, since we misclassify on average less than a single bit, which can appear only in 64 out of the 256 possible positions, and we may need to guess the value of the least significant bit of the scalar 1 out of 2 times on average.

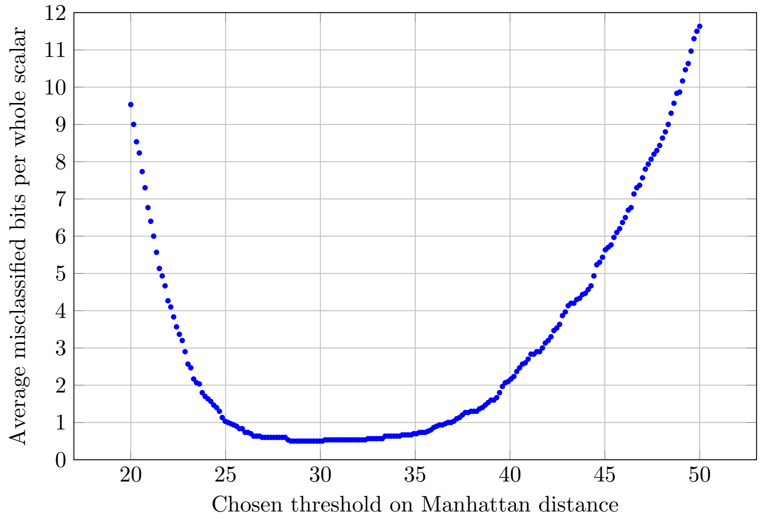

Fig. 3: Average number of mis-classified bits for an entire scalar, as a function of the chosen threshold on the Manhattan distance between the two sequences of regressed Hamming weights (all mis-classifications are caused by the point doubling operand matching)

6 Concluding Remarks

We presented a profiled side channel attack based on neural network against the elliptic curve scalar-point multiplication. We validated our attack technique on the elliptic curve scalar-point multiplication implemented in BearSSL, running on a Cortex-M4 microcontroller. Our attack strategy achieves a perfect success rate with an extremely small exhaustive search (128 possible scalar values for the P-256 curve), employing a single power trace from the attacked device, and a learning phase with 75k traces. The attack requires seven hours to perform the neural network training on an Intel(R) Xeon(R) Gold 6254 processor running at 3.1 GHz, while the actual key extraction and exhaustive search phase takes a couple of seconds.

We note that our attack can in principle be applied to non-windowing-based implementations, too. For instance, by looking at Algorithm 1 we can observe that $k_i = 0$ implies that the addition (line 7) at iteration i and the double (line 3) at iteration $i + 1$ share the same operand Q, similarly to what is exploited to attack Algorithm 3. We also note that the scalar blinding countermeasure becomes ineffective in a scenario where the attacker has full control of another instance of the device under attack, and she can perform the profiling phase disabling the countermeasures. In fact, our attack path relies on the knowledge of the precomputed values and the presence of internal collisions through subsequent iterations, which are not removed by scalar blinding. The well known countermeasures of *coordinates randomization* and *re-randomization*, would remove such requirements and thus block our attack. However, if the attacker has no access to a clone device without countermeasures, then she would not be able to mount the attack neither in the presence of scalar blinding, as she would not know the information needed to label the dataset for the profiling phase.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 29–45. Springer (2002). https://doi.org/10.1007/3-540-36400-5_4
2. Barthelmeh, J.: WolfSSL (formerly cyassl) library: a small, fast, portable implementation of TLS/SSL for embedded devices. https://github.com/wolfSSL/wolfssl (2016)
3. Batina, L., Hogenboom, J., van Woudenberg, J.G.J.: Getting More from PCA: First Results of Using Principal Component Analysis for Extensive Power Analysis. In: Dunkelman, O. (ed.) Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7178, pp. 383–397. Springer (2012). https://doi.org/10.1007/978-3-642-27954-6_24
4. Bauer, A., Jaulmes, É., Prouff, E., Wild, J.: Horizontal Collision Correlation Attack on Elliptic Curves. In: Lange, T., Lauter, K.E., Lisonek, P. (eds.) Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8282, pp. 553–570. Springer (2013). https://doi.org/10.1007/978-3-662-43414-7_28
5. Bishop, C.: Pattern Recognition and Machine Learning. Information Science and Statistics, Springer-Verlag New York (2006)
6. Brumley, D., Boneh, D.: Remote Timing Attacks Are Practical. In: Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003. pp. 1–13. USENIX Association (2003), https://www.usenix.org/conference/12th-usenix-security-symposium/remote-timing-attacks-are-practical
7. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Preprocessing. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 45–68. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_3
8. Carbone, M., Conin, V., Cornelie, M., Dassance, F., Dufresne, G., Dumas, C., Prouff, E., Venelli, A.: Deep Learning to Evaluate Secure RSA Implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 132–161 (2019). https://doi.org/10.13154/tches.v2019.i2.132-161
9. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3

10. Danger, J., Guilley, S., Hoogvorst, P., Murdica, C., Naccache, D.: Improving the Big Mac Attack on Elliptic Curve Cryptography. In: Ryan, P.Y.A., Naccache, D., Quisquater, J. (eds.) The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday. Lecture Notes in Computer Science, vol. 9100, pp. 374–386. Springer (2016). https://doi.org/10.1007/978-3-662-49301-4_23
11. Denis, F.: The Sodium cryptography library. Libsodium, https://doc.libsodium.org/ (2013)
12. Duchi, J.C., Hazan, E., Singer, Y.: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In: Kalai, A.T., Mohri, M. (eds.) COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010. pp. 257–269. Omnipress (2010), http://colt2010.haifa.il.ibm.com/papers/COLT2010proceedings.pdf#page=265
13. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2162, pp. 251–261. Springer (2001). https://doi.org/10.1007/3-540-44709-1_21
14. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual Information Analysis. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5154, pp. 426–442. Springer (2008). https://doi.org/10.1007/978-3-540-85053-3_27
15. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. Stochastic Methods. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4249, pp. 15–29. Springer (2006). https://doi.org/10.1007/11894063_2
16. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015. pp. 106–111. IEEE Computer Society (2015). https://doi.org/10.1109/HST.2015.7140247
17. Hanley, N., Kim, H., Tunstall, M.: Exploiting Collisions in Addition Chain-Based Exponentiation Algorithms Using a Single Trace. In: Nyberg, K. (ed.) Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9048, pp. 431–448. Springer (2015). https://doi.org/10.1007/978-3-319-16715-2_23
18. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980
19. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology. p. 104–113. CRYPTO '96, Springer-Verlag, Berlin, Heidelberg (1996)
20. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. p. 388–397. CRYPTO '99, Springer-Verlag, Berlin, Heidelberg (1999)
21. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: ICML Workshop on Deep Learning for Audio, Speech and Language Processing (2013)

22. Maghrebi, H.: Assessment of Common Side Channel Countermeasures With Respect To Deep Learning Based Profiled Attacks. In: 31st International Conference on Microelectronics, ICM 2019, Cairo, Egypt, December 15-18, 2019. pp. 126–129. IEEE (2019). https://doi.org/10.1109/ICM48031.2019.9021728
23. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking Cryptographic Implementations Using Deep Learning Techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10076, pp. 3–26. Springer (2016). https://doi.org/10.1007/978-3-319-49445-6_1
24. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
25. Masure, L., Dumas, C., Prouff, E.: A Comprehensive Study of Deep Learning for Side-Channel Analysis. IACR Cryptol. ePrint Arch. **2019**, 439 (2019), https://eprint.iacr.org/2019/439
26. Perin, G., Chmielewski, L., Batina, L., Picek, S.: Keep it Unsupervised: Horizontal Attacks Meet Deep Learning. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 343–372 (2021). https://doi.org/10.46586/tches.v2021.i1.343-372
27. Pornin, T.: BearSSL, a smaller SSL/TLS library. https://bearssl.org/index.html (2016)
28. Poussier, R., Zhou, Y., Standaert, F.: 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 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 534–554. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_26
29. Quisquater, J.J., Samyde, D.: Eddy current for Magnetic Analysis with Active Sensor. In: Proceedings of Esmart 2002, Nice, France. pp. 185–194 (9 2002)
30. Reed, R.D., Marks, R.J.: Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. MIT Press, Cambridge, MA, USA (1998)
31. Roelofs, N., Samwel, N., Batina, L., Daemen, J.: Online Template Attack on ECDSA: - Extracting Keys via the Other Side. In: Nitaj, A., Youssef, A.M. (eds.) Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12174, pp. 323–336. Springer (2020). https://doi.org/10.1007/978-3-030-51938-4_16
32. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
33. Weissbart, L., Chmielewski, L., Picek, S., Batina, L.: Systematic Side-Channel Analysis of Curve25519 with Machine Learning. J. Hardw. Syst. Secur. **4**(4), 314–328 (2020). https://doi.org/10.1007/s41635-020-00106-w
34. Weissbart, L., Picek, S., Batina, L.: One Trace Is All It Takes: Machine Learning-Based Side-Channel Attack on EdDSA. In: Bhasin, S., Mendelson, A., Nandi, M. (eds.) Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11947, pp. 86–105. Springer (2019). https://doi.org/10.1007/978-3-030-35869-3_8
35. Witteman, M.F., van Woudenberg, J.G.J., Menarini, F.: Defeating RSA Multiply-Always and Message Blinding Countermeasures. In: Kiayias, A. (ed.) Topics

in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6558, pp. 77–88. Springer (2011). https://doi.org/10.1007/978-3-642-19074-2_6
36. Zhou, Y., Standaert, F.X.: Simplified Single-Trace Side-Channel Attacks on Elliptic Curve Scalar Multiplication using Fully Convolutional Networks. In: 40th WIC Symposium on Information Theory in the Benelux (2019), https://dial.uclouvain.be/pr/boreal/object/boreal:226275
37. Zotkin, Y., Olivier, F., Bourbao, E.: Deep learning vs template attacks in front of fundamental targets: experimental study. IACR Cryptol. ePrint Arch. **2018**, 1213 (2018), https://eprint.iacr.org/2018/1213

A Details of Double-and-Add algorithm

Algorithm 4: Elliptic Curve Point Addition in the BearSSL library [27]

```
    Input: U = $(x_1, y_1, z_1) \in \mathbb{E}(\mathbb{F}_p)$; T = $(x_2, y_2, z_2) \in \mathbb{E}(\mathbb{F}_p)$
    Output: $(x_1, y_1, z_1) = \mathtt{U} + \mathtt{T}$
1  $t_3 = z_2^2$
2  $t_1 = x_1 \cdot t_3$
3  $t_4 = z_2 \cdot t_3$
4  $t_3 = y_1 \cdot t_4$
5  $t_4 = z_1^2$
6  $t_2 = x_2 \cdot t_4$
7  $t_5 = z_1 \cdot t_4$
8  $t_4 = y_2 \cdot t_5$
9  $t_2 = t_2 - t_1$
10 $t_4 = t_4 - t_3$
11 $t_4 = t_4 \bmod p$
12 $t_7 = t_2^2$
13 $t_6 = t_1 \cdot t_7$
14 $t_5 = t_7 \cdot t_2$
15 $x_1 = t_4^4$
16 $x_1 = x_1 - t_5$
17 $x_1 = x_1 - t_6$
18 $x_1 = x_1 - t_6$
19 $t_6 = t_6 - x_1$
20 $y_1 = t_4 \cdot t_6$
21 $t_1 = t_5 \cdot t_3$
22 $y_1 = y_1 - t_1$
23 $t_1 = z_1 \cdot z_2$
24 $z_1 = t_1 \cdot t_2$
25 return $(x_1, y_1, z_1)$
```

Algorithm 5: Elliptic Curve Point Doubling in the BearSSL library [27]

```
    Input: Q = $(x_1, y_1, z_1) \in \mathbb{E}(\mathbb{F}_p)$
    Output: $(x_1, y_1, z_1) = [2]\mathtt{Q}$
1  $t_1 = z^2$
2  $t_2 = x_1 + t_1$
3  $t_1 = x_1 - t_1$
4  $t_3 = t_1 \cdot t_2$
5  $t_1 = t_3 + t_3$
6  $t_1 = t_3 + t_1$
7  $t_3 = y_1^2$
8  $t_3 = t_3 + t_3$
9  $t_2 = x_1 \cdot t_3$
10 $t_2 = t_2 + t_2$
11 $x_1 = t_1^2$
12 $x_1 = x_1 - t_2$
13 $x_1 = x_1 - t_2$
14 $t_4 = y_1 \cdot z_1$
15 $z_1 = t_4 + t_4$
16 $t_2 = t_2 - x_1$
17 $y_1 = t_1 \cdot t_2$
18 $t_4 = t_3^2$
19 $t_4 = t_4 + t_4$
20 $y_1 = y_1 - t_4$
21 return $(x_1, y_1, z_1)$
```