Springer Journal manuscript No.
(will be inserted by the editor ITNG 2024)

An Unprofiled Single Trace Side-channel Attack for Asymmetric Cryptosystems

Isabella Piacentini[1] · Alessandro Barenghi[1] · Gerardo Pelosi[1] · Ruggero Susella[2]

Abstract Side-channel attacks aim at retrieving cryptographic secrets from a device by exploiting involuntary information channels, such as power consumption or electromagnetic emissions, measured in traces. Among them, *Horizontal attacks* are particularly powerful as they require only a single measurement of an algorithm execution from the target device. In our work, we propose a horizontal attack technique that autonomously detects intermediate value reuses, and extracts cryptographic secret information without needing to analyze or know the implementation of the cryptographic algorithm being run. Our technique, which only assumes the basic a-priori knowledge that the algorithm computes iteratively using a portion of the cryptographic secret in each iteration, both allows successful attacks, and provides the secure software developers with feedback on the vulnerable spots. We validate our attack through a case study on a square-and-multiply-always RSA implementation using the production grade *mbedtls* cryptographic library. Experimental results demonstrate that our approach can retrieve the entire secret RSA exponent from a single execution trace.

Keywords IoT Security, Cryptography, Side channel attacks, Embedded systems security, Applied cryptography

1 Introduction

Side-channel attacks (SCAs) exploit the ability to model the data dependent behaviour of a computing device, allowing attackers to derive the data being processed. Attackers deduce secret values by measuring physical phenomena such as power consumption [13], electromagnetic emissions [1], or execution time [12]. Indeed, an attacker collects one or more measurements of the desired side channel, say, power consumption, while the device under attack is computing a cryptographic algorithm involving a secret value. Such measurements, which take the form of discrete time series (a.k.a. traces) are matched against a set of models for the expected behaviour of the device, each model depending on a small portion of the cryptographic secret. Observing which model is the best fit for the measured device behaviour allows the attacker to deduce the value of the portion of the cryptographic secret. SCAs are traditionally classified according to whether the model of the device is a synthetic one (*unprofiled SCAs*), or derived in a data driven fashion from a similar device (*profiled SCAs*) A second axis for taxonomy is whether an attack exploits information coming from the same time instant in multiple executions (known as a *vertical attack*, from the custom of depicting traces from multiple executions one on top of another, keeping the time axis horizontal), or it exploits information coming from different time instants in a single execution (known as an *horizontal attack* from the same pictorial custom). In this work, we consider horizontal, unprofiled attacks, as they are highly efficient (needing only one trace) and provide explainable attack avenues, which in turn allow a developer to design effective countermeasures.

Contribution. This work introduces a horizontal attack, targeting asymmetric cryptographic primitives. Our method relies on a single execution trace and exploits collisions within the cryptographic process, without requiring prior knowledge of the specific algorithm's implementation or the operations it uses (e.g., the Instruction Set Architecture of the microprocessor involved). This attack is also effective against asymmetric cryptosystems implementations realized with constant-time coding practices, and indeed benefits from constant time code. Our attack strategy builds upon the so-called *collision-based attacks* [4], but allows a significantly weaker attacker, who does not know the specifics of the implementation, to successfully extract the cryptographic secrets. We employ as a case study a square-and-multiply-always algorithm, which is a ubiquitous building

[1]Politecnico di Milano, Piazza Leonardo da Vinci, 32, Milano, 20133, MI, Italy.
[2]STMicroelectronics, Via Camillo Olivetti, 2, Agrate Brianza, 20864, MB, Italy.
*Corresponding author(s). E-mail(s): isabella.piacentini@polimi.it
Contributing authors: alessandro.barenghi@polimi.it; gerardo.pelosi@polimi.it ruggero.susella@st.com

block in asymmetric cryptosystems: RSA encryption, signature, and the Diffie-Hellman key agreement all employ it. Our approach generalizes to any iterative algorithm where value reuse in the computation depends on portions of the secret key. This includes scalar-point multiplications, which are fundamental to elliptic curve-based cryptosystems, and iterative decoders from post-quantum cryptosystems. Our experimental validation uses the open-source *mbedtls* library, which is the TLS library of choice by ARM, targeted to embedded devices and a commercial off-the-shelf microcontroller. In our evaluation, we prove the effectiveness of our approach, and improve on the state-of-the-art methods from [4, 11]. Specifically, our approach enables the extraction of both direct, value-based information (i.e., the private key bit being employed in an iteration) and value-change-based information (i.e., whenever two private key bits differ between two iterations), together with providing a temporal hint on which phases of the computation are leaking information via side channel.

2 Horizontal side channel attacks

In the following, we will consider *horizontal* side-channel attacks, which leverage leakage from a single execution of a cryptographic algorithm [6, 16]. Horizontal side channel attacks exploit features of the trace of a single execution of a cryptographic algorithm to infer the value of the secret key by combining the information contained in multiple samples. The simplest case of horizontal attack employing the device power consumption as the side channel goes by the name of Simple Power Analysis (SPA) [13], and concerns the case of a cryptographic algorithm implementation where the presence of a feature in the trace directly maps to a value of a portion of the cryptographic secret. The root cause of cryptographic implementations being vulnerable to simple power analysis is often the dependence on the secret value by a portion of the control flow of the implemented algorithm. A concrete example is a modular exponentiation procedure performed with a schoolbook square and multiply approach. In this case, at each iteration of the procedure, a (multi-precision) multiplication is executed or not depending on the value of a given single bit of the secret exponent. This in turn translates to an easily detectable increase in power consumption whenever the multiplication is computed, which appears as a distinctive feature of the trace [13].

Mitigating horizontal attacks such as the one exemplified by SPA, has led to the development of unified formulas to perform an iteration of square and multiply exponentiations (and their correspondant on the group of points of an elliptic curve, double and add multiplications). Unified formulas, such as the ones in Chevallier et al.[5] and Giraud et al.[10] employ the same sequence of field operations to compute a correct square and multiply/double and add iteration. While unified formulas consider the removal of distinctive features on an algorithmic level, atomic patterns further detail this idea by providing operative descriptions, i.e., sequences of instructions to be executed in a specific order as to perform the desired computation while avoiding both variations in timing and in the kind of architectural instructions executed at a given time instant. While atomic patterns prevent attackers from leveraging features in the power trace depending on variations in the control flow, or the kind of instructions employed, they still allow to consider the effect of the processed data on the power consumption. This information is leveraged by the so-called *collision* attack strategy. Collision attacks aim to determine if two operations share a common input operand value or not: discerning whether or not two operations, say, in subsequent iterations of the algorithm, share an operand value or not may allow to derive the values being computed.

A first use of the collision technique, focusing on public key implementations, with a limitation on required traces, was reported in [8]. In [2], the authors identified a potential security risk in RSA cryptography through power analysis of the Binary with Initial Points algorithm. They noted that these attacks are especially effective during the algorithm modular multiplication stages. Meanwhile, [9] analyzed collision attacks to evaluate devices susceptible to information leakage, particularly targeting Differential Power Analysis (DPA). Their findings emphasized that collision attacks can uniquely exploit certain leakages, succeeding where other non-profiled attacks may not. In [3], the authors improved horizontal attack techniques, examining masking schemes, which work by splitting a secret into multiple shares to prevent leakage. Lastly, [7] presented a collision attack that investigates the reuse not only of input values but also of outputs. They show a case study on scalar multiplication over elliptic curves, focusing on the reuse of specific computations, and highlighting the need for stronger countermeasures, particularly through noise addition in the computation. In [4], Bauer et al. introduce a theoretical horizontal collision correlation attack on elliptic curve cryptography implementations using point-scalar randomization to achieve constant-time atomic execution. The attack exploits collisions within an a-priori known cryptographic algorithm, where identical internal states arise from different inputs. However, the authors do not demonstrate practical effectiveness, and the attack requires prior knowledge of the algorithm's behavior on the target device, which limits its applicability in real-world scenarios. Building on Bauer et al.'s work, Hanley et al. in [11] improve the attack's success rate by expanding the analysis to compare more than two operations simultaneously. These multiple-operation comparisons, enhance the attack's effectiveness. However, like Bauer et al.'s work, this attack relies on simulated results, lacking real-world validation.

3 Proposed attack methodology

In the following, we detail our horizontal attack methodology, using a square-and-multiply-always algorithm as a

Algorithm 1 Right-to-Left Square-and-Multiply-Always (for RSA)

Input: $b \in \mathbb{Z}_N$; $n = \lceil \log_2(N+1) \rceil$, $e \in \mathbb{Z}^*_{\varphi(n)}$, where $\varphi(N)$ is the Euler Totient function, and $e = (e_{n-1}e_{n-2}\dots e_0)_2$
Output: $res = b^e \mod N$

1: $res \leftarrow 1$
2: $d_0 \leftarrow 1$
3: $d_1 \leftarrow 1$
4: **for** $i \leftarrow 0$ to $n-1$ **do**
5: $\quad res \leftarrow res^2 \pmod N$
6: $\quad d_0 \leftarrow res \cdot \neg e_i$
7: $\quad d_1 \leftarrow b \cdot res \pmod N$
8: $\quad d_1 \leftarrow d_1 \cdot e_i$
9: $\quad res \leftarrow d_0 + d_1$
10: **end for**
11: **return** res

Algorithm 2 Multiple-precision Multiplication

Input: positive integers x and y having $n+1$ and $t+1$ base b digits, respectively.
Output: the product $x \cdot y = (w[n+t+1]\dots w[1]w[0])_b$ in radix b representation

1: Initialize $w[0\dots(n+t+1)]$ to an array of zeros
2: **for** $i = 0$ **to** $(n+t+1)$ **do**
3: $\quad w[i] \leftarrow 0$
4: **end for**
5: **for** $i = 0$ **to** t **do**
6: $\quad c \leftarrow 0$
7: $\quad$ **for** $j = 0$ **to** n **do**
8: $\quad\quad (uv)_b = w[i+j] + x[j] \cdot y[i] + c$
9: $\quad\quad w[i+j] \leftarrow v, c \leftarrow u$
10: $\quad$ **end for**
11: $\quad w[i+n+1] \leftarrow u$
12: **end for**
13: **return** $(w[n+t+1]\dots w[1]w[0])_b$

running example. We consider a single execution trace T made of n power consumption samples τ; the target algorithm performs λ iterations, each one acting on the state of the computation and a portion of the secret key. We denote the cryptographic secret employed s as a sequence of portions, $s = (s_0, \dots s_{\lambda-1})$ where s_i corresponds to the i-th portion of the secret used in the i-th iteration of the algorithm, $0 \le i \le \lambda - 1$.

We will consider the execution trace T as logically split into λ non overlapping subsequences $T = (T_0, \dots, T_i, \dots, T_{\lambda-1})$, each one corresponding to the power consumption of one of the λ iterations of the algorithm. We denote each one of the λ subsequences as *iteration trace* T_i, where the suffix i is the index of the iteration to which the iteration trace corresponds. Each iteration trace T_i has length $m = \frac{n}{\lambda}$. We will denote each sample of the execution trace as τ_i, $0 \le i \le n-1$, therefore we have $T = (\tau_0, \dots, \tau_{m\lambda-1})$ as $n = m\lambda$, and $T_i = (\tau_{i \cdot m}, \dots, \tau_{(i+1)\cdot m-1})$.

Each iteration trace is further split into d non overlapping subsequences, denoted as *sections*, each one of length $l = \frac{m}{d}$, therefore obtaining smaller sequences of operations to consider for our attack. We denote each section of the iteration trace T_i as $T_{i,j}$, with $0 \le j < d$; $T_{i,j}$ corresponds to the sequence of samples $T_{i,j} = (\tau_{i\cdot m+j\cdot l}, ..., \tau_{i\cdot m+(j+1)\cdot l-1})$. The rationale of this split is that a single section should contain a small number of instructions, which may be expected to act or not on the same information across iterations.

Our running example, the square-and-multiply-always technique is a constant-time modular exponentiation algorithm that exhibits the iterative execution pattern leveraged in our attack strategy. We consider the case where the technique is used to compute a modular exponentiation modulo N, where the exponent $e \in \mathbb{Z}^*_{\varphi(N)}$ is the cryptographic secret, as it is the case in RSA signatures and Diffie-Hellman key agreements.

Algorithm 1 provides a description of the square-and-multiply-always procedure, which computes $b^e \mod N$, taking as input $b \in \mathbb{Z}_N$ and $e \in \mathbb{Z}^*_{\varphi(N)}$. Algorithm 1 performs a number of iterations equal to the length $n = \lceil \log_2(N+1) \rceil$ of the binary representation of e, and, at each iteration computes the square of the value of res (line 5). After computing the square of res, two intermediate results, d_0 and d_1, are computed. One is null, while the other either holds res or $b \cdot res$, depending on the value of the i-th secret exponent bit e_i. d_0 and d_1 are subsequently added, and stored in res: this has the effect of either storing in res the value $b \cdot res$ (whenever $e_i = 1$) or leaving res unchanged (whenever $e_i = 0$) while performing the same amount of computations, regardless of the value of e_i.

In our proposed approach, we consider the full execution trace T as the sequence of power consumption values recorded during the loop running from lines 4–10, with each iteration trace T_i corresponding to a single loop iteration. We note that each one of the multiplications reported in Algorithm 1 requires a multiple precision multiplication, which is in turn not executable as a single instruction. We, therefore, report the multiple precision multiplication algorithm from [15] (Algorithm 14.12), as it is the one employed by most cryptographic libraries, including the one employed in our experimental evaluation.

Algorithm 2 describes the procedure of a schoolbook multiplication between two integers x and y, expressed as digits in base b, yielding the product $w = xy$. Typical values for the base b are determined by the architectural word size of the computing platform, so that digit-wise operations correspond to architectural operations between registers. Algorithm 2 iterates over the digits of the second factor y, $y[i]$, (lines 5–12) and, for each one of them, iterates over the digits of the first operand x, (lines 7–10), loading them, multiplying them by $y[i]$ (line 8) and accumulating in word $w[i+j]$ of the result the lower digit of the product, while storing the higher in the carry digit c (line 9). Finally, in line 11 is stored the carry-out resulting from the multiplication of the most significant digit of x by $y[i]$.

Our attack strategy focuses on operands remaining the same across two operations, since many architectural instructions will act on the same values. For instance, if the value of x is the same in two multiple precision multiplications, the repeated sequence of `load` instructions in Algorithm 2

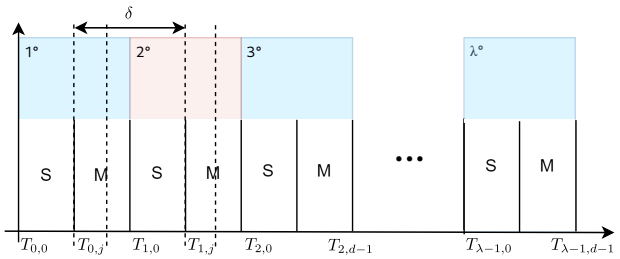

Fig. 1 Computation of a square (S in figure)-and-multiply (M in figure)-always algorithm. Cross-correlation computed between sections, one from the i-th iteration and the other from the $i+1$-th iteration, with λ total iterations. δ depicts the distance between sections of which we compute the cross-correlation. Each iteration of the square-and-multiply-algorithm is m samples long. The entire trace will have length of λm. Subsequent iterations highlighted alternating red and blue.

at line 8 will involve the same values and result in similar switching activity. Power consumption similarities can also be observed when the same value is loaded in one part of the algorithm and stored in another part during subsequent runs. A specific case occurs when the square operation at line 5 in Algorithm 2 produces a result used in the next iteration's square operation in the for loop at lines 4–10. The switching activity from the `store` and `load` operations is expected to result in nearly identical power consumption patterns, as the data transferred to and from memory remains unchanged.

Attack Strategy Our approach measures the similarity between pairs of iteration trace sections from consecutive iterations and extracts information from these comparisons.

To this end, we consider each pair of consecutive iteration traces, T_i and T_{i+1} as two periods of two periodic discrete functions f and g, and compute their cross-correlation $(f \star g)[\delta]$ defined as $(f \star g)[\tau] = \sum_{m=-\infty}^{\infty} \overline{f[m]} g[m+\tau]$, where $\overline{f[m]}$ is the complex conjugate of $f[m]$. The cross-correlation is a concrete measure of similarity between two time series, considered as a function of the time displacement τ of one relative to the other. We will denote as $T_i \star T_j$ the cross-correlation between two generic iteration traces, and as $T_{i,j} \star T_{i',j'}$ the one between two generic sections of iteration traces.

Figure 1 illustrates an execution of the loop in lines 4–10 of the square-and-multiply-always algorithm (Algorithm 1), assuming an exponent of λ bits. We analyze the iteration traces by pairing the ones in consecutive iterations, which are highlighted in alternating colors. Thus, each section of an iteration trace corresponds to part of one iteration and a portion of the next, separated by a time distance δ. The two j-th sections, $T_{0,j}$ and $T_{1,j}$, within the first two iteration traces T_0 and T_1 are indicated by dashed lines. Our goal is to examine the similarity between sections of different iteration traces, as variations in similarity are likely linked to the secret exponent bit e_i used in the i-th iteration. We note that high similarities can occur between sections separated by a time distance δ shorter than a complete iteration trace. For instance, in the case of two subsequent square-and-multiply-always iterations, where the first iteration is employing a null exponent bit, we expect that the section(s) containing the power consumption from the `store` operations of the result of the square operation will exhibit high similarity to the sections where the second iteration performs `loads` of the same value, as the same value is involved in both operation sequences. The latter sections will be closer to the beginning of their own iteration trace than the former sections will be to the beginning of the first iteration trace, resulting in a value of δ smaller than the length of a single iteration trace m.

We summarize the information provided by cross correlations in the similarity matrix $\mathbf{S}_\delta$, which captures the maximum value of the cross-correlation between any pair of sections of two consecutive iterations having distance δ, where the parameter δ takes values in $\{m, m+l, m+2l, \ldots, m+(d-1)l\}$. Denoting with γ the value $\frac{\delta - m}{l}$, we have that the matrix $\mathbf{S}_\delta$ is defined as follows:

$$\mathbf{S}_\delta = \begin{bmatrix} \max_\tau(T_{0,0} \star T_{1,0+\gamma}) \ldots \max_\tau(T_{(\lambda-2),0} \star T_{(\lambda-1),0+\gamma}) \\ \max_\tau(T_{0,1} \star T_{1,1+\gamma}) \ldots \max_\tau(T_{(\lambda-2),1} \star T_{(\lambda-1),1+\gamma}) \\ \vdots \qquad\qquad \ddots \qquad\qquad \vdots \\ \max_\tau(T_{0,l} \star T_{1,l+\gamma}) \ldots \max_\tau(T_{(\lambda-2),l} \star T_{(\lambda-1),l+\gamma}) \end{bmatrix}$$

We choose to use the maximum value of each cross-correlation $(T_{i,j} \star T_{i+1,j+\gamma})$ across all possible time displacements τ since our division of iteration traces into sections does not rely on the structure of the computed algorithm. Consequently, sequences of instructions with similar power consumption may be unevenly split across sections. Furthermore, the effect of differences in execution times for certain operations appearing despite constant time implementations (e.g., the ones caused by clock jitter), are mitigated by taking the maximum cross-correlation value over τ.

Employing the $\mathbf{S}_\delta$ matrix allows us to generalize on previous collision-based attacks [4, 11]: indeed, they exploit the information coming from a subset of the elements of the $\mathbf{S}_\delta$ matrix, which are hand-picked with the explicit knowledge of the algorithm being executed. The selection criterion of previous collision attacks is finding, in the algorithm being executed, operations that are expected to exhibit similarities, and measure only such similarities.

We consider only values of $\delta \in \{m, m+l, m+2l, \ldots, m+(d-1)l\}$, as such offsets between compared iteration trace sections are already enough to extract a significant amount of information from the single power trace. We note that, albeit at a larger computation cost, extending the set of possible values of δ to $\{m, m+l, m+2l, \ldots, m+(d-1)l, m+dl, m+(d+1)l \ldots m\lambda - l\}$, that is considering also the similarities of sections of non-consecutive iteration traces, may yield additional information. In the following, we restrict to extracting information out of sections $T_{i,j}$ and $T_{i+1,j}$ of contiguous iteration traces, considering only the values $\max_\tau(T_{i,j} \star T_{i+1,j+\delta})$ of $\mathbf{S}_\delta$ where $0 \leq j + \delta \leq d-1$, i.e., the valid values for the offset δ so that the cross-correlation is computed between sections of two subsequent iteration traces.

Finally, we map the information contained in $\mathbf{S}_\delta$ into actual values for the secret used in each algorithm iteration.

Consider the secret value e, divided into portions corresponding to each of the λ iterations. Each portion takes a value from a fixed finite set E. In our example, the portions of e are single bits, $e_i, i \in \{0, \ldots, \lambda - 1\}$, which belong to $E = 0, 1$. The columns of $\mathbf{S}_\delta$ are classified depending on the values of the exponent bit e_i used in computing the iteration trace T_i, which has its sections serving as the first operands in the cross-correlations defining the column. Alternatively, if the secret values per iteration are two, an effective approach is to classify pairs of adjacent columns of $\mathbf{S}_\delta$ according to whether the values of e_i and e_{i+1} match. In this case, while the extracted information doesn't directly reveal the entire secret e, an exhaustive search guessing the value of e_0 is sufficient to derive it.

Both the classification tasks mentioned can be performed with any classifier, from a simple threshold-based discriminator on the values of the columns of $\mathbf{S}_\delta$ to more refined machine learning methods.

4 Experimental validation

In this section, we detail our experimental validation of the proposed attack technique. To this end, we implemented the square-and-multiply-always procedure (Algorithm 1) using the multiple precision arithmetic library from the `mbedtls` open-source cryptographic library [14]. For the sake of representation, we depict the results of the attacks with a 16 bit secret exponent e, noting that our attack scales linearly with the increase in the number of exponent bits and requires only a repetition of this procedure to extract the whole secret. To measure power consumption traces from our implementation, we used a ChipWhisperer 1200 Pro digital sampling kit and ran our code on a 32-bit ARM Cortex-M4 microcontroller. Sampling was synchronized with the target device's clock, capturing one sample per clock cycle, with the victim being clocked with a 7.37 MHz external crystal.

To select the value d, i.e., the number of sections into which each iteration trace should be split, we tested our attack procedure with values of $d \in \{10, \ldots, 50\}$, In our case study, a value of $d = 22$ was sufficient to clearly distinguish the features in the $\mathbf{S}_\delta$ matrix. It should be noted that the optimal value of d may vary in other situations, but is easily fine-tuned, or aggressively overestimated, so that matrix features are evident before proceeding with the attack strategy.

Figure 2 shows the $\mathbf{S}_\delta$ matrices for $\delta = m$, calculated on pairs of execution traces with $\lambda = 4$ bit exponents $e' = (e'_3, e'_2, e'_1, e'_0)$ and $e'' = (e''_3, e''_2, e''_1, e''_0)$. We depict eight distinct exponent pattern differences, from an all-zeros exponent against other patterns to an all-ones exponent against other patterns (the first four bits are reported for each exponent). The difference matrices are color-coded to show correlation values, with brighter colors indicating higher cross-correlation. This depiction highlights which section pairs have higher or lower correlation based on the exponent bit value. By examining two subfigures comparing exponents

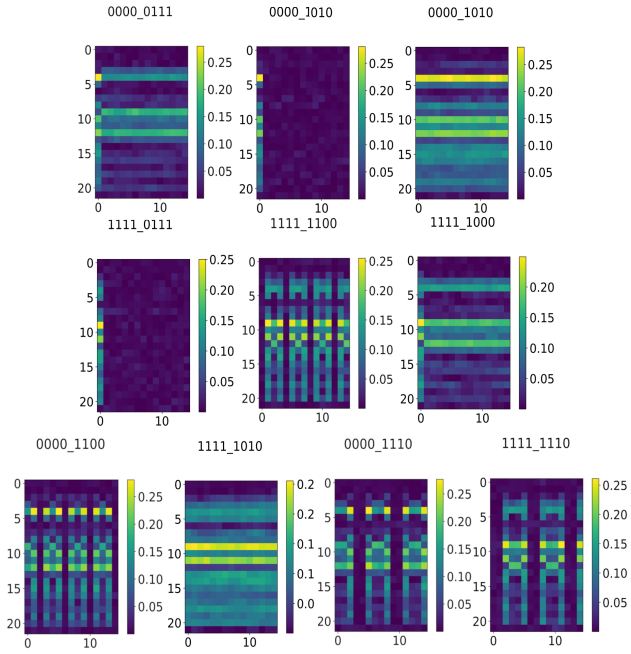

Fig. 2 Differences between pairs of $\mathbf{S}_\delta$ matrices for $\delta = m, \lambda = 4$ and $d = 22$. The pairs of matrices are obtained from execution traces with four bit exponents reported on top of each subfigure of differences between couples of cross-correlation matrices.

with a one-bit difference i.e., $(e', e'') = (0000, 1000)$ (as 0000_1000) and $(e', e'') = (1111, 0111)$ (as 1111_0111 in Figure 2), we note that iterations with identical secret values show no correlation differences with the following iteration (seen as columns in Figure 2). However, when exponent bits differ (e.g., $e'_3 \neq e''_3$), significant differences in correlation (yellow pixels in the first column) appear. These correlation differences occur between different section pairs, depending on whether e'_3 is zero or one, indicating that information about e'_3 is present in the column. Examining the cases $(e', e'') = (0000, 0111)$ and $(e', e'') = (1111, 1000)$ (as 0000_0111 and 1111_1000 in Figure 2), we observe that the $\mathbf{S}_\delta$ values also reveal whether a given exponent bit changes. Specifically, when comparing correlations on the first differing exponent bit (e'_2) with those from subsequent differing bits, we see distinct positions where correlation changes significantly. For example, in $(e', e'') = (0000, 0111)$, the correlation difference in the fifth section (shown as the fifth row of pixels in subfigure 0000_0111) lessens when the exponent bit e''_2 is 1 for the first time, compared to subsequent cases.

Building on these observations, which validate our expectations that $\mathbf{S}_\delta$ values for $\delta = m$ capture information about both the values and transitions of bits in the secret exponent e, we extend our analysis to cases where $\delta \neq m$. We analyzed four specific patterns that represent all possible transitions between exponent bit values in consecutive iterations (00, 01, 10, 11). For this analysis, we considered $\mathbf{S}_\delta$ in the $\lambda = 2$ case and consolidated various $\mathbf{S}_\delta$ matrices into a single column. Figure 3 visually reports the results, organized by the four possible cases. Four different patterns in the correlations

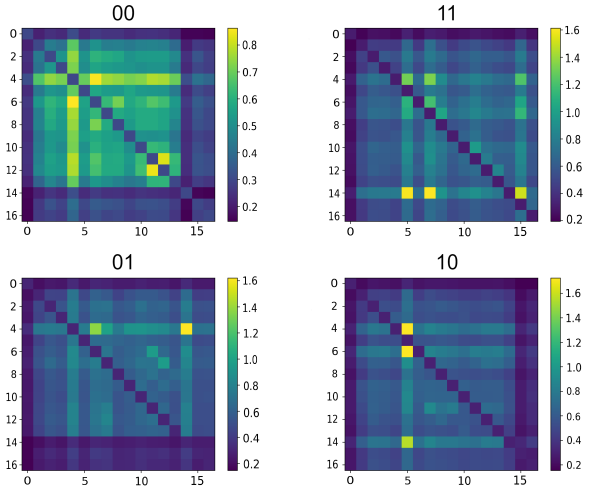

Fig. 3 Matrices of correlation computed between the one iteration and the following one of the square-and-multiply-always, focusing on the multiplication execution. Correlation from cells on the diagonal omitted as to highlighting the results when $\delta \neq 0$. We report four patterns, for each possible transition of the exponent bit in one iteration and the next. The color bars show the color code for the correlation values(from the lowest cross-correlation in blue to the highest in yellow).

emerge depending on both the value and the changing of the exponent, pointing to an explicit information leakage.

Although simple visual inspection, in the fashion of simple power analysis attacks [13] is sufficient to derive either the values of the secret exponent or the transitions across iterations, we completed our attack procedure by mechanizing the exponent value extraction. Due to the significant amount of information present in the $\mathbf{S}_\delta$ matrices, we opted for a simple threshold classifier, which employs the x-y positions of the peaks in cross correlations appearing in the $\mathbf{S}_\delta$ matrix of an iteration to decide to which of the classes the matrix belongs. While the classification logic can be extended to complex patterns, in our case study it is sufficient to distinguish between two fundamental classes of $\mathbf{S}_\delta$ matrices. In the first category, correlation peaks appear symmetrically w.r.t. the main diagonal point to cross-correlations associated with cases where δ equals the section length, i.e. between two instances of the same operation in different iterations. The second category consists of $\mathbf{S}_\delta$ matrices with non-diagonal symmetric peaks, indicating correlations between different operations. Using the described methodology, we retrieved with 100% accuracy the exponent bits for five different exponent patterns by analyzing correlation matrices focused on multiplication execution with variable δ, shown in Figure 3.

5 Conclusion

In this work, we proposed a novel single-trace side-channel attack targeting iterative cryptographic algorithms where the secret value is used piecewise across iterations. Our approach detects similarities between portions of the device's behavior across subsequent iterations, without requiring prior knowledge of the algorithm being executed, and was validated as working on a real world setup with a COTS microcontroller. As a side benefit, our methodology also finds and highlights to the developer the timing at which side channel leakages occur, providing help in designing countermeasures.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side-Channel(s). In: CHES 2002, 4th International Workshop, CA, USA, August 13-15, *LNCS*, vol. 2523, pp. 29–45. Springer (2002)
2. Amiel, F., Feix, B.: On the BRIP Algorithms Security for RSA. In: Second IFIP WG 11.2 International Workshop, WISTP Seville, Spain, May 13-16,, *LNCS*, vol. 5019, pp. 136–149. Springer (2008)
3. Battistello, A., Coron, J., Prouff, E., Zeitoun, R.: Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme. In: CHES 2016 - 18th International Conference, CA, USA, August 17-19, 2016, Proceedings, *LNCS*, vol. 9813, pp. 23–39. Springer (2016)
4. Bauer, A., Jaulmes, É., Prouff, E., Reinhard, J., Wild, J.: Horizontal Collision Correlation Attack on Elliptic Curves. IACR Cryptol. ePrint Arch. p. 321 (2019)
5. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. IEEE Trans. Computers **53**(6), 760–768 (2004)
6. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal Correlation Analysis on Exponentiation. In: 12th International Conference, ICICS Barcelona, Spain, December 15-17, 2010, *LNCS*, vol. 6476, pp. 46–61. Springer (2010)
7. Danger, J., Guilley, S., Hoogvorst, P., Murdica, C., Naccache, D.: Improving the Big Mac Attack on Elliptic Curve Cryptography. In: The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday, *LNCS*, vol. 9100, pp. 374–386. Springer (2016)
8. Fouque, P., Valette, F.: The Doubling Attack - *Why Upwards Is Better than Downwards*. In: CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, *LNCS*, vol. 2779, pp. 269–280. Springer (2003)
9. Gérard, B., Standaert, F.: Unified and optimized linear collision attacks and their application in a non-profiled setting: extended version. pp. 45–58 (2013)
10. Giraud, C., Verneuil, V.: Atomicity Improvement for Elliptic Curve Scalar Multiplication. In: 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 14-16, 2010, *LNCS*, vol. 6035, pp. 80–101. Springer (2010)
11. Hanley, N., Kim, H., Tunstall, M.: Exploiting Collisions in Addition Chain-Based Exponentiation Algorithms Using a Single Trace. In: Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings, *LNCS*, vol. 9048, pp. 431–448. Springer (2015)
12. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: CRYPTO '96, Santa Barbara, California, USA, August 18-22, 1996, Proceedings, *LNCS*, vol. 1109, pp. 104–113. Springer (1996)
13. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
14. Mbed-TLS Contributors: Mbed TLS cryptographic library. `https://github.com/Mbed-TLS/mbedtls` (2024)
15. Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)
16. Walter, C.D.: Sliding Windows Succumbs to Big Mac Attack. In: CHES 2001, Third International Workshop, France, May 14-16, 2001, *LNCS*, vol. 2162, pp. 286–299. Springer (2001)