Detecting Program Cycles through Power Consumption

František Dejm, David Pokorný, Petr Socha, Martin Novotný
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
frana.dejm@gmail.com, {david.pokorny, petr.socha, martin.novotny}@fit.cvut.cz

***Abstract*—We propose a method for identifying program cycles in power traces, estimating their iteration counts, and locating individual cycle iterations. The method applies a modified Rapid Alignment Method to detect and group recurring points of interest within a single trace, associates these groups with program cycles, and uses correlation-based matching of trace segments to recover iteration starts. Building on this information, the approach can support trace realignment, estimate unknown cycle counts such as those arising during sample rejection in lattice-based cryptography, and facilitate simple power analysis attacks based on program branching.**

***Index Terms*—side-channel analysis, power trace alignment, cycle detection**

I. Introduction

The security of modern digital systems depends on cryptographic algorithms to ensure the confidentiality, integrity, and authenticity of data. Widely used algorithms have been thoroughly analyzed for theoretical weaknesses and are considered secure in a mathematical sense. However, the security of a cryptographic system is dependent on the physical implementation of an algorithm as much as the algorithm itself. An insecure implementation can leak information through various channels, including timing, power consumption, and EM radiation, potentially allowing an attacker to recover sensitive data without breaking the underlying algorithm.

Paul Kocher first exploited such a vulnerability in 1996 [7], successfully attacking RSA and Diffie-Hellman key exchange implementations by measuring the execution time of modular exponentiation. A more general attack on hardware implementations of cryptographic algorithms called Differential Power Analysis (DPA) [6] was published in 1999. DPA relies on measuring the power consumption of a chip over several executions of an algorithm with the same secret key. Statistical analysis of the power traces then allows the attacker to reconstruct the key. Several similar attacks on the power consumption side channel, including Correlation Power Analysis [2], Mutual Information Analysis [5] and Template Attacks [3], have since been developed.

All of these methods typically target a particular instruction executed by the chip and are therefore dependent on an attacker's ability to repeatedly measure the power consumption of that instruction. In practice, power traces become misaligned due to the absence of a trigger indicating the start of the measurement, or when the traces themselves vary in length due to branching, or deliberate countermeasures, such as randomly executing dummy instructions [4] or altering the clock frequency during execution [1]. Misaligned traces make the aforementioned attacks significantly more difficult, typically requiring a much larger number of measurements to be effective. Preprocessing, in the form of restoring the alignment of measured traces, is therefore often performed to increase the success rate of an attack [8].

One particular cause of misalignment is a technique called rejection sampling, commonly used in post-quantum lattice-based cryptosystems, in which randomness is used to ensure that accepted signature values follow a desired probability distribution. If a candidate value is rejected, the sampling process is repeated, resulting in a loop with a random number of iterations. From the perspective of side-channel analysis, only the final iteration that produces an accepted sample is typically of interest.

A. Contribution

In this work, we propose a novel method, based on the Rapid Alignment Method (RAM) [9], for detecting cycles in power traces, determining their iteration counts, and identifying the start of each iteration. Alignment methods such as Elastic Alignment [10] and RAM focus on overcoming countermeasures like random process interrupts and offer no way to directly detect iteration starts and realign traces of algorithms that employ rejection sampling. The proposed method does not require any prior knowledge of the number of iterations of any cycle, only knowledge of the control flow structure of the program under analysis. In addition, the method does not require multiple measurements of the power consumption of an algorithm. All results can be extracted from a single power trace, making it a useful tool for horizontal attacks.

In Chapter II, we describe the Rapid Alignment Method [9], which forms the basis for our work. Chapter III describes how our method analyzes power traces of programs with non-nested cycles and recursive function calls, and Chapter IV extends this approach to algorithms which contain nested cycles. Finally, the method is experimentally validated on power traces of AES and RSA in Chapter VI.

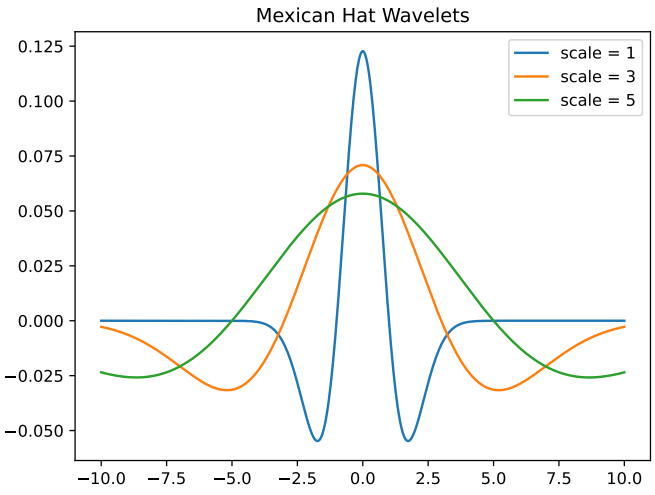

Fig. 1. Mexican hat wavelets at different scales

II. Rapid Alignment Method

The Rapid Alignment Method (RAM), published by Muijrers et al. [9], is inspired by image processing algorithms. It uses wavelet convolution to identify and describe points of interest (POIs) within a trace. The POIs of two traces are then matched to each other, and the traces are finally shifted to restore alignment of the POIs. The method is divided into four distinct procedures: the detector, descriptor, matcher, and warper.

The detector recursively identifies POIs within a trace. The convolution of a trace and a chosen wavelet is calculated, and points with a sufficient wavelet response are saved as POIs. The authors suggest mexican hat wavelets for this purpose, shown in Fig. 1. Each interval between two existing POIs is then investigated again using a scaled-down wavelet. The location and scale of each POI are saved. Selected POIs are pruned so that no two POIs of the same scale occur too close to one another. The allowed distance of two POIs is a parameter of the procedure.

The descriptor procedure creates a vector of features for each POI detected in the first step. The area around a POI is divided into several sections, see Fig. 2. For each section, the wavelet response of several equidistant samples is calculated and normalized by a Gaussian distribution centered at the POI. The wavelet scale used depends on the scale at which the POI was detected. It is recommended to use Haar wavelets, shown in Fig. 3. The sums of the normalized responses and the absolute values of the normalized responses are then stored in the feature vector to describe that section.

The matcher then finds pairs of similar POIs on the same scale. The distance of two POIs is calculated as the normalized Euclidean distance of their feature vectors, defined as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{N} \frac{(x_i - y_i)^2}{\sigma_i^2}}, \tag{1}$$

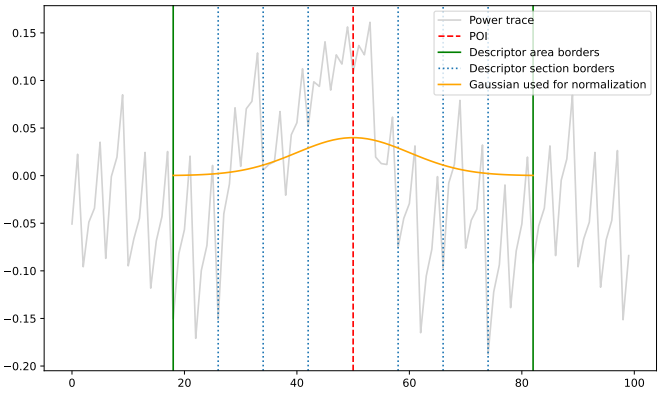

Fig. 2. The area around a POI is divided into sections. Each section is described by summing the wavelet response of its samples and normalized using a Gaussian distribution.

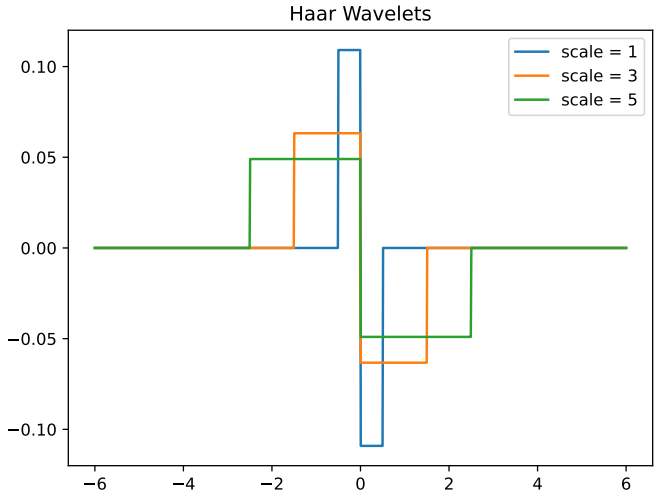

Fig. 3. Haar wavelets at different scales

where $\mathbf{x}$ and $\mathbf{y}$ are feature vectors, x_i and y_i denote their i-th components, and σ_i is the standard deviation of the i-th component across all feature vectors. A list of matches is maintained for each POI and the match with the smallest distance is assumed to be the correct one.

Cross-matching can occur at this point. Let p_1, p_2 be two POIs from one trace and let q_1, q_2 be two POIs from the other trace, such that p_1 occurs before p_2 and q_1 occurs before q_2. A cross match is a situation where p_1 is matched to q_2 and p_2 is matched to q_1. Such matches would violate the temporal qualities of power traces; therefore, one of them must be wrong. A confidence score for each match is calculated to solve this problem. The confidence that a match is correct is the difference between the distance of the best and second-best matches. Once all mutually exclusive matches have been identified, the one with the lowest confidence score is discarded. This is repeated until all cross matches have been resolved. The final part of the method, the warper, aligns all matching POIs and interpolates between them.

III. Analyzing Non-nested Cycles

In this chapter, we build upon RAM to create a method to identify cycles and find their iteration starts. Let us first consider the case of non-nested cycles. We assume that a power trace may contain several cycles and each cycle executed a random number of iterations as a result of either a loop in the code or a recursive function call. The core idea of our method is to apply RAM-based POI detection and matching within a single trace to identify patterns corresponding to the start or end of each iteration. This information is then used to detect individual iterations of each cycle.

The method consists of three main steps described in the following three sections. First, a modified RAM [9] detector, descriptor and matcher are used to create POI groups, where each group represents recurring features within the trace. Second, the method selects which POI groups describe each cycle. Finally, these POI groups are used to identify trace segments corresponding to each iteration, providing the information necessary for trace realignment, e.g., by discarding all but the last iteration of each cycle.

A. Creating POI Groups

The first step of our method is to detect points of interest (POIs) within the trace and organize them into POI groups. We define a POI group as a sorted list of POIs, ordered by the time of their occurrence in the power trace. The sample index of a POI is the index of the trace sample where the POI occurs. The sample index of the first POI in a group is referred to as the group's start, and the sample index of the last POI as the group's end. Two POI groups A and B are said to overlap if $A.start < B.start < A.end$ or $B.start < A.start < B.end$.

We adopt a modified version of the RAM detector to find POIs. Convolution between the trace and a Haar wavelet is calculated at each sample and samples with a sufficiently high wavelet response are considered as POIs. Unlike the original RAM detector, which recursively identifies POIs across multiple scales, we restrict detection to a single, fixed wavelet scale. As a result, the wavelet scale is an important parameter of the procedure.

A feature vector, containing information about trace features within a POI's vicinity, is then computed for each POI using the RAM descriptor. POIs are then grouped based on the Euclidean distance of their feature vectors: two POIs belong in the same group if their distance is smaller than a given threshold. This threshold is another key parameter. An additional constraint is placed on the created POI groups. The distance between every pair of POIs in a group must be smaller than the threshold. If this condition is violated, a list of POIs with mismatches is created. The POI with the highest number of mismatches is removed from the group and this process is repeated until the aforementioned condition is satisfied.

This step produces a list of POI groups, with each group representing a recurring feature within the trace. This forms the basis for the rest of the method, where we use these POI groups to analyze cycles and identify their individual iterations. An example of a POI group within a trace is shown in Fig. 4.

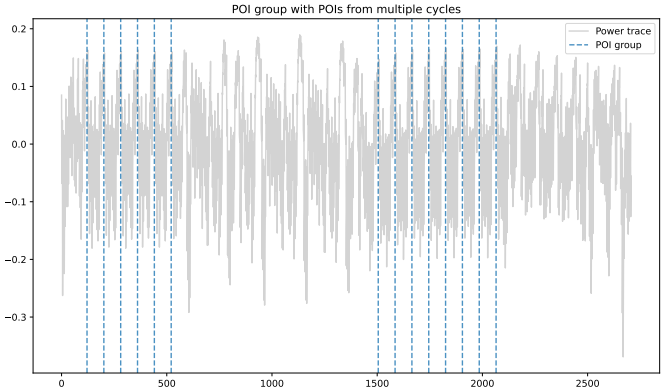

Fig. 4. A single POI group with POIs from two separate cycles

B. Associating Cycles with POI Groups

Once the POI groups have been created, the next step is to determine which groups correspond to each cycle. A single POI group may contain POIs from iterations of multiple cycles, e.g., when several cycles share identical iteration bodies, see Fig. 4. In such cases, the POI group must be partitioned into multiple subgroups, each containing only POIs from a single cycle.

The goal of this step is to select all POI groups that describe a given cycle and to restrict them so that they contain only POIs from iterations of that cycle. The method assumes prior knowledge of the algorithm's control flow structure. Specifically, it requires information about how many execution cycles the algorithm contains, as well as how many distinct loop-body variants may occur in each cycle. These variants typically arise from conditional branching within the loop body. However, the number of iterations of each cycle does not need to be known. Cycles are processed in the order in which they appear in the trace, and the following steps are executed for each cycle.

First, a single POI group that describes the cycle, i.e., contains POIs from iterations of that cycle, is selected. This POI group may also contain POIs from other cycles. If this is the case, the group is split into two new POI groups: one containing only the POIs from the current cycle, and one containing all remaining POIs from the original group, as described below. Since cycles are processed in accordance with their execution order on the chip, we can assume that the first n POIs from the original group are from the cycle currently under analysis, while the remaining POIs are from later cycles.

We introduce a splitting procedure that takes a POI group G and the number of possible iteration bodies n_{bodies} as input. We additionally assume that each possible iteration body appears at least once in the power trace, that is, that each possible control-flow path through the loop body was taken at least once. The output is a pair of POI groups: G_{cycle} contains

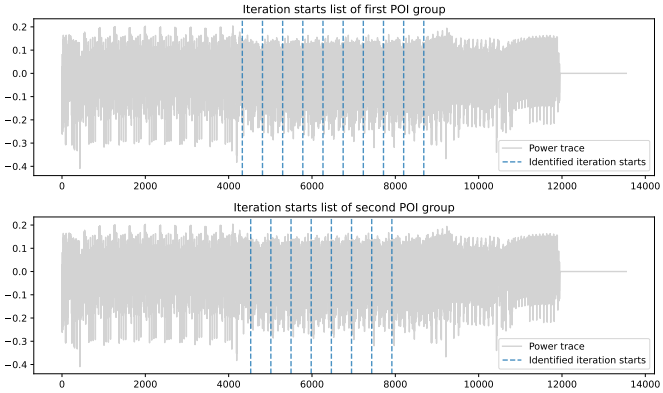

Fig. 5. Candidate iteration starts lists produced from two different POI groups. The first list has more iterations and would be chosen as the better candidate.

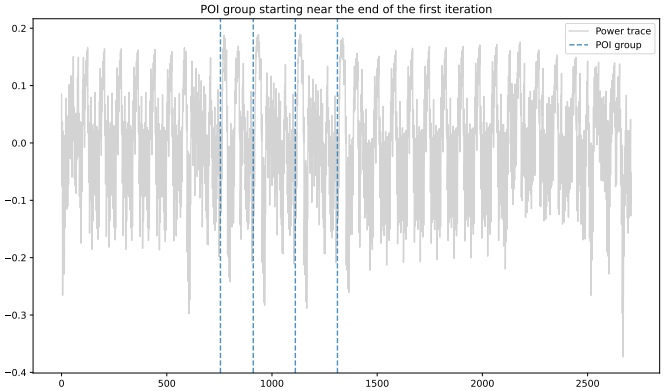

Fig. 6. A POI group whose first POI is at the end of the first iteration of the cycle.

all POIs in G that originate from the cycle currently under analysis, and G_{other} consists of all POIs in G that are not in G_{cycle}. The procedure iterates over gaps between consecutive POIs in G and uses the Pearson correlation coefficient to classify each trace segment. If a segment matches a known iteration body, its starting POI is added to G_{cycle}. When a segment fails to match a known iteration body and fewer than n_{bodies} bodies have been discovered so far, it is added to the list of known bodies. Otherwise, it is considered to be the boundary of the current cycle.

After the initially selected POI group has been split, a list of all POI groups G' that overlap with G_{cycle} is created. The following steps are executed for each of these groups:

1) Split G' into G'_{cycle} and G'_{other}.
2) Add G'_{cycle} to the list of groups associated with the current cycle.
3) Add all groups that overlap with G'_{cycle} and have not yet been processed to the list of overlapping groups.

This procedure produces a list of all POI groups that describe the current cycle, with no POIs from other cycles. In the following section, we describe how these POI groups are used to identify the start of each iteration within the cycle.

C. Finding Iteration Starts

The final step of our method is to determine the start of each iteration within a cycle. For each POI group selected in the previous step, we generate a candidate list of iteration start indices. The list with the largest number of iterations found is considered the best candidate. Iteration start lists produced by two different POI groups for the same power trace are shown in Fig. 5. The rest of this section describes how these candidate lists are created.

Similarly to the group-splitting procedure, the procedure for finding iteration starts takes a POI group G and the number of distinct iteration bodies n_{bodies} as input. Its output is a list of trace sample indices $iteration\_starts$. Each element of this list denotes the start index of an iteration of the cycle.

The first step of the procedure finds reference iteration bodies. Although this can be performed simultaneously with the splitting step in practice, we describe it separately here for better clarity. We once again iterate over gaps between POIs of G and use the Pearson correlation coefficient to identify distinct iteration bodies.

Finally, the list of reference bodies is used to find the starting points of individual iterations. The procedure begins by locating candidate first iterations through correlation: each reference body is matched against the trace to find positions where the first iteration might begin. Since the first POI in a group may occur near the end of the first iteration, see Fig. 6, the search window starts before the first POI of the group. Once a set of possible starting iterations has been identified, the procedure attempts to extend each candidate by chaining iterations. For example, if an iteration is detected at sample index 100 and its body length is 60 samples, the procedure will search for the next iteration near sample 160. The size of the area where the next iteration start is sought is defined by a parameter n_ϵ. This process continues until the chain can no longer be extended. This approach may produce multiple iteration chains. The procedure selects the longest chain in terms of number of iterations as the best candidate.

Having calculated a candidate iteration start list for each POI group, the longest of these lists is chosen as the best representation of the cycle, and its elements are taken as the definitive iteration starts. Two candidate iteration start lists are shown in Fig. 5.

IV. Extending the Approach to Nested Cycles

In this section, we describe how the method can be modified to analyze nested cycles. We consider the case where an outer cycle contains one or more inner, non-nested cycles. The core idea is that each iteration of the outer cycle is analyzed separately using the method described in the previous section. Although we focus on a single level of nesting, the same approach can be generalized to multiple levels of nesting.

The main requirement for analyzing nested cycles using our approach is that a POI group whose POIs separate individual iterations of the outer cycle must be manually selected. This POI group, denoted G_{outer}, is then used to analyze each itera-

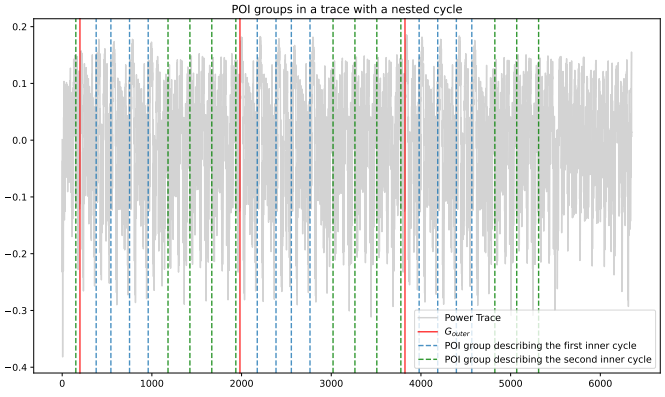

Fig. 7. A manually selected outer POI group with two POI groups describing the inner cycles in a trace with a nested cycle.

tion independently. The second requirement is the knowledge of how many cycles occur within each iteration of the outer cycle, along with the number of distinct iteration bodies each inner cycle may have. A possible selection of G_{outer} along with POI groups describing the inner cycles is shown in Fig. 7.

Given a POI group G_{outer} describing the outer cycle, we iterate over each pair of its consecutive POIs p_1 and p_2 and execute the following procedure. First, every POI group G that overlaps with $T[p_1 : p_2]$ is split into three new groups:

$$G_{before} = \{\, p \in G \mid p.sample\_index < p_1.sample\_index \,\},$$
$$G_{middle} = \{\, p \in G \mid p_1.sample\_index \leq p.sample\_index \wedge p.sample\_index \leq p_2.sample\_index \,\},$$
$$G_{after} = \{\, p \in G \mid p.sample\_index > p_2.sample\_index \,\}.$$

The group G_{middle} contains all POIs from G that are relevant for the analysis of that specific iteration of the outer cycle and is therefore added to the list of POI groups to be used.

Once the list of POI groups relevant to a given iteration of the outer cycle has been created, the method from the previous section can be applied to find the iteration starts of each inner cycle. This is done independently for every interval $T[p_1 : p_2]$ defined by consecutive POIs in G_{outer}.

After all such intervals have been processed, the method checks whether an additional iteration of the outer cycle occurred before the first POI of G_{outer} or after its last POI. The reference iteration bodies of the inner cycles are already known, so they are matched against the trace using the Pearson correlation coefficient within a window preceding and following the region covered by G_{outer}. If a match is detected, the corresponding segment is treated as an additional iteration of the outer cycle and analyzed using the method for non-nested cycles.

V. Parameters of the Method

In this section, we provide an overview of the parameters of our method. Most of these parameters are required for the procedures our method inherited from RAM [9]. A more detailed discussion of these parameters along with suitable values can be found in [9]. Experiments have shown that the method is particularly sensitive to the values of parameters that control POI detection.

Let us first discuss the RAM detector's parameters:

- *POI Detector wavelet scale* controls the scale of the mexican hat wavelet used for POI detection, see Fig. 1. A larger scale will produce stronger wavelet responses for wider peaks in the trace, whereas a smaller scale will emphasize more narrow peaks.
- *POI Detector threshold* defines how large the wavelet response of a sample must be for the sample to be considered a POI.
- *Minimal distance between POIs* dictates the smallest allowed gap (in samples) between two POIs. If two POIs are closer to each other than this minimal distance, one of them is discarded.

The second and largest group of parameters controls the RAM descriptor:

- *POI Descriptor area size* is the number of samples around a POI that contribute to that POI's feature vector, i.e, it is the size of the area around a POI that is considered when describing the POI.
- *POI feature count* defines the size of a POI's feature vector. The area around each POI is divided into this number of sections and each section is described by summing the Haar wavelet responses of samples within that section normalized by a Gaussian distribution centered at the POI, shown in Fig. 2.
- *POI Descriptor step size* determines the gap between samples when describing a section. If this value is equal to 1, then the wavelet response is calculated for each sample within that section, setting it to 2 would result in the wavelet response being calculated for every second sample etc.
- *POI Descriptor Gauss variance* defines the variance of the Gaussian distribution used for normalizing the wavelet responses. A higher value will result in less aggressive normalization, in effect making the sections further from a POI more important.
- *POI Descriptor wavelet scale* determines the scale of the Haar wavelet used for describing sections, see Fig. 3.

There is only one parameter for our modified RAM matcher:

- *POI match threshold* is the maximum allowed distance between two POIs that belong in the same POI group. The distance between two POIs is defined as the normalized Euclidean distance of their feature vectors, see Section II.

The last group of parameters controls the classification of trace segments:

- *Iteration match threshold* defines how strongly correlated two trace segments must be to be considered the same iteration body.
- *Iteration matching area* is the number of samples that will be tested as possible iteration starts when constructing iteration chains, see Section III-C.

TABLE I
PARAMETER VALUES USED IN OUR EXPERIMENTS

Parameter	RSA trace	AES traces
POI Detector wavelet scale	4	8
POI Detector threshold	0.25	0.35
Minimal distance between POIs	1	200
POI Descriptor area size	64	64
POI feature count	32	32
POI Descriptor Gauss variance	300	300
POI Descriptor wavelet scale	12	12
POI match threshold	0.05	0.05
Iteration match threshold	0.8	0.8
Iteration matching area	10	30

VI. EXPERIMENTAL RESULTS

The method was evaluated on power traces captured using a ChipWhisperer-Lite 32-bit platform with an STM32F303 target. We tested the method on power traces of RSA and AES. The parameter values used in these experiments are shown in Table I.

We implemented the individual steps of our method in a Jupyter notebook to visualize the outputs of each step and select suitable parameter values. Our experience shows that the method is particularly sensitive to parameters controlling POI detection. The optimal values of individual parameters is likely to vary based on the algorithm under analysis and measurement setup.

A. Horizontal Attack on RSA

The first test scenario was a horizontal attack on an unprotected implementation of RSA. A 32-bit key was used for the decryption for the sake of simplicity of implementation and better visualization of the results. Although such a key should never be used in practice, it was sufficient for evaluation and demonstration purposes. The encryption was implemented using the square and multiply algorithm, which produces one iteration for each bit of the key. Furthermore, the bodies differ based on the value of that secret bit.

Some care had to be taken when selecting suitable parameter values. The power signatures of individual iterations were quite short in terms of the number samples in each iteration. For this reason, the *Minimal distance between POIs* was set to 1. This choice traded performance for lower risk of discarding important POIs. However, only 103 POIs were discovered in this case, making the performance impact of processing more POIs negligible. The *POI Detector wavelet scale* was set to 4 to emphasize more narrow peaks and the *Iteration matching area* was set to 10 samples to accommodate short iteration bodies.

Our method correctly identified all iteration starts without any prior knowledge of the iteration count and classified the discovered iterations into two groups, see Fig. 8. Each iteration corresponds to one key bit, and each group contains iterations that were produced for the same key bit value. Two key guesses can be acquired using this information by either assuming that the first group contains iterations produced by key bits equal to 0 and the second group contains iterations

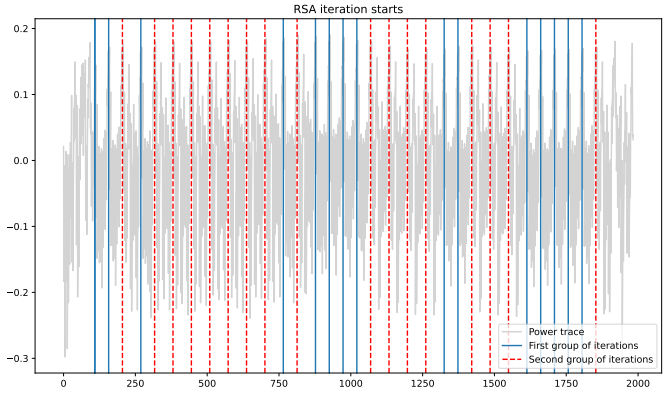

Fig. 8. Iterations detected within an RSA encryption power trace classified into two groups

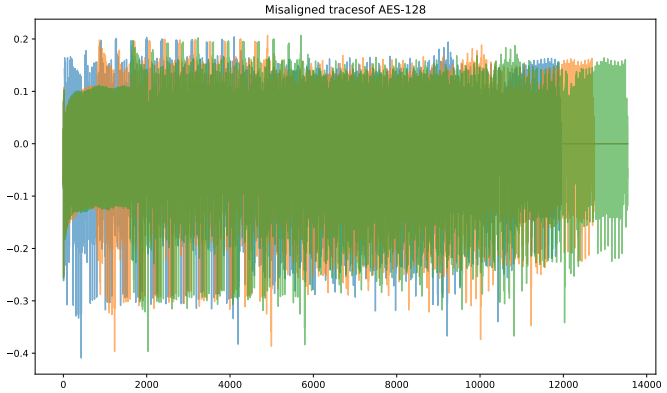

Fig. 9. Measured AES traces

produced for key bits equal to 1, or vice versa. This reduces the search space of possible keys from 2^{key_length} to 2. We successfully recovered the encryption key using this approach.

It is worth emphasizing that this attack required no user interaction, other than providing the information that the trace contained one cycle with two different iteration bodies and setting parameter values. As such, it could be easily scaled to attack much longer keys.

B. Aligning AES Traces

The second experiment was conducted on traces of a software implementation of AES-128, where key expansion is performed in a separate cycle before the encryption rounds. We simulated a scenario where no trigger is available to start measuring traces at the exact same point in execution. For this purpose, three traces were measured and a different number of irrelevant instructions was executed before the encryption for each trace, causing significant misalignment, see Fig. 9. Our implementation of the method requires all traces to have the same length. The traces were therefore padded with zeros to satisfy this requirement.

The power signatures of individual encryption rounds were much longer than those of the RSA encryption discussed in Section VI-A. As a result, many POIs were detected in each

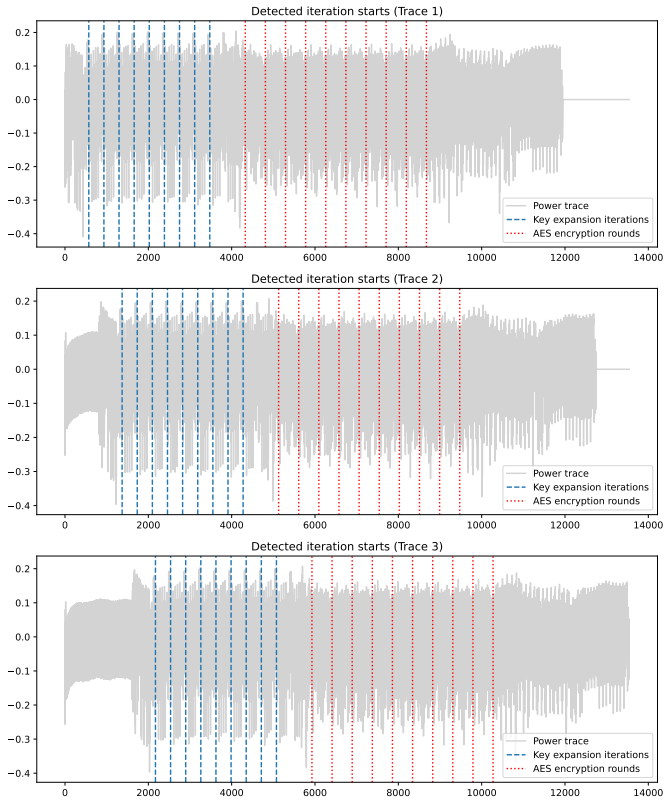

Fig. 10. Iteration starts detected within each AES power trace

iteration. This led us to use a higher value of the *Minimal distance between POIs* parameter, setting it to 200 samples. In comparison with *Minimal distance between POIs* equal to 1, this choice reduced the number of POIs from 177 to 41 and the number of POI groups from 17 to 4, while still preserving enough information to identify individual iteration starts.

Our method correctly determined the number of encryption rounds as well as the start of each round within each trace. The method had to be given the information that two different iteration bodies are to be expected in the encryption cycle because the last round of AES does not contain the MixColumns operation. However, only the first nine out of ten iterations of the key expansion cycle were found. The last iteration was not detected because its power signature was not sufficiently correlated with the previous iterations. The identified iteration starts within each trace can be seen in Fig. 10.

The traces were subsequently truncated to begin at the final encryption round, thereby restoring alignment of the traces, see Fig. 11. The truncated traces were once again padded with zeroes to ensure that they all had the same length. The choice of iteration from which the traces are truncated was arbitrary. It would be possible to align the traces to any chosen encryption round. This setup is useful for vertical attacks on hardware implementations targeting the last round, even without knowledge of whether a 128-, 192, or 256-bit key was used, as the key length determines the number of rounds.

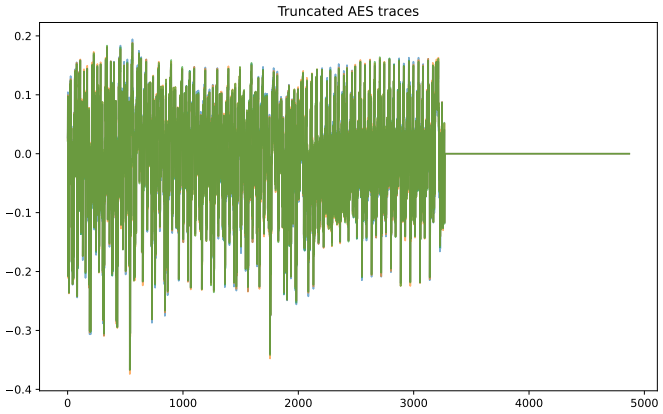

Fig. 11. AES traces truncated to start with the last encryption round

VII. Conclusion

We presented a method for detecting cycles and identifying the start of each iteration in non-nested cycles. The approach can be extended to nested cycles when the boundaries of the outer cycle's iterations are provided. The method is designed to detect cycles within power traces and identify the starts of individual iterations of each cycle without any prior knowledge of the iteration counts. Furthermore, the method only requires one captured power trace, making it suitable for automating horizontal attacks, such as Simple Power Analysis.

The method was tested on traces captured during RSA and AES-128 encryption. It successfully detected all iterations of the RSA encryption, thereby facilitating an automated SPA attack. It correctly detected both the key expansion cycle as well as the individual encryption rounds within the AES-128 traces, but only the first nine out of ten key expansion iterations were identified.

One downside of the proposed approach is the number of parameters used by the method. No general rule for selecting parameter values exists, and finding suitable values needs to be done on a case by case basis. Another weakness is the use of correlation for trace segment classification. On some occasions, even trace segments corresponding to the same iteration bodies were not sufficiently correlated, leading to the method missing some iterations. Finally, the method cannot analyze traces with nested cycles without user interaction. This makes it impractical for use on algorithms with many nested cycles, or deep levels of nesting, for example recursive algorithms.

Future work might include exploring ways to automate parameter value selection, creating more robust approaches to classifying trace segments, and automating the selection of POI groups that correspond to outer cycles. Exploring scenarios with a more powerful attacker, e.g., one that has access to reference traces with known iteration counts, might also lead to more powerful methods.

ACKNOWLEDGMENT

This work was supported by the Student Summer Research Program 2025 of FIT CTU in Prague and was carried out within The Advanced Chip Design and Research Center (ACDRC) project focused on innovation and collaboration in semiconductor research and education between Taiwan and the Czech Republic. This work was also supported by the Czech Technical University (CTU) grant No. SGS26/184/OHK3/3T/18 “Intelligent, safe, and secure embedded systems”.

REFERENCES

[1] Kean Hong Boey, Yingxi Lu, Maire O'Neill, and Roger Woods. Random clock against differential power analysis. In *2010 IEEE Asia Pacific Conference on Circuits and Systems*, pages 756–759, 2010.

[2] 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*, pages 16–29, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[3] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[4] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 252–263, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[5] 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*, pages 426–442, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[6] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[7] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[8] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, chapter Alignment of Power Traces, pages 206–209. Springer New York, 2007.

[9] Ruben A. Muijrers, Jasper G. J. van Woudenberg, and Lejla Batina. Ram: Rapid alignment method. In Emmanuel Prouff, editor, *Smart Card Research and Advanced Applications*, pages 266–282, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[10] 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*, pages 104–119, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.