

Exercise on Template Attacks

1 Dataset and Cryptographic Implementation

1. The provided dataset was captured using the Chipwhisperer-Lite device¹. The captured traces record 100k encryptions in an ARM-Cortex-M4 CPU using the AES-128 algorithm. The dataset can be downloaded here:

<https://doi.org/10.21942/uva.25377694>

The dataset is available in both Python `npz` format and in Matlab `mat` format. We recommend either language for coding the exercise.

2. The purpose of the exercise is to create templates that can distinguish between the 256 values k of a single byte i.e. $k \in \{0, 1, \dots, 255\}$
3. The provided dataset consists of a training set (`traces_train`), a validation set (`traces_valid`) and a test set (`traces_test`). The dataset partitioning is done in an 80-10-10 manner i.e. 80% of the data is used for training, 10% for validation and 10% for testing. Thus the training set contains 80k traces, the validation set contains 10k traces and the test set contains 10k traces. Every trace consists of 100 time samples.
4. The partitioned dataset also contains the related labels i.e. 80k labels for training (`labels_train`), 10k labels for validation (`labels_valid`) and 10k labels for testing (`labels_test`). Every label is a single-byte value $k \in \{0, 1, \dots, 255\}$ and is equal to the output of a single AES sbox during the 1st round of AES-128. Note that recovering the single-byte sbox output k through a template attack is equivalent to recovering a single byte of the AES-128 round key, since the plaintext is typically known.

2 Splitting and Preprocessing the Dataset

1. **Group Splitting.** In the training dataset, use the values of the 80k training labels to split the training traces into 256 groups G_0, G_1, \dots, G_{255} , each containing the data of every class, in the following manner:

¹<https://rtfm.newae.com/Capture/ChipWhisperer-Lite/>

$$\begin{aligned}
G_0 &= \{\text{all training traces such that training label} = 0\} \\
G_1 &= \{\text{all training traces such that training label} = 1\} \\
&\vdots \\
G_{255} &= \{\text{all training traces such that training label} = 255\}
\end{aligned}$$

Repeat the exact same process in the 10k-trace validation set (using the validation labels), producing groups V_0, V_1, \dots, V_{255} and finally in the 10k-trace test set (using the test labels), producing groups H_0, H_1, \dots, H_{255} .

2. **Preprocessing (optional task).** Before constructing the side-channel templates, many analysts perform preprocessing i.e. they apply techniques such as resampling, frequency filtering, denoising, interpolation and others in order improve the signal quality. After completing all the tasks of the template exercise, you may select any preprocessing technique and apply it to the dataset. Investigate the effect of the chosen technique on the accuracy of the template attacks.

3 Reduced Templates

1. **Template Building.** In the training set, we have groups G_0, G_1, \dots, G_{255} each representing a class. You will use these groups to construct the reduced templates for every class, by applying the following steps:

Assume that the group G_0 contains $|G_0|$ traces \mathbf{t}_i^0 with $i = 1, 2, \dots, |G_0|^2$. The dimensions of each trace \mathbf{t}_i^0 are 1×100 , since each trace is a vector with 100 time samples. Compute the reduced template \mathbf{r}_0 for class 0 as follows:

$$\mathbf{r}_0 = \frac{1}{|G_0|} \sum_{i=1}^{|G_0|} \mathbf{t}_i^0$$

Note that the dimensions of \mathbf{r}_0 are also 1×100 . Repeating this process for the rest of the groups (G_1, G_2, \dots, G_{255}), compute:

$$\mathbf{r}_k = \frac{1}{|G_k|} \sum_{i=1}^{|G_k|} \mathbf{t}_i^k, \quad \text{for } k = 1, 2, \dots, 255$$

where $|G_k|$ = number of traces in group G_k , \mathbf{t}_i^k are the traces from group G_k , indexed for $i = 1, 2, \dots, |G_k|$ and \mathbf{r}_k is the reduced template for class k .

²The term $|G_0|$ denotes the “cardinality” of group 0.

The 256 vectors $\mathbf{r}_0, \mathbf{r}_1, \dots, \mathbf{r}_{255}$ are the reduced templates for all 256 classes. To perform the necessary vector operations you can use Python `numpy.mean` or the Matlab `mean` function.

2. **Template Matching.** In the test set, we have groups H_0, H_1, \dots, H_{255} . You will use these groups to compute the total accuracy³ of the reduced template classifier. For a single trace \mathbf{t}_i^0 in group H_0 with $i \in \{1, 2, \dots, |H_0|\}$, you can compute the score w.r.t. class 0, denoted as $score_0$, using the inner product, as follows:

$$score_0 = (\mathbf{t}_i^0 - \mathbf{r}_0) * (\mathbf{t}_i^0 - \mathbf{r}_0)^\top$$

The $score_0$ quantifies the distance between the 100-dimensional vectors \mathbf{t}_i^0 and \mathbf{r}_0 i.e. it quantifies how well does the trace \mathbf{t}_i^0 match to the reduced template for class 0. Note that $score_0$ is a scalar i.e. it has dimensions 1×1 . You must compute the matching score between trace \mathbf{t}_i^0 and all 256 reduced templates.

$$score_k = (\mathbf{t}_i^0 - \mathbf{r}_k) * (\mathbf{t}_i^0 - \mathbf{r}_k)^\top, \quad \text{for all } k = 0, 1, \dots, 255$$

We now use k^* to denote the $k \in \{0, 1, \dots, 255\}$ for which the $score_k$ is minimum. Finding this k^* means that you found the best matching class for trace \mathbf{t}_i^0 , according to the reduced template classifier.

$$k^* = \operatorname{argmin}_k (score_k), \quad \text{with } k \in \{0, 1, \dots, 255\}$$

Using k^* you can compute the classifier accuracy. Since trace \mathbf{t}_i^0 belongs to group H_0 (and thus class 0), then classifying \mathbf{t}_i^0 with an ideal classifier should result in $k^* = 0$ and you would have a hit. If that doesn't happen, then you would have a miss. Using this principle, count the number of hits in group H_0 and continue by counting the number of hits in groups H_1, H_2, \dots, H_{255} . By summing the number of hits you can now compute the accuracy:

$$\text{accuracy} = \frac{\text{total no. of hits}}{\text{total no. traces in test set}} = \frac{\text{total no. of hits}}{10000}$$

3. **Random Classifier.** After training and computing the accuracy in the test set, a sanity check is to compare the obtained accuracy to that of a random classifier:

$$\text{accuracy} \leq \frac{1}{\text{no. of classes}} = \frac{1}{256}$$

³Note that in the side-channel literature the accuracy of a classifier is often referred to as the ‘‘Success Rate (SR)’’ of the attack.

4. **Higher-Order Accuracy (optional task).** The standard accuracy metric counts how many times the correct⁴ class is the top guess of the classifier. In the context of side-channel attacks, we are also interested whether the correct class is within the top- m guesses of the classifier and we refer to that metric as accuracy (or success rate) of order m .

$$\text{accuracy-order-}m = \frac{\text{total no. of top-}m \text{ hits}}{\text{total no. traces in test set}}$$

For example, assume test trace \mathbf{t}_i^0 from group H_0 (and thus from class 0). For that trace we compute 256 scores ($score_0, score_1, score_2, \dots, score_{255}$) and then order them from minimum (best) to maximum (worst) from left to right.

$$\text{ordered scores} = (score_{m_0}, score_{m_1}, score_{m_2}, \dots, score_{m_{255}})$$

The index m_0 denotes the index of the minimum score and the index m_{255} denotes the index of the maximum score. Ideally, since trace \mathbf{t}_i^0 comes from class 0, it should be that $score_{m_0} = score_0$, i.e. the best score index m_0 should be equal to the ground truth class 0 and since $m_0 = 0$ we would have a top-1 hit. However, if the index 0 is in the top-5 i.e. if $0 \in \{m_0, m_1, m_2, m_3, m_4\}$, then we can say that we have a top-5 hit instead.

Using this description, the constructed reduced templates ($\mathbf{r}_0, \mathbf{r}_1, \dots, \mathbf{r}_{255}$) and the test set, compute the accuracy-order-5 metric. Note that achieving a fairly high accuracy-order-5 across the 16 AES-128 key bytes implies that the adversary can brute-force the remaining candidates, right after the side-channel attack, to recover the full key. For example, consider 16 separate template attacks targeting the 16 key bytes of AES-128. Ideally, every attack should place the correct key byte at the top-1 position. Still, even if the correct key byte is placed in the top-5 instead, this implies that an adversary can recover the full 16-byte key after $5^{16} \approx 2^{38}$ key guesses.

4 Full Template Attack

1. **Dimensionality Reduction.** Before training the full templates, you will reduce the dataset dimensionality using principal component analysis (PCA)⁵. The goal is to reduce the 100-dimensional traces to f -dimensional traces with $f < 100$. Apply the following steps:

⁴Correct in the sense that it corresponds to the ground truth.

⁵Dimensionality reduction in the side-channel literature is often referred to as “Point of Interest (PoI) selection”. The PCA technique for full templates can be found here: <https://www.iacr.org/archive/ches2006/01/01.pdf>

- (a) Compute the mean vector $\bar{\mathbf{r}}$ of the 256 reduced template vectors \mathbf{r}_k with $k = 0, 1, \dots, 255$. The computed vector $\bar{\mathbf{r}}$ has dimensions 1×100 .

$$\bar{\mathbf{r}} = \frac{1}{256} * \sum_{k=0}^{255} \mathbf{r}_k$$

- (b) Compute the matrix B using the outer product $(\mathbf{r}_k - \bar{\mathbf{r}})^\top * (\mathbf{r}_k - \bar{\mathbf{r}})$ as follows. The computed matrix B has dimensions 100×100 .

$$B = \frac{1}{256} * \sum_{k=0}^{255} (\mathbf{r}_k - \bar{\mathbf{r}})^\top * (\mathbf{r}_k - \bar{\mathbf{r}})$$

- (c) Compute the singular value decomposition of matrix B and decompose B to matrices U , S , V such that $B = U * S * V^\top$.

$$(U, S, V) = \text{svd}(B)$$

In Python you can use the `scipy.linalg.svd` and in Matlab the `svd` function to perform the decomposition. The matrix U has dimensions 100×100 .

- (d) Select the number f of dimensions (principal components) that you want by selecting the first f columns $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_f$ of matrix U and produce the reduced matrix U_{red} . The dimension of every column \mathbf{u}_i is 100×1 .

$$U = [\mathbf{u}_1 \ \mathbf{u}_2 \ \dots \ \mathbf{u}_{100}]$$

$$U_{red} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \dots \ \mathbf{u}_f]$$

The computed matrix U_{red} has dimensions $100 \times f$. In Python you can use `U[:,0:f]` and in Matlab you can use `U(:,1:f)`.

- (e) Reduce the dimensionality of every trace \mathbf{t} in the training set, the validation set and the test set by projecting the trace \mathbf{t} to a lower dimension trace \mathbf{t}_{red} , using the reduced matrix U_{red} as follows:

$$\mathbf{t}_{red} = \mathbf{t} * U_{red}$$

Note that the the trace \mathbf{t} has dimensions 1×100 , thus the reduced trace \mathbf{t}_{red} has dimensions $1 \times f$ with $f < 100$.

- (f) After reducing the dimension of every trace, make sure that all the training, validation and testing groups G_k, V_k, H_k with $k = 0, 1, \dots, 255$ contain traces with f time samples instead of 100.

2. **Template Building.** After reducing the dimension of every trace from 100 to f , the training set contains reduced groups G_0, G_1, \dots, G_{255} , each representing a class. You will use these groups to construct the full templates, by applying the following steps:

- (a) **Mean Vector.** Compute the mean vector \mathbf{m}_k for all groups G_k with $k = 0, 1, \dots, 255$. The mean vector \mathbf{m}_k and the trace \mathbf{t}_i^k have dimensions $1 \times f$.

$$\mathbf{m}_k = \frac{1}{|G_k|} \sum_{i=1}^{|G_k|} \mathbf{t}_i^k, \quad \text{for } k = 0, 1, \dots, 255$$

- (b) **Covariance Matrix.** Compute the covariance matrix Σ_k for all groups G_k with $k = 0, 1, \dots, 255$. The covariance matrix Σ_k has dimensions $f \times f$.

Given vectors $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]$ and $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_n]$ we define the covariance between vectors \mathbf{x} and \mathbf{y} as follows:

$$\text{cov}(\mathbf{x}, \mathbf{y}) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu_{\mathbf{x}}) * (y_i - \mu_{\mathbf{y}})$$

$$\text{where } \mu_{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n x_i, \mu_{\mathbf{y}} = \frac{1}{n} \sum_{i=1}^n y_i$$

The covariance matrix Σ_k for class k is defined as the pairwise covariance between each column combination in the matrix G_k , with the dimensions of G_k being $|G_k| \times f$.

$$\Sigma_k(i, j) = \text{cov}(\text{column } i \text{ of } G_k, \text{column } j \text{ of } G_k), \quad \text{for } i, j \in \{0, 1, \dots, f\}$$

You can use the Python `numpy.cov` or the Matlab `cov` function to compute directly the formulas above for all $k = 0, 1, \dots, 255$.

- (c) **Pooled Covariance Matrix.** To improve numerical stability, aggregate all covariance matrices to a single one⁶. Compute the pooled covariance matrix across all 256 classes as follows:

$$\Sigma_{\text{pool}} = \frac{1}{256} \sum_{k=0}^{255} \Sigma_k$$

The 256 tuples $(\mathbf{m}_k, \Sigma_{\text{pool}})$ are the full templates for the 256 classes.

3. **Template Matching.** After reducing the dimension of every trace from 100 to f , the validation set contains groups V_0, V_1, \dots, V_{255} . You will use these groups to compute the total accuracy of the full template classifier, by applying the following steps:

- (a) **Probability Density Function Scoring.** For a single trace \mathbf{t} you could compute the score w.r.t. class k , denoted as score_k , using the multivariate normal probability density function (pdf), as follows:

⁶This improvement was suggested here: <https://eprint.iacr.org/2013/770.pdf>

$$score_k = \frac{1}{\sqrt{(2\pi)^f * det(\Sigma_{pool})}} * exp\left(-\frac{1}{2}(\mathbf{t} - \mathbf{m}_k) * \Sigma_{pool}^{-1} * (\mathbf{t} - \mathbf{m}_k)^\top\right)$$

The $score_k$ is the “likelihood” that the f -dimensional vector \mathbf{t} originates from the multivariate normal distribution $\mathcal{N}(\mathbf{m}_k, \Sigma_{pool})$. One should compute the matching score between the trace \mathbf{t} and all 256 full templates and find the maximum. In Python you can use `scipy.stats.multivariate_normal.pdf` and in Matlab `mvnpdf`.

- (b) **Simplified Score.** Unfortunately, the score formula above will only work for f being small. Large values in the dimension f may result in extremely small score values and may cause numerical errors. We thus simplify $score_k$ to $score'_k$ by applying the natural logarithm function:

$$\log_e(score_k) = \log_e\left(\frac{1}{\sqrt{(2\pi)^f * det(\Sigma_{pool})}}\right) + \left(-\frac{1}{2}(\mathbf{t} - \mathbf{m}_k) * \Sigma_{pool}^{-1} * (\mathbf{t} - \mathbf{m}_k)^\top\right)$$

The first term of the sum is constant across all 256 classes and can be ignored. Thus, we are left with the simplified score $score'_k$ whose formula will not encounter numerical problems for larger values of f .

$$score'_k = -\frac{1}{2}(\mathbf{t} - \mathbf{m}_k) * \Sigma_{pool}^{-1} * (\mathbf{t} - \mathbf{m}_k)^\top$$

We now use k^* to denote the $k \in \{0, 1, \dots, 255\}$ for which the $score'_k$ is maximum. Finding this k^* means that you found the best matching class for trace \mathbf{t} , according to the full template classifier.

$$k^* = \operatorname{argmax}_k (score'_k), \quad \text{with } k \in \{0, 1, \dots, 255\}$$

Using this principle, count the number of hits for the traces in the full validation set i.e. in groups V_0, V_1, \dots, V_{255} . Compute the accuracy on the validation set.

- (c) **Classifier Fine-Tuning.** Notice that your choice of f during the dimensionality reduction step affects the obtained accuracy in the validation set. Repeat the training-validation process with various choices of f in order to find one that maximizes the accuracy in the validation set. This classifier fine-tuning should be done using only the validation set and not using the test set.
- (d) **Classifier Testing.** Following the fine-tuning, use a good choice for the value f and compute the accuracy of the full template classifier in the test set by counting the hits in groups H_0, H_1, \dots, H_{255} .

- (e) **Multi-trace Attack (optional task).** So far the classifier performed a single-trace attack i.e. it tried to classify every trace in the test set independently and reported the achieved accuracy. A multi-trace attack aggregates traces from the same class before making a final decision.

Assume that test traces $\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_{10}$ originate from the same unknown class. The multi-trace attack could average the 10 traces to denoise the signal and then compute the classifier $score'_k$ for $k = 0, 1, \dots, 255$ to decide what is the best matching template.

$$\bar{\mathbf{t}} = \frac{1}{10} \sum_{i=1}^{10} \mathbf{t}_i$$

$$score'_k = -\frac{1}{2}(\bar{\mathbf{t}} - \mathbf{m}_k) * \Sigma_{pool}^{-1} * (\bar{\mathbf{t}} - \mathbf{m}_k)^\top, \quad \text{for } k = 0, 1, \dots, 255$$

Alternatively, the multi-trace attack could compute the $score_k$ product for all 10 test traces that originate from the same unknown class.

$$product_k = score_k(\mathbf{t}_1) * score_k(\mathbf{t}_2) * \dots * score_k(\mathbf{t}_{10})$$

where $score_k(\mathbf{t}_i)$ is the multivariate normal pdf of vector \mathbf{t}_i

Applying the natural logarithm function on $product_k$ and removing constant terms, we get the simplified multi-trace $score''_k$ as follows:

$$score''_k = \log_e(product_k) =$$

$$\log_e(score_k(\mathbf{t}_1)) + \log_e(score_k(\mathbf{t}_2)) + \dots + \log_e(score_k(\mathbf{t}_{10})) \approx \sum_{i=1}^{10} score'_k(\mathbf{t}_i)$$

$$\text{where } score'_k(\mathbf{t}_i) = -\frac{1}{2}(\mathbf{t}_i - \mathbf{m}_k) * \Sigma_{pool}^{-1} * (\mathbf{t}_i - \mathbf{m}_k)^\top$$

Using this principle, perform a multi-trace attack on the test set groups H_0, H_1, \dots, H_{255} , using either form of trace aggregation and compute the total accuracy. Make a plot that depicts the relation between the attack accuracy (y -axis) and number of aggregated attack traces (x -axis).