Side-channel cryptanalysis of a masked AES with SCALib

Gaëtan Cassiers

Download the dataset: https://tinyurl.com/ascad-5k (1.3 GB)
Slides available at https://github.com/simple-crypto/scalib-tutorial

Table of Contents

Content

Attack context

How secure is my implementation?

Worst-case attack:

- Assume all implementation details are known.
- Powerful profiling available (including knowledge of the masks).
- Profile & attack on the same device.

SCALib: A Side-Channel Analysis Library

- Flexible & Simple: a Python library working on numpy arrays.
- High performance: expensive computations are optimized with native code and multi-threaded.
- Scalability thanks to streaming APIs.

See docs at https://scalib.readthedocs.io/.

ASCAD: ANSSI's Side-Channel Analysis Dataset

ASCADv1 variable key:

- Released in 2018.
- 300k traces, 250k points in full traces.
- 8-bit smart card.
- Often used in (deep learning) SCA research.

A table-based masked AES implementation

Input: masked key bytes $(k_0^{(0)}, k_1^{(0)}), \ldots, (k_0^{(15)}, k_1^{(15)})$.
Input: masked plaintext bytes $(p_0^{(0)}, p_1^{(0)}), \ldots, (p_0^{(15)}, p_1^{(15)})$.
Input: randomness bytes r_{in}, r_{out}, masked sbox table $MskSbox$.
// Add round key
for $i = 0, \ldots, 15$ **do**
  $x_0^{(i)} \leftarrow k_0^{(i)} \oplus p_0^{(i)}$
  $x_1^{(i)} \leftarrow k_1^{(i)} \oplus p_1^{(i)}$
  $x_{rin}^{(i)} \leftarrow (x_0^{(i)} \oplus r_{in}) \oplus x_1^{(i)}$
  $x_{rout}^{(i)} \leftarrow MskSbox(x_{rin}^{(i)})$
  $y_0^{(i)} \xleftarrow{\$} \mathbb{F}_{256}$
  $y_1^{(i)} \leftarrow (x_{rout}^{(i)} \oplus y_0^{(i)}) \oplus r_{out}$.
end for
// ...

A simple attack on ASCAD

Profile all intermediate variables in the computation, recombine these to get the key.

A simple attack on ASCAD

Profile all intermediate variables in the computation, recombine these to get the key.

1. Selection of Points Of Interest (POIs) for each intermediate
   - To reduce leakage's dimensionality.
   - Select k points with highest Signal-to-Noise Ratio (SNR).

A simple attack on ASCAD

Profile all intermediate variables in the computation, recombine these to get the key.

1. Selection of Points Of Interest (POIs) for each intermediate
   - To reduce leakage's dimensionality.
   - Select k points with highest Signal-to-Noise Ratio (SNR).
2. Build templates with Linear Discriminant Analysis (LDA)
   - Gaussian Templates with pooled covariance matrix & dimensionality reduction.
   - Outputs probability distribution (over $\mathbb{F}_{256}$) for each intermediate.

A simple attack on ASCAD

Profile all intermediate variables in the computation, recombine these to get the key.

1. Selection of Points Of Interest (POIs) for each intermediate
   - To reduce leakage's dimensionality.
   - Select k points with highest Signal-to-Noise Ratio (SNR).
2. Build templates with Linear Discriminant Analysis (LDA)
   - Gaussian Templates with pooled covariance matrix & dimensionality reduction.
   - Outputs probability distribution (over $\mathbb{F}_{256}$) for each intermediate.
3. Recombine intermediates with Soft-Analytical Side-Channel Attack (SASCA)
   - Bayesian inference: find key distribution given intermediate's distributions.
   - (Approximate) solution: (loopy) belief propagation.

A simple attack on ASCAD

Profile all intermediate variables in the computation, recombine these to get the key.

1. Selection of Points Of Interest (POIs) for each intermediate
   - To reduce leakage's dimensionality.
   - Select k points with highest Signal-to-Noise Ratio (SNR).
2. Build templates with Linear Discriminant Analysis (LDA)
   - Gaussian Templates with pooled covariance matrix & dimensionality reduction.
   - Outputs probability distribution (over $\mathbb{F}_{256}$) for each intermediate.
3. Recombine intermediates with Soft-Analytical Side-Channel Attack (SASCA)
   - Bayesian inference: find key distribution given intermediate's distributions.
   - (Approximate) solution: (loopy) belief propagation.
4. Evaluate the attack: find the key rank

A simple attack on ASCAD

Profile all intermediate variables in the computation, recombine these to get the key.

1. Selection of Points Of Interest (POIs) for each intermediate
   - To reduce leakage's dimensionality.
   - Select k points with highest Signal-to-Noise Ratio (SNR).
2. Build templates with Linear Discriminant Analysis (LDA)
   - Gaussian Templates with pooled covariance matrix & dimensionality reduction.
   - Outputs probability distribution (over $\mathbb{F}_{256}$) for each intermediate.
3. Recombine intermediates with Soft-Analytical Side-Channel Attack (SASCA)
   - Bayesian inference: find key distribution given intermediate's distributions.
   - (Approximate) solution: (loopy) belief propagation.
4. Evaluate the attack: find the key rank
   - Also known as Guessing Entropy (GE).

A simple attack on ASCAD

Profile all intermediate variables in the computation, recombine these to get the key.

1. Selection of Points Of Interest (POIs) for each intermediate
   - To reduce leakage's dimensionality.
   - Select k points with highest Signal-to-Noise Ratio (SNR).
2. Build templates with Linear Discriminant Analysis (LDA)
   - Gaussian Templates with pooled covariance matrix & dimensionality reduction.
   - Outputs probability distribution (over $\mathbb{F}_{256}$) for each intermediate.
3. Recombine intermediates with Soft-Analytical Side-Channel Attack (SASCA)
   - Bayesian inference: find key distribution given intermediate's distributions.
   - (Approximate) solution: (loopy) belief propagation.
4. Evaluate the attack: find the key rank
   - Also known as Guessing Entropy (GE).
5. (Bonus) Leakage assessment with uni- and bi-variate T-test

A simple attack on ASCAD

Profile all intermediate variables in the computation, recombine these to get the key.

1. Selection of Points Of Interest (POIs) for each intermediate
   - To reduce leakage's dimensionality.
   - Select k points with highest Signal-to-Noise Ratio (SNR).
2. Build templates with Linear Discriminant Analysis (LDA)
   - Gaussian Templates with pooled covariance matrix & dimensionality reduction.
   - Outputs probability distribution (over $\mathbb{F}_{256}$) for each intermediate.
3. Recombine intermediates with Soft-Analytical Side-Channel Attack (SASCA)
   - Bayesian inference: find key distribution given intermediate's distributions.
   - (Approximate) solution: (loopy) belief propagation.
4. Evaluate the attack: find the key rank
   - Also known as Guessing Entropy (GE).
5. (Bonus) Leakage assessment with uni- and bi-variate T-test

Attack introduced in https://eprint.iacr.org/2021/817

Content

Setup

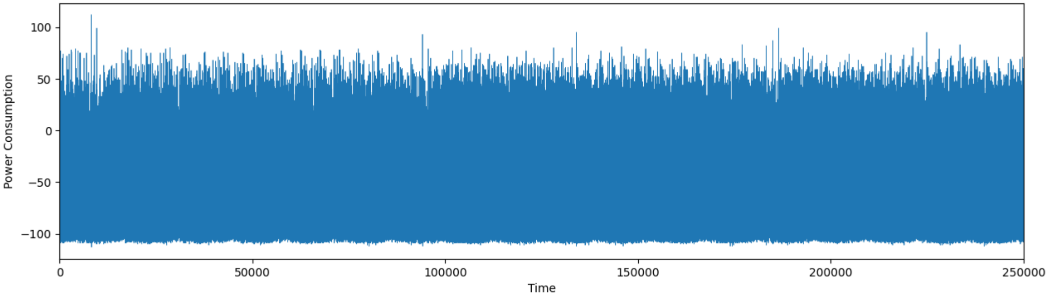
Follow instructions in the notebook:

`https://github.com/simple-crypto/scalib-tutorial/blob/main/Tutorial.ipynb`

You may run the code on your laptop (recommended) or use a Google Colab notebook.

Step 0: Plot a trace

Result:



Content

Step 1: SNR + POI

For every variable v among $r_{in}, r_{out}, x_0^{(i)}, x_1^{(i)}, y_0^{(i)}, y_1^{(i)}$,

1. Compute the SNR for each point k in the trace:

$$\mathrm{SNR}_{v,k} = \frac{\mathbb{V}_v\left(\mathbb{E}_{l_k}(l_k)\right)}{\mathbb{E}_v\left(\mathbb{V}_{l_k}(l_k)\right)}$$

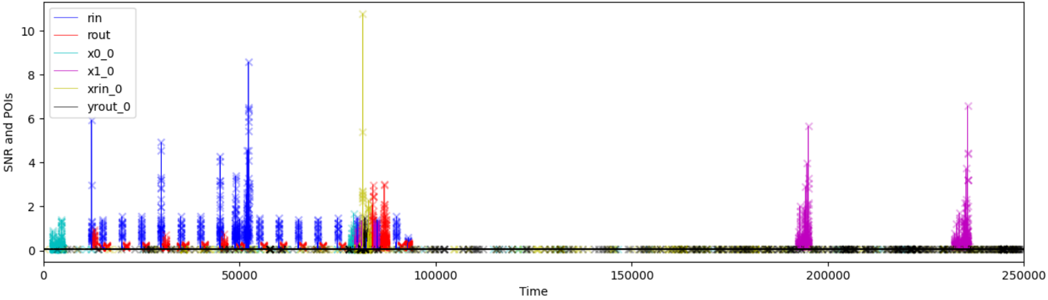
2. Select the $N_{\mathrm{POIs}} = 512$ points with higest SNR for each variable:

$$\mathrm{POIs}_v = \arg\max_k \mathrm{SNR}_{v,k}$$

Useful classes/functions: scalib.metrics.SNR, numpy.argsort

Trick: `right clik > New Console for Notebook`

Step 1: result



Step 1: Solution

Compute the SNR with:

```
for v in tqdm(variables, desc="SNR Variables"):
    snr = SNR(nc=256)
    x = labels[v].reshape((settings.profile, 1))
    snr.fit_u(traces, x)
    snrs[v] = snr.get_snr()[0, :]
```

Recover the POIs with:

```
for v, snr in snrs.items():
    poi = np.argsort(snr)[-settings.poi :].astype(np.uint32)
```

Step 2: LDA

For every variable v:

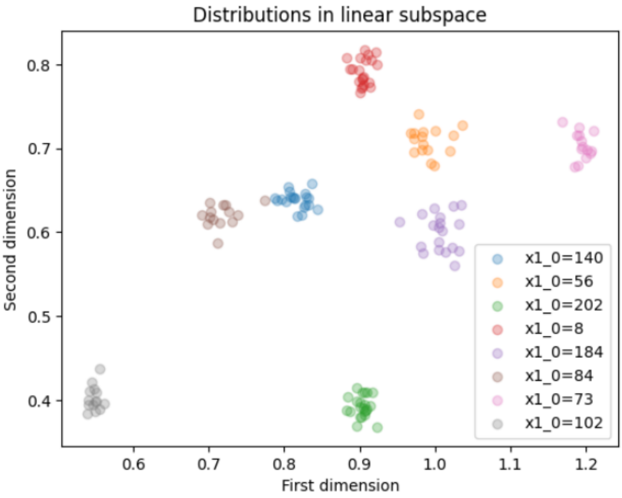
1. compute a linear discriminant analysis model based on the POIs,
2. make a scatter plot of the 2D-projected features,
3. predict a probability distribution of the variable for a new trace.

The LDA model is both

- a linear dimensionality reduction tool (that maximizes SNR in projected dimensions),
- a Bayesian classification tool (a.k.a. pooled Gaussian templates).

Useful classes/functions: scalib.modeling.LDAClassifier

Step 2: result



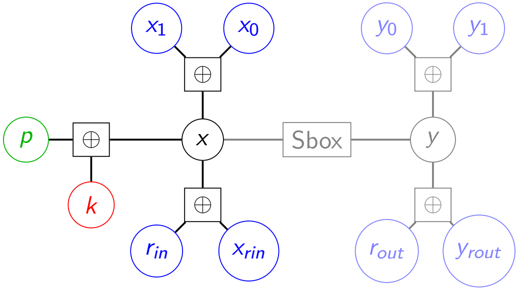
Step 2: Solution

Compute the LDA with:

```
for v in tqdm(variables, desc="LDA Variables"):
    lda = scalib.modeling.LDAClassifier(nc=256, p=settings.dim)
    lda.fit_u(traces[:, pois[v]], labels[v])
    lda.solve()
```

Step 3: SASCA

1. Implement factor graph
2. Add evidence from LDA
3. Run belief propagation
   - Remark: no cycle!



Useful classes/functions: scalib.attacks.FactorGraph.

Step 3: result

```
Correct Key: 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee 0xff
Guessed Key: 0x22 0x33 0xd4 0x95 0x66 0xc9 0xcc 0xdd 0xee 0xfc 0xcc 0xcd 0x8e 0x3f
```

Step 3: solution

For each of the bytes, run:

```
# Init the sasca with the factor graph for a single trace
graph = scalib.attacks.FactorGraph(sasca_graph, {"sbox": SBOX})
# Set the labels for the plaintext byte
bp = scalib.attacks.BPState(graph, 1, {"p": labels[f"p_{i}"].astype(np.uint32)})
# Set the initial distri. for target var. 'vs' if it is in the graph
for v in target_variables(i):
    vs = v.split('_')[0]
    if vs in graph.vars():
        # Assign the distribution of vs
        prs = ldas[v].predict_proba(traces[:, pois[v]])
        bp.set_evidence(vs, prs)
# Run 3 iterations of belief propagation
bp.bp_loopy(it=3, initialize_states=True)
byte_distribution = bp.get_distribution(f"k")
```

Step 4: Key rank estimation

Key rank (a.k.a. Guessing Entropy):

$$r = \left|\left\{k \in \mathcal{K} \mid \tilde{\Pr}[K = k] \geq \tilde{\Pr}[K = k*]\right\}\right|$$

where $k*$ is the correct key.

Easy to compute on single byte, harder for large keys.

1. Compute the rank for each of the key bytes.
2. Compute an estimation of the rank of the full key.

Useful classes/functions: numpy.count_nonzero, scalib.postprocessing.rankestimation.

Step 4: result

```
Evaluating the attack: 100%|██████████████████████████████████████████████████| 100/100 [00:16<00:00,  6.25it/s]
Success rate (rank 1): 1%
Success rate (rank 2**32): 95%
```

Step 4: solution

```python
def eval_rank(secret_key, key_distribution):
    """Returns the rank of the true key"""

    # Floor the key_distribution to avoid numerical issues
    key_distribution[np.where(key_distribution < 1e-100)] = 1e-100

    # Compute the rank of the
    log2distribution = np.log2(key_distribution)
    rmin, r, rmax = scalib.postprocessing.rank_accuracy(
        -log2distribution, secret_key, max_nb_bin=2**20
    )

    return r
```

Step 5: Improved SASCA

Improve the attack by using the Sbox output (see right-hand part of the factor graph in Step 3).

Step 5: result

```
Evaluating the attack: 100%|████████████████████████████████████████████████████████████████████████████████████████████████████| 100/100 [00:01<00:00, 52.05it/s]
Success rate (rank 1): 95%
Success rate (rank 2**32): 100%
```

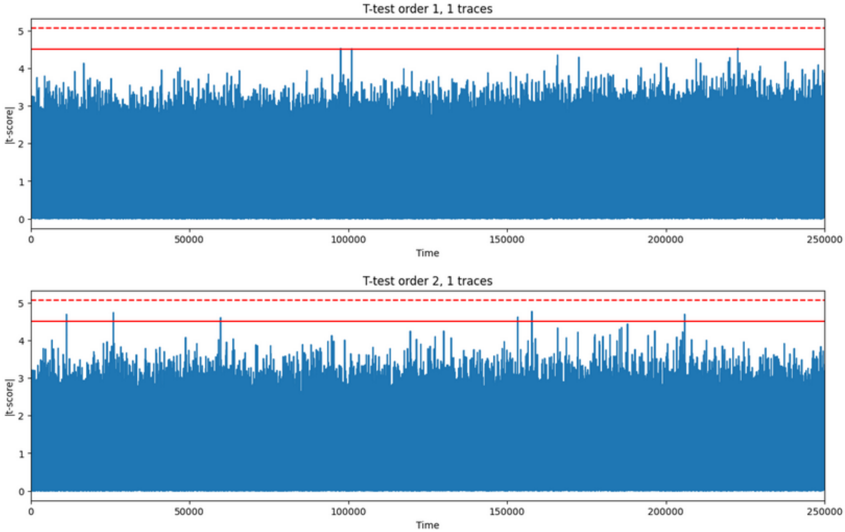
Step 5: solution

```
SASCA_GRAPH_IMPROVED = """
    [... previous graph ...]

    VAR MULTI y
    VAR MULTI y0
    VAR MULTI y1
    VAR MULTI yrout
    VAR MULTI rout

    PROPERTY y = sbox[x]
    PROPERTY y = y0 ^ y1
    PROPERTY y = yrout ^ rout
"""
```

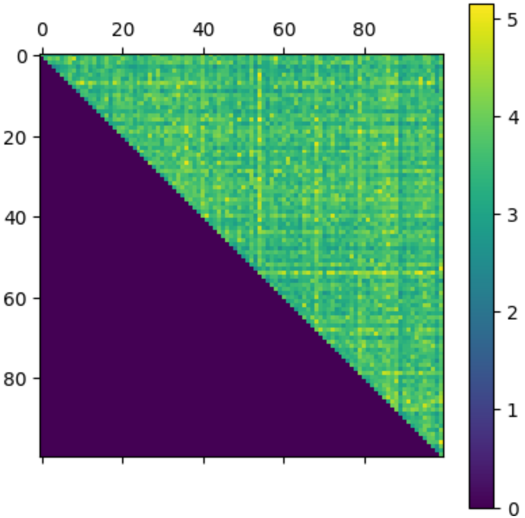
Step 6: (Bonus) Leakage assessment

We run univariate (first and second-order) T-tests as well as a bivariate T-test.
As classification bit, we cannot use the typical "fix-vs-random" with this dataset, we use a S-box input bit.

Step 6: univariate result



Step 6: bivariate result



Step 6: solution

Univariate

```
ttest = Ttest(d=2)
ttest.fit_u(traces, x)
t = ttest.get_ttest()
```

Bivariate

```
# Boolean matrix corresponding to the POI positions.
template = np.fromfunction(lambda i, j: i <= j, (n, n))
# Build POIS array from template
pois_idxs = np.nonzero(template)
pois_mat = np.vstack(pois_idxs).astype(np.uint32)
# Compute T-test
mttest = MTtest(d=2, pois=pois_mat)
mttest.fit_u(traces, x)
t = mttest.get_ttest()
```

Content

A versatile attack

- High robustness for hyperparameters (see https://eprint.iacr.org/2021/817).
- Similar to one of the winning attacks of the CHES 2023 challenge.
- Similar flow works against post-quantum crypto https://eprint.iacr.org/2023/1545.

Other SCALib features

- RLDA: a LDA that works with many classes (e.g. 16-bit or 32-bit states).
- Perceived information/Training information: evaluate quality of models.

See `https://scalib.readthedocs.io/en/stable/`
Feature requests: `https://github.com/simple-crypto/SCALib/issues/new`