Simplified Single-Trace Side-Channel Attacks on Elliptic Curve Scalar Multiplication using Fully Convolutional Networks

Yuanyuan Zhou
ICTEAM/ELEN/Crypto Group; IC&Card Lab
Université catholique de Louvain; Brightsight BV
Ottignies-Louvain-la-Neuve, Belgium; Delft, The Netherlands
zhou@brightsight.com

François-Xavier Standaert
ICTEAM/ELEN/Crypto Group
Université catholique de Louvain
Ottignies-Louvain-la-Neuve, Belgium
fstandae@uclouvain.be

***Abstract*—We aim to simplify the worst-case horizontal attack on scalar multiplication published at CHES 2017 [1] by making use of deep learning techniques, and to automate the critical steps of this previous work, namely the information identification, information extraction and information combination steps. For this purpose, we gradually increase the number of automated steps, targeting a very challenging assembly-level regular Montgomery ladder scalar multiplication implementation on a BeagleBone Black (BBB) Board running at 1 GHz. Our results demonstrate that the latter two steps can be simplified using deep learning techniques and lead to similar results as previous work. By contrast, the first step still requires some additional engineering. By showing that points of interest (POIs) selection can play an important (sometimes necessary) role for attacking asymmetric cryptography algorithms using deep learning techniques, we bring a more contrasted view on the advantage and limitations of such techniques. To the best of our knowledge, this is the first public report of deep learning based attack on ECC implementations. Besides, we propose the use of Fully Convolutional Networks as an alternative (deep) learning tool for side-channel analysis.**

***Index Terms*—ECC, Scalar Multiplication, Deep learning, Side-channel attack, Fully Convolutional Networks**

I. Introduction

Scalar multiplication is the cornerstone of Elliptic Curve Cryptography (ECC), thus its implementation security is critical for concrete deployment. As surveyed in a recent work by Poussier et al. [1], in the past decades, two categories of attacks targeting the scalar multiplication have been presented in the literature: *Divide and Conquer* (DC) ones and *Extend and Prune* (EP) ones. The DC approach is trying to recover the scalar bits independently, so each scalar bit is associated with a probability of score. The EP approach is trying to recursively recover the scalar bits, so that the attacker has to recover all the $i-1$ first bits to recover the i-th bit.

In this work, we are concerned with a very powerful type of EP attacks, denoted as Horizontal Differential Power Attack (HDPA) [2], [3]. As discussed in [1], these attacks are in the same time very interesting for security evaluations (since they are able to extract a lot of information from leakage traces) and cumbersome to mount. In order to systematize their analysis, the authors proposed a principled approach based on three steps – information identification, information extraction and information combination. Concretely, they performed the information identification with a correlation attack, and the next two steps with linear regression [4], which implies significant engineering efforts (in terms of time complexity, amount of data to collect and memory complexity). It also requires a good knowledge of the implementation and side-channel analysis techniques.

On the other hand, in the last few years, the development of machine learning/deep learning in the side-channel context [5]–[18] is booming as a powerful alternative to conventional profiling techniques. In this work, we are motivated to use deep learning techniques to simplify the complicated steps of the previous work [1]. For this purpose, we investigate the challenging case of a regular Montgomery ladder scalar multiplication implementation based on a BeagleBone Black (BBB) Board running at 1 GHz. We show that a good part of the single-trace attack by Poussier et al. can be automated, which is relevant to security evaluation labs since it provides them with an easier way to conduct worst-case security evaluations of ECC cryptosystems. Yet, our results also show that some alignment and preprocessing (i.e., information identification) remains necessary for the deep leaning attacks to succeed against such challenging targets. This conclusion is similar to the RSA case studied in a recent CHES'19 paper [6] by Carbone et al. using the signal-to-noise ratio (SNR) for points of interest (POIs) selection. Generally speaking, for asymmetric cryptography implementations, the number of sample points of target operations is normally pretty large, so POI selection is very important for the deep learning attack performance. To the best of our knowledge, this is the first public report of deep learning based attack on ECC implementations.

The rest of the paper is organized as followed. Section II introduces the necessary background on ECC scalar multiplication and the Fully Convolutional Networks (FCN) that we used. Section III shows the target implementation, the required preprocessing of the EM traces and deep learning experimental

results. Finally, we conclude and offer some directions for future research in Section IV. For the completeness and understating of our work, we still describe the target implementation details which are similar to [1].

II. Background

A. Notations

We use the same notations as in previous work [1]. In this work, capital letters and small caps are respectively used to denote random variables and their realizations. Functions (e.g. F) are denoted with sans serif font and calligraphic fonts for sets (e.g. $\mathcal{A}$). We use small bold caps for vectors (e.g. $\mathbf{v}$).

B. ECC Scalar Multiplication

We denote a finite field with a characteristic bigger than 3 as $\mathbb{F}_p$. The set of points $(x, y) \in \mathbb{F}_p^2$ (so-called affine coordinates) that satisfy the Weierstrass equation $y^2 = x^3 + ax + b$, $(a, b) \in \mathbb{F}_p^2$ with discriminant $\Delta = -16(4a^3 + 27b^2) \neq 0$ is defined as $\mathcal{E}(\mathbb{F}_p)$. A point at infinity $\mathcal{O}$ together with $\mathcal{E}(\mathbb{F}_p)$ build an Abelian additive group. We denote $[k]P$ as scalar multiplication, a k-times repeated point additions over $\mathcal{E}(\mathbb{F}_p)$, in which, $k \in \mathbb{N}$ is called a scalar and P is a curve point. $k \in [1, \#\mathrm{P}-1]$ and $\#P$ corresponds to the order of the subgroup generated by the point P.

For most of ECC cryptosystems, the scalar k is sensitive data because it is either directly used as a private key (e.g. ECDH key exchange) or used as an ephemeral key (e.g. ECDSA) related to the private key. Because of its regularity, Montgomery ladder scalar multiplication [19] as described in Algorithm 1 is representative of state-of-the-art implementations secure against SPA-like single trace attacks. As in previous work [1], we also target this representative implementation in this work. The results can be naturally applied to implementations using scalar randomization countermeasures against DPA attacks.

Algorithm 1 Montgomery ladder scalar multiplication (left-to-right).

Require: $\mathbf{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $\mathbf{k} = (k_{n-1}, ..., k_0)$
Ensure: $\mathbf{Q} = [k]\mathbf{P}$
  $\mathbf{R}_0 \leftarrow \mathcal{O}$; $\mathbf{R}_1 \leftarrow \mathbf{P}$;
  for $i = n - 1$ to 0 **do**
    $\mathbf{R}_{\neg k_i} \leftarrow \mathbf{R}_{\neg k_i} + \mathbf{R}_{k_i}$
    $\mathbf{R}_{k_i} \leftarrow [2]\mathbf{R}_{k_i}$
  end for
  return $\mathbf{R}_0$

In this work, we also convert the affine coordinates to Jacobian coordinates to avoid expensive filed inversion operations of point addition and point doubling as illustrated in Algorithm 2 and 3 in Appendix A. The cost of each point addition and each point doubling is $\mathtt{16MUL} + \mathtt{1ADD} + \mathtt{6SUB}$ and $\mathtt{10MUL} + \mathtt{9ADD} + \mathtt{4SUB}$, respectively.

C. Fully Convolutional Networks

Fully convolutional networks (FCN) have been first introduced for semantic segmentation on images [20] and have been achieved great success in that field. In addition, recently Facebook deployed their FCN-based speech recognition system [21]. FCN is a kind of CNN without fully connected layers, it recovers the pixel-level classification information from abstract features. We refer the readers to the original paper [20] for more details. The core difference between traditional Convolutional Neural Networks (CNN, such as AlexNet, VGGNet) and FCN is that traditional CNNs are suitable for image-level tasks but FCN is extended to be more suitable for pixel-level tasks. We consider sample points of side-channel traces as pixels and there is no previous work using FCN in side-channel context, so we want to introduce FCN into side-channel attacks. They have then been widely used for different applications, due to their compelling quality and efficiency. Classification is of course one of its application scenarios.

The design idea of FCN is simple: as depicted in Fig. 1, it is a stack of several basic blocks. Each basic block is composed of three layers: a convolutional layer γ followed by a batch normalization layer β [22] and a ReLU activation layer σ [23]. After stacking five basic blocks, a global average pooling layer δ is adopted (instead of a fully connected layer), in order to largely reduce the number of weights to be trained. Finally a softmax layer s is adopted to generate the class label of the input side-channel trace. We adopt the same strategy as in ResNet [24] to exclude any pooling operation for each basic block, in order to prevent overfitting. Batch normalization is applied to speed up the convergence and help improve generalization.

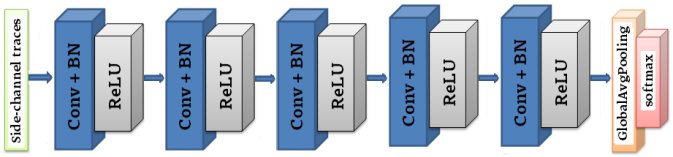


Fig. 1. Structure of FCN.

In summary, the FCN model can be written as:

$$\mathrm{FCN} = s \circ \delta \circ [\sigma \circ \beta \circ \gamma]^{n_1}, \tag{1}$$

where n_1 denotes the number of basic blocks (we set it to 5 for all our experiments). We further use 32, 128, 128, 256 and 256 filters for these five basic blocks. The kernel size of the first and last convolutional layer is 8 and 3, respectively. The other three convolutional layers are using the same kernel size of 5. Regarding our choice of hyperparameters, we followed the best practice of other deep learning application fields to do a small grid-search, that is, varying the number of basic blocks from 3 to 7, and then varying the optimizer among "Adadelta", "Adam", "Nadam", "RMSProp" and "SGD". For the number of filters we reuse the typical values used in most of computer

vision applications, and for the kernel size we also use small values since they showed better performance in other deep learning fields. We must mention that it is not a ideal way to decide the network structure and other hyperparameters, one of our future work direction is to use latest Neural Architecture Search (NAS) [25], [26] techniques to automatically designing effective neural network architecture for our data set.

D. Accuracy, Loss, First-Order Success Rate

Accuracy and *Loss* are twin metrics that are widely used in the machine learning community to monitor and evaluate neural network models. Training accuracy is the successful classification rate over the training data and training loss is the error rate over the training data. After each epoch, the trained model is applied to the validation data to calculate validation accuracy and validation loss. These two values indicate how good the trained model is at predicting outputs for inputs it has never seen before. Validation accuracy increases initially and saturates as the model starts to overfit.

For comparison with the previous work [1], we also use the same first-order Success Rate (1-O SR) metric, which is just the probability to recover the target n-bit scalar computed over repeated experiments.

III. Experimental Results on BeagleBone Black Implementation running at 1GHz

A. Target Implementation and Measurements

The finite field and elliptic curve arithmetic are implemented in assembly on our target BeagleBone Black Board. For comparison with the previous work, we also choose the NIST P-256 curve [27] as previous work and our attacks are independent of the curve being used, the regularity of the scalar multiplication implementation is the only requirement. We must note that, although our attacks are independent of the chosen curve, but the number of field multiplications for information identification step will be different. In our case, we have 26 field multiplications per iteration for information identification step. The regular Montgomery ladder scalar multiplication as described in Section II-B is implemented in ARM assembly using Jacobian coordinates. The point addition and doubling formulas are illustrated in Algorithm 2 and 3. An entire scalar multiplication takes approximately 17.000.000 clock cycles since our focus is constant time implementation without optimizations.

Regular scalar multiplication implementations (Montgomery ladder in this case) are composed of a fixed and predictable sequence of operations. All operations in the sequence that affect the internal state depending on the scalar bit value contain sensitive information. It is a hierarchical sequence. The top level is a loop of scalar bits handling. A fixed number of point additions and point doublings for each scalar bit are the second level. The third level is a fixed number of field operations per each point addition (resp. point doubling). At the bottom, a sequence of fixed number of register operations (such as register multiplications, additions and subtractions) forms a field operation. So the sequence of register operations for an n-bit scalar can be divided into n parts depending on the scalar bit index. We assume that each part consists of N register operations. In total, an entire regular binary scalar multiplication has n sequences of N sensitive operations. The N intermediate computation results occurring during the manipulation of the i-th scalar bit are denoted as $\mathbf{r}_i = (r_i^j), j \in [0, N-1]$. We denote by $\mathbf{l}_i = (l_i^j), j \in [0, N-1]$ side-channel leakages caused by each of these computations.

Field additions and subtractions are straightforwardly implemented using carry additions and subtractions. Field multiplications are conducted using the Long Integer Multiplication (LIM) followed by a modular reduction. LIM is easily implemented using the 32-bit unsigned multiplications `umull` and `umaal` (with accumulate) assembly instructions yielding a 64-bit result as depicted in Algorithm 4 in Appendix A. The modular reduction is implemented according to [27].

A modular reduction for both addition and multiplication is always executed for constant time against timing-type of attacks purpose. Both the results before and after reduction are saved in memory. A Boolean is computed whose value is true or false depending on the need or not of a reduction. This Boolean value will decide the result's pointer linked to the actual value to be returned. This Boolean value will be always true for multiplications since we need the modular reduction.

Our target BeagleBone Black board is a 32-bit AM335x 1GHz ARM Cortex-A8 linux-based single board computer.[1] This is a very challenging device in terms of side-channel analysis (see for example [28], [29]) as mentioned in previous work [1]. A full version of Ubuntu 14.04 is running on the board. A lot of noise and interruptions are introduced by the running Linux operating system and modern CPU design.

A Langer HV100-27 magnetic near field probe and a Lecroy WaveRunner 620Zi oscilloscope at a sampling rate of 10 GS/s are used to measure the EM emission. We set the CPU frequency to the highest 1 GHz and the CPU frequency governor to *'Performance'* during the measurements. We recorded the processing of the first 4 bits of the scalar. Each trace contains 2,000,000 sample points. We used the '`nohup`' trick as mentioned in [1] to avoid the long interruptions randomly appearing in the traces introduced by the running Linux system. Still, a lot of smaller interruptions are present in the EM traces.

B. Preprocessing of the traces

To handle those smaller interruptions, the preprocessing of the EM traces iterates over three steps. The first step is to align the traces around a field multiplication. Secondly we cut the traces around the aligned area into slices. Finally, each slice is concatenated to the set of preprocessed traces. We repeat these three steps for each field multiplication.

We use the same method as in previous work [1], which exploits correlation in order to synchronize the EM traces

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

focusing on the leakage part of the targeted sensitive data. This method works in three steps.

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

We use 100,000 preprocessed EM traces for profiling and 2,200 preprocessed traces to attack (similar to [1]). 20% of the profiling traces are used as validation data to improve the training of weights. Each preprocessed EM trace contains 518,491 sample points.

C. Neural Network Architecture

Our experiments are implemented using Keras [30] and Tensorflow [31] with Nvidia GTX 1080Ti GPU. We employ the aforementioned FCN neural network model in Section II-C. During our experiments, we use dropout of 0.2 for the last two basic blocks (after the ReLU activation layer) and of 0 for the rest [32]. All layers are randomly initialized with uniform initialization [33].

We use batch size of 1 due to the large number of sample points per trace and Adadelta optimizer with an initial learning rate of 1.0, adaptively reducing the learning rate with a factor of 10 if the validation accuracy is not increased within 30 consecutive epochs. The training will be stopped after 150 epochs or if the learning rate is getting lower than 10^{-3}.

D. BBB Results

As mentioned before, we are motivated to simplify the three critical steps of systematic approach proposed in previous work [1] using deep learning techniques. We take a two-step progressive strategy to conduct our experiments. We start with only simplifying the most sophisticated (information extraction and information combination) steps of the systematic approach by Poussier et al. We next try to automate all three steps (i.e. including the information identification).

1) Results with Information Identification: In this first experiment, we use the same information identification step as [1] to identify the POI to feed into the neural networks. This step consists of three parts: unprofiled correlation, partial SNR and optimizations.

We only focus on the higher 32-bit result of each umull and umaal instructions for all the available register operations $\mathbf{r}_i$ of each scalar bit.

We apply the unprofiled correlation attack and the partial SNR approach from [1] to efficiently identify the time positions of the corresponding registers r_i^j. The POI search is performed using the preprocessed $N_{poi} = 100,000$ traces $\mathbf{l}$ acquired using random known inputs (P^q) and scalars $(\mathbf{k}^q)$, $q \in [0, N_{poi} - 1]$. The N_{poi} leakages of the t-th time sample of each trace is denoted as a $\mathbf{l}[t]$ vector of size N_{poi}.

The unprofiled correlation approach boils down to compute Pearson's correlation coefficient ρ between each time sample for the N_{poi} internal values $\mathsf{r}_i^j = r_i^j(P^q, \lceil \mathbf{k}^q \rceil_i)$ and a Hamming weight leakage model HW. For our POI search, we compute $\rho(\mathsf{HW}(\mathsf{r}_i^j), \mathbf{l}[t])$, $t \in [0; 518,490]$.

To calculate partial SNR, the 32-bit values of r_i^j are first truncated to x bits. Then each trace is labeled with its truncated value and split into 2^x sets $\mathcal{S}_i$. The partial SNR of each time sample equals to $\frac{\mathsf{var}(\mathsf{mean}(\mathcal{S}_i))}{\mathsf{mean}(\mathsf{var}(\mathcal{S}_i))}$, in which, var and mean are the sample variance and mean functions. The time sample showing the highest SNR ratio is chosen as the time sample of r_i^j.

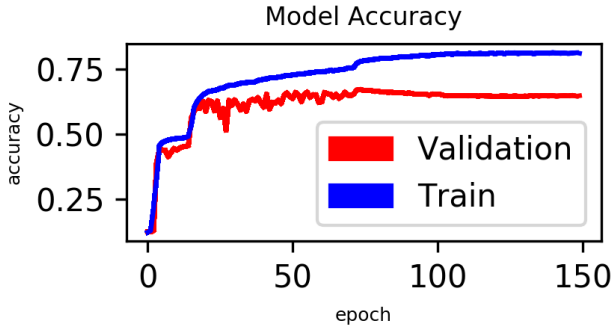
Applying each of these two methods on the full trace for all r_i^j's is computationally intensive. So using the fact that the time order of the registers is $(r_0^0, ..., r_0^{N-1}, r_1^0, ...r_i^j, ...r_{n-1}^{N-1})$, we first search r_0^0 among the first W time samples. Using correlation, we decide r_0^0's position by computing a p-value with a threshold of 5 [34]. We repeatedly move the window to the next W time samples until r_0^0 is found. The search of r_0^1 is similar with setting the initial offset of the window to r_0^0. We iterate this process to find all the registers. We use a window value W of 20,000 for our POI search.

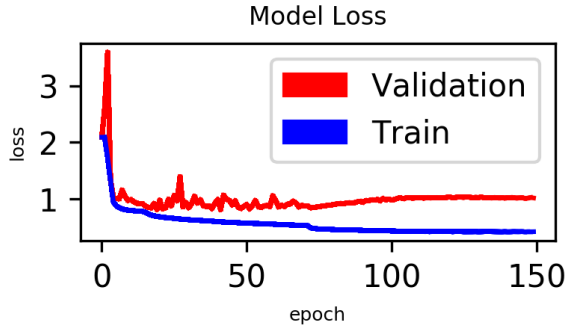
As in [1], we finally attack the first 4 bits of the scalar. The number of POIs is 3,492 so we feed those POI sample points into the neural network. We label each trace using the value of the first 4 bits of its scalar. The training accuracy/loss and validation accuracy/loss (as described in Section II-D) of the FCN model are given in the left graph of Fig. 2 when using 80,000 training traces and 20,000 validation traces out of 100,000 profiling traces. The training takes about 23 hours, it could be halved because the validation accuracy is getting stable after 70 epochs as can be seen from the accuracy graph.

After the training, we use the trained model to recover the 4 scalar bits of all 2,200 attack traces. We calculate the 1-O SR and use the same threshold mechanism as in [1] to discard the wrong attack results. That is, by setting a probability threshold under which some attack traces will be discarded. A higher threshold provides more confidence to have a successful partial nonce recovery, but it comes with the cost of increasing the number of discarded attack traces.

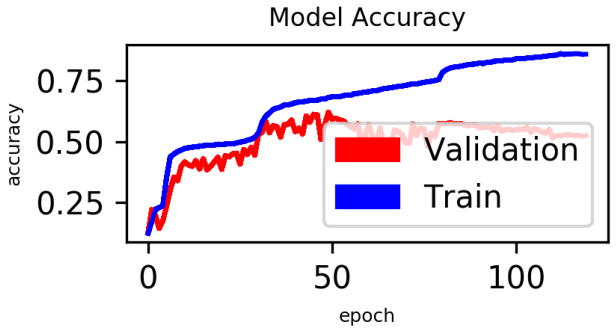
Concretely, we summarize in Table I the evolution of success rate in function of the probability threshold over the 2,200 attack traces. The scalar 1-O SR is 0.6692 when the threshold is set to 0.5. Yet we need 140 ECDSA nonces to recover the secret key with 4 bits of partial information using lattice attacks [1]. Since no error on the partial information is tolerated in a lattice attack context, the success rate of the key recovery is calculated as $0.6692^{140} \approx 3.7 \cdot 10^{-25}$. So we have to discard wrong attack results.

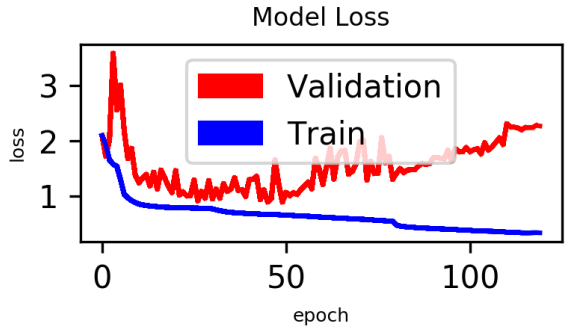
As can be seen, the first-order success rate is increasing with higher probability thresholds. We achieve a success rate of 1 using a threshold of 0.99. In that case, we discard 2,055 of the attack results and still remain 145 of them, which is remarkably similar to the results of Poussier et al.





(a) 80,000 training traces.





(b) 40,000 training traces.

Fig. 2. FCN Model Accuracy and Loss with POI selection.

TABLE I
EVOLUTION OF THE ECDSA SCALAR AND KEY RECOVERY SUCCESS RATE IN FUNCTION OF THE THRESHOLD

Threshold value	*Scalar 1-O SR*	*Key 1-O SR*	*# discarded result*	*# remaining result*
0.5	0.6691519105	$3.7 \cdot 10^{-25}$	764	1,436
0.75	0.7562225476	$1.0 \cdot 10^{-17}$	1,167	1,033
0.9	0.8251572327	$2.1 \cdot 10^{-12}$	1,564	656
0.95	0.8950495049	$1.8 \cdot 10^{-7}$	1,748	452
0.96	0.9090909091	$1.6 \cdot 10^{-6}$	1,800	400
0.97	0.9247910863	$1.7 \cdot 10^{-5}$	1,868	332
0.98	0.9503546099	0.0008	1,957	243
0.985	0.9629629630	0.0051	2,012	188
0.99	1	1	2,055	145

We further investigated the impact of reducing the number of training (resp. validation) traces to 40,000 (resp. 10,000). The bottom graph of Fig. 2 illustrates the training and validation accuracy/loss. The validation accuracy is worse than using 80,000 training traces. A similar observation holds for the scalar 1-O SR and key 1-O SR. Both suggest that the model is still improving for the amount of collected traces.

2) Results without Information Identification: We finally investigated the full automation of the three steps in Poussier et al.'s systematic approach.

For this purpose, we skipped the POI search and directly feed all the 518,491 sample points per trace into the neural networks. Due to the huge number of sample points per trace, the training is very time consuming. We use the same neural network architecture as in Section III-C to conduct the experiments, but in this case we only use 100 epochs because each epoch takes about 2.5 hours. Fig. 3 displays the training accuracy/loss and validation accuracy/loss. The training accuracy and validation accuracy are pretty low, the validation accuracy is stabilized at only 18%, which means very few scalar bit classes can be correctly identified, leading to unsuccessful attacks. The big drop of validation accuracy (spike of validation loss resp.) is caused by the dropout mechanism that we used during the training, and at that stage the weights of neurons have not converged yet. Later there is no big drop when the training is getting stabilized.

Regarding the failure of the attack, we provide the following tentative explanation. First, as we mentioned before, FCN model is designed for extracting pixel-level classification information from the raw data, taking into account that there are more than half million sample points (pixels) per trace, the tuning of neuron weights are depending on about 150 times more sample points (pixels) compared to the previous case of training with POIs. From deep learning point of view, generally it requires more data and more epochs to make the training converge, which we can not afford due to our computation power and will be considered in a future work. Secondly, the neural network model is directly reused from the previous experiment, which we did small grid-search based on the POIs dataset to choose the number of blocks and optimizer of FCN model. The adaptation of the network architecture

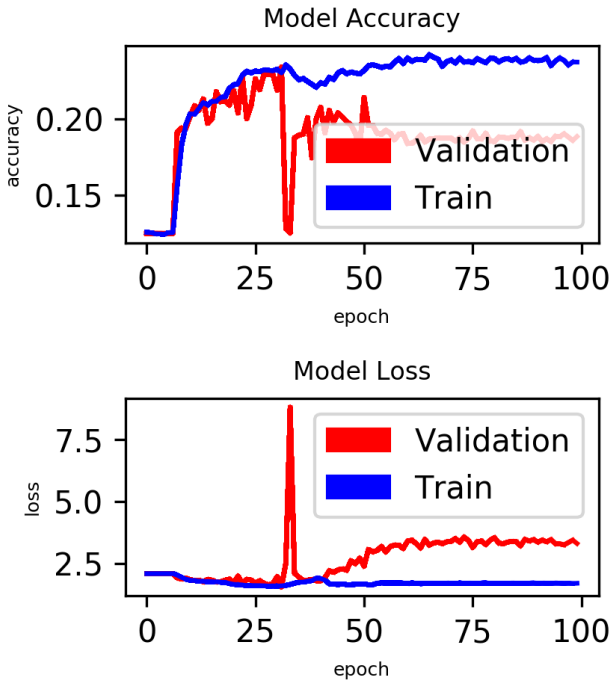


Fig. 3. FCN Model Accuracy and Loss without POI selection.

could be considered given our results. For example, NAS-type of techniques [25], [26] are interesting candidates for this purpose.

These results suggest that despite their versatility, deep learning based attacks benefit from a good selection of points of interest in case of challenging targets (in particular, it allows fitting the model with much less traces). The investigation of more suitable deep learning techniques for this kind of application is an interesting scope for further research.

IV. Conclusion and Future Work

Motivated by simplifying a previous systematic approach of worst-case HDPA attack on secure scalar multiplication implementations, we use deep learning techniques to automate the critical steps of such attacks. Our experimental results of an ARM Cortex-A8 running at 1 GHz (a presumably noisy target) suggest that the sophisticated information extraction and information combination steps can be performed in a quite black box manner using deep learning techniques. On the other hand, they also suggest the need of dimensionality reduction / detection of points of interest before launching the attack (or at least, our results show that such a preliminary step makes deep learning attacks significantly more efficient), which is a generic problem to tackle for attacking asymmetric cryptography implementations using deep learning techniques considering the large number of sample points per trace. Simple SNR-based POI selection could be a cheaper and faster alternative as demonstrated in single-trace attack on RSA case [6] using deep learning techniques.

For the developers of scalar multiplication implementations, our results confirm that scalar randomization activated implementations are generally at risk, considering both the huge amount of informative samples such implementations offer and the simplicity deep learning techniques provide. To mitigate this issue, point randomization is probably the most suitable solution. Evaluating that kind of implementations is an interesting track for further investigations. Another potential direction is to target register manipulations of point addition and point doubling operations from the elliptic curve arithmetic level.

V. Acknowledgment

François-Xavier Standaert is a Senior Research Associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the EU through the ERC project SWORD (Consolidator Grant 724725) and the H2020 project REASSURE.

References

[1] R. Poussier, Y. Y. Zhou, and F. X. Standaert, "A Systematic Approach to the Side-Channel Analysis of ECC Implementations with Worst-Case Horizontal Attacks," Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, Proceedings, pp 534–554, September 25-28, 2017.

[2] A. Bauer, É. Jaulmes, E. Prouff, and J. Wild, "Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations," Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco, CA, USA, Proceedings, pp 1–17, February 25-March 1, 2013.

[3] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil, "Horizontal Correlation Analysis on Exponentiation," Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, Proceedings, pp 46–61, December 15-17, 2010.

[4] W. Schindler, K. Lemke, and C. Paar, "A Stochastic Model for Differential Side Channel Cryptanalysis," Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, Proceedings, pp 30–46, August 29 - September 1, 2005.

[5] E. Cagli, C. Dumas, and E. Prouff, "Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing," Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, Proceedings, pp 45–68, September 25-28, 2017.

[6] M. Carbone, V. Conin, M. A. Cornelie, F. Dassance, G. Dufresne, C. Dumas, E. Prouff, and A. Venelli, "Deep Learning to Evaluate Secure RSA Implementations," IACR Trans. Cryptogr. Hardw. Embed. Syst., vol. 2, pp 132–161, February, 2019.

[7] B. Hettwer, S. Gehrer, and T. Güneysu, "Profiled Power Analysis Attacks Using Convolutional Neural Networks with Domain Knowledge," Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, Proceedings, pp 479–498, August 15-17, 2018.

[8] A. Heuser, and M. Zohner, "Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines," Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, Proceedings, pp 249–264, May 3-4, 2012.

[9] G. Hospodar, B. Gierlichs, E. D. Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," J. Cryptographic Engineering, pp 293–302, 2011.

[10] L. Lerman, G. Bontempi, and O. Markowitch, "A machine learning approach against a masked AES - Reaching the limit of side-channel attacks with a learning model," J. Cryptographic Engineering, pp 123–139, 2015.

[11] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F. X. Standaert, "Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis)," Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, Proceedings, pp 20–33, April 13-14, 2015.

[12] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking Cryptographic Implementations Using Deep Learning Techniques," Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, Proceedings, pp 3–26, December 14-18, 2016.

[13] C. Pfeifer, and P. Haddad, "Spread: a new layer for profiled deep-learning side-channel attacks," IACR Cryptology ePrint Archive, https://eprint.iacr.org/2018/880, 2018.

[14] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations," IACR Trans. Cryptogr. Hardw. Embed. Syst., vol. 1, pp 209–237, 2019.

[15] S. Picek, A. Heuser, A. Jovic, and A. Legay, "Climbing Down the Hierarchy: Hierarchical Classification for Machine Learning Side-Channel Attacks," Progress in Cryptology - AFRICACRYPT 2017 - 9th International Conference on Cryptology in Africa, Dakar, Senegal, Proceedings, pp 61–78, May 24-26, 2017.

[16] E. Prouff, R. Strullu, R. Benadjila, E. Cagli, and C. Dumas. "Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database," IACR Cryptology ePrint Archive, https://eprint.iacr.org/2018/053, 2018.

[17] P. Robyns, P. Quax, and W. Lamotte, "Improving CEMA using Correlation Optimization," IACR Trans. Cryptogr. Hardw. Embed. Syst., vol. 1, pp 1–24, 2019.

[18] B. Timon, "Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis," IACR Trans. Cryptogr. Hardw. Embed. Syst., vol. 2, pp 107–131, 2019.

[19] M. Joye, and S. M. Yen, "The Montgomery Powering Ladder," Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, Proceedings, pp 291–302, August 13-15, 2002.

[20] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, Proceedings, pp 3431–3440, June 7-12, 2015.

[21] N. Zeghidour, Q. T. Xu, V. Liptchinsky, N. Usunier, G. Synnaeve, and R. Collobert, "Fully Convolutional Speech Recognition," CoRR, http://arxiv.org/abs/1812.06864, 2018.

[22] S. Ioffe, and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, Proceedings, pp 448–456, 6-11 July 2015.

[23] V. Nair, and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, Proceedings, pp 807–814, June 21-24, 2010.

[24] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, Proceedings, pp 770–778, June 27-30, 2016.

[25] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient Neural Architecture Search via Parameter Sharing," Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, Proceedings, pp 4092–4101, July 10-15, 2018.

[26] H. Cai, L. G. Zhu, and S. Han, "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware," Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, Proceedings, May 5-9, 2019.

[27] PUB, NIST FIPS, "186-2: Digital Signature Standard (DSS)," National Institute for Standards and Technology, 2000.

[28] J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede, "DPA, Bitslicing and Masking at 1 GHz," Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, Proceedings, pp 599–619, September 13-16, 2015.

[29] J. Longo, E. D. Mulder, D. Page, and M. Tunstall, "SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip," Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, Proceedings, pp 620–640, September 13-16, 2015.

[30] C. François, and others, "Keras, " GitHub, https://github.com/fchollet/keras, 2015.

[31] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. F. Chen, C. Citro, et al., "TensorFlow: Large-scale machine learning on heterogeneous systems, " 2015, Software available from https://tensorflow.org.

[32] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, pp 1929–1958, 2014.

[33] X. Glorot, and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, Proceedings, pp 249–256, May 13-15, 2010.

[34] F. Durvaux, and F. X. Standaert, "From Improved Leakage Detection to the Detection of Points of Interests in Leakage Traces," Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, Proceedings, Part I, pp 240–262, May 8-12, 2016.

APPENDIX

Algorithm 2 Jacobian Addition.

Require: $\mathbf{P} = (X_1, Y_1, Z_1), \mathbf{Q} = (X_2, Y_2, Z_2), \mathbf{P} \neq \pm\mathbf{Q}$
Ensure: $\mathbf{P} + \mathbf{Q} = (X_3, Y_3, Z_3)$
$\mathsf{A} \leftarrow Z_1^2, \mathsf{B} \leftarrow Z_2^2, C \leftarrow X_1\mathsf{B}, D \leftarrow X_2\mathsf{A}, E \leftarrow C - D, F \leftarrow Y_1\mathsf{B}Z_2, G \leftarrow Y_2\mathsf{A}Z_1, H \leftarrow F - G, \mathsf{I} \leftarrow E^2, J \leftarrow \mathsf{I}E, K \leftarrow C\mathsf{I}$
$X_3 \leftarrow H^2 + J - 2K$
$Y_3 \leftarrow H(K - X_3) - FJ$
$Z_3 \leftarrow Z_1 Z_2 E$
return (X_3, Y_3, Z_3)

Algorithm 3 Jacobian Doubling.

Require: $\mathbf{P} = (X_1, Y_1, Z_1)$
Ensure: $\mathbf{P} + \mathbf{P} = (X_2, Y_2, Z_2)$
$\mathsf{A} \leftarrow X_1^2, \mathsf{B} \leftarrow Y_1^2, \mathsf{C} \leftarrow Z_1^2, D \leftarrow 3\mathsf{A} + a\mathsf{C}^2, E \leftarrow \mathsf{B}^2, F \leftarrow 4X_1B$
$X_2 \leftarrow D^2 - 2F$
$Y_2 \leftarrow D(F - X_2) - 8E$
$Z_2 \leftarrow 2Y_1 Z_1$
return (X_2, Y_2, Z_2)

Algorithm 4 Long Integer Multiplication in ARM Assembly.

Require: $x = (x_7, ..., x_0), y = (y_7, ..., y_0)$
Ensure: $m = x \times y = (m_{15}, ..., m_0)$

$m \leftarrow 0$
$\mathrm{r}_2 \leftarrow \mathsf{load}(x_7)$
$\mathrm{r}_3 \leftarrow \mathsf{load}(y_7)$
$\mathrm{r}_0, \mathrm{r}_1 \leftarrow \mathsf{umull}(\mathrm{r}_2, \mathrm{r}_3)$
$m_{15} \leftarrow \mathsf{store}(\mathrm{r}_0)$
for $i = 6$ to 0 **do**
  $\mathrm{r}_3 \leftarrow \mathsf{load}(y_i)$
  $\mathrm{r}_{i \bmod 2} \leftarrow 0$
  $\mathrm{r}_{1-i \bmod 2}, \mathrm{r}_{i \bmod 2} \leftarrow \mathsf{umaal}(r_2, r_3)$
  $m_{8+i} \leftarrow \mathsf{store}(\mathrm{r}_{1-i \bmod 2})$
end for
$m_7 \leftarrow \mathsf{store}(\mathrm{r}_0)$
for $i = 6$ to 0 **do**
  $\mathrm{r}_2 \leftarrow \mathsf{load}(x_i)$
  $\mathrm{r}_3 \leftarrow \mathsf{load}(y_7)$
  $\mathrm{r}_0 \leftarrow 0$
  $\mathrm{r}_1 \leftarrow 0$
  $\mathrm{r}_0, \mathrm{r}_1 \leftarrow \mathsf{umull}(\mathrm{r}_2, \mathrm{r}_3)$
  $\mathrm{r}_3 \leftarrow \mathsf{load}(m_{8+i})$
  $\mathrm{r}_0 \leftarrow \mathsf{adds}(\mathrm{r}_0, r_3)$
  $m_{8+i} \leftarrow \mathrm{store}(\mathrm{r}_0)$
  for $j = 6$ to 0 **do**
    $\mathrm{r}_3 \leftarrow \mathsf{load}(y_j)$
    $\mathrm{r}_{j \bmod 2} \leftarrow 0$
    $\mathrm{r}_{1-j \bmod 2}, \mathrm{r}_{j \bmod 2} \leftarrow \mathsf{umaal}(r_2, r_3)$
    $\mathrm{r}_3 \leftarrow \mathsf{load}(m_{i+j+1})$
    $\mathrm{r}_{1-j \bmod 2} \leftarrow \mathsf{adcs}(\mathrm{r}_{1-j \bmod 2}, \mathrm{r}_3)$
    $m_{i+j+1} \leftarrow \mathrm{store}(\mathrm{r}_{1-j \bmod 2})$
  end for
  $\mathrm{r}_0 \leftarrow \mathsf{adcs}(\mathrm{r}_0, 0)$
  $m_i \leftarrow \mathsf{store}(\mathrm{r}_0)$
end for
return m