Physical Attacks on Secure Systems (NWI-IMC068)

Second Assignment: Profiled Attacks and Fault Injection

IMPORTANT: do not distribute the files provided for this assignment!

Teacher	Email	Responsible for
Parisa Amiri Eliasi	parisa.amirieliasi@ru.nl	Part A – Template Attack
Azade Rezaeezade	azade.rezaeezade@ru.nl	Part B – Deep Learning Attack
Durba Chatterjee	durba.chatterjee@ru.nl	Part C – Fault Injection

Goals: After completing these exercises successfully, you should be able to perform the following tasks:

- Perform Template Attack against an unprotected Elliptic Curve Cryptography (ECC) implementations.
- Perform deep-learning assisted attacks against unprotected ECC implementations.
- Learn to work with an open-source fault simulator and evaluate the robustness of software code to fault injection attacks.

Grading: This assignment has 10 points, obtained by answering the questions.

Assignment	Points
Part A	4 points
Part B	3 points
Part C	3 points

Deadline: Sunday, June 9, 2024, 23:59 sharp!

Handing in your answers:

- You should hand in your solutions via the assignment module in Brightspace.
- Make sure that your name and student number for both team members are on top of the first page!

- Your answers shall be provided as a `.pdf` report giving full explanation and interpretation of any results you obtain. Results without explanation will be awarded 0 points.
- It is mandatory to submit your code that can *reproduce* your reported results. Submit your Python code. Answers not supported by reproducible results are *not* graded.

Your teacher will give feedback on the submitted report, which you will receive via email sent to you directly. You will see your grades in Brightspace later.

Before you start: This assignment should be done using Python. The steps required for solving the assignment are the following:

1. You can work on this assignment in a group of two or alone.
2. Download the provided files (data and code), see Table 1.

File name	**Description**
`Assignment_2A_TA.ipynb`	Part A – [CODE] `jupyter` notebook. Download it here.
`Assignment_2B_DL.ipynb`	Part B – [CODE] `jupyter` notebook. Download it here.
`databaseEdDSA.h5`	Part A&B – [DATA] contains power traces for implementation of the scalar multiplication operation for ECC. You need this dataset for the profiled attacks, namely Template and Deep Learning attacks. Download it here.

Table 1: Files to download for solving the second assignment – Part A and B

1 Profiled Attacks against an ECC implementation

This part of the assignment aims to implement profiled attacks (template and deep learning attacks) against elliptic-curve cryptography implemented on an ARM Cortex-M4 processor.

1.1 Preliminaries and Background

The algorithm used is Ed25519, the EdDSA algorithm with Curve25519 (see Algorithm 1, and the document that describes EdDSA).

Algorithm 1 EdDSA Signature generating and verification

Keypair Generation: (Used once, the first-time private key is used.)
Input: k, **Output:** a, b, P

Hash k such that $H(k) = (h_0, h_1, \ldots, h_{2u-1}) = (a, b)$
$a = (h_0, \ldots, h_{u-1})$, interpret as integer in little-endian notation
$b = (h_u, \ldots, h_{2u-1})$
Compute public key: $P = aG$.

Signature Generation:
Input: M, a, b, P **Output:** R, S

Compute ephemeral private key $r = H(b, M)$.
Compute ephemeral public key $R = rG$.
Compute $h = H(R, P, M) \mod l$.
Compute: $S = (r + ha) \mod l$.
Signature pair (R, S)

Signature Verification:
Input: M, P, R, S, **Output:** {True, False}

Compute $h = H(R, P, M)$
Verify if $8SG = 8R + 8hP$ holds in E

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

In this attack, we focus on recovering the ephemeral key by using *only one* trace of the full scalar multiplication.

The implementation of Ed25519 on Cortex-M4 is a window-based method with radix-16, making use of a pre-computed table containing results of the scalar multiplication of $16^i|r_i| \cdot G$, where $r_i \in [-8, 7] \cap \mathbb{Z}$ and G is the base point of Curve25519.

To recover the complete key, we will need to use the divide-and-conquer approach and recover the 16 parts of the ephemeral key associated with each iteration of the scalar multiplication.

The traces contain different iterations of the scalar multiplication. They are divided into ['Attack_traces', 'Profiling_traces'], and each set contains ['traces', 'label'] traces with associated labels.

- Compute the template prediction for each operation for all traces in the trace set: $p(k) = -\frac{1}{2}\sum_{i=1}^{N_t}(t_i - \mu_k)C_{inv}(t_i - \mu_k)^T$, where C_{inv} is the inverted pooled covariance matrix, and μ_k is the mean of operation k. The pooled covariance matrix is calculated by averaging over all computed covariance matrices for all classes/operations: $C_{inv} = (\frac{1}{N_c}\sum_{k=1}^{N_c} C_k)^{-1}$. $\frac{1}{N_c}\sum_{k=1}^{N_c} C_k$ is the pooled covariance matrix and C_k is the covariance matrix for class k.

4.1) What trace set(s) do we use to match our templates with?

4.2) Apply the templates and explain what does the result mean?

4.3) Once the probabilities for every trace are computed with every key hypothesis, compute the success rate of order from 1 to 16 for the test trace set and plot the result.

4.4) What happens if we increase/decrease the number of principal components that we use in the template building phase?

5. Compute and report the number of available traces for each template. Are the traces distributed equally among different templates? How could this impact the attack results? Explain your answer.

1.3 PART B: Deep Learning Attack

In this part of the assignment, you will use deep neural networks to attack the ECC implementation. The dataset is the same as the one you used for Part A.

The model used here is a CNN (convolutional neural network). The model's architecture is shown in Figure. 1. The input goes through 9 convolution blocks (blue blocks), and every second block performs batch normalization (orange blocks). Two fully connected layers (purple blocks) follow the convolution blocks, and before each fully connected layer, we add a dropout layer. Also, there is a flatten layer between the first dropout and the first fully connected layer (not shown in the picture). The architectural and training hyperparameters for the CNN model are listed in Table 2. The hyperparameters not listed in this table were not set, and their default values will be taken during the training.

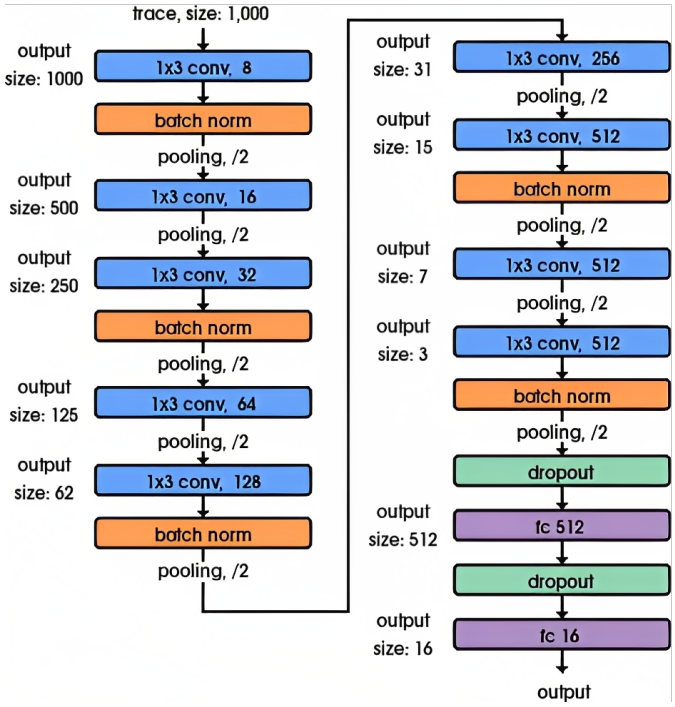

Figure 1: Convolutional Neural Network model to attack ECC

Table 2: Architectural and training hyperparameters

Hyperparameter	**Value**
CNN's Architecture Hyperparameters	
Number of convolution layers	9
Kernel size	3
Number of filters in $i^{(th)}$ layer	$min(8 * (2^i), 512)$
Pooling type	"Max"
Pooling stride	2
Number of dense layers	2
Number of neurons in dense layers	512, 16
CNN's Learning Hyperparameters	
Padding	"same"
Activation function (except for last layer)	"relu"
Activation function (last layer)	"softmax"
Optimizer	"Adam"
Loss	"sparse categorical crossentropy"
Epochs	25
Dropout rate	0.5

To complete this part of the assignment, you should train this CNN model. However, before that, you need to load the dataset, shuffle your training set, and reshape your traces to be compatible with the convolution layer input. You also need to split the

training set into training and validation sets using the split rate of 0.7. Considering the convolutional neural network in Figure 1 and the hyperparameters in Tabble 2, answer the following questions.

1. Using the demonstrated layers in Figure 1 and hyperparameters in Tabble 2, build your CNN.

2. Considering a single training (or test) example, how many inputs and outputs does our neural network have? Where do these numbers come from?

3. What are the outputs of your neural network, considering that you have a softmax activation function in the last layer?

4. Using the training set in the downloaded dataset and the training hyperparameters in Table 2, train your model. Report the training and validation accuracy and loss you get.

5. Use your trained model to guess about the key using a single trace in the attack set. To determine your single attack trace, use the sum of your student numbers' last two digits mod 16 as the following example. Let us take the following s-numbers: s1005614 belongs to team member 1, and s1235676 belongs to team member 2. In this example, the index of the single attack trace in the attack set is calculated as $(14 + 76) \bmod 16 = 10$. Please note that since the labels in the attack set are the key values, only the correct solution will give you points for this question.

6. The traces in the attack set of this dataset are collected using different values for the key. Considering this fact, can we calculate the guessing entropy as before to report our attack performance? Why? What metrics do you recommend using? Explain.

7. Print or plot the confusion matrix for the attack set using your model. Reflect on the acquired confusion matrix.

8. Fix number of epochs to 5. Then, rebuild and retrain your neural network again with the previous parameters. Using the confusion matrix, check the performance of your model. Repeat this procedure with the same values ten times. What do you notice? What is your guess about the source of your observation? What is your recommendation for preventing this effect during a real attack?

9. Try an MLP model with whatever architectural hyperparameters you like, but keep the training hyperparameters the same. Use the trained model to report the attack set's accuracy and confusion matrix. Compare the performance of this MLP with that of CNN. Is the performance better or worth comparing to CNN? What can you notice regarding the model selection and hyperparameter set?

10. Investigate the influence of training hyperparameters using the MLP you built. First, replace the current loss function with "categorical cross-entropy". Consider

using one-hot-encoding for this type of loss function. Rebuild and retrain your neural network. What is the effect of this change on the attack performance? Find two more hyperparameters that will affect your attack performance. Report the effect of those hyperparameters.

11. Considering questions 8 to 10, can you name at least two difficulties we need to overcome when using deep neural networks for side-channel analysis?