IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 1, 40 pages.
https://doi.org/10.62056/ah5w7ta5v

Unsupervised Horizontal Attacks against Public-Key Primitives with DCCA

- From Deep Canonical Correlation Analysis to Deep Collision Correlation Attacks -

Dorian Llavata[1,2], Eleonora Cagli[1], Rémi Eyraud[2], Vincent Grosso[2] and Lilian Bossuet[2]

[1] Univ. Grenoble Alpes, F-38000, Grenoble, France, CEA, LETI, MINATEC Campus, F-38054 Grenoble, France.

[2] Univ. Jean Monnet Saint-Etienne, CNRS, Institut d Optique Graduate School, Lab. Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France.

Abstract. In order to protect against side-channel attacks, masking countermeasure is widely considered. Its application on asymmetric cryptographic algorithms, such as RSA implementations, rendered multiple traces aggregation inefficient and led to the development of single trace horizontal attacks. Among these horizontal attacks proposed in the literature, many are based on the use of clustering techniques or statistical distinguishers to identify operand collisions. These attacks can be difficult to implement in practice, as they often require advanced trace pre-processing, including the selection of points of interest, a step that is particularly complex to perform in a non-profiling context. In recent years, numerous studies have shown the effectiveness of deep learning in security evaluation for conducting side-channel attacks. However, few attentions have been given to its application in asymmetric cryptography and horizontal attack scenarios. Additionally, the majority of deep learning attacks tend to focus on profiling attacks, which involve a supervised learning phase. In this paper, we propose a new non-profiling horizontal attack using an unsupervised deep learning method called Deep Canonical Correlation Analysis. In this approach, we propose to use a siamese neural network to maximize the correlation between pairs of modular operation traces through canonical correlation analysis, projecting them into a highly correlated latent space that is more suitable for identifying operand collisions. Several experimental results, on simulated traces and a protected RSA implementation with up-to-date countermeasures, show how our proposal outperformed state-of-the-art attacks despite being simpler to implement. This suggests that the use of deep learning can be impactful for security evaluators, even in a non-profiling context and in a fully unsupervised way.

Keywords: Horizontal Collision Side-Channel Attacks · Single Trace Attacks · Non-Profiling Attacks · RSA · Exponentiation · Unsupervised Deep Learning

1 Introduction

General context. Nowadays, side-channel attacks (SCA) are recognized as a critical attack vector, representing a serious security threat. This type of attack can allow an attacker or security evaluator to compromise the security of a cryptographic device, even if

E-mail: dorian.llavata@cea.fr (Dorian Llavata), eleonora.cagli@cea.fr (Eleonora Cagli), remi.eyraud@univ-st-etienne.fr (Rémi Eyraud), vincent.grosso@univ-st-etienne.fr (Vincent Grosso), lilian.bossuet@univ-st-etienne.fr (Lilian Bossuet)

Received: 2025-01-13 Accepted: 2025-03-11

the embedded cryptographic protocols are robust against theoretical cryptanalysis. Indeed, the implementation of these cryptographic protocols on embedded devices, such as smart cards, can lead to physical leakages during their execution. An in-depth analysis of these side-channel leakages (sometimes called traces), such as power consumption [KJJ99] or electromagnetic emissions [GMO01], can allow a security evaluator to retrieve sensitive information from the implementation.

One of the most powerful types of SCA is the profiling attack (*e.g.* template attacks [CRR03]), involving a scenario in which the evaluator has full control over a clone device, which he can use to perform a fine characterization of the leakages through a supervised manner. If this scenario is not available, the evaluator should consider a non-profiling attack, that is, any attack that does not require a leakage characterization through supervised manner. In SCA, we can also distinguish two attack modus operandi [BJPW13]. One is *vertical attack*, which exploits multiple traces leakage from several executions of the implementation (by varying the inputs) and use statistical tools to aggregate and extract the information [KJJ99, BCO04]. The other is *horizontal attack*, which thoroughly exploits information of a single trace (divided into several parts) from a single execution of the implementation [Wal01].

In RSA-based protocols, modular exponentiation is the most common critical operation targeted by a key recovery side-channel attack. Nowadays, modern implementations of RSA cryptosystems in embedded devices incorporate countermeasures to vertical attacks, thus numerous works have investigated the application of horizontal attacks. These include *horizontal collision attacks* [Wal01, CFG$^+$10, CFG$^+$12] which exploit the leakage from the manipulation of the same operand in two computationally expensive operations (*e.g.* long integer multiplications) as well as *horizontal clustering attacks* [HIM$^+$13, SHKS15, PC15] which exploit the leakage from directly bit-dependent operations (*e.g.* the bit reading itself). This type of attacks, while highly effective, can be extremely complex to implement in practice. Indeed, they require the acquisition of additional internal information, such as detailed knowledge of the implementation of modular multiplication. In addition, horizontal attacks suffer from the same constraints as vertical attacks (*e.g.* noise, dimension, points of interest selection), but to an even greater extent due to the need to find the secret exponent using a single exponentiation trace.

Since the last decade, deep learning techniques have been widely used to perform profiling SCA, proving their effectiveness in both vertical [MPP16, CDP17, BPS$^+$20, ZBHV19] and horizontal contexts [CCC$^+$19, ZBHV21, SKF$^+$22, BCM$^+$22]. More recently, non-profiling vertical attacks using deep learning, known as *Differential Deep Learning Attacks* and *Collision Deep Learning Attacks*, have begun to emerge as a popular research direction [Tim19, DLH$^+$22, SM23]. In these works, even in absence of profiling, deep learning is used in a supervised way and several traces may be used jointly to gain information about the target. These two properties make these works very different from the approach proposed in the present paper. Concerning the horizontal side and the single trace attack context, few studies have investigated the interest of unsupervised deep learning. To the best of our knowledge, only two papers propose the use of unsupervised deep learning to perform horizontal clustering attacks [PCBP21, LHK22]. On balance, both of these works are relevant, however, they may be difficult to implement in realistic attack scenarios, where the acquired traces are highly noisy and extensive, while containing few points of interest. In addition, if there is no leakage from directly bit-dependent operations, these attacks cannot be trivially implemented. Consequently, in most real-world scenarios, only horizontal collision attacks are available.

Contributions. In this paper, we propose a non-profiling horizontal collision attack on a binary Square-and-Multiply Always RSA implementation, using an unsupervised deep learning method called Deep Canonical Correlation Analysis. Our approach, introduces the use of a siamese neural network to extract common features from pairs of modular operation traces. To the best of our knowledge it is the first time that siamese neural network are used for non-profiling horizontal collision attacks.

We take advantage of properties of siamese neural network that maximize the correlation in a latent space, *i.e.* a space of smaller dimension that project important information and try to discard non informative data. In this way, the modular operation traces are put into a latent space where operand collisions are more easily distinguishable, enabling more efficient collision correlation attacks. Our approach allows us to be more robust to noise than classical technique of the literature.

Our theoretical approach is validated thanks to a broad experimental exploration of this approach, in simulation and against real traces measured from a device, to assess the soundness of such a technique in non-profiling contexts and to highlight its advantages and inconvenients. In our experiments, we first provide results on simulated traces, demonstrating the benefit of the proposed attack, the DCCA, in distinguishing the presence of collisions between operands compared to state-of-the-art methods. Our proposed attack simplifies the trace pre-processing and leakage characterization phases, making it possible to attack high-dimensional noisy traces without the need to select Points Of Interest (POIs). Our approach requires that traces are properly cut into sub-traces and realigned. To the best of our knowledge, all unsupervised horizontal attacks in the literature operate under this assumption.

Then, we assess the benefits of our proposed attack from a closer point of view to a real evaluation on the protected RSA implementation introduced in [CCC+19]. While state-of-the-art correlation collision attacks reach an accuracy of 96.21%, our attack reaches an accuracy of 99.33%, reducing significantly the remaining brute-force complexity required by the evaluator to recover the erroneous bits. Notably, our proposed attack enables key recovery with feasible remaining complexity in a realistic scenario for potential adversaries, making it a major security flaw. Thus, we believe that the practicability of our DCCA attack can turn potential vulnerabilities into exploitable ones, raising serious security concerns and the need for dedicated countermeasures against horizontal attacks. Moreover, our DCCA attack yields results close to the supervised one reported in [CCC+19], whereas the one we propose works in a completely unsupervised way. In this way, we considerably reduce the gap between supervised and unsupervised attacks in the context of horizontal collision attacks.

In this paper, we present a case study of an operand collision attack against a binary Square-and-Multiply Always RSA implementation. As our attack uses a single exponentiation trace, classical exponent masking countermeasure is automatically ineffective. However, our attack scheme is inherent to this type of exponentiation and is therefore ineffective against other exponentiation approaches such as the Montgomery Ladder or Windowed exponentiation, which operate on fundamentally different principles that do not present the same operand collision vulnerabilities. Nevertheless, our attack can be effectively extended to elliptic curve cryptography (ECC), as long as the implementation is based on similar algorithms, such as the Double-and-Add Always method. We also believe that the siamese DCCA model approach, which consists in seeking for a highly correlated trace representation that improves collision detection, could be generalized to other cryptographic contexts involving horizontal collision vulnerabilities, although its adaptability may require adjustments to align with the specific properties of the targeted implementation.

Paper organization. The paper is organized as follows. Section 2 provides background about RSA embedded implementations, horizontal attacks, neural networks and deep canonical correlation analysis. Section 3 presents our deep collision correlation attack and explains how to implement it efficiently. Section 4 presents our experiment settings, in particular the datasets used along with the attack baselines and the evaluation metrics considered. Section 5 validates the benefits of our proposed attack through experimental results. Section 6 discuss about results. Section 7 reports a brief investigation of hyperparameterization in order to deduce some general properties for DCCA model design. Finally, Section 8 and Section 9 discuss about applicability to other implementations and future research directions.

2 Preliminaries

2.1 Notations

In this paper, we use the calligraphic letter, *e.g.* $\mathcal{X}$ to denote sets. The uppercase letter X (resp. X_i when an indexation is needed) to denote random variables and the corresponding lowercase letter x (resp. x_i) to denote their realizations. Vectors and matrices are denoted with bold uppercase letters $\boldsymbol{X}$. The SCA traces are considered as realizations of a random vector $\boldsymbol{X} \in \mathbb{R}^d$, where d defines the dimension of each trace. The i-th entry of a vector $\boldsymbol{X}$ (resp. $\boldsymbol{x}$) is denoted by $\boldsymbol{X}[i]$ (resp. $\boldsymbol{x}[i]$). We denote the non-linear mapping implemented by a deep neural network model as $F(\boldsymbol{X}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the vector of learnable parameters of the model (including weights and biases).

Our attack exploits a single trace of the exponentiation, called the *exponentiation trace*, but we chop this trace into sub-traces corresponding to modular operations. We call these sub-traces *modular operation trace*.

2.2 RSA Embedded Implementations

RSA is an asymmetric cryptographic primitive, widely adopted in smart cards and embedded systems. Other algorithms such as DSA, ECDH and ECDSA, based on modular exponentiation or scalar multiplication, are also commonly used. All these algorithms exploit underlying modular operations of long integers. The RSA algorithm, in particular, is based on modular exponentiation of very large integers, typically 1 024, 2 048 or 4 096 bits. Such exponentiation can be easily performed on an ordinary workstation, but is very computationally expensive for smart cards and other embedded devices. From a practical point of view, the RSA exponentiation is subject to many attacks if straightforwardly implemented. The basic implementation of binary RSA exponentiation, called *Square-and-Multiply*, works by sequentially scanning the secret exponent bits and performing a modular square operation followed by conditional modular multiplication which is performed only when the corresponding exponent bit is equal to 1. This type of exponentiation is particularly vulnerable to SCA, which aim to distinguish a difference in behavior on modular operation traces when an exponent bit is a 0 or a 1.

Most of the exponentiation algorithms use a Long Integer Multiplication (LIM) algorithm to perform long integer multiplication by repeatedly calling an internal multiplier operating on t-bit words. The *Schoolbook* method (depicted in Algorithm 2) represents the most straightforward way to realize a long integer multiplication. Let x and y be long integers of the same size and their respective decompositions are $x = (x[l-1], x[l-2], \cdots, x[0])_b$ and $y = (y[l-1], y[l-2], \cdots, y[0])_b$ in base $b = 2^t$ with $l = \lceil \log_b(x) \rceil$. The product $x \times y$ is performed by the algorithm with a nested-loop structure and performs l^2 inner-products on t-bit integers $x[i] \times y[j]$. Thus, this algorithm gives l^2 intermediate results on $2t$-bits. Other more advanced LIM algorithms may also be used such as Comba [Com90], Karatsuba [Kar63], and Montgomery multiplication [Mon85]. For a detailed analysis of the different LIM, the reader may refer to [RMH17].

Nowadays, modern implementations of RSA incorporate some countermeasures to prevent SCA, that act at two different levels. The masking of the modulus, of the

Algorithm 1: Square-and-Multiply Always

```
Input: $m, n \in \mathbb{N}, m < n$,
    $d = (d[k-1], d[k-2], \cdots, d[0])_2$
Output: $m^d \mod n$
1  $R_0 \leftarrow 1$
2  $R_1 \leftarrow 1$
3  for $i = k-1$ to $0$ do
       // Square $R_1 \times R_1$
4      $R_1 \leftarrow \text{LIM}(R_1, R_1) \mod n$
       // Multiply $R_1 \times m$
5      $R_{d_i} \leftarrow \text{LIM}(R_1, m) \mod n$
6  end
7  return $R_1$
```

Algorithm 2: Schoolbook Long Integer Multiplication (LIM)

```
Input: $x, y$
    $x = (x[l-1], x[l-2], \cdots, x[0])_b$
    $y = (y[l-1], y[l-2], \cdots, y[0])_b$
Output: $x \times y = w$
    $= (w[2l-1], w[2l-2], \cdots, w[0])_b$
1  $w \leftarrow (00 \cdots 0)$
2  for $i = 0$ to $l-1$ do
3      $c \leftarrow 0$
4      for $j = 0$ to $l-1$ do
5          $(uv)_b \leftarrow w[i+j] + x[i] \times y[j] + c$
6          $w[i+j] \leftarrow v$
7          $c \leftarrow u$
8      end
9      $w[i+l] \leftarrow c$
10 end
11 return $w$
```

message and of the exponents [Cor99] acts directly on sensitive data and has as effect the randomization of the intermediate values handled during the exponentiation. In this way, it is effective against multiple trace attacks. Countermeasures that act at the operation level consist in the choice of regular and/or atomic exponentiation algorithms, and aim at protecting the implementation against single trace attacks. Common example of this type of regular exponentiation with constant operation sequences is the *Square-and-Multiply Always* with the addition of dummy multiplications (depicted in Algorithm 1) [Cor99]. This exponentiation always performs squaring and multiplication operations, regardless of bit values. For this purpose, when the current bit is 0 (line 5 of Algorithm 1 when $d_i = 0$), a dummy multiplication is performed and stored in a register dedicated to dummy values (R_0), which does not impact the exponentiation flow stored in the valid register (R_1). This ensures a consistent operation sequence and side-channel leakage. Other algorithms may also be used such as the *Montgomery Ladder* [JY03], or the atomic exponentiation *Multiply Always* which uses the same function for both modular operations [CCJ04].

In the rest of the paper we will consider implementation using masking of the modulus, the message and the exponent as protection for the implementation, the exponentiation use Square-and-Multiply Always method with a same atomic operation for both squaring and multiplication, thus, the result patterns display the same processing time.

2.3 Horizontal Attacks

Horizontal attacks are constrained to exploit a single exponentiation trace, due to exponent masking, which makes the key ephemeral and prevents multi-trace attacks. Hence, horizontal attacks need to extract as much information as possible from the exponentiation trace. To achieve this, most of the horizontal attacks aim to exploit the leakages at modular level rather than at exponentiation level. In other words, the evaluator is looking for finer granularity in the leakage analysis, coming from each t-bit words multiplication of the LIM, in order to obtain much more information from a single exponentiation. One common strategy in horizontal attacks is the search for collisions. This type of attack employs an analogous methodology to the chosen-message collision attack [FV03, HMA+08]. Instead of looking for collisions between two related executions of exponentiation, the horizontal collision attack aims to determine whether or not two distinct modular operations performed at different times during the same execution share common operands. It must be noted that chosen-message collision attacks are not effective in presence of countermeasures such as message randomization, while the horizontal collision attacks can be.

Attack scheme for the Square-and-Multiply Always implementation. This exponentiation (depicted in Algorithm 1) always performs squaring and multiplication operations, regardless of bit values. For this purpose, when the current bit is 0 (line 5 of Algorithm 1 when $d_i = 0$), a dummy multiplication is performed and stored in a register dedicated to dummy values (R_0), which does not impact the exponentiation flow stored in the valid register (R_1). Although this implementation is regular and protects against numerous side-channel attacks, a horizontal collision attack may allow to identify dummy multiplications and infer secret key bits. Indeed, despite the presence of dummy multiplications, the content of $R1$ will not be modified during the multiplication operation when the exponent bit d_i is equal to 0. Thus, for the processing of the next exponent bit d_{i-1}, the algorithm will manipulate the same value of $R1$ during the square operation. A potential collision occurs between one operand of the multiplication operations M_{d_i} and one operand of the consecutive square operations $S_{d_{i-1}}$. By detecting these collisions, the evaluator can infer all bits of the secret exponent except the last one. In the context of a profiling attack, Carbone *et al.* [CCC$^+$19] realized this attack scheme with a deep learning technique against an RSA implementation running on a certified arithmetic co-processor and equipped with full masking countermeasures *i.e.*, masking of the message, masking of the exponent and masking of the modulus, we refer the reader to [CCC$^+$19] for more details.

The limitation of the profiling scenario is that the evaluator must have in his possession a clone under evaluation, which the evaluator must be able to control fully. Indeed, if the device contains an exponent masking countermeasure, the evaluator must be able to turn it off in order to perform his profiling phase. In other words, the evaluator must know/choose the random values used during their executions. This is generally not the case in practice, as these values are generated in a protected manner and are inaccessible (even in many evaluation contexts). Consequently, in most real-world scenarios, only non-profiling horizontal attacks are available.

Non-Profiling Horizontal Attacks. In the non-profiling scenario, Walter presents the first horizontal collision attack with the *Big Mac* attack [Wal01]. This attack performs a signal pre-processing by averaging modular operation traces according to one of the long integer multiplication operands. This pre-processing improves collision detection and makes it easier to distinguish between square and multiplication operations from a single exponentiation trace. Clavier *et al.* [CFG$^+$10] extend this work with the so-called horizontal correlation analysis, which exploits correlations between intermediate values and leakage traces (assuming knowledge of the message). Subsequent works have proposed other horizontal collision attacks [CFG$^+$12, BJPW13, PZS17]. An alternative to collision is the use of clustering techniques, *e.g.* K-means, that have been studied for horizontal attacks. In these studies, the aim is to directly attack the secret key bits by automatically detecting patterns in the modular operation traces [HIM$^+$13, PITM14, SHKS15, PC15]. Recently, iterative deep learning frameworks have emerged to simplify or redress horizontal clustering attacks. Perin *et al.* [PCBP21] proposed an iterative deep learning framework to perform wrong bits correction after an unsupervised horizontal clustering attack on a protected ECC implementation. Subsequently, Lee *et al.* [LHK22] proposed another horizontal clustering attack framework on the same implementation based on one-shot learning and a convolutional siamese neural network for iterative identification of points of interest. On balance, both of these works are relevant, however, they may be difficult to implement in realistic attack scenarios, where the acquired traces are highly noisy and extensive, while containing few points of interest. In addition, if there is no leakage from directly bit-dependent operations, these attacks cannot be trivially implemented. Consequently, in most real-world scenarios, only horizontal collision attacks are available.

Rise of Collision Attacks on Square-and-Multiply Always. In the case of an attack on a Square-and-Multiply Always implementation with dummy multiplications, some works have proposed horizontal collision attacks using the cross-correlation of modular operations. Witteman *et al.* [WvM11] are the first to propose the use of correlation to exploit the collision of operands between two consecutive operations (multiply, square). However, their attack requires several exponentiation traces to work. This attack was generalized to use a single exponentiation trace by Hanley *et al.* [HKT15], and then improved by Sugawara *et al.* [SSS15] with the integration of the Big Mac signal pre-processing which we describe below.

Let $M = R1 \times m$ be a modular multiplication and let $S = R1 \times R1$ be the following modular squaring operation in Algorithm 1. Both include long integer multiplication in the form $X \times Y$ and their partial products $x[i] \times y[j]$ (line 5 of Algorithm 2). The leakage of each $x[i] \times y[j]$ is denoted by $t_{i,j}$. To highlight the left-hand operand and improve collision detection, the modular operation trace is compressed into l-dimensional vectors $\boldsymbol{t_x}$ such that:

$$\boldsymbol{t_x}[i] = \frac{1}{l}\sum_{j=0}^{l-1} t_{i,j}. \quad (1)$$

The averaged trace $\boldsymbol{t_x} = [\boldsymbol{t_x}[0], \ldots, \boldsymbol{t_x}[l-1]]$ is the concatenation of each compressed segments traces. The averaged traces of multiplication and squaring operations denoted by $\boldsymbol{t_x}$ and $\boldsymbol{t'_x}$ are compared with correlation distinguisher (as depicted in Figure 1). The correlation coefficient is assumed to be high if there are collisions.

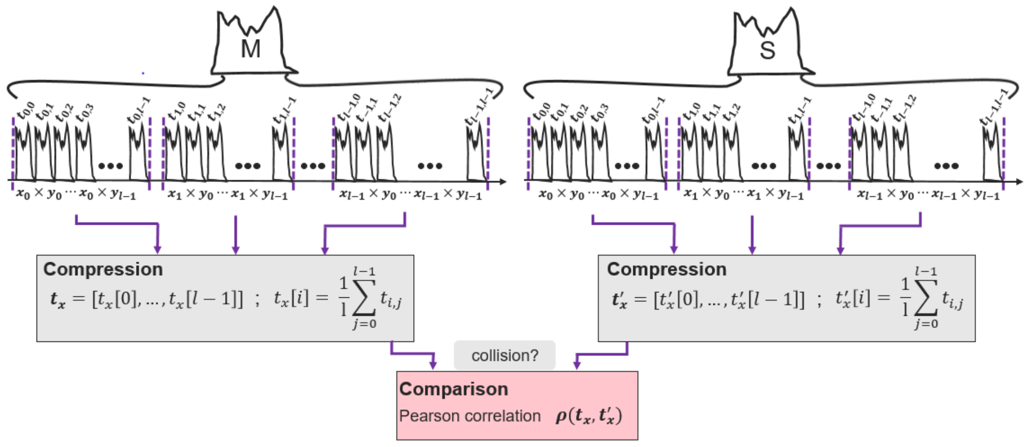

M
S
$t_{0,0}$ $t_{0,1}$ $t_{0,2}$ $t_{0,3}$ $t_{0,l-1}$
$t_{1,0}$ $t_{1,1}$ $t_{1,2}$ $t_{1,l-1}$
$t_{l-1,0}$ $t_{l-1,1}$ $t_{l-1,2}$ $t_{l-1,l-1}$
$x_0 \times y_0 \cdots x_0 \times y_{l-1}$
$x_1 \times y_0 \cdots x_1 \times y_{l-1}$
$x_{l-1} \times y_0 \cdots x_{l-1} \times y_{l-1}$
Compression
$t_x = [t_x[0], ..., t_x[l-1]]$; $t_x[i] = \frac{1}{l}\sum_{j=0}^{l-1} t_{i,j}$
Compression
$t'_x = [t'_x[0], ..., t'_x[l-1]]$; $t'_x[i] = \frac{1}{l}\sum_{j=0}^{l-1} t_{i,j}$
collision?
Comparison
Pearson correlation $\rho(t_x, t'_x)$

Figure 1: Collision correlation attack with Big Mac pre-processing between a multiply modular operation M and its consecutive square modular operation S.

Non-Profiling Horizontal Attacks in Practice. Horizontal attacks are theoretically effective approaches, but their implementation remains a major issue, particularly in non-profiling contexts. They require advanced trace pre-processing, involving careful leakage assessment. Indeed, an accurate extraction of points of interest is necessary, as a high proportion of non-informative points can compromise the success of most of the above-mentioned attacks. In particular, the feasibility of a horizontal collision attack, which exploits leakage information from LIM multiplier, depends heavily on the assumption that the evaluator is able to extract these leakages from a raw trace. For this purpose, the evaluator must have specific informations, such as detailed knowledge of the implementation of modular multiplication. Furthermore, even with this knowledge, performing this analysis in a non-profiling context is challenging, as the leakage assessment techniques traditionally used, such as SNR/TVLA, cannot generally be applied in a non-profiling context.

In the specific case of Square-and-Multiply Always exponentiation scheme (*e.g.*, RSA-1024), a tricky unsupervised approach may be used. In such an exponentiation, there are 1023 pairs of consecutive multiplication and square operations. Roughly half of them are colliding and there are also numerous pairs that we know not to be colliding, those that are not consecutive. Hence, comparing these two sets offers a potential strategy for identifying POIs. For that, an evaluator could compute the time-sample wise absolute difference between the modular operation trace of each multiplication and their respective consecutive squares, denoted as $|M_{d_i} - S_{d_{i-1}}|$, assuming they have the same number of samples. Then, classical leakage assessment tools such as SNR/TVLA may be applied between this set, labelled *e.g.* by 1, and a set of absolute difference of non-consecutive operation modular traces, labelled *e.g.* by 0. Even if roughly a half of the data labelled by 1 is wrongly labelled, if the leakage is strong enough this should give the POIs. However, this strategy is not trivially applicable in a real-world scenario, with low leakage due to noise. Indeed, if leakage is not sufficiently high, the leakage assessment is likely to be strongly impacted by the erroneous pseudo-labeling.[1]

In another way, Sugawara *et al.* [SSS15] performed unsupervised leakage assessment using a correlation matrix obtained by applying Witteman *et al.* collision correlation attack [WvM11]. However, the authors had to disable the masking countermeasure in order to exploit many exponentiation traces. In practice, this requirement is only satisfied if the same co-processor for exponentiation is used for other purposes without exponent masking. For example, in encryption or signature verification operations where no secret is involved. When such a scenario is not feasible, unsupervised selection of points of interest is currently mainly performed by projecting the modular operation traces into a lower-dimensional subspace using linear transformations such as Principal Component Analysis (PCA) [SHKS15] or univariate analysis with heuristic algorithms [PITM14]. Applying these methods to highly dimensional data is also a challenge. As a result, these methods may not be applicable in realistic attack scenarios, as the typical length of a modular operation trace is often very high. In addition, the presence of high noise and the low proportion of points of interest considerably reduces their effectiveness. Consequently, in certain experimental situations, existing horizontal attacks may not be applicable.

2.4 Neural Networks and Siamese Architecture

In the field of deep learning, we can distinguish two kinds of learning tasks: *classification* and *verification*. The first one consists in assigning a label or class to a given observation, while the second one consists in determining whether or not two observations belong to the same class. Let us assume a profiling scenario where an evaluator has acquired a training set of traces such that for each trace $\boldsymbol{X} \in \mathbb{R}^d$ we have the correct associated label $Z \in \mathcal{Z}$ (*e.g.* the value of the target variable handled during the acquisition). These two tasks can be expressed as:

- *Classification scenario*: if the evaluator wishes to classify some side-channel traces $\boldsymbol{x_i}$ in order to associate to each of them a label z_i, classical neural networks, such as feedforward neural networks, are perfectly suited to handle this classification problem [BPS+20].

[1]In this work, we explore a strategy that does not depend on POI selection. Studying the practical implementation of this unsupervised leakage assessment on real devices is left for future works.

- *Verification scenario*: if the evaluator is looking to associate for some pairs of traces $(\boldsymbol{x_i}, \boldsymbol{x_j})$ a label $z_{i,j} \in \{0, 1\}$ that indicates whether or not $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ belong to a same class, a siamese neural network is specifically designed to handle this task. In comparison with classification, which can be extremely complex for an attack scenario involving a large number of classes, verification removes this complexity by focusing on the comparison of two traces to determine whether or not they belong to the same class, making it more scalable for some attack scenarios involving a large number of classes.

Neural Network. The non-linear mapping implemented by a deep neural network model can be represented as a function $F(\boldsymbol{X}, \boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}^{|Z|}$ where $\boldsymbol{X}$ is the trace input, and $\boldsymbol{\theta}$ denotes a set of trainable parameters. These parameters are usually initialized with small random values and are iteratively tuned during the training phase to minimize a defined loss function L based on the desired output Z of the training set. In deep learning, a common optimization method is Stochastic Gradient Descent (SGD) with backpropagation algorithm [GBC16], enabling the network to progressively optimize its parameters and enhance its performance. There are various neural network architectures, the most commonly used to date in SCA are MultiLayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). Notably, CNNs are particularly effective for feature extraction in side-channel contexts, due to their translation invariance property, which is beneficial when dealing with desynchronized traces [CDP17]. For a detailed description of the MLPs and CNNs models and how they work in SCA context, the reader may refer to [BPS+20].

Siamese Neural Network. The architecture of a siamese neural network is specifically designed to compare two inputs [BC93]. It consists in evaluating in parallel two instances of the neural network F by passing it two inputs $\boldsymbol{X_1}$, $\boldsymbol{X_2}$ through two distinct branches. The outputs of these branches represent the latent spaces (sometimes called embeddings) of the two instances, and the role of the network is to extract class discriminative features from these two latent spaces. Indeed, their two outputs $F(\boldsymbol{X_1}, \boldsymbol{\theta})$ and $F(\boldsymbol{X_2}, \boldsymbol{\theta})$ are interpreted as new representations of the two inputs and a distance function D is applied to them. Such a function is assumed to show a somehow small output when applied to representations of two class-colliding inputs, and a somehow large output when applied to representations of two non-class-colliding inputs. The output of the distance function $D(F(\boldsymbol{X_1}, \boldsymbol{\theta}), F(\boldsymbol{X_2}, \boldsymbol{\theta}))$ is used to compute a loss function L that must be minimized during the training phase. The trainable parameters $\boldsymbol{\theta}$ of the two network instances are constrained to be the same, to ensure that both network branches learn similar features in a common latent space. Various distance and loss functions have been proposed in the literature, a commonly used approach being contrastive loss and euclidean distance [CHL05]. For some applications of such models in SCA context, the reader may refer to [MBPK22, LHK22, LLO24]. Modern siamese architectures are often composed of two main blocks: a feature extraction block, typically involving convolutional layers for their strong ability to efficiently capture spatial information, and a projector block, usually consisting of dense layers to project the data into a fully connected latent space. A classification head may potentially be added onto the projector to perform the contrastive task.

Behind Convolution lies Cross-correlation. The convolution operation is very similar to the cross-correlation [LHRB16]. Both slide a kernel window through an input by computing a weighted combination with the kernel values. The two operations mainly differ because the convolution flips the kernel along all spatial dimensions before computing the weighted combination while the cross-correlation does not. Here are the equations for a basic example of cross-correlation $(\star)$ and convolution $(*)$ side-by-side for h-size signal $\boldsymbol{S}$

and kernel $\boldsymbol{K}$:

$$\begin{aligned}\boldsymbol{G} = \boldsymbol{S} \star \boldsymbol{K} : \boldsymbol{G}[i] &= \sum_{u=0}^{h} \boldsymbol{S}[i+u]\boldsymbol{K}[u], \\ \boldsymbol{G} = \boldsymbol{S} * \boldsymbol{K} : \boldsymbol{G}[i] &= \sum_{u=0}^{h} \boldsymbol{S}[i-u]\boldsymbol{K}[u].\end{aligned} \tag{2}$$

When using the well-known convolutional layers in a deep learning model, the kernel weights are trainable parameters learned during the training process. Thus, these layers operations may be interpreted as both convolutions or cross-correlations. Interestingly, the commonly used TensorFlow[2] and Pytorch[3] deep learning libraries are implemented using cross-correlation for efficient implementation. Since the kernel parameters in a siamese CNN model are shared for both inputs, the learning process consists in finding the kernel parameters such that cross-correlation on both inputs minimizes the learning objective. This strong relationship between convolution and cross-correlation makes a siamese CNN a well-suited tool for performing correlation collision attacks, assuming that we can properly control the learning of the kernel parameters. In this work, we propose unsupervised parameter learning using a canonical correlation loss as objective function, as it will be detailed in next section.

2.5 Deep Canonical Correlation Analysis

In this section, we begin with a brief introduction to the canonical correlation analysis (CCA), then we provide details of the deep learning extension known as deep canonical correlation analysis (DCCA). In particular, we provide a detailed description of the canonical correlation loss used to train the DCCA models. Finally, we explain how we use these tools to extract the maximally correlated latent spaces from pairs of modular operation traces.

Basics of Canonical Correlation Analysis. The CCA [Ket71] is a multivariate statistical technique used to project two sets of variables into a common latent space that maximizes their correlation through linear combinations called *canonical variates*. CCA is often used in the field of multi-view learning, which aims to exploit information from multiple data sources (sometimes called views) to improve model performances. In this context, CCA can notably help to combine information from heterogeneous sources (*e.g.* text, audio and video [SSSL20]) by projecting them into a common latent space which is as correlated as possible, in order to obtain a more complete and accurate representation of a given problem.

Let $\boldsymbol{X_1} \in \mathbb{R}^{d_1}$ and $\boldsymbol{X_2} \in \mathbb{R}^{d_2}$ denote random vectors. CCA aims to find pairs of linear combinations from the two vectors, $(\boldsymbol{W_1^*}\boldsymbol{X_1}, \boldsymbol{W_2^*}\boldsymbol{X_2})$, such that the resultant projections are maximally correlated:

$$\begin{aligned}(\boldsymbol{W_1^*}, \boldsymbol{W_2^*}) &= \mathrm{argmax}_{W_1,W_2} \mathrm{Corr}\left(\boldsymbol{W_1}^T\boldsymbol{X_1}, \boldsymbol{W_2}^T\boldsymbol{X_2}\right), \\ &= \mathrm{argmax}_{W_1,W_2} \frac{\boldsymbol{W_1}^T\boldsymbol{\Sigma_{12}}\boldsymbol{W_2}}{\sqrt{\boldsymbol{W_1}^T\boldsymbol{\Sigma_{11}}\boldsymbol{W_1}\boldsymbol{W_2}^T\boldsymbol{\Sigma_{22}}\boldsymbol{W_2}}},\end{aligned} \tag{3}$$

where $(\boldsymbol{\Sigma_{11}}, \boldsymbol{\Sigma_{22}})$ and $\boldsymbol{\Sigma_{12}}$ are the covariance and cross-covariance matrices, and $(\boldsymbol{W_1} \in \mathbb{R}^{d_1}, \boldsymbol{W_2} \in \mathbb{R}^{d_2})$ are the pairs of canonical directions. Several solutions exist to solve the problem. One of the methods proposed by Martin and Maes [MM79] performs

[2] https://www.tensorflow.org/api_docs/python/tf/nn/convolution
[3] https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

singular value decomposition (SVD) on a matrix $\boldsymbol{T} = \boldsymbol{\Sigma_{11}}^{-1/2}\boldsymbol{\Sigma_{12}}\boldsymbol{\Sigma_{22}}^{-1/2}$.

Classical CCA is limited due to its ability to discover only *linear* relationships between two views. To overcome this limitation, various non-linear extensions of CCA have been introduced, including Kernel CCA [LF00] and Deep CCA [AABL13].

Deep Canonical Correlation Analysis. The DCCA, is a non-linear extension of canonical correlation analysis using neural networks [AABL13], which has grown in popularity due to its ability to handle complex non-linear relationships, as opposed to traditional CCA. The use of neural networks provides also a particular flexibility to address the problem of high-dimensional CCA computation, which is an open issue in the literature [SXW$^+$23]. DCCA has been explored in various feature learning applications, including image and text matching [YM15] and cross-modal subspace clustering [GLWS20]. A DCCA model conjointly trains two distinct neural networks with the aim of finding non-linear transformations that maximize the correlation of the two latent spaces. For this purpose, each input is passed through a neural network (encoder) and a canonical correlation analysis is performed on their outputs.

Let $\boldsymbol{X_1} \in \mathbb{R}^{d_1}$ and $\boldsymbol{X_2} \in \mathbb{R}^{d_2}$ denote two random vectors. $\boldsymbol{X_1}$ and $\boldsymbol{X_2}$ are passed as inputs to deep encoders, and we obtain the latent representations $F_1(\boldsymbol{X_1}, \boldsymbol{\theta_1}) \in \mathbb{R}^o$ and $F_2(\boldsymbol{X_2}, \boldsymbol{\theta_2}) \in \mathbb{R}^o$, where o is the output dimension of the encoders and $\boldsymbol{\theta_1}$, $\boldsymbol{\theta_2}$ are their corresponding parameters. The objective of a DCCA architecture is to determine the parameters of the two encoders such that:

$$(\boldsymbol{\theta_1^*}, \boldsymbol{\theta_2^*}) = \operatorname{argmax}_{(\theta_1,\theta_2)} \operatorname{Corr}\Big(F_1(\boldsymbol{X_1}, \boldsymbol{\theta_1}), F_2(\boldsymbol{X_2}, \boldsymbol{\theta_2})\Big). \tag{4}$$

To find $(\boldsymbol{\theta_1^*}, \boldsymbol{\theta_2^*})$, one estimates the correlation objective from the training data according to the Andrew *et al.* calculation method [AABL13]. Let $\boldsymbol{Z_1}$ and $\boldsymbol{Z_2}$ be the representation matrices produced by the two encoders for n training data.

$$\begin{aligned}\boldsymbol{Z_1} &= \Big(F_1(\boldsymbol{x_{1,1}}, \boldsymbol{\theta_1}), \cdots, F_1(\boldsymbol{x_{1,n}}, \boldsymbol{\theta_1})\Big) \in \mathbb{R}^{o\times n}\\ \boldsymbol{Z_2} &= \Big(F_2(\boldsymbol{x_{2,1}}, \boldsymbol{\theta_2}), \cdots, F_2(\boldsymbol{x_{2,n}}, \boldsymbol{\theta_2})\Big) \in \mathbb{R}^{o\times n}\end{aligned} \tag{5}$$

In the seminal work, Andrew *et al.* [AABL13] uses full-batch gradient descent to train the model. In this way n is set to the size of the training set. Subsequent works [JYP17, QLL18, SSSL20] proved that stochastic gradient descent can also be used, whereby n refers to the size of a mini-batch.

Let $(\boldsymbol{\Sigma_{11}}, \boldsymbol{\Sigma_{22}})$ and $\boldsymbol{\Sigma_{12}}$ be the covariance and cross-covariance matrices. When the dimensionality of the features d_1 (or d_2) is high, the covariance matrix $\boldsymbol{\Sigma_{11}}$ (or $\boldsymbol{\Sigma_{22}}$) may be singular, making the optimization problem underdetermined. To address this issue, the covariance matrices can be regularized [DBDM03] and $\boldsymbol{Z_1}$, $\boldsymbol{Z_2}$ are centralized beforehand:

$$\begin{aligned}\boldsymbol{\hat{\Sigma}_{11}} = \frac{1}{n-1}\boldsymbol{\bar{Z}_1}\boldsymbol{\bar{Z}_1}^T + r_1\boldsymbol{I}, \quad \boldsymbol{\hat{\Sigma}_{22}} = \frac{1}{n-1}\boldsymbol{\bar{Z}_2}\boldsymbol{\bar{Z}_2}^T + r_2\boldsymbol{I}, \quad \boldsymbol{\hat{\Sigma}_{12}} = \frac{1}{n-1}\boldsymbol{\bar{Z}_1}\boldsymbol{\bar{Z}_2}^T,\\ \text{with } \boldsymbol{\bar{Z}_1} = \boldsymbol{Z_1} - \frac{1}{n-1}\boldsymbol{Z_1}\boldsymbol{1}, \quad \boldsymbol{\bar{Z}_2} = \boldsymbol{Z_2} - \frac{1}{n-1}\boldsymbol{Z_2}\boldsymbol{1},\end{aligned} \tag{6}$$

where $r_1 > 0$ and $r_2 > 0$ are regularization constants that must be set to relatively small values,[4] $\boldsymbol{I}$ and $\boldsymbol{1}$ are respectively the identity matrix and an all-1 matrix of dimension $n \times n$.

[4]The official implementation of the mvlearn library fixes both constants to $1e^{-3}$.

The total correlation of $\boldsymbol{Z_1}$ and $\boldsymbol{Z_2}$ can be calculated by the sum of the top k singular values of the matrix $\boldsymbol{T} = \hat{\boldsymbol{\Sigma}}_{\boldsymbol{11}}^{-1/2}\hat{\boldsymbol{\Sigma}}_{\boldsymbol{12}}\hat{\boldsymbol{\Sigma}}_{\boldsymbol{22}}^{-1/2}$. If we consider the case of $k = o$, the correlation is exactly the matrix trace norm of $\boldsymbol{T}$:

$$(\boldsymbol{\theta_1^*}, \boldsymbol{\theta_2^*}) = \text{argmax} \sqrt{tr(\boldsymbol{T}^T\boldsymbol{T})}, \tag{7}$$

where tr(.) is the trace function of the matrix.

The final optimization goal for DCCA loss is:

$$L_{DCCA} = -\text{min} \sqrt{tr(\boldsymbol{T}^T\boldsymbol{T})}. \tag{8}$$

In this work, we consider stochastic gradient descent and the backpropagation algorithm. In this way, a random mini-batch of n pairs of training modular operation traces is fed forward to the DCCA model, enabling us to calculate $\boldsymbol{Z_1}$ and $\boldsymbol{Z_2}$ and the correlation matrix $\boldsymbol{T}$, to adjust the encoders parameters according to the aforementioned objective.[5]

3 Deep Collision Correlation Attack

In this section, we first present the attack concept and the underlying intuitions that led to its design. Then, we describe the different implementation steps from a more practical point of view.

3.1 Attack Motivation

Inspired by the profiling deep learning attack on operand manipulation present in [CCC+19], and the design of traditional horizontal collision attacks such as the Big Mac [Wal01], we have developed a deep collision correlation attack performed by an unsupervised neural network. In this approach, we propose to train a DCCA model to maximize the correlation between pairs of modular operation traces through canonical correlation analysis (as depicted in Figure 2), projecting them into a highly correlated latent space that is more suitable for identifying operand collisions. We modified the DCCA approach by using a siamese architecture. This choice results from the fact that in our targeted implementation, squares are not optimized and are performed as multiplications (to avoid trivial simple power analysis). This leads squares and multiplications execution to be enough similar to bring to indistinguishable patterns with the same processing time. Therefore, unlike the traditional method, which aims to combine data of heterogeneous natures (*e.g.* audio/video [SSSL20] or image/text [YM15]) and requires architectures adapted to each type of input, our approach uses a single architecture adapted to both inputs.

The use of a siamese architecture introduces an additional regularization constraint, as the trainable parameters $\boldsymbol{\theta}$ of the two network instances are constrained to be the same. This ensures that both network branches learn similar features in a common latent space, facilitating the correlation objective while providing a commutative function. Note that, in our collision attack, both inputs correspond to the same exponentiation operation, hence the leakage are similar, and the use of the same parameters for both inputs should enhance the latent representation. In this way, the latent representation assigned to a modular operation input trace remains the same, regardless of which network branch it was given as input. Therefore, the correlation between the two latent spaces remains the same regardless of the order of the modular operation input traces. As a result, our siamese DCCA model seeks to determine network parameters such that:

$$\boldsymbol{\theta^*} = \text{argmax}_\theta \, \text{Corr}\Big(F(\boldsymbol{X_1}, \boldsymbol{\theta}), F(\boldsymbol{X_2}, \boldsymbol{\theta})\Big). \tag{9}$$

[5]We provide in Appendix A a TensorFlow implementation of the DCCA loss as well as a skeleton algorithm that instantiates a siamese DCCA model and performs the deep collision attack.

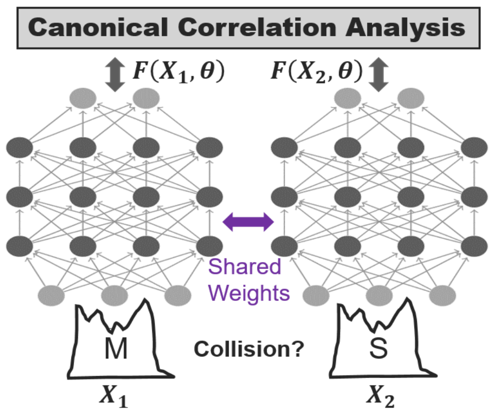


Figure 2: Deep Collision Correlation Attack with siamese DCCA model.

The application of this constraint of weights sharing results in the search for a single transformation for both modular operation input traces that maximizes their correlation. Such a constrained canonical correlation analysis using a single transformation has already been suggested in the literature under the name of *common canonical variates* [NF95].

Our proposal is based on the assumption that the presence of operand collisions should enhance the ability of the DCCA model to maximize the correlation of the two modular operation traces. Consequently, we expect latent spaces to be more strongly correlated if modular operations share some operands. This would make collision correlation attacks more effective. For the sake of clarity, we hereafter refer to the siamese DCCA neural network as the *DCCA model* and to our collision correlation attack performed on the correlated latent space as the *DCCA attack*.

3.2 Attack in Practice

Unsupervised neural networks training can be a difficult process, due mostly to the impact of random weights initialization. In order to address this issue and optimize our DCCA attack, we may need to train the DCCA model several times, using different random weights initializations. This process allows us to explore different starting points in the parameter space, increasing the chances of finding an initial configuration suitable for convergence towards latent representations favorable to the detection of collisions. As a result, the practical implementation of our attack may involve several steps, which we describe below.

Step 1. Correlation on DCCA model latent space. In this work, we consider a collision correlation attack on a regular Square-and-Multiply Always implementation. We follow the attack scheme described in Section 2.3. Therefore, we use a siamese DCCA model to maximize the latent space correlation of all pairs of modular operation traces (one multiplication and one squaring) that potentially contain collisions. Some pairs contain collisions while others do not, but in our unsupervised setting, we do not have access to the class's information during the learning phase. The DCCA loss function does not take any labeling into account.

Once all pairs of latent spaces ($\hat{\boldsymbol{M}} = F(\boldsymbol{X_1}, \boldsymbol{\theta}), \hat{\boldsymbol{S}} = F(\boldsymbol{X_2}, \boldsymbol{\theta}) \in \mathbb{R}^o$) have been obtained, we compute their correlation using the Pearson correlation (obtaining a cloud of correlation values, as it is depicted later in Figure 5).

$$\rho(\hat{\boldsymbol{M}}, \hat{\boldsymbol{S}}) = \frac{\sum_{i=1}^{o} (\boldsymbol{m}[i] - \bar{\boldsymbol{m}})(\boldsymbol{s}[i] - \bar{\boldsymbol{s}})}{\sqrt{\sum_{i=1}^{o} (\boldsymbol{m}[i] - \bar{\boldsymbol{m}})^2} \sqrt{\sum_{i=1}^{o} (\boldsymbol{s}[i] - \bar{\boldsymbol{s}})^2}} \tag{10}$$

Step 2. Collision threshold identification. A threshold must be selected to determine the presence of operand collisions (*i.e.* assign colors in Figure 5). When attacking a regular algorithm with a random secret exponent, it is often assumed that the distribution of null and non-null bits is balanced. In this respect, some approaches have suggested dividing the correlation coefficients into upper and lower halves, using a median [SSS15]. However, even if the bit distribution is balanced, it is very unlikely that the number of 0 and 1 bits are strictly identical,[6] so that using the median would inject errors in the threshold placement. Therefore, to propose a more generalizable solution, we decided to identify the collision/no-collision correlation threshold using a clustering algorithm. We chose to use the K-means algorithm due to its relevance in the image binarization process [GBB06, LY09]. The K-means algorithm aims to form k[7] distinct clusters from n unlabeled data. It starts by choosing k initial centroids and assigns each data point to the nearest centroid, then updates the centroids by recalculating the average distance of the points in each cluster. This process is repeated until convergence is achieved and the objective is to minimize total intra-cluster variance:

$$L_{K-means} = \sum_{j=1}^{k} \sum_{i=1}^{n} \|(x_i^j - c_j)\|^2. \tag{11}$$

Remark. Although the K-means algorithm is effective in determining a threshold to separate correlation coefficients, in some cases, a poor DCCA model weights initialization can lead to overly correlated latent spaces, resulting in a failure of the K-means algorithm to identify a collision threshold. In addition, the presence of outliers (*i.e.* points that differ significantly from the rest of the data) can significantly affect the result. Indeed, a single outlier with a large distance can strongly influence the result, leading to the formation of a cluster around this outlier. To address these issues, the next two steps in our attack may be considered.

Step 3. Rejecting unlikely attacks. Under the assumption that the secret exponent is roughly balanced, a significant cluster imbalance is considered to be an unlikely attack. Therefore, in order to reject all such unlikely attacks, we propose to identify those clusters with non-negligible class imbalance, using the cumulative distribution function (CDF) which gives the probability of a random variable X taking a value less than or equal to a value k.

$$CDF(k, n, p) = P(X \leq k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i} \tag{12}$$

Where k is the value for which we want to calculate the cumulative probability, n is the total number of trials and p is the probability of success in an individual trial.

Based on the CDF analysis, we define a threshold (arbitrarily set to 1^{-32}) beyond which we consider the cluster to be non-negligibly imbalanced. Therefore, if one of the two clusters is below this threshold, we consider the attack unlikely and we reject it.

Step 4. Ranking likely attacks. Given a set of clustering outputs from multiple attacks, we computed an unsupervised cluster validity index to evaluate the best among them. Arbelaitz *et al.* [AGM+13] propose an experimental work that compares 30 cluster validity indices in many different environments, and the so-called *silhouette score* obtained the best results in many of them. The silhouette score [Rou87] is a measure of clustering quality that evaluates how points in the same cluster are similar to each other, and how

[6]See Step 3 for a more formal statistical test.

[7]In our proposed framework, K-means clustering is used to identify a flexible collision/no-collision threshold in the correlation coefficients. Consequently, we set $k = 2$.

different they are from points in other clusters. It assumes that a good clustering solution encompasses compact and well-separated clusters. The silhouette score $S(x)$ computation proceeds by evaluating the individual *silhouette coefficient* $s(x_i)$ for each data point x_i. This coefficient is calculated as the difference between *intra-cluster cohesion* $a(.)$ and *inter-cluster separation* $b(.)$, divided by the higher of these two values.

$$\begin{aligned} S(x) &= \frac{1}{n}\sum_{i=1}^{n} s(x_i), \text{ with } s(x_i) = \frac{b(x_i)-a(x_i)}{\max(a(x_i), b(x_i))}, \\ a(x_i) &= \frac{1}{|C_I|-1}\sum_{j\in C_I, i\neq j} d(x_i,x_j), \ b(x_i) = \min_{J\neq I}\frac{1}{|C_J|}\sum_{j\in C_J} d(x_i,x_j), \end{aligned} \tag{13}$$

where $d(x_i, x_j)$ denotes the distance between points x_i and x_j, $a(x_i)$ is the average distance between point x_i and the other points in the same cluster C_I and $b(x_i)$ is the average distance between point x_i and all points in the nearest cluster C_J, where the nearest cluster is the one that minimizes this distance. For a global evaluation of a clustering, the silhouette score $S(x)$ represents the average $s(x_i)$ coefficients for all n points x_i. An overall silhouette score close to 1 indicates a good clustering quality, while a score close to -1 indicates that the points could be better allocated to a different cluster.

4 Experiments Settings

In this section, we present the experiment settings, including the datasets (simulated and real), model architectures and evaluation metrics we used during our experiments. All the experiments are implemented in Python 3.9 using the Keras 2.8 library with TensorFlow 2.8 backend and are run on a workstation equipped with 32GB RAM and an NVIDIA Quadro P4000 with 8GB memory.

4.1 Datasets and Neural Network Architectures

Simulated traces. We generated simulated exponentiation traces of a Square-and-Multiply Always 2 048-bit exponentiation, implementing Schoolbook LIM with an internal multiplier of 32×32 bits. For each exponentiation trace, we randomly generated the message, modulus and exponent to simulate a fully masked environment. As generally considered in the literature, we assume a linear leakage model with respect to the Hamming weight of the intermediate values manipulated during algorithm execution. For our simulated traces, we deliberately chose to place the leakage only on the LIM output.[8] In this way, each t-bit words multiplication of the LIM generates a single leakage point $HW(x[i] \times y[j])$. The result is a modular operation trace of 4096 time points. The complete exponentiation trace is $2\,048 \times 2 \times 4\,096$ points long (*i.e.* 16 777 216 time points). Additionally, Gaussian noise with mean $\mu = 0$ and standard deviation σ is added to the traces to test the behavior of the attacks in different noise conditions. To get closer to the experimental setup of Clavier *et al.* [CFG+12], we used similar levels of noise (*i.e.* $0 \leq \sigma \leq 7$). In the following experiments, we notably consider $\sigma = 0$ a null noise, $\sigma = 2$ a moderate noise, $\sigma = 4$ a substantial noise and $\sigma = 7$ a strong noise.

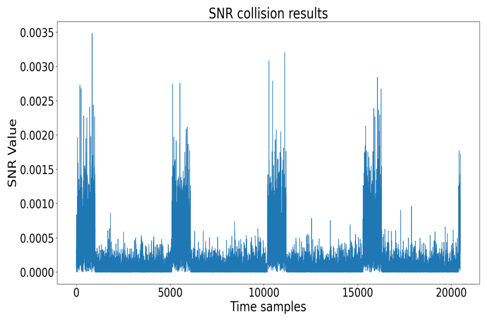
During our experiments, we also added non-informative points within the modular operation traces that will be denoted *PONI*. For this purpose, we used a normal distribution $\mathcal{N}(\mu, \sigma)$ with a mean $\mu = 30$ and a standard deviation σ according to the noise. These parameters have been arbitrarily chosen to ensure that non-informative points blend visually with the rest of the trace, making them indistinguishable from visual inspection.

[8] By removing the leakage from the input operands $HW(x[i])$, $HW(y[j])$, we get a more complex problem, as operand collisions will have to be identified only upon the output. We also experimented with input leakage and found it was a trivial problem.

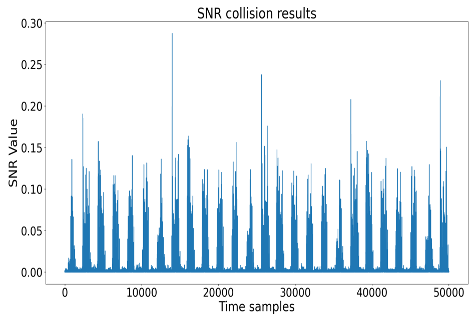
Table 1: Configuration of the addition of non-informative points (PONI) in traces.

Dataset	Modular trace dimension	Nb PONI	% of PONI
Without PONI	4 096	-	-
Moderate PONI	8 192	$4 \times 1\,024$	50
Substantial PONI	12 288	$4 \times 2\,048$	66
Strong PONI	20 480	$4 \times 4\,096$	80

We tested several configurations with moderate, substantial and strong addition of PONI. Table 1 illustrates the different configurations and their impact on trace dimensions. In order to simulate an environment close to the traces of the protected RSA dataset, we added these non-informative points in distinct blocks of the modular operation traces (*i.e.* every 1 000 time samples of the original traces). Figure 3a illustrates the operand collision leakage on the resultant traces with strong PONI configuration and a null noise $\sigma = 0$.



(a) Simulated dataset with Strong PONI and Null noise ($\sigma = 0$).



(b) Protected RSA dataset.

Figure 3: SNRs of operand collision leakage computed with the time-sample wise absolute difference between the modular operation traces of multiplications and their respective consecutive squares, denoted as $|M_{d_i} - S_{d_{i-1}}|$. Here the labelling is correct, obtained with the perfect knowledge of the colliding and non-colliding consecutive modular operations.

Protected RSA dataset. We use the dataset used in [CCC+19] to validate our approach with real traces, the targeted constant time RSA implementation is based on a 1 088 bits binary Square-and-Multiply Always exponentiation algorithm combined with three countermeasures: message randomization, modulus randomization and exponent randomization. Note that in the implementation the multiplication and the squaring are called with the same function. The detailed implementation is given in Appendix B.

For two 512-bit primes p and q, the combination of the three masking countermeasures corresponds to the following equation:

$$m^d \bmod N = \Big((m + k_1 \cdot N)^{d + k_2 \cdot \phi(N)} \bmod (k_0 \cdot N)\Big) \bmod N, \tag{14}$$

where m is the plaintext, d is the private exponent, k_0, k_1, k_2 denote three random positive integers of bit-length 64, $N = p \times q$ the modulus of 1 024 bits and $\phi(N) = (p-1)(q-1)$, is the Euler's totient function. By consequence all the values involved in the exponentiation have size $n = 1\,024 + 64 = 1\,088$ bits.

In addition, the RSA multiplications and squarings are performed with the same operation of the embedded arithmetic co-processor which includes a dedicated memory area based on Montgomery arithmetic. Therefore, both modular operations display similar patterns with the same processing time. For more information on the embedded arithmetic co-processor and the acquisitions campaign, we suggest readers refer to [CCC+19]. The implementation runs on a 0.13um 32-bit contact Smartcard IC and features an ARM core SC 100 with an EAL4+ arithmetic co-processor certified in Asia. The software part of the targeted RSA implementation does not provide specific security mechanisms to defeat horizontal attacks. For more information on the target device, we suggest readers refer to [CCC+19]. For the sake of completeness, the electromagnetic exponentiation traces we used have been cut by modular operation, and each modular operation trace consists of 50 000 points. The modular operation traces have been realigned,[9] and we can clearly identify by SNR computation the leakage of operand collisions during the LIM computations as shown in Figure 3b.

We are aware that this dataset has not yet been published. It was part of a challenge organized by the hardware security laboratory of the French Cybersecurity Agency (ANSSI) for industrial partners. We note that this dataset has already been used in the literature on previous works, presenting supervised deep learning attacks against RSA implementations [CCC+19, ZBHV21]. Therefore, this dataset provides the perfect context to allow a fair comparison with supervised attacks. Obviously, as we do not own the dataset, we are not allowed to publish it. By now, there does not exist a publicly available dataset that would allow us to carry out the operand collision attack, and validate the effectiveness of our works. To overcome this lack of other available datasets, we supported our experimentations with various experiments on simulated traces.

Neural Network models. Two convolutional siamese neural network architectures were used in our experiments.

- The first one used for simulated datasets is a siamese CNN with a feature extractor block consisting of one convolution layer of 4 filters of size 64, in order to take the same window size as the average segment size of the Big Mac pre-processing in the scenario without PONI. This is followed by a pooling layer of size 32 to reduce dimensionality. Finally, the projector uses a dense layer of 400 neurons to obtain the latent space. We used the same architecture for all simulation experiments, even in cases where we added non-informative points.
- The second one used for the experiments on the RSA dataset has been slightly modified. This architecture has been defined thanks to a brief hyperparameterization study reported in Section 7. The convolutional feature extractor uses two convolutional layers. In addition, we experimentally found on real traces that a large convolution window produced overly correlated latent spaces, resulting in an inability to distinguish collisions. To address this issue, we reduced the size of the convolution window to 8. The pooling was removed to avoid losing collision leakage information in the downsizing. Latent space is obtained with a dense layer projector of 1600 neurons.

In our simulation experiments, convolution layers use the ReLU activation function, while for experiments on the protected RSA dataset, we use SeLU activation to avoid potential vanishing and exploding gradient problems [KUMH17]. The dense projector always uses Sigmoid activation function.

The correlation objective is optimized using the RMSProp optimizer, with a learning rate set to 0.0001 and a mini-batch size of 100. This mini-batch size was chosen due to its satisfying performance in works using DCCA models on EEG signals [QLL18, LQZL19].

[9]We also tried our attack on desynchronized traces, nevertheless tests showed unsatisfying results. As it is common in unsupervised horizontal attack literature to assume that the trace realignment and cutting steps have been properly executed, we did not report the results.

The DCCA models were trained during 5 learning epochs for simulation experiments and 20 learning epochs for RSA dataset experiments. To enhances training stability and convergence speed, we pre-processed each dataset such that all trace samples in a batch are normalized between 0 and 1 [IS15].

The hyperparameters optimization is out of the scope of the experimental campaign. An investigation of the hyperparameters, and notably the impact of the convolutional window size on DCCA model performance, is left for future works.

4.2 Evaluation Metrics and Baselines

Evaluation Metrics. Unlike vertical attacks, where the accuracy is not considered to be an appropriate evaluation metric [CDP17], in the case of horizontal attacks, the accuracy is perfectly appropriate due to the constraint of finding all the secret bits of the exponent using a single exponentiation trace. Therefore we use this metric to evaluate the performance of our attacks. In most cases, a horizontal attack does not reveal 100% of the secret bits, and the missing bits must be recovered by some brute-force. The remaining attack complexity is an open problem in the literature, Zaid *et al.* [ZBHV21] experimentally evaluate how the accuracy impacts the final attack complexity. Following the same approach as the authors, we define the brute-force complexity for N remaining operations as $\log_2(N)$. According to the European SOG-IS evaluation scheme,[10] a practical threshold for brute-force complexity is considered to be around 2^{100} operations, thus beyond this threshold, we consider the remaining attack infeasible.

From the point of view of an evaluator who knows the location of each potential erroneous bits, each assumption error has 2 possible values of equal probability. Assuming that the evaluator has recovered K bits of the secret exponent and the total size of the exponent is N bits, the remaining brute-force complexity for the $N - K$ bits can be expressed as:

$$C_{2^n} = \log_2\left(2^{N-K}\right). \tag{15}$$

In the case where the position of the potential erroneous bits is not known to the evaluator (*i.e.* from a point of view closer to an attacker), the evaluator has to try all possible combinations for each wrong assumption, which increases complexity. We call this complexity *naive complexity*, expressed as:

$$C_{NC} = \log_2\left(\sum_{i=0}^{N-K}\binom{N}{i}\right). \tag{16}$$

Helpful post-analysis techniques called key enumeration algorithms [PSG16] may help to reduce this brute-force complexity thanks to a more efficient key enumeration. However, such an approach is not considered in this work.

Baselines. In order to compare our proposed DCCA attack, we perform several collision correlation attack baselines[11] on different trace representations which we summarize below:

- *Raw traces*: collision attack on raw modular operation traces as a fundamental baseline

[10]https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf

[11]We do not compare our results with horizontal clustering methods, as the attack paths are different from collision attacks: for horizontal clustering attacks the exploited vulnerability is the presence of directly bit-dependent operations, often very fast and low consuming (*e.g.* the bit reading itself), while for horizontal collision attacks, the exploited vulnerability is the manipulation of a same operand in two computationally expensive operations (such as long integer multiplications).

- *PCA*: collision attack after PCA dimensionality reduction. We set the number of principal components to the same value as the dimension of the latent space of our DCCA model for fair comparison.
- *Big Mac*: collision attack after Big Mac signal pre-processing. In order to stay in a realistic scenario, we always used the method naively in a totally unsupervised way, *i.e.* without extracting the LIM leakage points beforehand.

As multiplication and squaring operations are performed in the same way and display similar patterns with the same processing time, this makes it possible to apply Big Mac averaging pre-processing. Similarly, this uniformity supports the use of a siamese architecture for our proposed DCCA model.

In the following sections, we present an experimental analysis of our proposed DCCA attack. Notably, in Section 5 we present the results of our experiments on both simulated and real-trace datasets, highlighting the strengths and limitations of our approach. Then, Section 6 addresses the instability issue observed during our experiments due to the random initialization of DCCA model weights, proposing a practical way to handle this limitation. Additional details on the study of our DCCA attack on the real-trace dataset are presented in Section 7, in order to deepen our insights in a real-world context.

5 Experiments Results

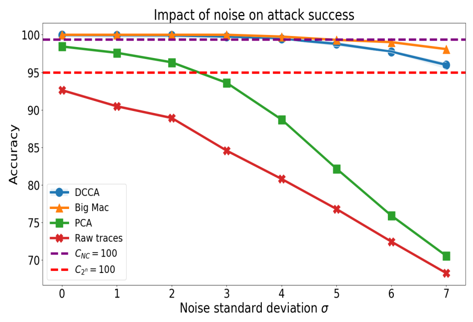
This section proposes an experimental exploration of our Deep Collision Correlation Attack. Notably, in Section 5.1 we evaluate our attack on simulated traces with a progressive addition of Gaussian noise and non-informative points (PONI). Next, in Section 5.2 we apply our DCCA attack on the protected RSA dataset introduced in [CCC+19].

5.1 Results on Simulated Dataset

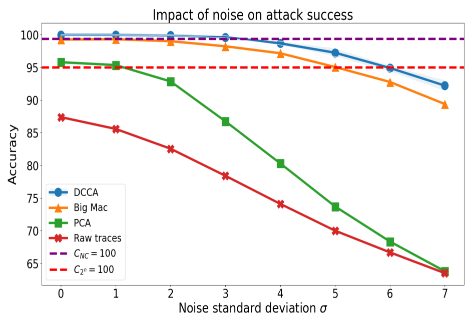
With regard to the simulation experiments, we reduced the experimental scope to the first two steps of the proposed attack. Therefore, we did not attempt to identify the most likely attack. For training our DCCA models, a total of 10 exponentiation traces were generated, 9 traces were used for training the DCCA models (without using label knowledge) and 1 exponentiation trace was kept and used to evaluate the attacks (test traces). In other words, our training set (resp. test set) contains 18 423 (resp. 2 047) pairs of modular operation traces (multiply/square). To take into account the impact of the weights initialization, we trained our DCCA models 10 times on each experiment and we consider the 95% confidence interval of the attack performance. The results of our experiments are summarized in Figure 4.

Impact of noise. First of all, we investigated the impact of noise on the performance of our attacks (Figure 4a). We observe a significant improvement in the performance of our DCCA attack in comparison with collision correlation attacks on raw modular operation traces or those reduced by PCA.[12] Indeed, the DCCA models have successfully put the modular operation traces into highly correlated latent spaces, preserving and amplifying

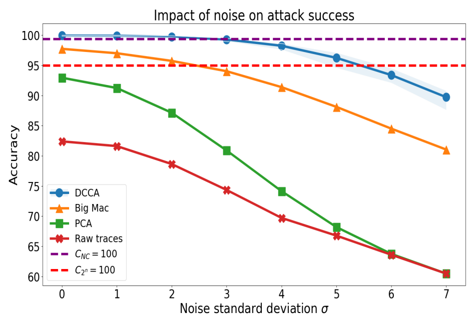
[12]Interestingly, despite all the points in the simulated traces being informative, the dimensionality reduction via PCA still improved the detection of collisions in the correlation coefficients. This improvement may be explained by the fact that, although the points are all informative, they are not entirely independent. For instance, during the multiplication of two integers $x = (x[l-1], x[l-2], \cdots, x[0])_{32}$ and $y = (y[l-1], y[l-2], \cdots, y[0])_{32}$, intermediate operations such as $x[0] \times y[0]$ and $x[0]] \times y[1]$ have dependencies, and these dependencies extend across many operations. Thus, PCA projection successfully extracted the most relevant variance from the leakage of these intermediate operations.



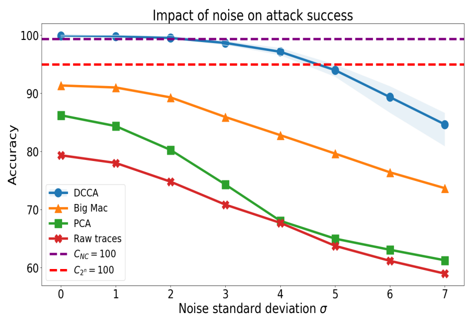
(a) Without PONI.



(b) Moderate PONI.



(c) Substantial PONI.

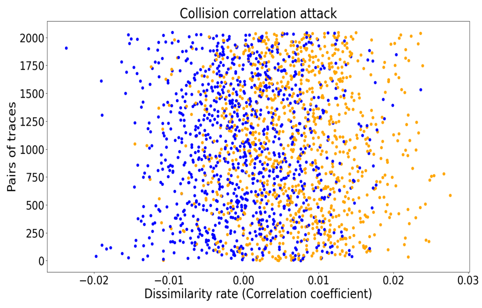


(d) Strong PONI.

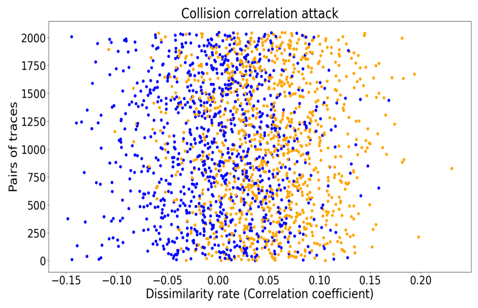
Figure 4: Experiment results on simulated datasets for different levels of noise and proportions of non-informative points. The blue curve represents the average performance of our DCCA attacks over 10 runs, along with the 95% confidence interval.

collisions, even in the presence of significant noise. However, the Big Mac attack was the most effective. It is worth noting that the Big Mac attack excels since the context is favorable for it (*i.e.* the modular operation traces correspond only to the leakage of the multiplier). In addition, pre-processing involving segment averaging in the Big Mac attack reduces the impact of Gaussian noise, while highlighting the targeted operand. The DCCA attack achieves comparable results to the Big Mac attack up to a substantial level of noise ($\sigma = 4$), but is less robust in the case of a strong level of noise ($\sigma = 7$). Despite this, the DCCA attack may result in successful attack for all considered levels of noise (*i.e.* $C_{2^n} \leq 100$). Furthermore, in cases where the Big Mac attack is more effective ($\sigma > 4$), both of these attacks fail to allow a naive remaining attack (*i.e.* $C_{NC} \leq 100$).

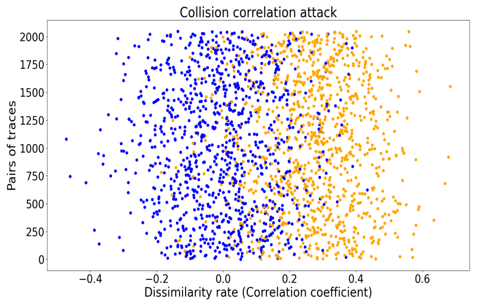
Impact of non-informative points (PONI). Then we analyzed the impact of the addition of non-informative points in the modular operation traces (Figure 4b,Figure 4c,Figure 4d). The DCCA attack proved to be more robust in the presence of non-informative points than the Big Mac attack, used here in a scenario that is no longer optimal for the latter. The Big Mac attack is particularly impacted by a strong level of PONI (Figure 4d), while there is only a slight impact on the DCCA attack up to a substantial level of noise ($\sigma = 4$). The robustness of DCCA attack can be explained by the high ability of CNN to extract features, notably thanks to their sliding window approach on the input traces. We note that in the most extreme scenario, with strong PONI, the DCCA attack is the only one able to allow successful remaining attacks (*i.e.* $C_{2^n} \leq 100$). In addition, the DCCA attack allows remaining naive attacks (*i.e.* $C_{NC} \leq 100$) up to a moderate level of noise ($\sigma = 2$). Figure 5 depicts the distinguishability of collisions in the correlation coefficients for a



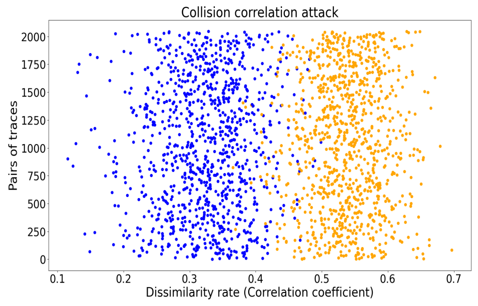
(a) Raw Traces.



(b) PCA.



(c) Big Mac.



(d) DCCA.

Figure 5: Distinguishability of collisions with Strong PONI and Substantial noise ($\sigma = 4$). The correlation coefficients are colored with their ground-truth labels. Blue (resp. orange) values represent pairs of modular operation traces without collision (resp. with collision).

substantial level of noise and a strong level of PONI. We can clearly visualize the benefit of our DCCA attack to distinguish the presence of collisions in comparison to the attack baselines. It is worth noting that the DCCA attack becomes increasingly unstable above a substantial level of noise ($\sigma = 4$). This instability is even more pronounced in the presence of a large number of non-informative points (Figure 4d). This suggests that the DCCA attack could be particularly unstable in a real attack environment. This assumption is confirmed in the next section, where we present our experiments on the protected RSA dataset, thus confronting our approach with conditions closer to a real-world application.

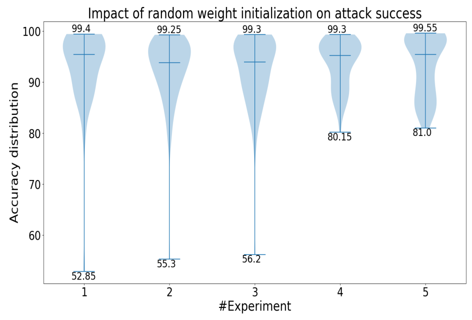
5.2 Results on Protected RSA Dataset

In this section, we propose to apply our DCCA attack on real traces from the protected RSA dataset and assess the gain in remaining attack complexity. For that purpose, we randomly selected 10 000 pairs of modular operation traces (multiply/square) to train the DCCA models (without using label knowledge) and 2 000 pairs of modular operation traces to evaluate the performance of our DCCA attack. This experiment used a single DCCA model architecture, designed using the hyperparameterization campaign reported in Section 7. This experiment was repeated 5 times, and for each repetition, the DCCA models were trained 100 times to assess the impact of random weights initialization.[13]

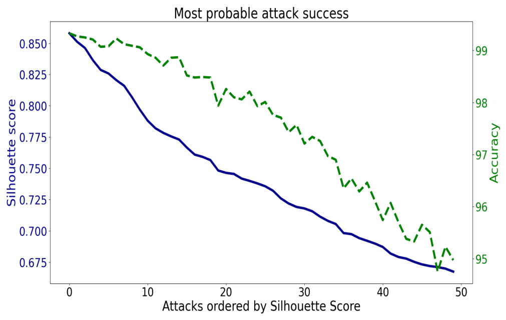
The results of our experiments are depicted in Figure 6a, which represents the accuracy distribution of the 100 DCCA attacks from the 5 experiments. We observed a high degree of instability in our attacks, whose success depended heavily on the initialization of random

[13]Training a DCCA model 100 times takes around 22 hours on our workstation.

weights in the DCCA models. Despite this instability, we observed similar behavior across our 5 experiments, with similar median and maximum attack performance for all experiments. Some variability was noted with regard to the worst attacks, generally due to the presence of outliers resulting from an under-correlated or, conversely, over-correlated pairs of modular operation traces. Using the procedure described in Section 3.2 (steps 3 and 4), those attacks considered unlikely due to an excessive class imbalance were discarded and the remaining attacks were ranked by decreasing likelihood of success according to the silhouette score calculated on the clustering results. Figure 6b depicts the top 50 most likely DCCA attacks with their true accuracy. We can observe the close relationship between silhouette score and true attack accuracy. We observe that the accuracy, although less stable in the rank, follows the same trend as the silhouette score. Therefore, ranking attacks according to silhouette score allows us to approximate in a fully unsupervised way their accuracy and correctly distinguish the most likely successful attacks. Notably, we observe a high confidence for the 5 attacks with the highest silhouette scores.



(a) Accuracy distribution of the 100 DCCA attacks from the 5 experiments.



(b) Top 50 most likely attacks ranking by silhouette score (blue curve) and their true corresponding accuracy (green dotted curve).

Figure 6: Experiment results on protected RSA dataset.

Table 2: Attack evaluation for 1088 bits exponent (average over the 5 experiments). Green (resp. red) values are considered as practicable complexity (resp. unpracticable).

Attack	Acc.	C_{NC}	C_{2^n}
Raw traces	92.31	428	84
PCA	96.21	258	42
Big Mac	58.62	1068	451
DCCA	99.33	68	8

Table 2 summarizes our average 5-experiments and illustrates the benefits of our DCCA attack in comparison with the attack baselines. The average performance of the DCCA attack has been calculated, taking into account for each experiment the DCCA attack identified in an unsupervised way as the most likely. First of all, we note that the dataset is particularly vulnerable to horizontal collision attacks, since a simple collision correlation attack on raw modular operation traces allows us to recover more than 92% of the secret exponent bits and may result in successful attack (*i.e.* $C_{2^n} \leq 100$), assuming the position of the potential erroneous bits is known to the evaluator. This is due to the fact that the dataset does not contain any specific countermeasure for horizontal attacks, and the noise level is relatively low. It is noteworthy that the second best approach on simulation experiments, namely the Big Mac attack, did not work on this dataset despite the absence of countermeasures against horizontal attacks. In fact, Big Mac pre-processing has even led to a loss of information and a drastic drop of performance compared to the collision attack on raw traces. This highlights once again that, despite the applicability of Big Mac pre-processing has been confirmed by simulation experiments, it still needs to be assessed with a real device [CCC+19, SIUH22]. It is worth noting that Big Mac pre-processing relies heavily on the assumption that the evaluator is able to extract LIM multiplier leakage from a raw trace. Indeed, even in our simulation experiments, we observed the strong impact of non-informative points on Big Mac pre-processing. Moreover, this pre-processing is not effective if noise cannot be sufficiently suppressed by the segment averaging. Under these conditions, we experimentally found that collision detection with Big Mac pre-processing was not successful in our real trace environment.

We observe a significant improvement when DCCA attack is used in comparison with attack baselines. While a collision correlation attack performed on a linear dimensionality reduction by PCA reaches 96.21% of accuracy,[14] the DCCA attack reaches 99.33% of accuracy and the remaining brute-force complexity is reduced from 2^{42} to 2^{8} (resp. from 2^{258} to 2^{68}) when the evaluator wants to evaluate a C_{2^n} remaining attack complexity (resp. C_{NC} remaining attack complexity). As a result, the remaining brute-force complexity to be performed by the evaluator is significantly reduced. Notably, our DCCA attack may enables key recovery with naive remaining complexity (*i.e.* $C_{NC} \leq 100$) in a realistic scenario for potential adversaries, making it a major security flaw.

Indeed, according to the criteria established by the French National Security Agency[15] during evaluations carried out as part of the French scheme, if the complexity of completing the attack is equal to, or less than, 2^{70} invocations of the cryptographic algorithm concerned (DES, AES, modular exponentiation, scalar multiplication, etc.), the cost of the effort required to recover the residual information is deemed to be negligible. In this context, our DCCA attack achieves performances that meet these criteria, underlining the seriousness of the threat it poses to the targeted implementation previously considered robust against unsupervised attacks.

Moreover, our DCCA attack yields similar results to the supervised one reported in [CCC+19], whereas the one we propose works in a completely unsupervised way, implying that our current work reduces the gap between supervised and unsupervised attacks.

All our experiments where run on (re)-aligned traces. We tried to apply our methodology to not so well aligned traces, and obtain poor result, especially for real traces. The trace were realigned with classical tool of signal processing. However, it could be an interesting challenge to see if better result could be obtain for DCCA with other network architectures in presence of desynchronization countermeasure, to have a more automatic tool.

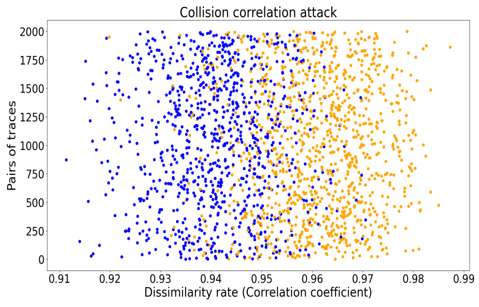
6 Handling Attack Instability

In this section, we discuss the instability of our proposed attack, related to the weights initialization and the unsupervised learning objective of the DCCA model. The impact of weights initialization remains an open issue in deep learning literature [NBS22]. A poor initialization may have a significant impact on the learning process, making convergence more difficult or even impossible. In the context of an unsupervised objective, weights initialization is particularly important since there are no labels to guide the task. Thus, the unsupervised model is expected to discover patterns and representations by itself. Indeed, most unsupervised deep learning approaches require some form of pre-training. Furthermore, the DCCA model is trained to maximize the correlation between pairs of

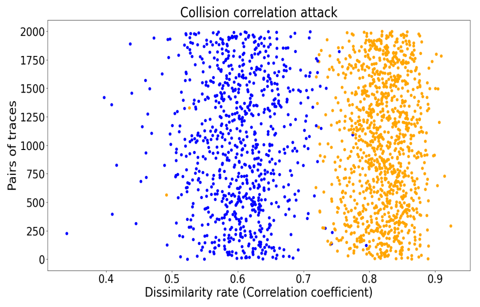
[14] With regard to the PCA attack results, considered the most representative baseline in this experiment, we thoroughly investigated it by varying the number of principal components from 100 to 1600 by steps of 100. As these alternative number of principal components did not yield better performances, we ensured a fair comparison with the DCCA attack results.

[15] `https://cyber.gouv.fr/sites/default/files/2022-08/anssi-cc-note-23-remaining-strength_v1.0[1].pdf`

modular operation traces. Hence, the DCCA model can correlate patterns that do not refer to collisions and successfully solve its learning objective. As a result, an overly correlated latent space may result in an inability to distinguish collisions (as depicted in Figure 7a).



(a) Unsuccessful DCCA attack.



(b) Successful DCCA attack.

Figure 7: Visual interpretation of successful attack. The correlation coefficients are colored with their ground-truth labels. Blue (resp. orange) values represent pairs of modular operation traces without collision (resp. with collision).

With regard to the DCCA model, in the seminal paper Andrew *et al.* [AABL13] initializes the parameters of each layer of the DCCA model with a denoising autoencoder. This instability handling approach based on autoencoder proposed in the DCCA literature is not well suited to the collision SCA attack problem. In particular, training an autoencoder with high-dimensional traces requires very complex architectures with a large number of parameters to be learned. Furthermore, the reconstruction loss does not guarantee the preservation of the collision points of interest.

To handle the instability, we propose to explore different starting points in the parameter space. In this way, we are looking for a weights initialization allowing convergence towards latent representations suitable to the distinguishability of collisions. For this purpose, our unsupervised framework consists of training the DCCA model several times with different weight initalizations, then identifying the best attacks in an unsupervised way, thanks to the silhouette score (which effectively approximates the accuracy one could estimate in a supervised way).

Our experiments proved that the silhouette score is a suitable tool for identifying successful attacks among several runs in a completely unsupervised way, by analyzing the separability of clusters in the correlation coefficients. The reason behind the success of the silhouette score is that, in a one-dimensional context with only two classes, clustering becomes remarkably interpretable and easy to evaluate visually. Indeed, a simple scatter plot can be used to display the correlation coefficients, enabling an intuitive assessment of the effectiveness of the attack. We expect a clear separation between the two clusters for a successful attack (as depicted in Figure 7b).

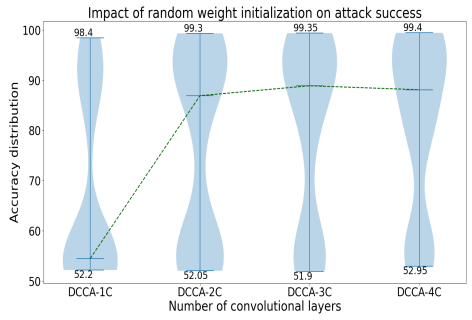
7 DCCA Model Design on Protected RSA Dataset

In this section, we present a brief study of some of the hyperparameters of the siamese DCCA model. The aim here was to deduce some general properties in order to identify and construct a suitable model for our experiments on the protected RSA dataset reported in Section 5.2.

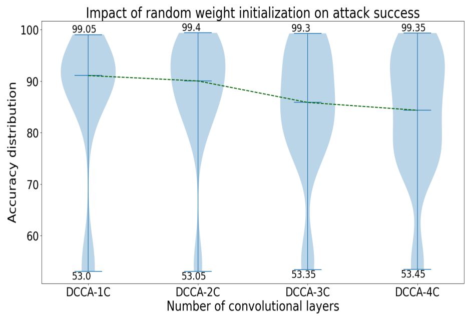
General settings. For this campaign, we randomly selected 10 000 pairs of modular operation traces (multiply/square) to train the DCCA models (without using label knowledge) and 2 000 pairs of modular operation traces to evaluate the performance of our DCCA attack. To enhance training stability and convergence speed, we pre-processed the modular operation traces such that all samples in a batch are normalized between 0 and 1.

In order to analyze the impact of learning time, the models were trained from 5 to 40 epochs. The RMSProp optimizer, with a learning rate of 0.0001 and a mini-batch size of 100, is used to optimize the correlation objective. For all architectures, convolutional layers used SeLU activation function and dense layers used Sigmoid activation function.

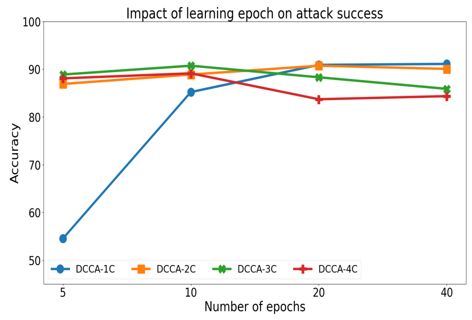
The DCCA models were trained 100 times to assess the impact of random weights initialization. In order to analyze both the success and stability of our attack, we evaluated the median attack performance as well as the performance of the best attack, *i.e.* the most likely one, ranked after performing the unsupervised framework described in Section 3.2 (steps 3 and 4). The latter reflects the actual capability an attacker could have by exploiting the framework in a real attack scenario.



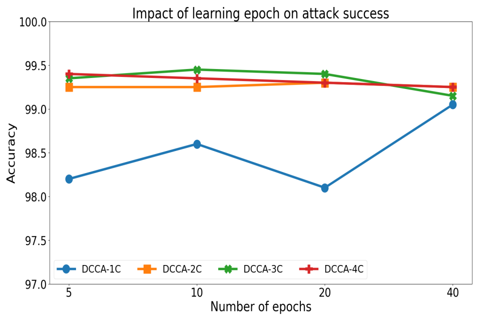
(a) DCCA results for 5 training epochs.



(b) DCCA results for 40 training epochs.



(c) Median of DCCA attacks.



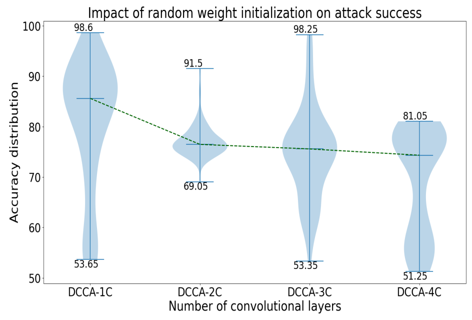
(d) Most likely DCCA attack.

Figure 8: Experiment results on protected RSA dataset for different convolutional feature extractors without downsizing.

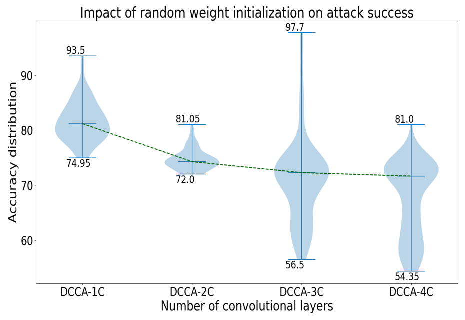
Feature extractor. We first studied the impact of the feature extractor block on the success of the DCCA attack. For this purpose, we experimented with several DCCA models based on a convolutional feature extractor block comprising a variable number of convolutional layers (from 1 to 4) without downsizing (*i.e.* without pooling operations), each with 4 filters and a convolution window of size 8. Following this, the extracted representations are projected into a fully connected latent space in the same way as the simulation experiments, using a dense layer of 400 neurons acting as a projector. This exploration summarized in Figure 8 enabled us to identify the complexity required for the feature extractor to successfully extract fine-grained operand collision leakage.

We observed that using a feature extractor with a large number of convolutional layers does not necessarily offer significant advantages. Although the addition of convolutional layers does enable finer-grained feature extraction, this advantage seems to be limited to the early learning phases, as depicted in Figure 8a. Indeed, during the first learning epochs, a more complex feature extractor may speed up initial convergence thanks to the richness of the representations. However, as learning progresses, increasing the size of the feature extractor may lead to a drop in attack stability, as depicted in Figure 8b, with the median attack performance decreasing as the feature extractor grows.

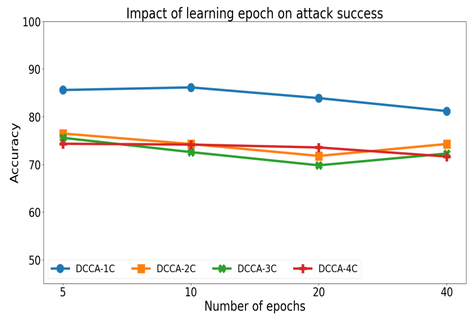
Figure 8c, and Figure 8d provide a detailed overview of these trends, showing respectively the median DCCA attack performance and the most likely DCCA attack identified in the unsupervised way by our framework for different learning stages. First of all, we observed that, except for single convolutional feature extractor (DCCA-1C), extended training of DCCA models had only a slight impact on the performance of the most likely DCCA attacks (depicted in Figure 8d). Then, we observed that simple architectures, such as those with one or two convolutional layers, tended to lead to DCCA models whose performance gradually stabilized during training (depicted in Figure 8c). Conversely, models with three or four layers feature extractor tended towards a gradual drop in stability beyond 10 learning epochs. Indeed, we observed that these more complex models tended to over-correlate latent spaces, preventing collisions detection. As a result, a two convolutional feature extractor trained for 20 epochs, seems to be a good balance between stability and best attack performance. Therefore, we used these settings for our experiments.



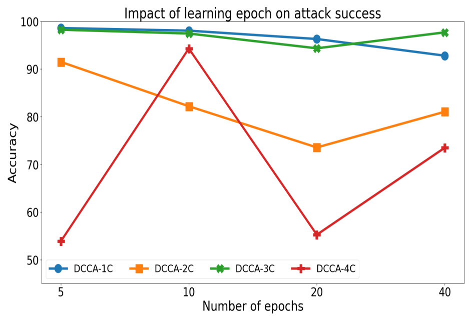
(a) DCCA results for 5 training epochs.



(b) DCCA results for 40 training epochs.



(c) Median of DCCA attacks.



(d) Most likely DCCA attack.

Figure 9: Experiment results on protected RSA dataset for different convolutional feature extractors with downsizing.

Dimensionnality reduction The feature extractor block previously considered only involved convolutional layers, without intermediate pooling layers. In this way, the spatial dimensionality is not compressed throughout the feature extractor, meaning that dimensionality reduction is entirely carried out in the fully connected dense projection layer. Thus, the projector, whose job is to produce the final latent space where the loss function is computed, acts as a genuine bottleneck, compressing drastically the extracted features. This architectural design comes with its own challenges. Although this design preserves spatial granularity up to the projection stage, which in some cases may be advantageous for capturing meaningful local relationships in the traces, the lack of downsizing in the feature extractor may limit the ability of the DCCA model to reduce noise and redundant information. Furthermore, the drastic bottleneck constraint imposed on the dense projector can lead to a significant loss of collision leakage.

In order to reduce the complexity of intermediate representations in a structured way, while at the same time reducing the downsizing task of the projector, we experimentally tried to integrate a progressive dimensionality reduction directly into the feature extractor block. To this end, we added successive averaged pooling layers with a pool size and stride of 2 after each convolutional layer.

The results of these experiments are depicted in Figure 9. We observed significant negative impacts from this downsizing. The overall DCCA attack performance was considerably degraded with increased instability. These results highlight the inherent difficulty of performing effective dimensionality reduction in unsupervised contexts. In comparison to the simulation, where large pooling layers efficiently reduced noise and improved collision detection (in a similar way to the Big Mac pre-processing), performing such a downsizing on real traces remains an open issue. As a result, it is preferable to keep the spatial granularity to ensure that collision leakage is preserved.

Need for translational invariance. The lack of downsizing with pooling operations in the feature extractor limits the DCCA model's ability to learn translational invariant representation [JSZK15, MMD20]. Indeed, pooling operations ensure that small spatial shifts or translations in the input data do not significantly affect the extracted features. In contrast, for a feature extractor without such a downsizing, the translational invariance property is limited, relying solely on convolution operations due to shared weights and kernel shifts. This limitation may explain why some experiments we performed on badly resynchronised datasets[16] showed unsatisfying results, given that translation invariance property is the basis for CNN to handle desynchronization [CDP17, Tim19]. Exploring alternative methods to improve translation invariance, such as strided convolutions [HZRS16] or specific data augmentation techniques [CDP17, Kau18], and determine whether better results could be achieved for DCCA in presence of desynchronization, is left for future tracks.

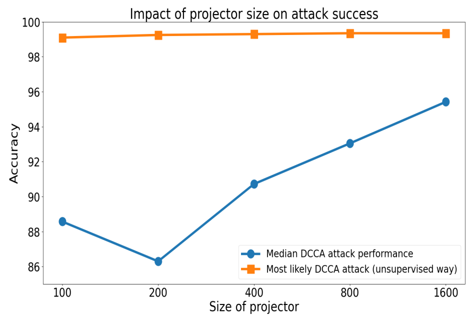


Figure 10: Impact of projector size on the DCCA attack.

[16]Such experiments are not reported in the present paper.

Projector. Since downsizing is performed entirely by the projector, we studied the impact of its size on our DCCA attack. The results of these experiments are depicted in Figure 10. We observed that, although the projector size does not have a direct impact on the performance of the best attacks, it can strongly impact their stability. By choosing a large projector size, we limited the drastic bottleneck constraint and improved the overall performance of the DCCA models.

However, considering a large-scale projector brings some architectural constraints. First of all, it significantly increases the number of model parameters, due to the fully dense connections with the feature extractor output, which is extremely high (*i.e.* 200 000 dimensions in our experiments[17]). This increases the memory and computational costs of training.[18] Furthermore, DCCA loss requires costly operations such as matrix inversions, which cannot be efficiently computed on such a high dimension. As a result, these limitations underline the importance of striking a balance between the feature extractor that expands dimensionality and the preservation of information in the projector to handle the practical constraints imposed by available computing resources. Therefore, we performed our experiments with a projector of 1600 neurons.

General remarks. The results of our experiments validated the relevance of the DCCA model for projecting modular operation traces into a highly correlated latent space, amplifying collisions and significantly improving their detection. However, our approach suffers from some limitations: the model's dependence for proper random weight initialization makes our proposed attack unstable. Furthermore well-aligned traces are still mandatory. These limitations stem from the unsupervised nature of the DCCA loss function, which seeks only to maximize the correlation without any regularization mechanism to control the learning process. Consequently, the DCCA model may produce an overly correlated latent space, compromising the ability to distinguish collisions.

In addition, this issue brings several architectural constraints. Notably, we observed in our experiments that a complex feature extractor may produce an over-correlated latent space, which may increase the instability of the DCCA attack. Beyond this, integrating dimensionality reduction into the DCCA model proved to be a challenging task on real traces. During our experimental campaign, we observed that it is preferable to keep the spatial dimensionality in the feature extractor and only perform the dimensionality reduction at the dense projector level. This architectural constraint may explains the limitation of our approach on desynchronized traces, as the absence of pooling limits the translational invariance properties of the convolutional feature extractor. Furthermore, the dimensionality reduction performed only at the dense projector level acts as a bottleneck that may lead to a loss of collision leakage, especially since it is difficult to consider a high-dimensional projector for computational capacity reasons. This bottleneck constraint may explain the dependence on proper initialization of random DCCA model weights and the instability of our attack.

As a result, to effectively handle this instability issue, the evaluator needs to carefully design the DCCA model according to its computational capacity, striking a balance between the richness of the feature extractor that can explode latent dimensionality and the preservation of essential information within the projector. For our protected RSA dataset, we identified an architecture with a 2-layer convolutional feature extractor, each consisting of 4 filters of size 8 followed by a dense projector with 1600 neurons as a suitable DCCA model.

In order to further handle the instability issue, we also proposed in our framework a weight initialization search strategy, where the attack is run multiple times with different

[17]The input modular operation traces consist of 50 000 points and all convolution layers contain 4 filters, thus the feature extractor always keeps a dimensionality of 200 000 up to its output.

[18]With our computational resources, we could not exceed a projector size of 1600.

starting weights, and the most likely attacks are identified in an unsupervised manner using the silhouette score, a widely used clustering metric in machine learning community. Exploring the integration of a loss function regularization or architectural design improvements to better handle dimensionality reduction within the feature extractor could be an interesting challenge for future tracks, moving towards better attack stability and a more automated approach to deal with desynchronized traces.

8 Applicability to Other Cryptosystems

We presented our analysis on a standard implementation of the RSA algorithm, which mainly involves binary Square-and-Multiply Always exponentiation with a secret exponent. However, as our attack uses a single exponentiation trace and no message or modulus information is required, the methodologies and tools we propose are widely applicable to other implementations. Therefore, RSA-CRT exponentiation and other exponentiation schemes using random exponents, such as DSA and Diffie-Hellman exponentiation, are prone to this attack. It is also worth noting that elliptic curve primitives are as well vulnerable to our attack, as long as operand collisions reveal information about the secret and that the scalar multiplication uses the same type of algorithm (*i.e. Double-and-Add Always*). We also believe that the siamese DCCA model approach, which consists in seeking for a highly correlated trace representation that improves collision detection, could be generalized to other cryptographic contexts involving horizontal collision vulnerabilities, although its adaptability may require adjustments to align with the specific properties of the targeted implementation. With regard to Post Quantum Cryptography (PQC), recent works such as those presented in [GCCD23, BDK$^+$25] involve the detection of operand collisions in computational loops. Hence, we conjecture that the ability to identify and exploit operand collisions of our approach may be exploited in the context of PQC to improve the effectiveness of such attacks. However, the application may not be trivial and could be an interesting direction for future research.

Considering the ability of our attack to affect a wide range of cryptosystems while simplifying the implementation of horizontal attacks, it becomes crucial to consider dedicated countermeasures against such attacks. Classical exponent masking countermeasure in asymmetric implementations prevent multiple exponentiation trace attacks and in most cases make supervised attacks impractical. As our approach is unsupervised and uses a single exponentiation trace, this kind of masking is automatically ineffective. Although various countermeasures against horizontal side-channel attacks aiming to implement masking [ISW03] or shuffling [HOM06] at the modular operation level (*i.e* in the underlying long integer multiplication algorithm) have been proposed in the literature [BJP$^+$15], they are generally not implemented in practice as unsupervised horizontal attacks are often considered too difficult to implement. Indeed, state-of-the-art unsupervised horizontal attacks require in-depth knowledge of the implementation to be effective, especially knowledge about the underlying long integer multiplication algorithm, to accurately select points of interest. Furthermore, even with this knowledge, performing this points of interest selection in a non-profiling context is challenging, as the leakage assessment techniques traditionally used, such as SNR/TVLA, cannot be trivially applied in a non-profiling context. In contrast, our attack is robust even with many non-informative points, allowing us to avoid the need for points of interest selection. Thus, we believe that the practicability of our DCCA attack, making it possible to attack high-dimensional noisy traces,[19] can turn potential vulnerabilities into exploitable ones, raising serious security concerns and the need for dedicated countermeasures against horizontal attacks.

[19]Our attack only requires preliminary cutting of the exponentiation trace into modular operation traces and their realignment. In the unsupervised horizontal attack literature, it is common to assume that these steps have been properly executed.

9 Conclusion

This paper presents a new method for conducting horizontal collision attacks using an unsupervised deep learning method called Deep Canonical Correlation Analysis. Among the thousands of existing deep learning architectures, we propose in this approach to train a siamese DCCA model to maximize the correlation between pairs of modular operation traces, projecting them into a highly correlated latent space that is more suitable for identifying operand collisions. Our proposal is supported by several experimental results on simulated traces and a protected RSA implementation with state-of-the-art vertical attack countermeasures. In particular, we reduce some of the practical problems of horizontal attacks thanks to the flexibility of neural networks, notably the ability to consider large-scale noisy traces with a large number of non-informative points. This property represents a major practical advantage over state-of-the-art non-profiling horizontal collision attacks, which require extensive and complicated trace pre-processing to be effective. To assess the benefits from an evaluation point of view, we evaluate the remaining attack complexity and show a significant complexity reduction with our proposed deep collision attack in a real attack scenario, raising the need to implement dedicated countermeasures against horizontal attacks. Following the SOG-IS and the French National Security Agency security guidances, the improvement in remaining complexity provided by our DCCA attack lead to a reconsideration of the security level of the targeted system.

However, our attack sometimes proved to be unstable, especially when experimenting on real traces. The performance of the attack strongly depends on the random initialization of the model weights. Consequently, to be effective, our attack may need to be performed several times. To address this well-known issue in unsupervised deep learning, we proposed a strategy to identify and rank the most likely attacks based on an unsupervised cluster validity metric called silhouette score. Although our study is based on standard Square-and-Multiply Always RSA implementations, the ideas and tools we propose may also be applied to other collision attack contexts, such as RSA-CRT and elliptic curve algorithms, as long as operand collisions reveals information about the implementation.

Future works. We plan to investigate a way to stabilize our attack in order to limit the impact of the random initialization of the DCCA model weights. For this purpose, we are looking for an approach to modify the canonical correlation loss to take into account the silhouette score during the learning process, in order to ensure latent spaces suitable for collision detection. We also think it would be interesting to adapt the method to target atomic algorithms without dummy multiplications. It could be interesting to adapt our method to desynchronize traces to remove the signal pre-processing step to realign the traces. Finally, we believe that the fusion method of DCCA model could be useful in multi-probe attacks to pre-process the signal captured by different probes during the same operation, which is expected to be highly correlated between the different probes.

Acknowledgements

This work was financially supported by the Defense Innovation Agency (AID) from the french ministry of armed forces. The protected RSA implementation used in this paper was developed by CryptoExperts (https://www.cryptoexperts.com/) as part of a challenge organized by the ANSSI hardware security laboratory for industrial partners.

References

[AABL13] Galen Andrew, Raman Arora, Jeff A. Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *Proceedings of the 30th International Conference on*

Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1247–1255. JMLR.org, 2013. URL: http://proceedings.mlr.press/v28/andrew13.html.

[AGM+13] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M. Pérez, and Iñigo Perona. An extensive comparative study of cluster validity indices. *Pattern Recognit.*, 46(1):243–256, 2013. URL: https://doi.org/10.1016/j.patcog.2012.07.021, doi:10.1016/J.PATCOG.2012.07.021.

[BC93] Pierre Baldi and Yves Chauvin. Neural networks for fingerprint recognition. *Neural Comput.*, 5(3):402–418, 1993. URL: https://doi.org/10.1162/neco.1993.5.3.402, doi:10.1162/NECO.1993.5.3.402.

[BCM+22] Alessandro Barenghi, Diego Carrera, Silvia Mella, Andrea Pace, Gerardo Pelosi, and Ruggero Susella. Profiled side channel attacks against the RSA cryptosystem using neural networks. *J. Inf. Secur. Appl.*, 66:103122, 2022. URL: https://doi.org/10.1016/j.jisa.2022.103122, doi:10.1016/J.JISA.2022.103122.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, Heidelberg, August 2004. doi:10.1007/978-3-540-28632-5_2.

[BDK+25] Sebastian Bitzer, Jeroen Delvaux, Elena Kirshanova, Sebastian Maaßen, Alexander May, and Antonia Wachter-Zeh. How to lose some weight: a practical template syndrome decoding attack. *Designs, Codes and Cryptography*, pages 1–17, 2025. doi:10.1007/s10623-025-01603-1.

[BJP+15] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, Jean-René Reinhard, and Justine Wild. Horizontal collision correlation attack on elliptic curves - - extended version -. *Cryptogr. Commun.*, 7(1):91–119, 2015. URL: https://doi.org/10.1007/s12095-014-0111-8, doi:10.1007/S12095-014-0111-8.

[BJPW13] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and vertical side-channel attacks against secure RSA implementations. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 1–17. Springer, Heidelberg, February / March 2013. doi:10.1007/978-3-642-36095-4_1.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, June 2020. doi:10.1007/s13389-019-00220-8.

[CCC+19] Mathieu Carbone, Vincent Conin, Marie-Angela Cornélie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff, and Alexandre Venelli. Deep learning to evaluate secure RSA implementations. *IACR TCHES*, 2019(2):132–161, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7388. doi:10.13154/tches.v2019.i2.132-161.

[CCJ04] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Trans. Computers*, 53(6):760–768, 2004. doi:10.1109/TC.2004.13.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 45–68. Springer, Heidelberg, September 2017. doi:10.1007/978-3-319-66787-4_3.

[CFG$^+$10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *ICICS 10*, volume 6476 of *LNCS*, pages 46–61. Springer, Heidelberg, December 2010. doi:10.1007/978-3-642-17650-0_5.

[CFG$^+$12] Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylène Roussellet, and Vincent Verneuil. ROSETTA for single trace analysis. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 140–155. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34931-7_9.

[CHL05] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 539–546. IEEE Computer Society, 2005. doi:10.1109/CVPR.2005.202.

[Com90] Paul G. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Syst. J.*, 29(4):526–538, 1990. URL: https://doi.org/10.1147/sj.294.0526, doi:10.1147/SJ.294.0526.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer, Heidelberg, August 1999. doi:10.1007/3-540-48059-5_25.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Heidelberg, August 2003. doi:10.1007/3-540-36400-5_3.

[DBDM03] Tijl De Bie and Bart De Moor. On the regularization of canonical correlation analysis. *Int. Sympos. ICA and BSS*, pages 785–790, 2003.

[DLH$^+$22] Ngoc-Tuan Do, Phu-Cuong Le, Van-Phuc Hoang, Van-Sang Doan, Hoai Giang Nguyen, and Cong-Kha Pham. Mo-dlsca: Deep learning based non-profiled side channel analysis using multi-output neural networks. In *2022 International Conference on Advanced Technologies for Communications (ATC)*, pages 245–250. IEEE, 2022. doi:10.1109/atc55345.2022.9943024.

[FV03] Pierre-Alain Fouque and Frédéric Valette. The doubling attack - *Why Upwards Is Better than Downwards*. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2003. doi:10.1007/978-3-540-45238-6_22.

[GBB06] Ursula Gonzales-Barron and Francis Butler. A comparison of seven thresholding techniques with the K-means clustering algorithm for measurement of bread-crumb features by digital image analysis. *Journal of food engineering*, 74(2):268–278, 2006. doi:10.1016/j.jfoodeng.2005.03.007.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. URL: http://www.deeplearningbook.org/.

[GCCD23] Vincent Grosso, Pierre-Louis Cayrel, Brice Colombier, and Vlad-Florin Dragoi. Punctured syndrome decoding problem - efficient side-channel attacks against classic mceliece. In Elif Bilge Kavun and Michael Pehl, editors, *Constructive Side-Channel Analysis and Secure Design - 14th International Workshop, COSADE 2023, Munich, Germany, April 3-4, 2023, Proceedings*, volume 13979 of *Lecture Notes in Computer Science*, pages 170–192. Springer, 2023. doi:10.1007/978-3-031-29497-6_9.

[GLWS20] Quanxue Gao, Huanhuan Lian, Qianqian Wang, and Gan Sun. Cross-modal subspace clustering via deep canonical correlation analysis. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3938–3945. AAAI Press, 2020. URL: https://doi.org/10.1609/aaai.v34i04.5808, doi:10.1609/AAAI.V34I04.5808.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, Heidelberg, May 2001. doi:10.1007/3-540-44709-1_21.

[HIM+13] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering algorithms for non-profiled single-execution attacks on exponentiations. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 79–93. Springer, 2013. doi:10.1007/978-3-319-08302-5_6.

[HKT15] Neil Hanley, HeeSeok Kim, and Michael Tunstall. Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 431–448. Springer, Heidelberg, April 2015. doi:10.1007/978-3-319-16715-2_23.

[HMA+08] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. Collision-based power analysis of modular exponentiation using chosen-message pairs. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2008. doi:10.1007/978-3-540-85053-3_2.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06*, volume 3989 of *LNCS*, pages 239–252. Springer, Heidelberg, June 2006. doi:10.1007/11767480_16.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi:10.1109/CVPR.2016.90.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_27.

[JSZK15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2017–2025, 2015. URL: https://proceedings.neurips.cc/paper/2015/hash/33ceb07bf4eeb3da587e268d663aba1a-Abstract.html.

[JY03] Marc Joye and Sung-Ming Yen. The Montgomery powering ladder. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer, Heidelberg, August 2003. doi:10.1007/3-540-36400-5_22.

[JYP17] Zhong Ji, Xuejie Yu, and Yanwei Pang. Zero-shot learning with deep canonical correlation analysis. In Jinfeng Yang, Qinghua Hu, Ming-Ming Cheng, Liang Wang, Qingshan Liu, Xiang Bai, and Deyu Meng, editors, *Computer Vision - Second CCF Chinese Conference, CCCV 2017, Tianjin, China, October 11-14, 2017, Proceedings, Part III*, volume 773 of *Communications in Computer and Information Science*, pages 209–219. Springer, 2017. doi:10.1007/978-981-10-7305-2_19.

[Kar63] Anatolii Karatsuba. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, pages 595–596, 1963.

[Kau18] Eric Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. *CoRR*, abs/1801.01450, 2018. URL: http://arxiv.org/abs/1801.01450, arXiv:1801.01450.

[Ket71] Jon R Kettenring. Canonical analysis of several sets of variables. *Biometrika*, 58(3):433–451, 1971.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_25.

[KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 971–980, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/5d44ee6f2c3f71b73125876103c8f6c4-Abstract.html.

[LF00] Pei Ling Lai and Colin Fyfe. Kernel and nonlinear canonical correlation analysis. *Int. J. Neural Syst.*, 10(5):365–377, 2000. doi:10.1142/S012906570000034X.

[LHK22] Nayeon Lee, Seokhie Hong, and Heeseok Kim. Single-trace attack using one-shot learning with siamese network in non-profiled setting. *IEEE Access*, 10:60778–60789, 2022. doi:10.1109/ACCESS.2022.3180742.

[LHRB16] Shan Sung Liew, Mohamed Khalil Hani, Syafeeza Ahmad Radzi, and Rabia Bakhteri. Gender classification: a convolutional neural network approach. *Turkish Journal of Electrical Engineering and Computer Sciences*, 24(3):1248–1264, 2016. doi:10.3906/elk-1311-58.

[LLO24] Di Li, Lang Li, and Yu Ou. Side-channel analysis based on siamese neural network. *J. Supercomput.*, 80(4):4423–4450, 2024. URL: https://doi.org/10.1007/s11227-023-05631-3, doi:10.1007/S11227-023-05631-3.

[LQZL19] Wei Liu, Jie-Lin Qiu, Wei-Long Zheng, and Bao-Liang Lu. Multimodal emotion recognition using deep canonical correlation analysis. *CoRR*, abs/1908.05349, 2019. URL: http://arxiv.org/abs/1908.05349, arXiv:1908.05349.

[LY09] Dongju Liu and Jian Yu. Otsu method and k-means. In Ge Yu, Mario Köppen, Shyi-Ming Chen, and Xiamu Niu, editors, *9th International Conference on Hybrid Intelligent Systems (HIS 2009), August 12-14, 2009, Shenyang, China*, pages 344–349. IEEE Computer Society, 2009. doi:10.1109/HIS.2009.74.

[MBPK22] Naila Mukhtar, Lejla Batina, Stjepan Picek, and Yinan Kong. Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. In Steven D. Galbraith, editor, *Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings*, volume 13161 of *Lecture Notes in Computer Science*, pages 297–321. Springer, 2022. doi:10.1007/978-3-030-95312-6_13.

[MM79] Nick Martin and Hermine Maes. Multivariate analysis. *London, UK: Academic*, 1979.

[MMD20] Coenraad Mouton, Johannes C. Myburgh, and Marelie H. Davel. Stride and translation invariance in cnns. In Aurona J. Gerber, editor, *Artificial Intelligence Research - First Southern African Conference for AI Research, SACAIR 2020, Muldersdrift, South Africa, February 22-26, 2021, Proceedings*, volume 1342 of *Communications in Computer and Information Science*, pages 267–281. Springer, 2020. doi:10.1007/978-3-030-66151-9_17.

[Mon85] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016. doi:10.1007/978-3-319-49445-6_1.

[NBS22] Meenal V. Narkhede, Prashant P. Bartakke, and Mukul S. Sutaone. A review on weight initialization strategies for neural networks. *Artif. Intell. Rev.*, 55(1):291–322, 2022. URL: https://doi.org/10.1007/s10462-021-10033-z, doi:10.1007/S10462-021-10033-Z.

[NF95] Beat E Neuenschwander and Bernard D Flury. Common canonical variates. *Biometrika*, 82(3):553–560, 1995.

[PC15] Guilherme Perin and Lukasz Chmielewski. A semi-parametric approach for side-channel attacks on protected RSA implementations. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 34–53. Springer, 2015. doi:10.1007/978-3-319-31271-2_3.

[PCBP21] Guilherme Perin, Łukasz Chmielewski, Lejla Batina, and Stjepan Picek. Keep it unsupervised: Horizontal attacks meet deep learning. *IACR TCHES*, 2021(1):343–372, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8737. doi:10.46586/tches.v2021.i1.343-372.

[PITM14] Guilherme Perin, Laurent Imbert, Lionel Torres, and Philippe Maurine. Attacking randomized exponentiations using unsupervised learning. In Emmanuel Prouff, editor, *COSADE 2014*, volume 8622 of *LNCS*, pages 144–160. Springer, Heidelberg, April 2014. doi:10.1007/978-3-319-10175-0_11.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 61–81. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53140-2_4.

[PZS17] Romain Poussier, Yuanyuan Zhou, and François-Xavier Standaert. A systematic approach to the side-channel analysis of ECC implementations with worst-case horizontal attacks. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 534–554. Springer, Heidelberg, September 2017. doi:10.1007/978-3-319-66787-4_26.

[QLL18] Jie-Lin Qiu, Wei Liu, and Bao-Liang Lu. Multi-view emotion recognition using deep canonical correlation analysis. In Long Cheng, Andrew Chi-Sing Leung, and Seiichi Ozawa, editors, *Neural Information Processing - 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part V*, volume 11305 of *Lecture Notes in Computer Science*, pages 221–231. Springer, 2018. doi:10.1007/978-3-030-04221-9_20.

[RMH17] Ciara Rafferty, Máire McLoone, and Neil Hanley. Evaluation of large integer multiplication methods on hardware. *IEEE Trans. Computers*, 66(8):1369–1382, 2017. doi:10.1109/TC.2017.2677426.

[Rou87] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[SHKS15] Robert Specht, Johann Heyszl, Martin Kleinsteuber, and Georg Sigl. Improving non-profiled attacks on exponentiations based on clustering and extracting leakage from multi-channel high-resolution EM measurements. In Stefan Mangard and Axel Y. Poschmann:, editors, *COSADE 2015*, volume 9064 of *LNCS*, pages 3–19. Springer, Heidelberg, April 2015. doi:10.1007/978-3-319-21476-4_1.

[SIUH22] Kotaro Saito, Akira Ito, Rei Ueno, and Naofumi Homma. One truth prevails: A deep-learning based single-trace power analysis on RSA-CRT with windowed exponentiation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):490–526, 2022. URL: https://doi.org/10.46586/tches.v2022.i4.490-526, doi:10.46586/TCHES.V2022.I4.490-526.

[SKF+22] Seiya Shimada, Kunihiro Kuroda, Yuta Fukuda, Kota Yoshida, and Takeshi Fujino. Deep learning-based side-channel attacks against software-implemented RSA using binary exponentiation with dummy multiplication. In Hiroki Nishikawa and Xiangbo Kong, editors, *Proceedings of the 4th International Symposium on Advanced Technologies and Applications in the Internet of Things (ATAIT 2022), Ibaraki and Virtual, Japan, August 24-26, 2022*, volume 3198 of *CEUR Workshop Proceedings*, pages 75–84. CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3198/paper10.pdf.

[SM23] Marvin Staib and Amir Moradi. Deep learning side-channel collision attack. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):422–444, 2023. URL: https://doi.org/10.46586/tches.v2023.i3.422-444, doi:10.46586/TCHES.V2023.I3.422-444.

[SSS15] Takeshi Sugawara, Daisuke Suzuki, and Minoru Saeki. Two operands of multipliers in side-channel attack. In Stefan Mangard and Axel Y. Poschmann:, editors, *COSADE 2015*, volume 9064 of *LNCS*, pages 64–78. Springer, Heidelberg, April 2015. doi:10.1007/978-3-319-21476-4_5.

[SSSL20] Zhongkai Sun, Prathusha Kameswara Sarma, William A. Sethares, and Yingyu Liang. Learning relationships between text, audio, and video via deep canonical correlation for multimodal language analysis. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8992–8999. AAAI Press, 2020. URL: https://doi.org/10.1609/aaai.v34i05.6431, doi:10.1609/AAAI.V34I05.6431.

[SXW+23] Xiang-Jun Shen, Zhaorui Xu, Liangjun Wang, Zechao Li, Guangcan Liu, Jianping Fan, and ZhengJun Zha. Extraordinarily time-and memory-efficient large-scale canonical correlation analysis in fourier domain: From shallow to deep. *IEEE Transactions on Neural Networks and Learning Systems*, 2023. doi:10.1109/tnnls.2023.3282785.

[Tim19] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR TCHES*, 2019(2):107–131, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7387. doi:10.13154/tches.v2019.i2.107-131.

[Wal01] Colin D. Walter. Sliding windows succumbs to big mac attack. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 286–299. Springer, Heidelberg, May 2001. doi:10.1007/3-540-44709-1_24.

[WvM11] Marc F. Witteman, Jasper G. J. van Woudenberg, and Federico Menarini. Defeating RSA multiply-always and message blinding countermeasures. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 77–88. Springer, Heidelberg, February 2011. doi:10.1007/978-3-642-19074-2_6.

[YM15] Fei Yan and Krystian Mikolajczyk. Deep correlation for matching images and text. In *IEEE Conference on Computer Vision and Pattern Recognition,* *CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3441–3450. IEEE Computer Society, 2015. doi:10.1109/CVPR.2015.7298966.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR TCHES*, 2020(1):1–36, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8391. doi:10.13154/tches.v2020.i1.1-36.

[ZBHV21] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Efficiency through diversity in ensemble models applied to side-channel attacks. *IACR TCHES*, 2021(3):60–96, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8968. doi:10.46586/tches.v2021.i3.60-96.

A DCCA Implementation Hints

This section provides some DCCA implementation hints using the Keras 2.8 library and TensorFlow 2.8 backend.

```python
import tensorflow as tf
#CCA loss for DCCA model
def cca_loss():
  def inner_cca_objective(y_true, y_pred):
    """
    It is the CCA loss function as introduced in the original paper: Andrew et al., "Deep Canonical Correlation Analysis.", ICML, 2013.
    It uses the Keras library with the TensorFlow V2 backend.
    The implementation is based on github@VahidooX's Theano implementation. As the loss is unsupervised y_true is just ignored
    """
    r1, r2, eps = 1e-3, 1e-3, 1e-9
    o1 = o2 = int(y_pred.shape[1] // 2)
    # unpack (separate) the output of networks for view 1 and view 2
    H1 = tf.transpose(y_pred[:, 0:o1])
    H2 = tf.transpose(y_pred[:, o1:o1 + o2])
    m = tf.shape(H1)[1]

    H1bar = H1 - tf.cast(tf.divide(1, m), tf.float32) * tf.matmul(H1, tf.ones([m, m]))
    H2bar = H2 - tf.cast(tf.divide(1, m), tf.float32) * tf.matmul(H2, tf.ones([m, m]))

    SigmaHat12 = tf.cast(tf.divide(1, m - 1), tf.float32) * tf.matmul(H1bar, H2bar, transpose_b=True)
    SigmaHat11 = tf.cast(tf.divide(1, m - 1), tf.float32) * tf.matmul(H1bar, H1bar, transpose_b=True) + r1 * tf.eye(o1)
    SigmaHat22 = tf.cast(tf.divide(1, m - 1), tf.float32) * tf.matmul(H2bar, H2bar, transpose_b=True) + r2 * tf.eye(o2)
    # Calculating the root inverse of covariance matrices by using eigen decomposition
    [D1, V1] = tf.linalg.eigh(SigmaHat11)
    [D2, V2] = tf.linalg.eigh(SigmaHat22)
    # get eigen values that are larger than eps
    posInd1 = tf.where(tf.greater(D1, eps))
    D1 = tf.gather_nd(D1, posInd1)
    V1 = tf.transpose(tf.nn.embedding_lookup(tf.transpose(V1), tf.squeeze(posInd1)))

    posInd2 = tf.where(tf.greater(D2, eps))
    D2 = tf.gather_nd(D2, posInd2)
    V2 = tf.transpose(tf.nn.embedding_lookup(tf.transpose(V2), tf.squeeze(posInd2)))

    SigmaHat11RootInv = tf.matmul(tf.matmul(V1, tf.linalg.diag(D1 ** -0.5)
  ), V1, transpose_b=True)
    SigmaHat22RootInv = tf.matmul(tf.matmul(V2, tf.linalg.diag(D2 ** -0.5)
  ), V2, transpose_b=True)

    Tval = tf.matmul(tf.matmul(SigmaHat11RootInv, SigmaHat12),
  SigmaHat22RootInv)
    corr = tf.sqrt(tf.linalg.trace(tf.matmul(Tval, Tval, transpose_a=True)
  ))
    return -corr
  return inner_cca_objective
```

Implementation 1: DCCA loss function.

```
def DCCA_model(input_shape=50000,nb_neurons=400):
  Xinput = Input(shape=(input_shape,1))
  Yinput = Input(shape=(input_shape,1))
  model = Sequential()
  model.add(Conv1D(filters=4,kernel_size=8,activation="selu",padding="same
  ",name="conv1"))
  #model.add(AveragePooling1D(2,2))
  model.add(Flatten())
  model.add(Dense(nb_neurons,activation="sigmoid"))
  encoded_l = model(Xinput)
  encoded_r = model(Yinput)
  #Concatenate latent spaces for CCA loss
  sub_layer = Concatenate(name="encoded")([encoded_l,encoded_r])
  dcca_model = Model([Xinput,Yinput],sub_layer)
  return dcca_model

#Instantiation of the siamese DCCA model
dcca_model = DCCA_model(input_shape=input_shape,nb_neurons=nb_neurons)
optimizer = RMSprop(learning_rate=learning_rate)
dcca_model.compile(optimizer=optimizer,loss=[cca_loss()])

#train DCCA model to put pairs of modular operation traces in a correlated
   latent space
dcca_model.fit(training_generator,epochs=nb_epochs,verbose=verbose)
#get correlated latent space for the attack
new_traces = dcca_model.predict(testing_generator)
#compute the correlation on the new trace representation
corr = compute_corrcoeff(new_traces)
#compute Kmeans to identify the collision threshold
pred = compute_kmeans(corr)
```

Implementation 2: Collision attack with a siamese DCCA model.

B Protected RSA Implementation Hints

This section provides some details about the software part of the protected RSA implementation targeted in our experiments. For more information on the embedded arithmetic co-processor and the acquisitions campaign, we suggest readers refer to [CCC+19].

Implementation details (described in [CCC+19]). The targeted RSA implementation given in Algorithm 3 is based on a left-to-right binary Square-and-Multiply Always exponentiation algorithm combined with three countermeasures: message randomization, modulus randomization and exponent randomization. The processing flow is regular and independent of the exponent value thanks to the addition of dummy multiplications. Morever, the modular multiplications and squarings operations are performed with the same LIM operation of the embedded arithmetic co-processor which includes a dedicated

Algorithm 3: Exponentiation Square-and-Multiply Always (from [CCC+19])

```
Input: the masked input $m'$ in Montgomery representation, the masked exponent
       $d'$ of size $n$ bits, the function MMM initialized with the modulus $N'$ and
       the Montgomery factor $R$, and four memory segments $@(j)$ with $j \in [1..4]$.
Output: address $@(1)$ contains $m'^{d'}$ mod $N'$
1  $@(1) \leftarrow m$
2  $@(2) \leftarrow 1$
3  $\mathrm{seg_{in}} \leftarrow 1$
4  $\mathrm{seg_{acc}} \leftarrow 2$
5  $\mathrm{seg_{dum}} \leftarrow 3$
6  for $i = 0$ to $n - 1$ do
7      $s \leftarrow d'[n - 1 - i]$                                        ▷ Read from left to right
8      $\mathrm{seg_{free}} \leftarrow 9 - \mathrm{seg_{acc}} - \mathrm{seg_{dum}}$
9      MMM$(\mathrm{seg_{free}}, \mathrm{seg_{acc}}, \mathrm{seg_{acc}})$                  ▷ SQUARE
10     $\mathrm{seg_{acc}} \leftarrow \mathrm{seg_{free}}$
11     $\mathrm{seg_{free}} \leftarrow 9 - \mathrm{seg_{acc}} - \mathrm{seg_{dum}}$
12     MMM$(\mathrm{seg_{free}}, \mathrm{seg_{acc}}, \mathrm{seg_{in}})$                   ▷ MULTIPLY
13     $\mathrm{seg_{acc}} \leftarrow s \times \mathrm{seg_{free}} + (1 - s) \times \mathrm{seg_{acc}}$
14     $\mathrm{seg_{dum}} \leftarrow s \times \mathrm{seg_{dum}} + (1 - s) \times \mathrm{seg_{free}}$
15 end
16 $@(\mathrm{seg_{dum}}) \leftarrow 1$
17 MMM$(\mathrm{seg_{acc}}, \mathrm{seg_{acc}}, \mathrm{seg_{dum}})$ ▷ Montgomery representation to integer normal form
18 return $@(1) \leftarrow @(\mathrm{seg_{acc}})$
```

memory area based on Montgomery arithmetic, referred as MMM in Algorithm 3. The implementation uses 4 memory segments that respectively contain the input of the exponentiation and the intermediate values resulting from the bit-by-bit exponentiation processing. These addresses are denoted by $@(j)$ with $j \in [1..4]$. The input is stored at address $@(j)$ with $j = \mathrm{seg_{in}}$ and the value of $\mathrm{seg_{in}}$ does not vary during the processing. At each loop, a squaring then a multiplication are always executed. So two computations are processed during the exponentiation: the true one is stored at address $@(j)$ with $j = \mathrm{seg_{acc}}$, whereas the dummy one is stored at address $@(j)$ with $j = \mathrm{seg_{dum}}$. The values of the two indices $\mathrm{seg_{acc}}$ and $\mathrm{seg_{dum}}$ vary according to the value of the exponent bit which is treated. Their updating is done without conditional branch thanks to the use of a third index $\mathrm{seg_{free}}$.