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

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

    **Signature Verification:**
    **Input:** $M, P, R, S$, **Output:** {True, False}
10: Compute $h = H(R, P, M)$
11: 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.

1.2 Part A: Template Attack

In this attack, our operations match the possible labels or classes. To create templates, perform the following procedure.

1. Why are we using profiled attacks to recover the ephemeral key? Explain your answer.

2. How many classes/operations do we have to build templates for?

3. Build the templates:

   - Group the training data according to the training labels (i.e., the different nibble values). Compute the mean of each group, μ_k, where $k \in \{1, .., N_c\}$, and N_c is the total number of classes.
   - Compute the mean of all the groups, $\bar{\mu} = \frac{1}{N_c}\sum_{k=1}^{N_c} \mu_k$.
   - Compute $B = \frac{1}{N_c}\sum_{k=1}^{N_c}(\mu_k - \bar{\mu})(\mu_k - \bar{\mu})^T$. The result is a matrix.
   - Perform the Singular Value Decomposition (SVD) of B. You can use `U,S,V = np.linalg.svd(B)` to perform SVD. *Hint: look at Numpy API reference for SVD computation.*
   - Choose the number of principal components (dimensions), m, and build $\text{U}_{\text{reduced}}$. Select the number of principal components by adding your two student numbers mod four and looking at the following table. You will see two different values. Try your attack once by selecting the smaller number of principal components and once with the larger one!
   - perform `U_reduced = U[:,:m]`.

(sNum1+SNum2)mod 4	#principal components
0	$m = 2$, $m = 8$
1	$m = 3$, $m = 9$
2	$m = 4$, $m = 10$
3	$m = 5$, $m = 11$

   - Project all datasets (profiling and attack) using $\text{U}_{\text{reduced}}$, i.e., perform matrix multiplication to produce matrices of shape $(n_{\text{training\_traces}}, m)$ and $(n_{\text{test\_traces}}, m)$.
   - Compute the mean vector and the covariance matrix from the projected training dataset.

   3.1) What trace set(s) do we use to build our templates upon?

   3.2) How many templates we will have?

   3.3) What are those templates? Explain.

   3.4) What is the dimension of the B matrix? Why?

   3.5) Explain what is Principal Component Analysis (PCA) and why do we use it?

   3.6) Explain what is SVD and why do we use it?

   3.7) What is the meaning of this step: `U_reduced = U[:,:m]`.

4. Match the templates:

- 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.