IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 1, pp. 345–366. DOI:10.46586/tches.v2022.i1.345-366

Semi-Automatic Locating of Cryptographic Operations in Side-Channel Traces

Jens Trautmann[1], Arthur Beckers[2], Lennert Wouters[2], Benedikt Gierlichs[2], Stefan Wildermann[1], Ingrid Verbauwhede[2] and Jürgen Teich[1]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstraße 11, 91058 Erlangen, Germany, firstname.lastname@fau.de

[2] imec-COSIC, KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium, firstname.lastname@esat.kuleuven.be

Abstract. Locating a cryptographic operation in a side-channel trace, i.e. finding out where it is in the time domain, without having a template, can be a tedious task even for unprotected implementations. The sheer amount of data can be overwhelming. In a simple call to OpenSSL for AES-128 ECB encryption of a single data block, only 0.00028% of the trace relate to the actual AES-128 encryption. The rest is overhead. We introduce the (to our best knowledge) first method to locate a cryptographic operation in a side-channel trace in a largely automated fashion. The method exploits meta information about the cryptographic operation and requires an estimate of its implementation's execution time.

The method lends itself to parallelization and our implementation in a tool greatly benefits from GPU acceleration. The tool can be used offline for trace segmentation and for generating a template which can then be used online in real-time waveform-matching based triggering systems for trace acquisition or fault injection. We evaluate it in six scenarios involving hardware and software implementations of different cryptographic operations executed on diverse platforms. Two of these scenarios cover realistic protocol level use-cases and demonstrate the real-world applicability of our tool in scenarios where classical leakage-detection techniques would not work. The results highlight the usefulness of the tool because it reliably and efficiently automates the task and therefore frees up time of the analyst.

The method does not work on traces of implementations protected by effective time randomization countermeasures, e.g. random delays and unstable clock frequency, but is not affected by masking, shuffling and similar countermeasures.

Keywords: Side-channel analysis, locating of cryptographic operations

1 Introduction and related work

Side-channel attacks are well-known methods to extract secret information from embedded cryptographic devices. They are based on the dependency between the data being processed and a physical observable (instantaneous power consumption, electromagnetic radiation, etc.) of the device. Techniques such as DPA [KJJ99], CPA [BCO04] or MIA [GBTP08] exploit these side-channels to extract secret data protected via Cryptographic Operations (COs) from such devices. However, all these techniques have in common that they require measurement data from a very large number of executions of the same CO with different inputs. The evaluator needs to locate all the targeted COs within the side-channel trace(s) and carefully align them.

When evaluating an implementation against side-channel attacks in a lab setting, it is often straightforward to find the targeted CO segments within a side-channel trace as

Received: 2021-07-15 Accepted: 2021-09-15 Published: 2021-11-19

one has full control over the device. Trigger signals (e.g. generated by GPIO pins) can be used to indicate the exact locations of the target operations. These trigger signals greatly simplify the processing of the collected side-channel trace(s), in particular the segmentation of the trace(s) and the alignment of the COs, which ultimately also speeds up the evaluation. In a real-world setting, however, the attacker often has to locate the targeted COs in the trace(s) among other operations without having a clear view on where the targeted COs are situated.

In certain scenarios it may be possible to roughly determine the location of the COs by relying on logic events in e.g. communication lines. In an ideal case the evaluator will end up with a set of segmented traces which however still have to be aligned. Trace alignment can then be achieved using, for example, peak based alignment, SAD alignment or Dynamic Time Warping (DTW) [vWWB11]. However, in less ideal and more realistic scenarios it may not be possible to acquire traces in a segmented form.

An example for such a scenario would be an attack on a secure boot process. An embedded device implementing secure boot reads the encrypted and signed firmware from non-volatile storage when booting. Only after a successful signature verification the device decrypts the (many blocks large) firmware and executes it. When taking a side-channel measurement of this boot process with the goal of attacking the decryption, the side-channel trace will encompass a signature verification and many block decryptions (our target COs). The location of each CO within the trace is unknown.

To segment a trace in such a scenario one could rely on a waveform-matching based triggering systems such as the one proposed by Beckers et al. [BBGV16]. Commercial tools, e.g. the Chipwhisperer Pro from NewAE [Newa] and icWaves from Riscure [Ris], with similar capabilities exist and allow to generate trigger signals in real-time by matching a given template of the CO with the side-channel trace. However, all of these tools assume that a template of the CO is *already* available, i.e., they require that the pattern of a CO on a side-channel measurement is known.

Unfortunately, classical leakage assessment techniques are not a silver bullet when it comes to locating COs in unsegmented traces, particularly in real-world scenarios such as the aforementioned secure boot example. Many of these techniques (e.g. TVLA [GGJR+11], SNR and known-key correlation [KJJR11]) require some prior knowledge or control over the input, and cannot always be used to locate the COs. Similarly, a differential or correlation analysis based on known inputs and outputs (as described in [KJJR11]) to locate the COs may not be applicable. During a secure boot process the evaluator can only capture one long side-channel trace that contains multiple COs, this trace is unsegmented. In this scenario control over, or manipulation of the input data will be detected in the initial signature verification step, aborting the boot process. Similarly, to apply a known input analysis the evaluator would require multiple inputs with valid signature, an unlikely scenario that is further hindered by the deployment of anti-rollback countermeasures.

This paper closes the gap by providing a semi-automated tool that can segment a full side-channel trace into individual sub-traces per CO. Besides locating the COs, our semi-automated tool is also able to output a template of the CO that can be used in waveform-matching based triggering systems. The basic idea of the proposed approach is to exploit an inherent feature of many cryptographic primitives: The use of a round function that is iterated for a specified number of times. Our approach therefore searches for repeating patterns with a matching number of rounds in a side-channel trace.

Our contribution.

- We explore this alternative approach and present a largely automated method for locating particular COs in a side-channel trace.
- We implement our method in a tool that makes heavy use of parallel computation and GPU acceleration.

- We evaluate our tool in various real-world scenarios using hardware and software implementations of different COs executed on diverse platforms.
- We highlight strengths and weaknesses of the method.
- We published the tool under an open-source license[1] with walk-through Python Notebooks to test the tool.

The tool does not work on traces of implementations protected by effective time randomization countermeasures, e.g. random delays and unstable clock frequency, but is not affected by masking, shuffling and similar countermeasures.

2 Method

From a high-level perspective our proposed method consists of the following steps:

1. Identify the most probable location of one CO.
2. Generate a CO template.
3. Use the CO template to find all trace segments which correspond to other occurrences of the same CO.
4. Use those trace segments to refine the CO template (optional).

2.1 Requirements

Before we go into more detail we first discuss the requirements of our method, as these ultimately determine in which scenarios the tool can be applied. We assume we have a side-channel trace $\boldsymbol{t}$ that was obtained with known sampling rate f_{sample}. We further assume that the trace includes a known number $n \geq 1$ of COs. We need to know the number r of *successive* and *identical* rounds that are part of the CO, e.g. typically $r = 16$ for DES, $r = 9$ for AES-128, etc. Additionally, an estimate for the number of clock cycles in one round (m) helps to bound the computational complexity. Further, we need to know the target device clock frequency f_{device}, or we need to be able to estimate it, e.g. based on a Fast Fourier transform (FFT) of the side-channel trace $\boldsymbol{t}$. With this information the width w of one round pattern (in numbers of samples) can be calculated or estimated as

$$w = \frac{f_{\text{sample}}}{f_{\text{device}}} \cdot m\,. \tag{1}$$

2.2 Locating COs if the round pattern length w is known

To simplify the explanation we first assume that the round pattern length w is known. Later, in Section 2.3, we will relax this assumption. Algorithm 1 illustrates the procedure of locating COs for a known round pattern length w. The first step is to identify the most probable location of one CO in the trace. The basic idea is to analyse *all* possible trace segments of length $r \cdot w$. As we are looking for r identical rounds, we select the segment that is comprised of r most similar patterns. In more detail, this exhaustive search works by looping over the entire trace $\boldsymbol{t}$ (Alg. 1, line 1). At each iteration i a trace segment of length $r \cdot w$ samples is analysed. The loop index i is taken as the segment's starting point. One segment can be cut into r pieces of length w. These pieces will be used to assign a *similarity score* $q_{i,w}$ to this segment and thus to this sample i. One way to compute this similarity score is to first average the r pieces in the segment, resulting in a candidate

[1] https://github.com/FAU-LS12-RC/Finding-COs-on-Side-Channel-Traces

template $\boldsymbol{a}_{i,w}$ of one single round (Alg. 1, line 5). Afterwards, the average similarity (quantified by, for example, Sum of Absolute Differences (SAD) or Pearson correlation) between this candidate round template and the r pieces is used as the similarity score $q_{i,w}$ for this segment (Alg. 1, line 6). Finally, the index $\hat{i} = \arg\max_i q_{i,w}$ with maximal score is considered as the most likely starting point of a CO that consists of r identical rounds of width w. We further denote the most likely candidate round template, i.e. the average of the r most similar patterns, as the *round template* $\boldsymbol{a}_{\hat{i},w}$ (Alg. 1, line 8).

The second step is to generate a template for the pattern of the CO that begins at index $\hat{i}$ in the trace. This can be achieved by selecting the original trace segment of length $r \cdot w$ starting from index $\hat{i}$, or by concatenating r copies of the round template. We call this the *CO template* $\boldsymbol{c}_w$ (Alg. 1, line 9).

The third step is to find all segments within the trace that correspond to the remaining COs. To achieve this, we compute a similarity vector $\boldsymbol{p}_w$ between part of the power trace at each index i and the obtained CO template. We expect to find n COs and therefore select the n indices yielding the highest similarity scores that are at least $r \cdot w$ samples apart from each other (Alg. 1, line 11). Using this approach, we thus obtain the starting points of the n most probable CO patterns in the side-channel trace.

2.3 Locating COs if the round pattern length w is unknown

In this section we relax the assumption of knowing w by testing for multiple candidate widths. In practice it is necessary that the analyst is able to roughly estimate lower and upper bound on the width w in order to keep the number of candidate widths at a feasible level. Later, in Section 5 we will discuss the impact of the number of candidate widths on the method's computational complexity and show concrete timings from our tool.

In this scenario we have to determine the correct round width w within a set of candidate widths $\boldsymbol{w}$. By iterating over all $w \in \boldsymbol{w}$, we first calculate the most likely starting point of a CO as in Section 2.2 for each candidate width. However, we then cannot simply select width w for which the round template achieves the highest similarity score $q_{i,w}$. That is because we may have tested a candidate width for which the similarity score is accidentally greater than the similarity score for the correct width. This can for example happen if the candidate width is the length of a single clock cycle. In this case the similarity of different clock cycles when the device is idle is overwhelming.

Instead, we also make use of our knowledge of the expected number n of COs in the trace and the structure of one CO. The idea is as follows. For each candidate width $w \in \boldsymbol{w}$, we generate a specific CO template and use that to find all other COs in the trace $\boldsymbol{t}$. If we find exactly the expected number n of COs, the respective w is the correct width. This approach yields the correct result with a high probability as we accept a

Algorithm 1 Simplified pseudocode for locating COs in a trace $\boldsymbol{t}$ with known w

1: **for all** i in $\boldsymbol{t}$ **do** ▷ Step 1
2: **for all** v in [0,w) $\{v \in \mathbb{N}_0\}$ **do**
3: $a_{v,i,w} \leftarrow \frac{1}{r} \cdot \sum_{z=0}^{r-1} t_{i+v+z \cdot w}$ ▷ create candidate round template
4: **end for**
5: $\boldsymbol{a}_{i,w} \leftarrow [a_{v=0,i,w}, \ldots, a_{v=w-1,i,w}]$
6: $q_{i,w} \leftarrow$ calculateSimilarityScoreStep1($\boldsymbol{t}$, $\boldsymbol{a}_{i,w}$)
7: **end for**
8: $\boldsymbol{a}_{\hat{i},w}$ with $\hat{i} = \arg\max_i q_{i,w}$
9: $\boldsymbol{c}_w \leftarrow$ createCOTemplate($\boldsymbol{t}, w, \hat{i}, a_{\hat{i},w}$) ▷ Step 2
10: $\boldsymbol{p}_w \leftarrow$ calculateSimilarityVectorStep3($\boldsymbol{t}$, $\boldsymbol{c}_w$) ▷ Step 3
11: COStartingPoints $\leftarrow$ getNPeaks($\boldsymbol{p}_w, n$)

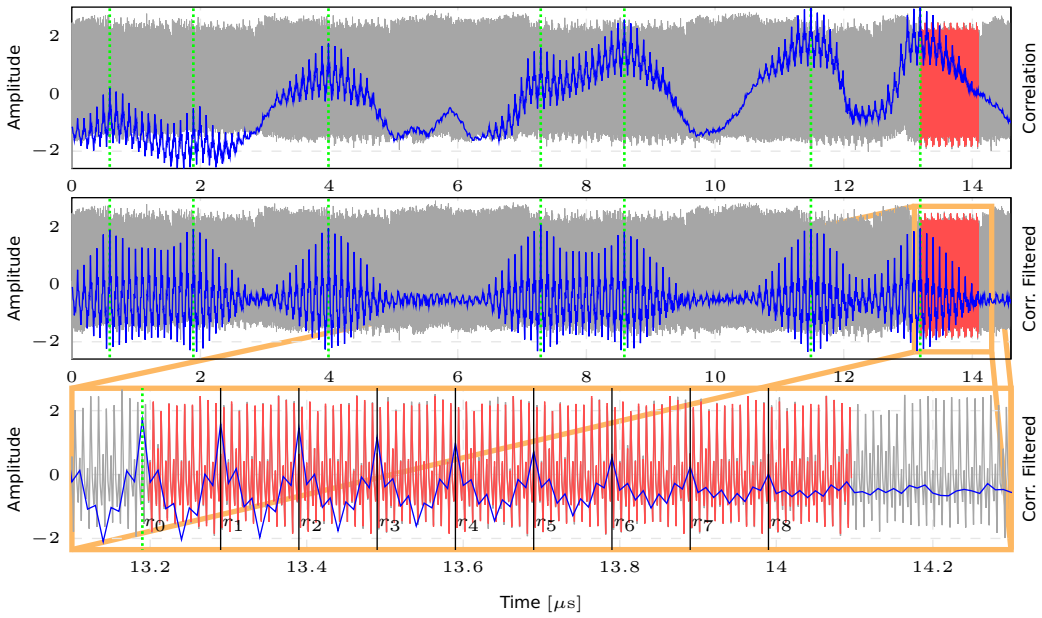

Figure 1: Top: EM trace (grey) with 7 COs; the CO template (red); overlayed correlation between EM trace and CO template (blue). Middle: as top but the correlation was filtered with a moving average filter. The 7 highest peaks in the correlation are marked by green dotted lines and correspond to the starting points of the COs in the EM trace. Bottom: a zoomed window of the middle plot.

possible CO template only if it is based on r similar rounds and occurs exactly n times in the given trace. The remainder of this section describes the procedure (also illustrated in Algorithm 2) in more detail.

For each candidate width $w \in \boldsymbol{w}$, we first generate a CO template $\boldsymbol{c}_w$ as in Section 2.2. This procedure can be seen in Algorithm 1 lines 1-9 and is substituted in Algorithm 2 line 2 by getPotentialCOTemplate. Afterwards, we calculate a similarity score $p_{i,w}$, between $\boldsymbol{c}_w$ and segments of size $r \cdot w$ of the side-channel trace $\boldsymbol{t}$ at each index i, resulting in a similarity vector p_w for each width w (Alg. 2, line 3). For example, Figure 1 (top) shows an EM trace $\boldsymbol{t}$ (gray), containing $n = 7$ COs, and the CO template $\boldsymbol{c}_w$ found within it (red). The similarity scores $p_{i,w}$ between the CO template and the power trace are depicted in blue.

From Figure 1, it can be observed that this similarity score can suffer from undesirable temporal effects (e.g. low-frequency voltage drift or clock jitter), such that a smaller

Algorithm 2 Simplified pseudocode for locating COs in a trace $\boldsymbol{t}$ with unknown w

```
1: for all w ∈ w do
2:     c_w ← getPotentialCOTemplate(t, w, r)                ▷ Step 1 & 2
3:     p_w ← calculateSimilarityVectorStep3(t, c_w)         ▷ Step 3
4:     p_w ← applyMovingAverageFilter(p_w, w)
5:     possibleStartPoints ← findPossibleStartPoints(p_w, c_w, t, minSubPeaks)
6:     if #possibleStartPoints == n then
7:         return (w, possibleStartPoints)
8:     end if
9: end for
10: return false
```

similarity score is assigned to a CO further away from the starting point of the CO template $\boldsymbol{c}_w$. To reduce the impact of these temporal effects, a moving average filter (with a window-size of length w) is applied to the similarity vector (Alg. 2, line 4). The middle plot in Figure 1 shows the resulting filtered similarity score in blue.

Afterwards, the filtered similarity vector is analysed for patterns that are a direct result of the round-based nature of the CO. Our technique is based on two observations. The first observation is that if we calculate the similarity score between the CO template $\boldsymbol{c}_w$ and a trace segment of size $r \cdot w$ at index i that covers exactly one CO, there will be a significant peak in the similarity vector $p_{i,w}$, which we denote by *main peak*.

The second observation is that if we now shift the CO template, the similarity will drop until we shifted it by exactly one round width w. Then, the template will cover $r-1$ rounds of the CO which produces a further peak in the similarity vector, only smaller than the *main peak*. This repeats for the next shifts of exactly one round width w. Therefore, besides the *main peak*, there will ideally be $r-1$ *sub-peaks* (or local maxima) present in the similarity vector. In other words, if one or more rounds of the CO template $\boldsymbol{c}_w$ overlap with one or more rounds of another CO a peak will be observable in the similarity vector. The more rounds overlap, the higher the similarity and thus the higher the peak will be. If all rounds overlap the similarity will be at its maximum, resulting in a *main peak*. Subsequent peaks or *sub-peaks* will have a lower similarity score and will be the result of a partial yet round-aligned overlap between the $\boldsymbol{c}_w$ and a CO in the trace. Figure 1 (bottom) shows a pattern consisting of a *main peak* and its subsequent *sub-peaks*. Note that the same pattern of *sub-peaks* is also present in front of the *main peak*.

Following these observations, we mark an index i in $\boldsymbol{t}$ as the start of a CO, if there is a *main peak* at this location followed and/or preceded by $r-1$ *sub-peaks* spaced w samples apart. Using this technique, we end up with a set of indices that are possible starting points of COs (Alg. 2, line 5). If we found exactly n locations, we terminate (Alg. 2, line 6). Else, this procedure is repeated for the remaining widths $w \in \boldsymbol{w}$ and terminates when exactly n COs are detected with $r-1$ *sub-peaks*.

However, in a practical or noisy scenario, we may not always detect exactly $r-1$ *sub-peaks* per CO. In such a scenario, we lower the required minimum number of sub-peaks in an iterative fashion (see minSubPeaks in Alg. 2, line 5). For example, if we are unable to detect n *main peaks* with their corresponding $r-1$ *sub-peaks* for any candidate width, we restart and try to find n *main peaks* with at least $r-2$ *sub-peaks*, and so on. The algorithm completes when the most likely candidate width is determined which results in n detected COs with at most $r-1$ *sub-peaks* each. The algorithm returns this width w producing the best match as well as the starting indices of the identified COs (Alg. 2, line 7).

2.4 Refinement of the Template

The starting points of COs identified by the method described above are not always perfectly accurate. Discrepancies can be caused, e.g. by noise present in the CO template, or because of the first CO in a batch of COs being slightly different. Such effects can lead to small variances in accuracy.

One way to improve the obtained results is to refine the CO template (step 4), by using multiple detected COs from step 3. To obtain a refined template we first align multiple detected COs (e.g. using SAD) and then average them. Template refinement can significantly increase the accuracy, as detailed in Section 4. The refinement method can be applied iteratively to further increase accuracy. In Section 4, we use the term *unlimited refinement* which corresponds to iteratively applying the refinement method until the detected CO locations do not change anymore.

3 In practice

In this section, we describe our implementation of the method (the tool), walk the reader through an exemplary trace analysis visualizing the intermediate results of the main steps and discuss a number of details that should be considered when using the tool in practice.

3.1 Our implementation / the tool

The implementation is written in Python 3 and mainly uses the NumPy and SciPy libraries. Acceleration of several computationally intensive parts is achieved using Numba and OpenCL. Our method requires the computation of a similarity score for every index i in the trace $\boldsymbol{t}$. As there are no dependencies between these computations, this is an embarrassingly parallel task that greatly benefits from the parallel execution capabilities of a GPU. We use OpenCL to distribute the computations over all GPU cores, resulting in a significant speedup as can be seen in Section 5.

Our experiments and benchmarks are performed on a consumer-grade desktop PC with a Ryzen 3600 CPU clocked at 3.6 GHz and 32 GB of DDR4-3200 RAM. Furthermore, the PC is equipped with a NVIDIA GeForce RTX 3070 GPU with 5,888 CUDA cores and 8 GB of GDDR6 RAM. In 2021 the cost of the setup is approximately 1 000 EUR.

3.2 An exemplary step-by-step walk-through

In Section 2, we provided a textual description of how the proposed method works. In contrast, this section will serve as a walk-through of the described method on a real power trace. For this particular example we use a power trace acquired from an STM32F415 microcontroller running a software AES-128 implementation ($r = 9$). More details about the measurement setup are available in Section 4.1.

As per the requirements of our method (Section 2.1), we know that the trace $\boldsymbol{t}$, consisting of roughly 10M samples, was sampled at a rate f_{sample} of 24 MS/s. The microcontroller was operating at a f_{device} of 24 MHz. A synchronous measurement setup linking the device and oscilloscope clock was used. We thus use, in this example, a trace containing one sample per clock cycle. Additionally, the trace contains a known number $n = 50$ of hidden COs in which one round is executed in 3 035 cycles resulting in a width w of 3 035 samples. Figure 2 shows the full trace (top) and the locations of the hidden COs. Note that the indicated starting locations of the hidden COs are merely used as a visual aid and are not provided as an input to the tool.

As per the method described in Section 2.2, our first step is to find the most likely starting point for one CO. To that end we analyse all trace segments of size $r \cdot w$, or in this example all trace segments with a length of $9 \cdot 3\,035 = 27\,315$ samples. The second row in Figure 2 shows two such trace segments. The left trace segment perfectly covers an AES operation, whereas the segment on the right was created at a random offset in the trace.

These trace segments are next segmented into r pieces of length w (indicated by the vertical lines in the plot). By averaging these pieces, we obtain a candidate round template. The third row of plots in Figure 2 shows for both segments an overlay of the $r = 9$ pieces as well as the candidate round template as their average in blue. Afterwards, we use this candidate round template and compute the correlation coefficient between it and each of the $r = 9$ pieces. Their average is assigned as the similarity score $q_{i,w}$ for the respective segment. By doing this at each index i in the trace, we obtain the similarity vector (step 1) that is shown in the penultimate row of Figure 2. The blue triangles indicate the similarity score obtained from the two example segments. Put differently, a segment starting at index i in the power trace will generate a value in the similarity vector at index i. The larger the similarity value, the more similar each of the r pieces in this segment are. Therefore,

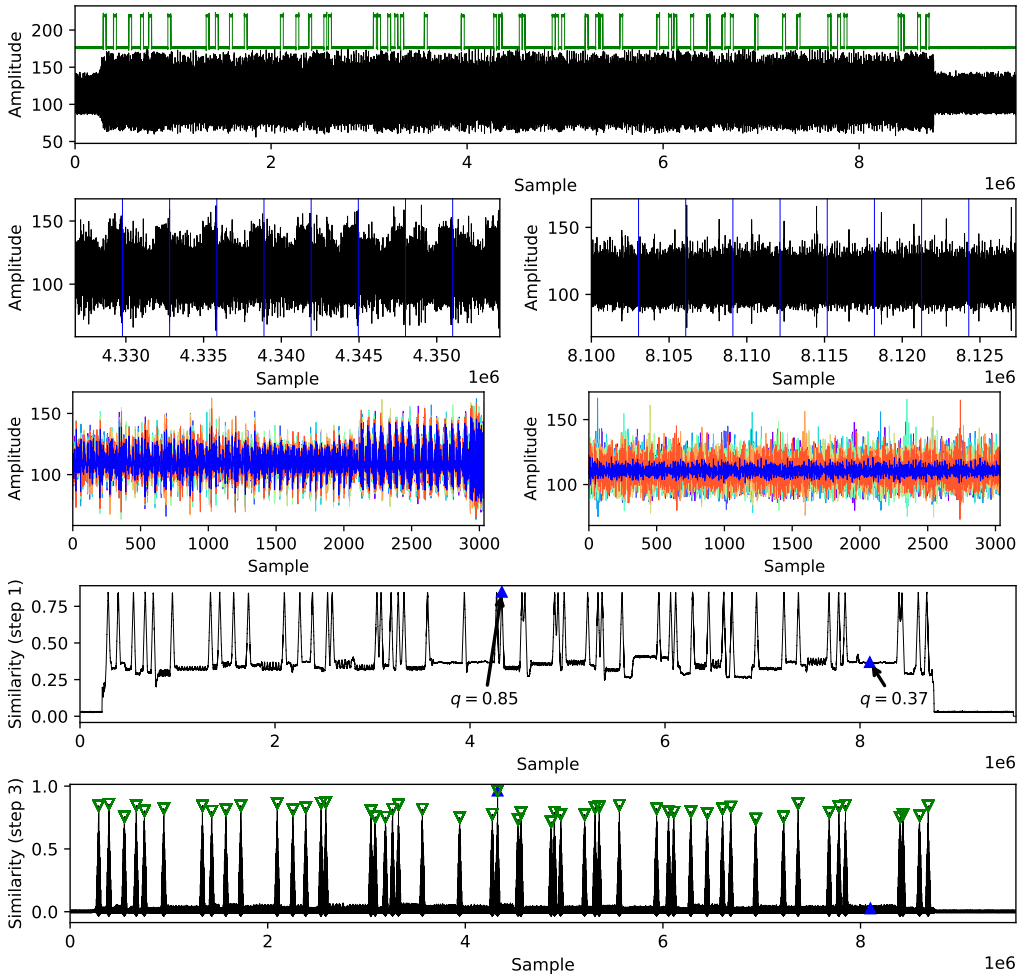

Figure 2: A power trace containing 50 hidden AES COs (top). The second row shows two trace segments of length $r \cdot w$. The segment on the left coincides with an AES operation whereas the segment on the right is taken at a random offset in the trace. The third row of plots show an overlay of the r pieces that make up each segment and the average of those pieces (a candidate round template) in blue. The penultimate row contains a plot of the similarity vector $q_{i,w}$ (step 1), with the similarity value resulting from the two example segments indicated by blue triangles. The last row shows the similarity vector (step 3), and the identified CO starting locations are marked with green triangles.

the index where the similarity is maximal corresponds to the most likely starting point for a CO.

Having identified the most likely starting point for a CO, we can now use it to create a CO template $\boldsymbol{c}_w$ (step 2). For this example, we generate the CO template by concatenating the average round template r times, the template itself can be seen in Figure 3 (right).

Afterwards, in step 3, a similarity score between the CO template and the full trace can be calculated using e.g. Pearson correlation. By doing so, we again obtain a similarity vector. A moving average filter is applied to the similarity vector to reduce the impact of temporal effects (see Section 2.3). Finally, we analyse the filtered similarity vector to determine the n most likely starting points of the hidden COs. The starting points can be determined by selecting the n largest values in the similarity vector that are at least $r \cdot w$ samples apart. Figure 2 (bottom) shows the filtered similarity trace. The identified

starting points are indicated by green triangles.

Note that for this basic example the similarity from step 1 would have been sufficient to recover the starting locations for all COs. This is not the case in some of the more advanced examples that are discussed in Section 4.

3.3 Practicalities

The tool can be used in a fully-automated approach, provided that the requirements laid out in Section 2.1 are met. However, as side-channel traces can originate from a virtually unlimited spectrum of targets it can be advantageous to use the tool in a semi-automated approach in which the tool is guided by the user or evaluator. In a semi-automated approach the user can visually inspect certain intermediate results and tweak parameters to guide the tool to a desirable outcome while still reducing manual effort to a minimum.

We now explain a number of details that should be considered when using the tool in practice. Their impact will be evaluated in Section 4.

3.3.1 The number of rounds r

Many cryptographic algorithms are round based, but not all rounds are necessarily identical. AES-128 for instance has ten rounds of which nine are identical. An informed choice of the expected number of identical rounds may speed up the tool and is important for its success rate.

It can actually happen that the number of rounds is unknown. One example is, that the analyst does not know if the trace contains AES-128 or AES-256 operations. In this case one can work with $r = 9$ similar rounds, which should work for both cases, or with $r = 13$ similar rounds which should work only for AES-256.

Another example to mention here is the SHA-2 family which is typically implemented using two types of rounds that appear a different number of times. In SHA-256 we can have for example 16 rounds of type A and 48 rounds of type B, depending on the implementation. We discuss analysis of a SHA-256 trace in Section 4.5.

3.3.2 Building the CO template

There are two options for constructing the CO template. The first option is to concatenate the round template r times. Since the round template is the average of multiple rounds it contains less noise than the original trace, therefore this option leads to a low-noise CO template which will make the further steps in the processing more robust. The second option is to create a CO template by cutting out an entire trace segment of length $r \cdot w$ corresponding to a CO. This has the advantage that, if the clock frequency is not exactly determined, or not stable, or sampling is done asynchronously, small differences between the calculated width w and the true width do not add up. Further, a template constructed in this way is less sensitive to – or more precisely is better able to capture – low frequency variations in the trace from round to round which persist through all the COs. We discuss a concrete example in Section 4.3.

3.3.3 Detecting sub-peaks

Jitter or clock drift can be detrimental to step 3 of our algorithm – the recovery of all COs – if not accounted for. The detection of the sub-peaks assumes them to be evenly spaced by a width w. If f_{device} is unstable, the width of one round can be off by an integer or non-integer offset. Consequently, an acceptable deviation ϵ needs to be introduced into the algorithms. With this deviation, a sub-peak can now be off by some samples and still count as a sub-peak. When evaluating the tool in Section 4, the parameter ϵ is always set

to 2. This is an arbitrary choice, simply motivated by the fact that it worked well for us across a wide range of scenarios.

3.3.4 Integer round length w

We already mentioned above the susceptibility of the tool to a varying round length w within one or multiple COs. We therefore recommend to set the sampling rate as an integer multiple of the device clock frequency f_{device}, because the tool requires w to be an integer. If the sample rate is not an integer multiple of the device clock frequency, artificial jitter is introduced requiring the user to set a larger ϵ. An alternative approach is to resample the trace such that sample rate and clock form integer multiples.

3.3.5 Unexpected number of COs

One of the requirements listed in Section 2.1 is that we need to know the number n of COs in the trace $\boldsymbol{t}$. This seems to be a weak requirement as the number of COs can be trivially derived from input or output data, e.g. the number of plaintext blocks that get encrypted.

In practice, it can however happen that the tool does not find all n COs. The user then needs to decide how to proceed. More interestingly and surprisingly, it can happen that the tool finds *more* than n proper COs in the trace. We encountered this when we worked with the mbedTLS and OpenSSL cryptographic libraries and report more details in Section 4.4 and 4.6. In this case, we recommend user interaction: the tool can for example mark all found COs (or more precisely main peaks) in the trace and let the user select which n of them to proceed with.

3.3.6 Trace preprocessing

We do not consider trace preprocessing part of the tool, but clearly preprocessing steps like trace decimation or filtering can improve the tool's speed and accuracy.

4 Evaluation

In order to evaluate the effectiveness of our tool in practice we use traces of different hardware and software implementations of several cryptographic primitives that are executed on diverse target platforms. Furthermore, we evaluate our tool on two scenarios that target executions on the application/protocol layer. The first one is an OpenSSL library call (Section 4.6) and the second a secure boot process (Section 4.7). Both scenarios are characterised by the fact that several COs have to be found in a huge trace, which at the same time also includes a multitude of other primitives and routines that also have a round-like nature in the power trace. Moreover, we also show for the secure boot scenario how our approach enables to ultimately extract the secret key used in the targeted COs via Correlation Power Analysis (CPA).

4.1 Evaluation platforms

STM32F415 Our first evaluation platform is the STMicroelectronics STM32F415 ARM Cortex-M4 microcontroller. This particular family of microcontrollers is widely used as a benchmarking platform for cryptographic implementations (e.g. in the NIST PQC competition [NIS]) and is commonly used to evaluate the side-channel security of software implementations. Furthermore, this microcontroller embeds hardware accelerators for cryptographic primitives such as AES and DES. Concretely, we use the NewAE CW308 UFO board [Newc] in combination with a NewAE STM32F415 UFO target [Newb]. The STM32F415 target is configured to use an 8 MHz external clock from which the internal

CPU-clock of 24 MHz is derived. To acquire the power traces we use a Tektronix DPO7254C oscilloscope sampling at 125 MS/s or at 500 MS/s depending on the case study (note that these are not integer multiples of the 24 MHz operating clock). The power consumption signal is sampled synchronously to minimize clock jitter and clock drift, i.e. the clock to the microcontroller is provided by a signal generator whose reference clock is synchronized with the oscilloscope.

STM32F303 Our second evaluation platform is the STM32F303. This is a different ARM Cortex-M4 microcontroller with maximum clock frequency of 72 MHz. The target microcontroller is measured on a custom target board specifically designed for side-channel measurements. We measure the power consumption over a 1 Ω shunt resistor in GND. No analog filtering is applied before capturing the traces. The sampling rate is therefore set to 1.25 GS/s on the Tektronix DPO70404C oscilloscope to avoid aliasing. The microcontroller is clocked at 48 MHz or 72 MHz depending on the case study (note that the sampling rate is not an integer multiple of either). In this setup we rely on the internal high speed 8 MHz oscillator of the STM32F303 to derive the system clock. The use of the internal oscillator as a reference for the main clock will result in a noticeable amount of clock jitter as these internal oscillators are known to drift.

BeagleBone Black Our third evaluation platform is a BeagleBone Black. The BeagleBone Black is equipped with a Texas Instruments AM335x ARM processor [Bea]. It runs Linux based on Debian 10 with kernel version `4.19.94-ti-r42`. Dynamic frequency scaling was disabled on the CPU and the clock fixed to 1 GHz. The measurement setup is based on the one described by Balasch et al. [BGRV15]. For our setup we placed a Langer magnetic near field probe, model RF-B, on one of the decoupling capacitors of the IC. The EM signal was then amplified by a wideband 30 dB low-noise amplifier from Langer, model PA 303. The contactless power measurement was recorded using a Teledyne HDO6054A oscilloscope sampling at 10 GS/s.

4.2 Software AES-128 on the STM32F415

Our first case study uses tinyAES [Kok] on the STM32F415. The sampling rate of the oscilloscope is set to 125 MS/s. A single power trace $\boldsymbol{t}$ is approximately 50M samples long and contains 50 AES-128 COs which are randomly distributed over the trace and interleaved with independent randomly ordered ShiftRows and MixColumns operations to make the task more difficult. One AES-128 execution is roughly 155 000 samples long while one round is approximately $w = 15\,175$ samples long before decimation or resampling was applied to the trace. As we are using the STM32F415 setup there is no clock jitter in this scenario.

As the used sample rate of 125 MS/s is not an integer multiple of the target's operating frequency (24 MHz), we first resample the trace at 120MS/s. Furthermore, before processing the trace with the tool we decimate it by a factor of 5 to speed up the computation. After resampling and decimation a single round will cover $w = 3\,035$ samples.

Figure 2 (top) shows the resampled and decimated version of the full trace that is used as the input to our tool. The trigger signal is plotted in green and indicates the starting point of each CO in this trace. Figure 3 shows a single tinyAES encryption on the left and a template after refinement generated by the tool on the right.

Success metric We set a GPIO trigger high immediately before each CO and set it back to low right afterwards. We obtain a separate trigger trace which we do **not** give to the tool and which we only use as ground truth for the evaluation. It allows us to verify the potential starting points of the COs indicated by the tool against their actual positions.

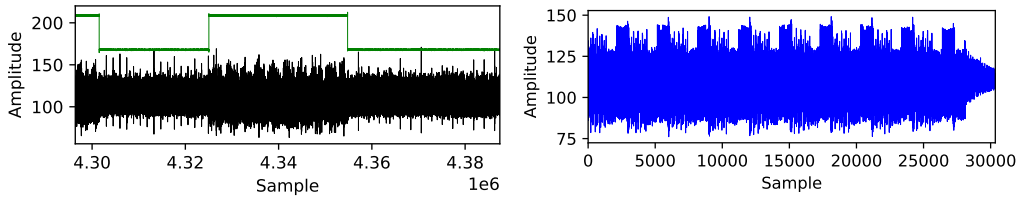

Figure 3: Power trace with one AES-128 operation (tinyAES) on STM32F4 with trigger (left) and one CO template (right).

Our metric to judge the tool's accuracy is then based on the distance between the true start index and the CO start index given by the tool. In practice the true start index is not exactly the GPIO trigger position, but it is a constant and known number of clock cycles later. We therefore correct the distance numbers by this offset. Further, in order to make the results and the accuracy analysis comparable over all case studies with different clock frequencies, sample rates and round widths, we report the distance not in the number of samples unit but normalized to the number of rounds unit. That is, if we determined a distance of 5 samples and one round is 15175 samples long, we work with a distance of $\delta = \frac{5}{15175} = 0,000329489$ rounds, and report a rounded value. This reflects the spirit that we are primarily interested in finding the COs and that we do not require sample-accuracy. As long as we find the COs any remaining misalignment can be corrected using standard alignment techniques.

In order to reduce the information to a single figure of merit, we report the number of hits. We consider a location a hit if the absolute value of the distance δ is at most $r - 1$, in other words, if there is at least one round length overlap between the CO template and the CO segment in the trace.

Results In this case study and all the case studies that follow we only show the evaluation when using a round template as a CO template (step 2). Additionally, we only use the Pearson correlation as the similarity measure when generating the round template (step 1), and for finding the CO in the trace (step 3). We further evaluate the impact of a single refinement step and of an unlimited number of refinement steps (step 4).

We also tried using a round trace segment as a CO template and using SAD as a similarity measure. The presented combination of round template and Pearson correlation was empirically determined to be the most robust across all case studies. Therefore, we only include these results in the paper.

Table 1: STM32F4 tinyAES accuracy results.

CO template	similarity step 1	similarity step 3	hits no refine.	hits 1 refine.	hits unlim. refine.
round template	Corr	Corr	100%	100%	100%

We provide the results for the tinyAES use case in Table 1. The tool performs very well in this scenario. It achieves not only 100% hit rate, but furthermore the rounded distance δ is zero rounds in nearly all cases. Next we look at a more challenging scenario.

4.3 Hardware AES-128 on STM32F415

Our second case study uses the hardware AES-128 accelerator on the STM32F415. There are again 50 AES-128 COs randomly distributed over a single 50M samples long trace. The hardware AES is called through the libopencm3 library [lib]. The AES-128 function calls are spaced out randomly over the trace and interleaved with independent ShiftRows and

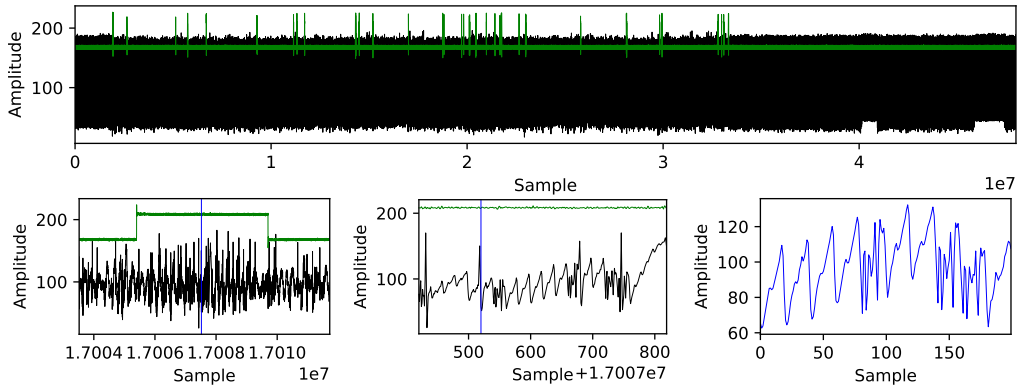

Figure 4: Power trace with 50 hardware AES-128 operations on STM32F415 and their starting positions (top). One AES operation with trigger (bottom left), zoomed (bottom centre) and one CO template (bottom right).

MixColumns operations implemented in software to make it more difficult. The sample rate of the oscilloscope is set to 500 MS/s. According to the reference manual [STM21] a single AES-128 operation takes 14 clock cycles of which a few are setup of the registers, hence we assume it is a round-based implementation and the actual encryption is 10 clock cycles long. An entire AES-128 operation is around 200 samples long while a single round is 20 samples long. There is no clock jitter in our STM32F415 setup.

Figure 4 shows a decimated version of the trace that we use. The trigger signal is plotted in green and indicates the starting point of each CO in this trace. The top figure shows the full trace. The figure on the bottom left shows a single hardware AES-128 encryption, the blue vertical line marks the exact starting point of the AES. We plot a large time window to show that the AES is not trivially distinguishable from the rest of the trace. The figure in the bottom centre shows a zoom on the actual hardware AES-128 (from roughly time index 520 to 720). The figure on the bottom right shows a template after refinement generated by the tool.

Results In this case study we evaluate the same range of parameters and choices as in the previous case study. Before processing the trace with the tool we resample it at 480 MS/s and do not decimate it. We provide the results for the hardware AES in Table 2. The results confirm that this scenario is more challenging for our tool.

Table 2: STM32F4 hardware AES accuracy results.

CO template	similarity step 1	similarity step 3	hits no refine.	hits 1 refine.	hits unlim. refine.
round template	Corr	Corr	16%	48%	100%

It is hard to say beforehand if refinement steps will improve or worsen the accuracy as this depends on the concrete features and defects which are present in the CO template before refinement. They can for example cause that the COs in the trace are not located exactly but with some offset, or that they are not aligned properly, and then get averaged to generate the refined template.

In Figure 5 the templates used by the tool before and after refinement are shown. The trace from the STM32F415 hardware AES contains some random noisy peaks which, if they coincide with the hardware AES, can skew the result. If a template is constructed from a relatively large number of samples, the sparse noisy peaks will have a small impact on the template's quality. But if the template is relatively short, as is the case for the hardware

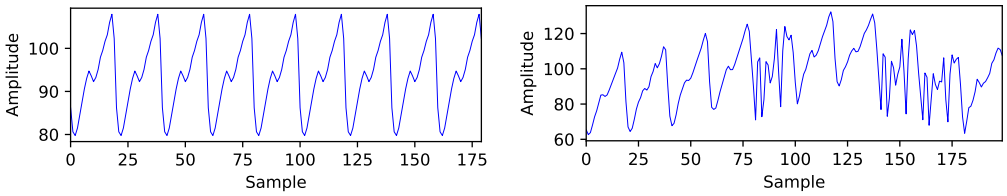

Figure 5: Template for HWAES on STM32F4 before and after refinement.

AES, a big noise spike in the wrong location can result in a bad template selection. With the round template in Fig. 5 (left) we circumvent those noisy peaks by averaging all rounds but do not incorporate the low frequency pattern of the entire CO. This effect is visible as an upwards trend in Fig. 4 (bottom centre). There is a significant improvement in the hit rate after refinement, because the new template in Fig. 5 (right) includes the low frequency pattern. This effect is observable in Table 2, even though the refined template contains noisy imperfections.

4.4 Mbed TLS AES-128 on STM32F303

For our third case study we run Mbed TLS version 2.24 on the STM32F303 target board and set the clock frequency to 48 MHz. We use the Platform Security Architecture (PSA) interface of Mbed TLS to encrypt a single plaintext block with AES-128. Before the AES we call the SHA-256 hash function also through the PSA interface to hash 1 block of data, in order to have some other pattern present in the trace. The PSA interface is a high level interface which has a relatively large overhead as a single function call takes approximately 8.3 million samples of which only roughly 60 000 are spent on the AES encryption. The trace is 31.25M samples long and contains a noticeable amount of clock jitter as the microcontroller is running of its internal clock.

Figure 6 shows a decimated version of the trace that we use. The trigger signal is plotted in green and indicates the starting point of any AES CO in the trace. The top figure shows the full trace. Roughly the first half of the trace corresponds to the execution of the SHA-256 and the rest of the trace to the AES-128 encryption of a single block. Clearly and surprisingly, the AES-128 routine in which we added our code to produce a trigger signal is executed much more often than once. And it is also executed after our call of the SHA-256 operation. The tool finds in total 47 AES-128 COs in the trace. The very last one of them is the encryption of the single data block we expected. The others are needed for Mbed TLS / PSA interface initialization (set-up of a DRBG).

The figure on the bottom left shows a single Mbed TLS AES-128 encryption and the figure on the bottom right shows a template before refinement generated by the tool.

Results Before processing the trace with the tool we resample it at 1.2 GS/s and decimate it by a factor of 5. We provide the results for the Mbed TLS AES-128 case study in Table 3. Since there is a noticeable amount of clock jitter present in the long trace due to drift of the internal oscillator one would expect a low success rate. The jitter is however countered by setting the ϵ introduced in Section 3.3.3 sufficiently high. For this use case ϵ was set to 2. Using the tool in this particular use case with $\epsilon = 0$ reduced the number of hits significantly. The results show that the tool works very well for the shown configuration. However, using SAD as a similarity measure for step 1 will lead to no success.

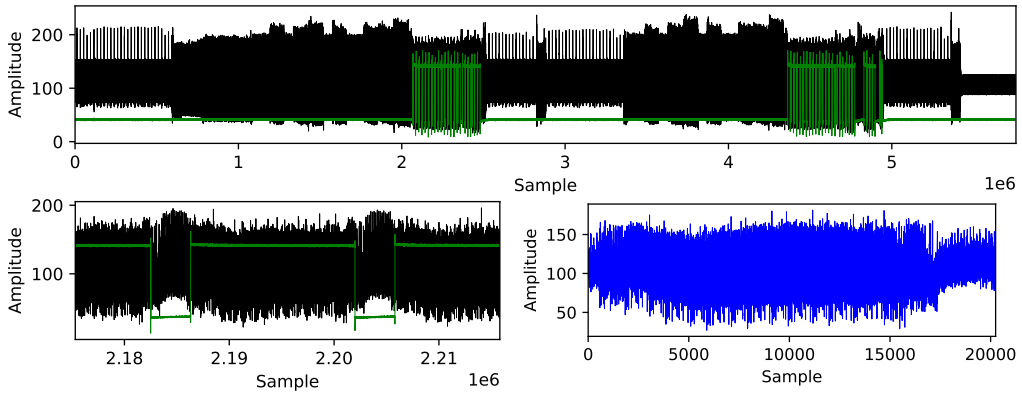

Figure 6: Power trace with 47 Mbed TLS AES-128 operations on STM32F303 (top). One AES operation with trigger (bottom left) and one CO template (bottom right).

Table 3: STM32F3 Mbed TLS AES-128 accuracy results.

CO template	similarity step 1	similarity step 3	hits no refine.	hits 1 refine.	hits unlim. refine.
round template	Corr	Corr	100%	100%	100%

4.5 Mbed TLS SHA-256 on STM32F303

For our fourth case study we run Mbed TLS on the STM32F303 target board. This time we call the PSA interface to hash a block of data using SHA-256 [Dan15]. For this experiment the microcontroller's main clock frequency was set to 72 MHz and the reference clock from which this clock was derived was either provided by an external or by an internal reference. This is to show the impact clock jitter can have on the success rate of the tool. The sampling rate of the oscilloscope was lowered to 250MS/s and its internal 20 MHz low pass filter was enabled. The trace is 50M samples long. A single SHA-256 operation is roughly 440 000 samples long.

Figure 7 shows a trace of the Mbed TLS implementation of SHA-256 with the full trace (top), the CO (bottom left) and the template after refinement (bottom right). The trace contains 33 SHA-256 COs.

There is a small but noticeable difference in the power trace of the CO between the first 16 rounds and the last 48 rounds. While the first 16 rounds are roughly 6 490 samples long the last 48 rounds are 6 950 samples long. This is because the implementation of the first 16 rounds of SHA-256 is slightly different from the implementation of the remaining 48 rounds.

If the user does not take this difference into account and sets up the tool to look for $r = 64$ identical rounds when locating COs, the tool will not be able to find the correct number of required sub-peaks and decrease the number until it eventually works. If the user knows about the difference and takes it into account he can immediately set up the tool to look for $r = 48$ or for $r = 16$ identical rounds and save processing time.

Results Table 4 and Table 5 show the results for the Mbed TLS implementation of SHA-256 running from either the external or the internal reference clock. We took the difference between the first 16 and the last 48 rounds into account when configuring the tool's parameters. The reported results are for locating $n = 33$ COs and looking for $r = 16$ identical rounds.

When comparing Table 4 and Table 5 it becomes obvious that a bit of clock jitter and thus a difference in length w of the rounds has a noticeable impact on the success

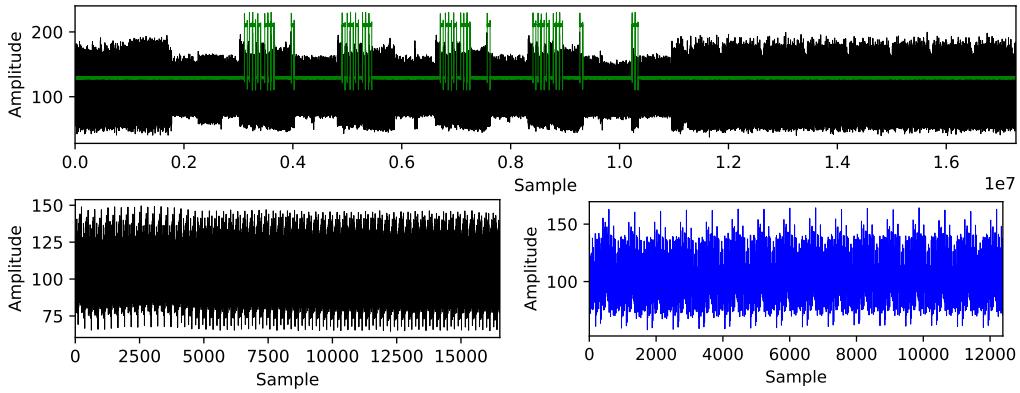

Figure 7: Power trace with 33 SHA-256 operations on STM32F303 (top). One SHA-256 operation with 64 rounds (bottom left) and one CO template with 16 rounds (bottom right).

Table 4: STM32F4 SHA-256 on **stable external clock**, accuracy results.

CO template	similarity step 1	similarity step 3	hits no refine.	hits 1 refine.	hits unlim. refine.
round template	Corr	Corr	100%	100%	100%

Table 5: STM32F4 SHA-256 on **unstable internal clock**, accuracy results.

CO template	similarity step 1	similarity step 3	hits no refine.	hits 1 refine.	hits unlim. refine.
round template	Corr	Corr	82%	97%	100%

rate. This is particularly true as the jitter accumulates over the long trace. While the tool effortlessly finds all COs when the clock is derived from a stable external reference we can observe a noticeable drop in success rate when we switch over to the unstable internal reference clock.

Table 5 further shows that if a good enough CO template was initially constructed from the round template, the tool will correctly locate all the COs in the trace after some refinement steps.

If we build the CO template for the 48 rounds instead of the 16 rounds, the clock jitter becomes a bigger problem because of the increased length of the template. We tested this option and the results were worse, even with an increased value of ϵ.

4.6 BeagleBone Black OpenSSL AES-128

For our fifth case study we use a default OpenSSL (`1.1.1d`) AES-128 encryption running on the BeagleBone Black. By default we mean that we made no attempt to enable a certain code variant, hardware acceleration or similar.

Due to the high clock speed of the processor we cannot place individual triggers around every CO. We therefore opted to place a trigger around the entire function call which contains multiple COs. This however means that we can not establish a ground truth and therefore do not include a results table for this case study. The oscilloscope is triggered right before the OpenSSL call to encrypt 32 blocks of data.

This use case corresponds best to a real-world scenario in which the user does not know what the pattern of the CO looks like and does not know the exact width of one round of the CO he tries to locate. Assuming the AES-128 is a t-table based implementation we set the search space for the round width w from 1 to 50 clock cycles, which is equivalent to 2

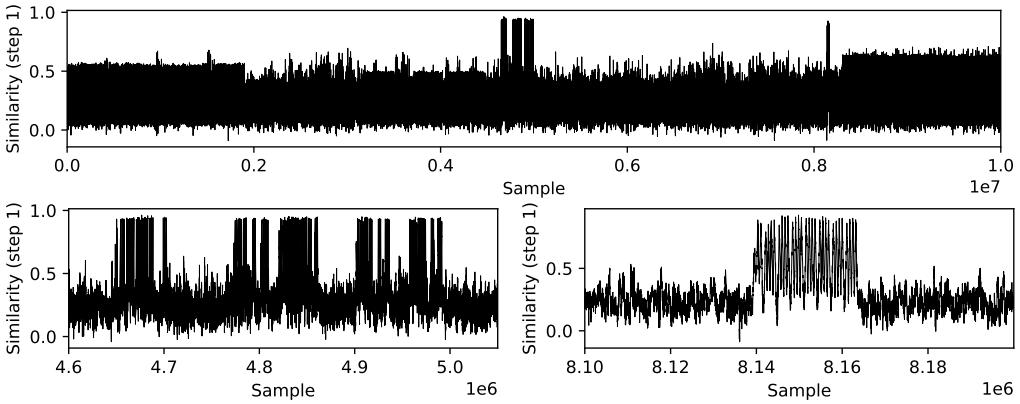

Figure 8: Similarity (step 1) for each sample in the entire trace (top). Zoom in on left block of higher similarity (bottom left) and zoom in on right block (bottom right)

to 100 samples. The length of the entire trace is 10M samples after decimation by factor 5. We do not use any refinement steps.

Surprisingly, the method does not terminate because it claims to have found 140 COs which is significantly more than the 32 COs that should be in the trace. Figure 8 (top) shows a plot of the similarity vector created in step 1. We can see two blocks of higher similarity, one in the middle of the plot and one further to the right. The two plots at the bottom show a zoom on each block in the similarity plot. Visual inspection of the two blocks leads to the conclusion that the block on the right hand side, which has 31 peaks, should be the correct block of COs, because it almost matches the number of expected COs. We assume that the peak corresponding to the first CO is missing because the first execution of the code is not cached and therefore looks different in the trace than the subsequent executions. This effect has been previously observed by Balasch et al. [BGRV15]. We further verified that the peaks in this block are the AES encryptions we are looking for, by calling OpenSSL to encrypt 100 data blocks and observing the number of peaks increase to 99.

Figure 9 (top) shows the entire trace. By running the tool on only the right half of the trace we are able to locate exactly 31 COs as we expect. The tool again iterates over all different widths in order to find the correct starting points for each of the 31 COs. It is able to recover the starting points for all 31 AES-128 encryptions in less than one minute. The presence of the other COs in the left half of the trace can again be attributed to AES being used in initialization of OpenSSL. The tool determined the round width w to be 56 samples which matches with the round width observed in the trace of a single AES-128 operation shown in Figure 9 (bottom left).

The template for the CO generated by the tool (Figure 9, bottom right) can be used to recover the COs from all subsequent traces, or as reference in a waveform-matching based triggering system.

4.7 AES-128 in a Secure Boot Scenario

For our final case study we target a secure boot example implemented using the Mbed TLS library. The example follows the same sequence of operations which one would expect in a real secure boot process. We assume that the evaluator is able to obtain the encrypted firmware image. The target code was executed on the same STM32F303 target platform with the same settings as were used for our third use case discussed in Section 4.4. The target platform is a custom development board geared towards side-channel measurements. The secure boot example first computes a hash digest over 8 kB (i.e. 500 16-byte blocks)

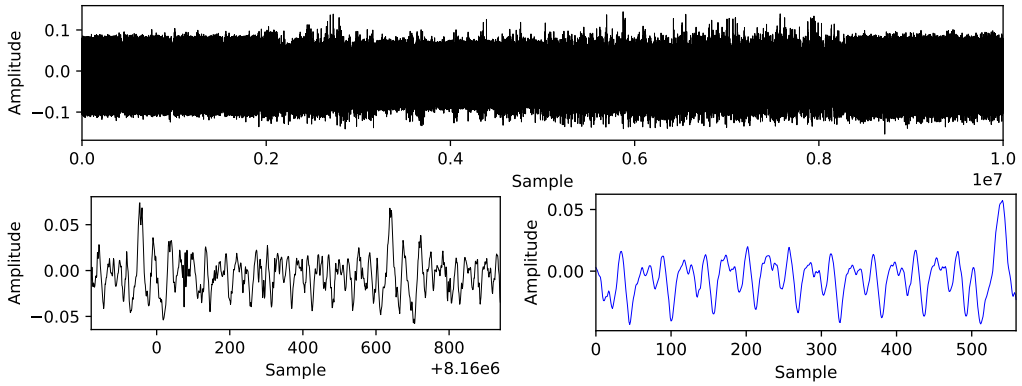

Figure 9: Power trace with 140 AES-128 operations on the BeagleBone Black (top). One AES operation (bottom left) and one CO template (bottom right).

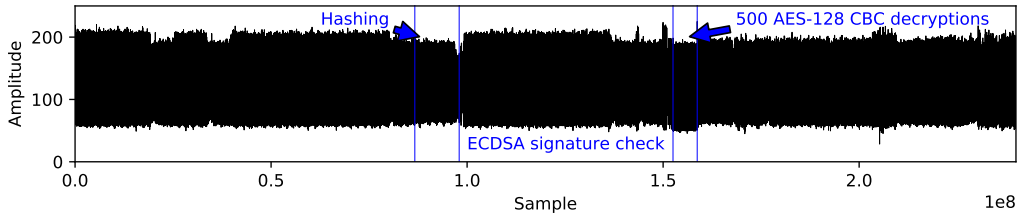

Figure 10: Power trace with 500 Mbed TLS AES-128 operations on the STM32F303.

of data followed by an ECDSA signature verification. If the signature verification succeeds the data is decrypted using AES-128 in Cipher Block Chaining (CBC) mode. The trace capture was started on the release of the reset pin of the microcontroller. Figure 10 shows a single trace consisting of 250M samples centered around the entire secure boot sequence. Note that this trace contains information related to multiple cryptographic primitives and other routines (e.g. `memcpy` and `memcmp`), all of which consist of repeating operations and thus result in repeating patterns in the side-channel trace. We demonstrate the effectiveness of our tool by performing a CPA-based key recovery attack. Successful recovery of the key demonstrates a correct segmentation of the COs, as failing to detect a single CO breaks the relation between trace segments and ciphertexts.

Concretely, the tool was first used to identify potential starting points of the targeted COs. Our method requires knowing the number n of COs. Usually, when the device under attack is booting from an external Non-Volatile Memory (NVM) it is possible to extract the sequence of blocks read during the boot process. As each block has to be decrypted the number of blocks corresponds to n. Now, in order to identify the correct width w the procedure from Section 2.3 was followed. We set the deviation of the tool to $\epsilon = 2$ as in all experiments before. Furthermore, we also used Pearson correlation as a similarity measure and the error margin was set to 1%, allowing for some misclassifications to happen before refinement. 40 different widths were tested by the tool to find out the correct length of one round in the CO template. The tool considered three of those widths to be plausible. The two false positives were ruled out by briefly inspecting the found matches and the created CO template after an additional refinement step. Further confirmation of the correct width was gained by looking at the correlation between the Hamming weight of the incoming ciphertext with the respective parts of the power trace $\boldsymbol{t}$ found by the tool. In total it took three minutes for every tested width, three minutes for the refinement of the three candidates and the key recovery itself took less than a minute. Overall, the automated unsupervised analysis to derive the three waveform template candidates took approximately 2 hours after which the correct candidate was manually selected within a few minutes.

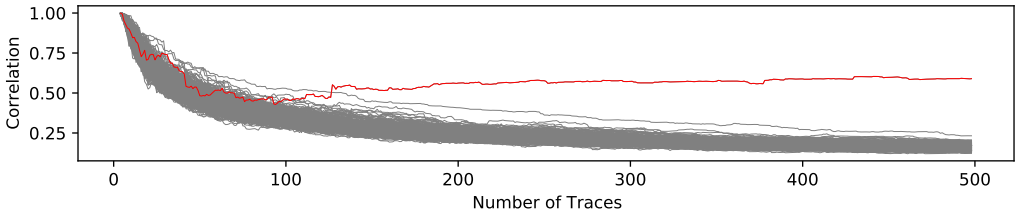

Figure 11: CPA result when targeting the first key byte with an increasing number of traces. Correlation for the correct key byte value is shown in red.

Each segment corresponds to a single CO. No further post-processing or alignment steps were performed on the individual segments. Afterwards, we performed CPA to extract the key of the AES-128 decryption. For this attack, we require the separated individual trace segments per CO as provided by our tool. Moreover, we require the ciphertext used per CO. The sequence of blocks read from NVM during the boot process can be extracted. They can then be matched with the sequence of identified CO trace segments. With this information (pairs of CO trace segments and the ciphertext used for the CO), we perform the CPA. Figure 11 shows the successful CPA result on these segments targeting the first key byte in the first round of the AES-128 decryption.

This use case further demonstrates the practical applicability of our tool in a scenario where classical techniques would not help. The tool was able to successfully, and semi-automatically, segment one trace containing different cryptographic primitives into individual target segments. A successful CPA attack was mounted, allowing to recover the full cryptographic key.

5 Complexity and Timing Analysis

The runtime of the tool will be determined by the parameter choices made by the user and the properties of the CO. In this section we study the complexity of the proposed method and provide concrete timings achieved by our tool.

5.1 Complexity Analysis of the Proposed Method

Before describing the runtime complexity of the tool as a whole we look at the complexity of the different steps the method takes to locate the COs as described in Section 2.

1. Identifying the most probable location of one CO (step 1) requires the computation of a similarity score between the averaged round template of length w and each of the r rounds. This computation needs to be performed at each offset in the trace $\boldsymbol{t}$. In $\mathcal{O}$ notation this becomes $\mathcal{O}(w \cdot |\boldsymbol{t}| \cdot r)$.

2. The generation of a CO template has a negligible overhead.

3. Using the generated template to recover the remaining COs requires us to correlate the CO template of width $r \cdot w$ with the trace at every offset using a similarity measure. This reduces to a complexity of $\mathcal{O}(w \cdot |\boldsymbol{t}| \cdot r)$.

4. When using trace refinement, step 3 needs to be repeated for every refinement step. If we denote the number of refinement steps by z the complexity of steps 3 and 4 combined becomes $\mathcal{O}(w \cdot |\boldsymbol{t}| \cdot r \cdot z)$.

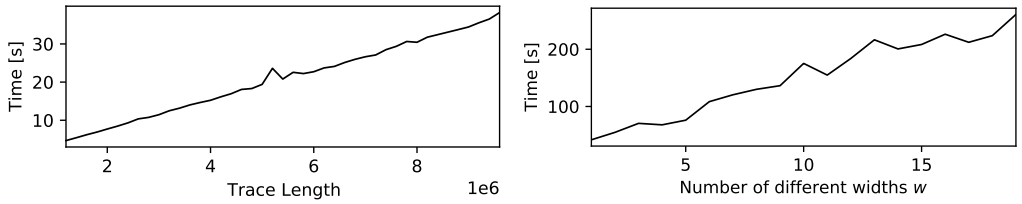

Figure 12: Time needed for the entire method with the TinyAES trace of Section 4.2. Different trace lengths and a fixed width of $w = 3\,035$ samples (left) and different number of widths and a fixed trace length $|\boldsymbol{t}| = 10$ million (right).

The timing complexity of the method as a whole when testing a single w is the combination of the above steps and thus equals $\mathcal{O}((w \cdot |\boldsymbol{t}| \cdot r) \cdot (1 + z))$. As the w is generally not known to the user he will have to iterate over multiple different w increasing the complexity. Some smaller operations such as locating the different sub-peaks in step 3 are omitted from the complexity analysis as their overhead is negligible compared to the calculation of the similarity scores in step 1 and 3.

The speed of the tool will thus depend mainly on the similarity score used in step 1 and 3 of the algorithm.

5.2 Pearson Correlation timing analysis

Pearson correlation has multiple necessary computations as can be seen in Eq. 2.

$$r_{xy} = \frac{\sum_{i=1}^{w}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{w}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{w}(y_i - \overline{y})^2}} \tag{2}$$

This takes up a lot of time, especially if the algorithm is implemented in a sequential manner. However, an implementation on a GPU can significantly cut on the execution time. In our experiments we are comparing a Ryzen 5 3600 CPU with an RTX 3070 GPU. The GPU implementation of the Pearson correlation is over 400 times faster than the CPU implementation for traces longer than 200k samples. The limiting factor of the GPU is then not the computation itself but the memory accesses. Computing SAD instead of the Pearson correlation is therefore not faster on a GPU with sufficient computational power. For example, when calculating a similarity score between a template of width $w = 10\,000$ and a trace $\boldsymbol{t}$ of length 1 million (as in step 3), the GPU accelerated version needs approximately 0.22 seconds whereas the same calculation on the CPU needs 100 seconds.

We plot the execution time for the complete processing of a trace in Figure 12. We observe that the execution time scales linearly with trace length $|\boldsymbol{t}|$ (left). Experiments with different round widths w show that it also scales linearly with the width. Figure 12 (right) shows the time needed if the tool has to check an increased number of widths which, again, scales linearly.

6 Conclusion

Pinpointing the exact location of a particular CO without having a template within a side-channel trace can be a difficult task. Especially when the execution time of the CO is negligible in comparison with the overhead, searching the CO becomes like searching for a needle in a haystack. The tool presented in this work aids an evaluator in the process of locating a particular CO within a side-channel trace in a (semi-)automatic manner,

requiring only a minimal amount of user input. The evaluator needs to provide the tool with certain meta information about the CO and its implementation. The proposed method however has a a high degree of parallelizability which allows the evaluator to trade off between time overhead and computation power.

We demonstrated the effectiveness of the tool on multiple different platforms and executing different COs. For each use case the tool was able to recover all the COs present in the trace, provided that the correct parameter values were used. The proposed method is not affected by masking, shuffling and similar countermeasures, because they do not hide the overall structure of the CO. As the method requires the width of a round to be constant it will only be effective on traces of constant time implementations. COs protected by effective time randomization countermeasures, e.g. random delays and unstable clock frequency can not be located by the tool. Our open-source implementation enables others to extend the tool in the future. One such extension could include the use of DTW as a similarity measure, potentially making the tool more robust against effective time randomization countermeasures.

Acknowledgments

This work was supported in part by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 "Cybercrime and Forensic Computing" (grant number 393541319/GRK2475/1-2019). This work was supported in part by CyberSecurity Research Flanders with reference number VR20192203, in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058, and in part by the European Commission through the Horizon 2020 research and innovation programme under grant agreement Cathedral ERC Advanced Grant 695305.

References

[BBGV16] Arthur Beckers, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Design and implementation of a waveform-matching based triggering system. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*, volume 9689 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2016.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[Bea] BeagleBoard.Org. Beaglebone black. https://beagleboard.org/black. Accessed: 2021-07-15.

[BGRV15] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. DPA, Bitslicing and Masking at 1 GHz. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293, pages 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[Dan15] Quynh Dang. NIST FIPS 180-4: Secure Hash Standard. In *Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD*, 2015.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008.

[GGJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KJJR11] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *J. Cryptogr. Eng.*, 1(1):5–27, 2011.

[Kok] Kokke. tinyAES. https://github.com/kokke/tiny-AES-c. Accessed: 2021-07-15.

[lib] libopencm3. libopencm3. https://libopencm3.org/. Accessed: 2021-07-15.

[Newa] NewAE. ChipWhisperer Pro custom trigger feature. https://wiki.newae.com/Tutorial_P1_Using_a_Custom_Trigger. Accessed: 2021-07-15.

[Newb] NewAE. Cw308 stm32f. https://rtfm.newae.com/Targets/UFO%20Targets/CW308T-STM32F.html. Accessed: 2021-07-15.

[Newc] NewAE. Cw308 ufo. https://rtfm.newae.com/Targets/CW308%20UFO.html. Accessed: 2021-07-15.

[NIS] NIST. Post-Quantum Cryptography. https://csrc.nist.gov/Projects/post-quantum-cryptography. Accessed: 2021-07-15.

[Ris] Riscure. Inspector hardware, icWaves. https://www.riscure.com/security-tools/inspector-hardware. Accessed: 2021-07-15.

[STM21] STMicroelectronics. *STM32F405/415, STM32F407/417, STM32F427/437 andSTM32F429/439 advanced Arm®-based 32-bit MCUs*, February 2021. Rev. 19.

[vWWB11] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving Differential Power Analysis by Elastic Alignment. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 104–119. Springer, 2011.