Deep Learning to Evaluate Secure RSA Implementations

Mathieu Carbone[1], Vincent Conin[1], Marie-Angela Cornélie[2], François Dassance[3], Guillaume Dufresne[3], Cécile Dumas[2], Emmanuel Prouff[4] and Alexandre Venelli[3]

[1] SERMA Safety and Security, France, {m.carbone,v.conin}@serma.com
[2] CEA LETI, France, {cecile.dumas,marie-angela.cornelie}@cea.fr
[3] Thales ITSEF, France,
{francois.dassance,guillaume.dufresne,alexandre.venelli}@thalesgroup.com
[4] ANSSI, France, emmanuel.prouff@ssi.gouv.fr

Abstract. This paper presents the results of several successful profiled side-channel attacks against a secure implementation of the RSA algorithm. The implementation was running on a ARM Core SC 100 completed with a certified EAL4+ arithmetic co-processor. The analyses have been conducted by three experts' teams, each working on a specific attack path and exploiting information extracted either from the electromagnetic emanation or from the power consumption. A particular attention is paid to the description of all the steps that are usually followed during a security evaluation by a laboratory, including the acquisitions and the observations pre-processing which are practical issues usually put aside in the literature. Remarkably, the profiling portability issue is also taken into account and different device samples are involved for the profiling and testing phases. Among other aspects, this paper shows the high potential of deep learning attacks against secure implementations of RSA and raises the need for dedicated countermeasures.

Keywords: Side-Channel Attacks · RSA · Deep Learning

1 Introduction

Side-channel analysis (SCA) is a class of cryptanalytic attacks that exploit the physical environment of a cryptosystem implementation to recover some leakage about its secrets. It is often much more efficient than a cryptanalysis mounted in the so-called black-box model where no leakage occurs, and dedicated countermeasures are usually implemented to protect the execution of cryptographic algorithms on embedded systems. In most of secure products like smart-cards, the security is achieved by combining techniques applied at the software level (e.g. *masking/blinding* [Cor99] or *shuffling* [MOP07]) with mechanisms acting at the hardware level (e.g. *clock jittering*, *white noise addition* or *power consumption balancing* [MOP07]). This is especially true for RSA implementations where resistance against side-channel attacks is achieved by defining a secure exponentiation algorithm (at software level) on the basis of a secure arithmetic co-processor (e.g. implementing a fast Montgomery modular arithmetic while limiting information leakage) [BMV05, BÖPV03].

When it comes to assess the robustness of an RSA implementation against the SCA threat, several attacks are performed by experts, usually after a leakage characterization step. To help security evaluators, some technical reports like the AIS-46 published by BSI [fIS18] give overviews and recommendations on the relevant side-channel attacks that have to be applied against implementations. Among them, the so-called family of *profiled*

IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925, Vol. 2019, No. 2, pp. 132–161
DOI:10.13154/tches.v2019.i2.132-161

SCA is recognized as the most effective ones since the underlying adversary is assumed to priorly use an open copy of the final target to precisely tune all the parameters of the attack. It includes Templates Attacks [CRR02] and Stochastic modelling (a.k.a. Linear Regression Analysis) [DPRS11, Sch08, SLP05]. This attack strategy where the adversary precedes the attack by a supervised training phase may be viewed as a classical Machine Learning problem and a recent line of works has started to build connections between the world of side-channel analysis and the world of Machine Learning (with a particular focus on Deep Learning).

Related Works

Many studies have been published to argue on the efficiency of profiled attacks against (secure) block cipher implementations (see e.g. [BLR13, GHO15, HZ12, HGM+11, LBM14, LMBM13, LPB+15, PSB+18] for the most recent ones). However, few works have been published on asymmetric cryptographic implementations like ECDSA or RSA. Some like [MAB+18] have followed a machine learning approach to demonstrate successful detection of state-of-the-art cache-based SCAs. Other profiled attacks (like e.g. [LBM14]) have been tested against RSA but the targeted implementation was not secure (neither on the software nor on the hardware level). The main challenge when attacking secure RSA implementations is that the attack must be able to recover more than 90% of the secret parameter from a single execution[1] (due to exponent randomization [Cor99]) without knowing neither the message nor the modulus which are processed (due to input and modulus randomizations [Gir06]). The so-called class of *horizontal collisions attacks* has been introduced to specifically address this issue [Wal01]. However, even if the subsequent works like [BJPW13, CFG+10, PZS17, vWWB11] have improved the original idea, none of them has been shown to be efficient against an implementation running on a defensive arithmetic co-processor (like e.g. those whose security has been certified in a Common Criteria framework [Arr18]).

Contributions

To the best of our knowledge, this paper presents the first full profiled attack against an RSA implementation running on a certified arithmetic co-processor and equipped with classical side-channel countermeasures (blinding of the message, blinding of the exponent and blinding of the modulus as e.g. detailled in [Cor99]).[2] Several attacks, exploiting either the electromagnetic emanation or the power consumption measured during the processing, are presented that combine the principles of address-bit DPA attacks [IIT02, MDS99b] and horizontal attacks [BJPW13] with machine learning techniques like Convolutional Neural Networks (CNN) to accurately recover the value of the exponent bits. A special attention is paid to the description of the acquisition campaigns and of the pre-processing of the measured traces, as the latter steps are known to pose practical difficulties when it comes to attack arithmetic co-processors (e.g. synchronization issues, size of the traces

[1] In this case, [SW14a] shows that the secret RSA parameter may be efficiently reconstruct from several randomized representations if the random value used for the blinding has bit-length 64, which is common in today implementations.

[2] A certified arithmetic co-processor comes with guidelines that detail how to use the crypto-coprocessor in a secure manner. As a cautionary note, we alert the reader on the fact that we did not have access to the code in order to verify the compliance with these guidelines. Moreover, as discussed in the conclusion of the paper, the software part of the targeted RSA implementation does not embed specific security mechanisms to defeat horizontal or address-bit side-channel attacks. This choice has been done deliberately by CryptoExperts' team (https://www.cryptoexperts.com/) who was responsible for the development of the RSA software part. Actually, the latter implementation was part of a challenge organized by ANSSI's laboratory of hardware security for industrial partners. This paper shows that the application of advanced profiling attacks like those based on Deep Learning renders security mechanisms against horizontal and address-bit attacks mandatory to achieve a high level of security.

and choice of the sampling rate, few information on the details of the hardware processing, etc.). Also, the question of *profiling portability* (*i.e.* the ability to apply a profiling done on a device A to attack a device B with the same architecture) is addressed by using different device samples. Our results practically demonstrate that, in our context, the latter portability does not significantly impact the effectiveness of our attacks. More generally, our work shows that the ability of the deep learning algorithms to efficiently operate on highly-dimensional inputs (which are leakage measurements in our context) even in the presence of signal jittering makes them a tool of choice for the secure evaluation of RSA implementations (in addition to the methods already in the toolbox of the evaluator – see e.g. [fIS18] –).

Paper Organization

Section 2 is dedicated to the presentation of the software and hardware aspects of the target RSA implementation and the identification of two attack paths. Then, we describe in Section 3 the acquisitions' campaigns that have been launched to measure the power consumption and the electromagnetic emanation during the RSA processing. A leakage characterization analysis has afterwards been done on the measured traces. It is discussed in Section 4. Eventually, Section 5 presents our profiled attacks' results and our main observations related to the application of deep learning attacks against RSA implementations.

2 Target Presentation & Attack Paths

2.1 Target Presentation: Hardware Part

2.1.1 Brief Hardware Description

The hereby target of evaluation is a software RSA SFM (*StraightForward Mode* in opposition to the *Chinese Remainder Mode*) running on a 0.13um 32-bit Contact Smartcard IC. The IC has been EAL4+ certified in Asia (out of the European SOG-IS scheme [Mem18]). It features an ARM core SC 100 with 18 KB of RAM, 8 KB of ROM and 548 KB of FLASH. An overview of the IC die and of the main blocks is given in Figure 1.

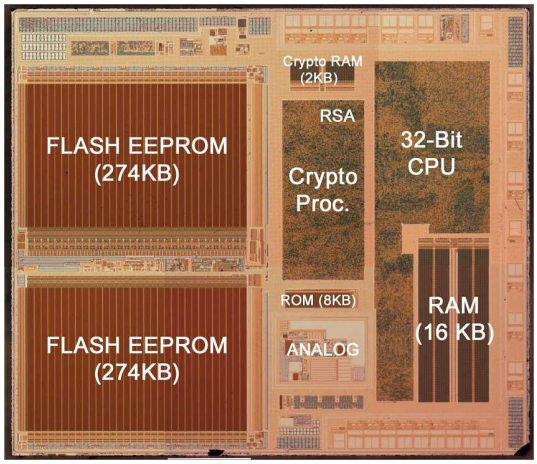


Figure 1: General view of the die after sample opening; mag 50X

The RSA multiplications and squarings are performed with the same operation of the embedded arithmetic co-processor which includes a dedicated memory area, the so-called

crypto-RAM, composed of 8 segments of 64 words (2048 bits). Moreover, to speed-up the processing, the modular arithmetic routines of the library are based on Montgomery arithmetic. It is detailed here-after.

2.1.2 Low Level Library and Montgomery Arithmetic

The co-processor of the device studied in this paper implements the so-called *Montgomery arithmetic* (see e.g. [BSS05] for a clear and short introduction). Roughly speaking, the core idea of this arithmetic is to replace several costly modular reductions with the same modulus by some pre-processing and then efficient divisions by a well-chosen power of 2 (we recall that such divisions simply consist in right binary shifts).

Let N denote a modulus and let us assume that we want to compute a sequence of operations in the form $x \cdot y \bmod N$ for $x, y \in \mathbb{Z}/N\mathbb{Z}$. Thanks to the arithmetic co-processor, this processing is done by calling a so-called *Montgomery multiplication*. Before running such a multiplication, the operands must be normalized, i.e. put in *Montgomery representation* $\widetilde{x} = x \cdot R \bmod N$ and $\widetilde{y} = y \cdot R \bmod N$, where R is the *Montgomery constant* defined in our context by $R \doteq 2^{\omega} \bmod N$ with ω being the smallest multiple of 32 which is greater than or equal to $\log_2(N)$. Then, a hardware routine computing $u \cdot v \cdot R^{-1} \bmod N$ is executed for $u = \widetilde{x}$ and $v = \widetilde{y}$. It may be checked that the following equality holds:

$$\widetilde{x} \cdot \widetilde{y} \cdot R^{-1} \bmod N = (x \cdot y) \cdot R \bmod N = \widetilde{z} \ ,$$

so that the output is indeed the Montgomery representation of $z = x \cdot y \bmod N$.

Once the initial operands of a sequence of modular multiplications have been put in Montgomery representation, all operations are performed under this representation. In particular, modular multiplications are replaced by Montgomery multiplications with the pre-defined RSA modulus N and the Montgomery constant R. In our context, this is actually done through the procedure $\texttt{MMM}(i_{\text{dest}}, i_1, i_2)$ which stores in the segment of index i_{dest} the result of the Montgomery multiplication between values stored in segments of indexes i_1 and i_2. The output and input values are in Montgomery representation. When the computation is over, the final result is put back in its natural integer form (which simply consists in multiplying the result by R^{-1} modulo N).

2.2 Target Presentation: Software Part

The targeted RSA implementation is based on a *Left-to-Right Square & Multiply Always* exponentiation algorithm [Cor99] combined with three countermeasures: *input randomization*, *modulus randomization* and *exponent randomization*. Moreover the code is branch-less as the exponent bit is not tested and since no conditional jump is performed.

2.2.1 Countermeasures

Modulus Randomization. This technique is based on the following equality:

$$m^d \bmod N = (m^d \bmod k_0 \cdot N) \bmod N, \tag{1}$$

where k_0 is any (random) positive integer co-prime to N. Note that any integer lower than p and q is co-prime to N, and in particular, any integer of bit-length 64 (or less) is co-prime to N. The modulus randomization technique then simply consists (1) in picking up a random integer k_0, (2) in computing the exponentiation modulo $k_0 \cdot N$ and (3) in reducing the result modulo N.

Input Randomization. The input randomization technique involved in our target implementation can only be used jointly with the modulus randomization technique. It is based on the following equation:

$$m^d \mod N = ((m + k_1 \cdot N)^d \mod k_0 \cdot N) \mod N, \tag{2}$$

where k_0 and k_1 are any (random) positive integers, k_0 being co-prime to N. The input randomization technique hence consists in picking a random integer k_1 and adding $k_1 \cdot N$ to the input m, before the exponentiation (modulo $k_0 \cdot N$).

Exponent Randomization. This technique is based on the following equality:

$$m^d \mod N = m^{d+k_2 \cdot \phi(N)} \mod N, \tag{3}$$

where ϕ denotes the Euler's totient function, and where k_2 is any (random) positive integer. In case of an RSA modulus $N = p \cdot q$ (with p and q prime), we have $\phi(N) = (p-1)(q-1)$. The principle of the exponent randomization countermeasures is to pick a random integer k_2 and evaluate the RSA exponentiation with exponent $d' = d + k_2 \cdot \phi(N)$.

Final Implementation. Values p and q are prime integers of bit-length 512. So the modulus N is a 1024-bit integer. The combination of the three masking countermeasures corresponds to the following computation:

$$((m + k_1 \cdot N)^{d+k_2 \cdot \phi(N)} \mod k_0 \cdot N) \mod N, \tag{4}$$

where m is the plaintext in Montgomery representation, k_0, k_1 and k_2 are some random positive integers of bit-length 64. By consequence, the masked input $m' \doteq m + k_1 \cdot N$, the masked modulus $N' \doteq k_0 \cdot N$ used to perform the `MMM`s, the masked exponent d' and all the values involved in the modular exponentiation have size $n = 1024 + 64 = 1088$ bits. This implementation is completed by the first step that transforms the plaintext in Montgomery representation $m \mapsto \tilde{m} = m/R \bmod N'$) and the last step doing the inverse operation putting back the result in its natural integer form to obtain the ciphertext (see Sect. 2.1 for more details).

2.2.2 Implementation of the modular exponentiation

The detailed implementation is given in Algorithm 1. It is based on a classical *Square & Multiply Always* strategy to get a processing flow which is independent of the exponent value. It uses 4 memory segments that respectively contain the input of the exponentiation and the intermediate values resulting from the bit-by-bit exponentiation precessing. These addresses are denoted by $@(j)$ with $j \in [1..4]$.

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 $j = \text{seg}_{\text{acc}}$, whereas the dummy one is stored at address $@(j)$ with $j = \text{seg}_{\text{dum}}$. The values of the two indices seg_{acc} and 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 seg_{free}. Moreover the input is stored at address $@(j)$ with $j = \text{seg}_{\text{in}}$ and the value of seg_{in} does not vary during the processing.

After the processing of Algorithm 1, the result is reduced modulo N to get the correct signature as described in (4).

2.3 Attack Paths

As the use of randomization techniques breaks the dependency between the intermediate values of the processing and the side-channel observations, an attacker cannot work

Algorithm 1 Modular Exponentiation *Square & Multiply Always*

Require: 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 $j \in [1..4]$.
Ensure: address $@(1)$ contains ${m'}^{d'} \bmod N'$

1: $@(1) \leftarrow m$
2: $@(2) \leftarrow 1$
3: $\text{seg}_{\text{in}} \leftarrow 1$
4: $\text{seg}_{\text{acc}} \leftarrow 2$
5: $\text{seg}_{\text{dum}} \leftarrow 3$
6: **for** $i = 0$ **to** $n - 1$ **do**
7: $\quad s \leftarrow d'[n-1-i]$ ▷ Read from left to right
8: $\quad \text{seg}_{\text{free}} \leftarrow 9 - \text{seg}_{\text{acc}} - \text{seg}_{\text{dum}}$
9: $\quad \texttt{MMM}(\text{seg}_{\text{free}}, \text{seg}_{\text{acc}}, \text{seg}_{\text{acc}})$ ▷ SQUARE
10: $\quad \text{seg}_{\text{acc}} \leftarrow \text{seg}_{\text{free}}$
11: $\quad \text{seg}_{\text{free}} \leftarrow 9 - \text{seg}_{\text{acc}} - \text{seg}_{\text{dum}}$
12: $\quad \texttt{MMM}(\text{seg}_{\text{free}}, \text{seg}_{\text{acc}}, \text{seg}_{\text{in}})$ ▷ MULTIPLY
13: $\quad \text{seg}_{\text{acc}} \leftarrow s \times \text{seg}_{\text{free}} + (1 - s) \times \text{seg}_{\text{acc}}$
14: $\quad \text{seg}_{\text{dum}} \leftarrow s \times \text{seg}_{\text{dum}} + (1 - s) \times \text{seg}_{\text{free}}$
15: $@(\text{seg}_{\text{dum}}) \leftarrow 1$
16: $\texttt{MMM}(\text{seg}_{\text{acc}}, \text{seg}_{\text{acc}}, \text{seg}_{\text{dum}})$ ▷ Montgomery representation to integer normal form
17: $@(1) \leftarrow @(\text{seg}_{\text{acc}})$

vertically (*i.e.* with several executions of the RSA processing). He has to mount an attack *horizontally* (*i.e.* with a *single* trace). The target implementation described in previous sections contains two main vulnerabilities which can be exploited in a profiled attack scenario. The first one focuses on the address indices manipulation and may be viewed as an example of address-bit DPA [IIT02, MDS99a], while the second one focuses on the operands' manipulation and exploits the same principles as the horizontal collision attacks [BJPW13]. They are detailed in the next two sections. It is here important to remember that, in both cases, the challenging point is to be able to successfully apply the attack to recover almost all the secret exponent bits by exploiting *a single* leakage trace (after profiling). This will be discussed in the second part of the paper.

2.3.1 Address Indices Manipulation.

The exponentiation *Square & Multiply Always* described in Algorithm 1 contains a potential vulnerability related to the manipulation of the index seg_{free} (that is linked to the memory address where the result of the Montgomery multiplication is stored).

It may be observed that the address index seg_{free} can take three different values (theoretically unknown); in other terms, seg_{free} successively belongs to three different classes during the whole modular exponentiation. Remarkably, the sequence of classes depends on the exponent bits, and we will see that this dependency may be exploited to recover sensitive information. More precisely, seg_{free} stays unchanged for two consecutive exponentiation loop indices i and $i + 1$ if and only if the corresponding exponent bit $d'[n - 1 - i]$ equals 1. Thus, by knowing whether seg_{free} stays in the same class or not, an attacker may learn the values of all the exponent bits except the last one.

From a more formal point of view, it may be checked that the value of seg_{free} at input of the MMM processing during the ith squaring (resp. during the ith multiplication) takes the form $2 + \#\text{dec}(i)$ where, for every i in $[0..n-2]$, the value $\#\text{dec}(i)$ is recursively defined

Table 1: Registers Manipulation (Example).

i	$d'[n-1-i]$	seg_{free} during squaring $\text{seg}_{\text{free}} = \#\text{offset} + \#\text{dec}(i)$	seg_{free} during mult. $\text{seg}_{\text{free}} = \#\text{offset} + \#\text{dec}(i)$
0	0	$2+2$	$2+0$
1	0	$2+1$	$2+2$
2	1	$2+0$	$2+1$
3	1	$2+0$	$2+1$
4	1	$2+0$	$2+1$
5	0	$2+0$	$2+1$
6	1	$2+2$	$2+0$
7	1	$2+2$	$2+0$
8	0	$2+2$	$2+0$
9	0	$2+1$	$2+2$

as follows:

$$\#\text{dec}(i+1) = \begin{cases} \#\text{dec}(i) - (1 - d'[n-1-i]) \bmod 3 & \text{, for the squaring} \\ \#\text{dec}(i) + 2(1 - d'[n-1-i]) \bmod 3 & \text{, for the multiplication} \end{cases}, \quad (5)$$

with $\#\text{dec}(0)$ equalling 2 and 0 respectively for the squaring and the multiplication.

Table 1 gives an illustration of the dependency between the exponent bits and the consecutive values of $\#\text{dec}(i)$ when the 10 most significant bits of d' equal 0011101100.

2.3.2 Operands' Manipulations

Another attack path is based on a property of the algorithm *Square & Multiply Always*. Indeed, when the exponent bit $d'[n-1-i]$ treated at loop index i is zero, then the multiplication is useless and hence, the result is thrown. This implies that the subsequent operation (a squaring) shares an operand with the previous multiplication. If, at the opposite, $d'[n-1-i]$ equals 1, then the multiplication and the subsequent squaring are not likely to share the same operand because the result of one is the entry of the other one. This property is convenient to retrieve the involved bit of the secret exponent. Indeed, if we denote by $\mathbf{a}_i^{\text{sq.}}$ (resp. $\mathbf{a}_i^{\text{mult.}}$) the left-hand side operand of the squaring at loop index i (resp. of the multiplication at loop index i), this attack path consists in comparing $\mathbf{a}_i^{\text{mult.}}$ and $\mathbf{a}_{i+1}^{\text{sq.}}$ in order to retrieve the bit exponent $d'[n-1-i]$ at loop i.

3 Presentation of the Acquisitions Campaign

For the two attack paths identified in previous section, it is important to precisely identify in each trace/observation the sub-parts corresponding to the `MMM` processings. In the next sections, we show how this can be done for both the power consumption and electromagnetic measurements, in the context of a defensive arithmetic co-processor. In the first (Icc) case, the measurements correspond to the full RSA processing, while they only correspond to the processing of the 7 MSB of the exponent in the second (EM) case. This difference is a direct consequence of the fact that our EM attacks (that target internal operations of the co-processor) required measurements at a much greater sampling rate than that needed for our Icc attacks (namely 2.5 GS/s *versus* 50 MS/s).[3]

[3] Our SNR characterization of the EM measurements show that the sampling rate could have been maybe smaller without too much impacting the attack efficiency, but we did not experimentally validate this observation.

3.1 Power Consumption Measurements

A first acquisition campaign has been launched to measure the power consumption (Icc) during the processing of the RSA implementation described in previous section (namely Algorithm 1 with masked exponents, masked messages and masked moduli of size $n = 1088 = 1024 + 64$). The power consumption (Icc) was measured at 50 MS/s using a Lecroy WaveRunner 625Zi oscilloscope. This (low) sampling rate suggests that an alternative to the traditional bench-top oscilloscope, e.g. a PC oscilloscope such as Picoscope, could have been used in the present case.

Each acquisition, denoted by $\mathbf{L}$, was composed of around $25,000,000$ time samples that have been split into $2n$ sub-traces $\mathbf{L}_0$, ..., $\mathbf{L}_{2n-1}$. This splitting has been done by following a so-called *correlation pattern matching* [VKMJ10] approach (i.e. the signal pattern corresponding to the `MMM` is correlated as a sliding window with the full trace, then the maximums of correlation allow the extraction of the different occurrences of patterns from the trace). A new set has been rebuilt with these patterns ensuring a *per multiplication* fast resynchronization (see Figure 2a and Figure 2b). Afterwards, each of these multiplication patterns has been pre-processed, *i.e.* decimated by a factor 5 after being filtered by a linear low pass filter (see Figure 2c). The resulting `MMM` processing sub-traces $\mathbf{L}_j$ are each composed of around $2,247$ time samples after this pre-processing step. Note that the pre-processing step, e.g. the choice of the decimation parameter, was guided by the strength of the resulting univariate first-order leakage assessment (see Subsubsection 4.1.1). This allows us to get a stronger leakage strength for the attacks with a lower computation cost.

Finally all the `MMM` processing sub-traces $\mathbf{L}_i^{\text{sq.}}$ corresponding to a squaring (*i.e.* defined s.t. $\mathbf{L}_i^{\text{sq.}} = \mathbf{L}_{2i}$) have been gathered into a same set, and the same processing has been done for the sub-traces $\mathbf{L}_i^{\text{mult.}}$ corresponding to a multiplication (*i.e.* defined s.t. $\mathbf{L}_i^{\text{mult.}} = \mathbf{L}_{2i+1}$). The fact that the mean of 100 sub-traces does not fade away (see Figure 2c) was considered as a valuable experimental validation of our resynchronization effectiveness.

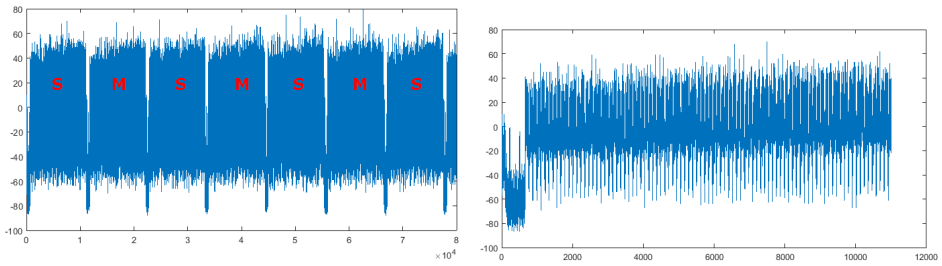
The campaign and the pre-treatment have been repeated on several smart-cards.

3.2 Electromagnetic Measurements

A second acquisition campaign has been launched to measure the electromagnetic emanations during the processing of the seventh most significant bits of the masked exponent of the RSA. The implementation was similar to that targeted in previous section (namely the masked modulus, the masked message and the masked exponent were all of size 1088 bits). The signal has been acquired with a 2.5 GS/s sampling rate over 200 μs (see Figure 3 for the processing of the 7 first MSB of the exponent). Each acquired trace was composed of $5,000,000$ time samples which correspond to the 7 MSB of the masked exponent. A standard laboratory equipment has been used to perform the campaign (Lecroy WaveRunner 625Zi oscilloscope and Langer ICR EM probe). An example of measurement is plotted in Figure 3.

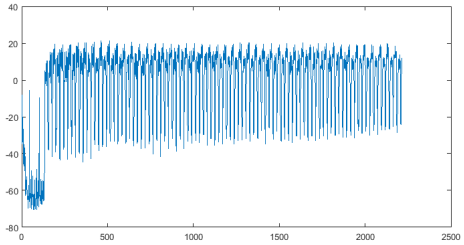
As done for the power consumption campaign, each trace has been roughly split into sub-traces $\mathbf{L}_j$ corresponding to `MMM` processings. Then, the latter traces have been reorganized to group all the sub-traces $\mathbf{L}_i^{\text{sq.}}$ corresponding to a squaring in one set and all sub-traces $\mathbf{L}_i^{\text{mult.}}$ corresponding to a multiplication in another set (due to the regularity of Algorithm 1, $\mathbf{L}_i^{\text{sq.}} = \mathbf{L}_{2i}$ and $\mathbf{L}_i^{\text{mult.}} = \mathbf{L}_{2i+1}$). In the following, we however continue to use the notation $\mathbf{L}_j$ when the discussion is valid whatever the nature of the operation.

As it may be observed in Figure 4, two peculiar types of patterns can be identified in each sub-trace $\mathbf{L}_j$: the first one corresponds to the *initialization* of the operation which



(a) Illustration of a part of the signal, with the succession of squaring and multiplication by `MMM` processing.

(b) A single `MMM`.



(c) Mean of 100 `MMM` processing patterns after filtering and decimation by a factor 5.

Figure 2: Alignment steps of the `MMM` processing traces.

e.g. includes Steps 7, 8, 10 and 11 in Algorithm 1, while the second one corresponds to the multi-precision `MMM` *operation* itself (squaring or multiplication). These two patterns are likely to contain information on the processed secret bit and they have therefore been analyzed.

`MMM` initialization alignment. The parts corresponding to the `MMM` initialization have been extracted from each $\mathbf{L}_j$. Each part contained around $30,000$ time samples. Figure 5a shows the overlapping of 10 such parts. A desynchronization is clearly observable, in particular around the two large EM peaks marked in red. Figure 5 presents the different resynchronization steps that have been followed to correct this issue. A first alignment step has been applied by fixing a threshold EM amplitude value and by aligning all EM traces when they reach this threshold. The dotted line in Figure 5 refers to the selected threshold. Figure 5b shows the traces after this first alignment step and it may be observed that the traces are indeed correctly aligned on the first large EM peaks (marked in green). However, a timing desynchronization is still present as EM peaks on the right part of the figure are clearly not aligned. Hence, another alignment step is required. Figure 5c shows traces after the application of a so-called *sliding correlation alignment method* which may be viewed as a variant of the *correlation pattern matching* approach [VKMJ10] applied in previous section. It first consists in selecting a small timing window on a reference EM trace and in setting maximum shift offsets. Then, for each shift, Pearson's correlation is computed between the current trace to align and the reference trace for the given window. The maximum correlation value implies that a correct shift has been found. Then, the window slides over time and the algorithm is re-applied. This technique allows for the realigning of traces in cases where the desynchronization changes over time.

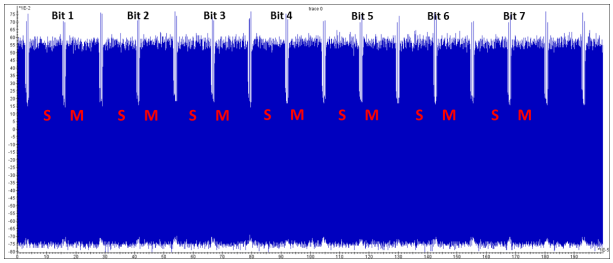


Figure 3: Sequence of MMM processings for the first MSBs of the exponent.

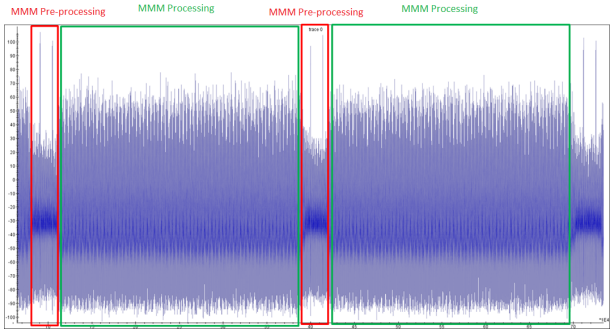


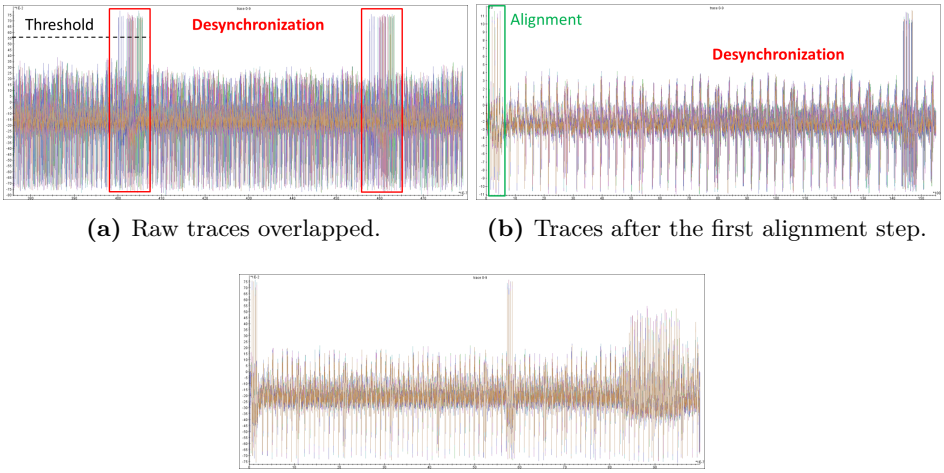
Figure 4: EM emanation during a single loop of Algorithm 1.

MMM operation alignment. The EM patterns corresponding to the MMM squaring operation have also been extracted from each sub-trace $\mathbf{L}_i^{\text{sq.}}$ and then resynchronized. The size of those patterns was roughly $300,000$ time samples (ten times more compared to the initialization). Figure 6 presents the different resynchronization steps needed to obtain aligned traces. Figure 6a shows the overlapping of 10 extracted patterns. As expected, it may be observed that the MMM multi-precision operation is composed of small sub-operations (marked in black) which correspond to the arithmetic co-processor unitary operations. Figure 6b shows a zoom on some of these sub-operations. As previously, a timing desynchronization is observable. Hence, the same sliding correlation alignment technique has been applied. Figure 6c presents some overlapped traces after this alignment, zoomed on some sub-operations. The sub-operations are clearly aligned with each others and over time during the whole MMM operation.

After all realignment steps, no trace was discarded for both the dataset containing the MMM-initialization patterns and the dataset containing the MMM-operation patterns. This is due to the fact that no strong desynchronization countermeasures are implemented. The campaign and the pre-treatment have been repeated on several smart-cards.

4 Leakage Assessment

To verify that the (Icc and EM) sub-traces pre-processed as detailed in Subsection 3.1 and Subsection 3.2 contain information that are exploitable to test the attack paths presented in Subsection 2.3, a leakage assessment has been done. First, we have verified that both Icc and EM sub-traces could be used to perform the attack related to the address indices manipulation (Subsubsection 2.3.1). From this leakage assessment, it is



(a) Raw traces overlapped.

(b) Traces after the first alignment step.

(c) Traces after the second alignment step.

Figure 5: Alignment steps of the `MMM` pre-processing traces.

not clear whether the address indices leak during their manipulation by the CPU or by the arithmetic co-processor (without full open access to the code, such a conclusion is actually difficult to draw). Secondly, we have verified that the EM sub-traces (and more precisely the portion related to the `MMM` processing part) contain enough information to test the attack related to the operand's manipulation (Subsubsection 2.3.2).

4.1 Address Indices Manipulation

4.1.1 Power Consumption

To characterize the information leakage, a univariate first-order leakage assessment related to the first attack path identified in Subsection 2.3 has been launched on $2,016$ RSA processing observations pre-treated as described in Subsection 3.1. This led to two labelled databases of sub-traces $(\mathbf{L}_i^{\text{sq.}})_{i \leqslant 1088 \times 2016}$ and $(\mathbf{L}_i^{\text{mult.}})_{i \leqslant 1088 \times 2016}$ respectively corresponding to squarings and multiplications with the `MMM` routine. By using the fact that the random masks and the secret exponents were known, the *Normalized Inter-Class Variance* (NICV) [BDGN14] has been estimated for all the $2,247$ time samples composing the sub-traces in each dataset. The goal is to detect statistical dependency with the value of the memory index seg_{free}. We recall that the NICV at time sample t, denoted by $\text{NICV}[t]$, is defined as:

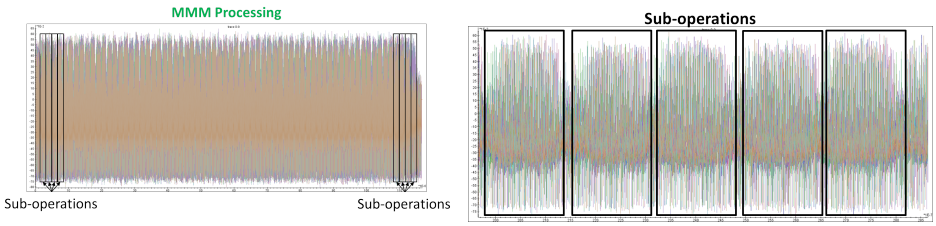
$$\text{NICV}[t] \doteq \frac{1}{\frac{1}{\text{SNR}[t]} + 1} \ ,$$

with

$$\text{SNR}[t] = \frac{\mathbb{V}_{\text{seg}_{\text{free}}}[\mathbb{E}[\mathbf{L}^{\text{sq.}} \mid \text{seg}_{\text{free}}]]}{\mathbb{E}_{\text{seg}_{\text{free}}}[\mathbb{V}[\mathbf{L}^{\text{sq.}} \mid \text{seg}_{\text{free}}]]} \text{ or } \text{SNR}[t] = \frac{\mathbb{V}_{\text{seg}_{\text{free}}}[\mathbb{E}[\mathbf{L}^{\text{mult.}} \mid \text{seg}_{\text{free}}]]}{\mathbb{E}_{\text{seg}_{\text{free}}}[\mathbb{V}[\mathbf{L}^{\text{mult.}} \mid \text{seg}_{\text{free}}]]} \ ,$$

where $\mathbf{L}^{\text{sq.}}$ (res. $\mathbf{L}^{\text{mult.}}$) denotes the random variable associated to the observations $(\mathbf{L}_i^{\text{sq.}})_i$ (resp. $(\mathbf{L}_i^{\text{mult.}})_i$).

The NICV has also been computed with respect to the value of the masked exponent bit associated to a multiplication processing (i.e. dataset $(\mathbf{L}_i^{\text{mult.}})_i$). For comparison purpose, the three characterizations are plotted on top of each other in Figure 7.



(a) Raw traces overlapped.

(b) Zoom on some sub-operations.

(c) Aligned traces zoomed on some sub-operations.

Figure 6: Alignment steps of the `MMM` operations traces.

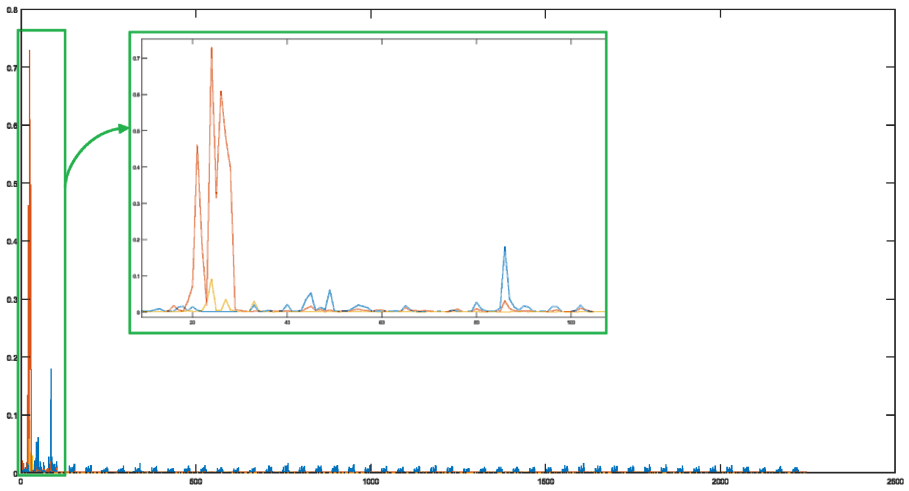


Figure 7: Icc Campaign – NICV scores w.r.t. time samples; in blue, seg_{free} values for $\mathbf{L}^{\text{sq.}}$, in orange seg_{free} values for $\mathbf{L}^{\text{mult.}}$, in magenta the exponent bit values for $\mathbf{L}^{\text{mult.}}$.

The most significant information leakage (orange trace) is found for seg_{free} and a `MMM` processing corresponding to a multiplication (*i.e.* random variable $\mathbf{L}^{\text{mult.}}$). This makes seg_{free} as a leakage of choice to perform SCA: leakages related to seg_{free} during a squaring (blue trace) is significantly smaller and also leakages related to the bits of the randomized exponent associated to the multiplications patterns (magenta trace).

The NICV computations performed on another cards reveal similar leakages and no significant difference has been found.

4.1.2 Electromagnetic

First order univariate leakage assessment has also been conducted on $45,000$ electromagnetic traces acquired as described in Subsection 3.2. For both sets of sub-traces $(\mathbf{L}_i^{\text{sq.}})_{i<7\times45,000}$ and $(\mathbf{L}_i^{\text{mult.}})_{i<7\times45,000}$, a SNR characterization has been performed w.r.t. the value of register index seg_{free}. Exploiting the observation reported in Subsection 3.2, the SNR has only been computed on the first $30,000$ points of the sub-traces (which correspond to the `MMM` initializations). Figure 8 shows, on top and for temporal reference, an original aligned EM trace of the initialization part of the $\mathbf{L}_i^{\text{sq.}}$ sub-traces and, on bottom, the corresponding SNR results.

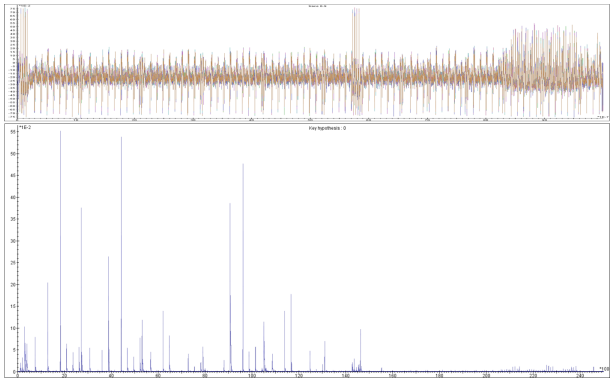


Figure 8: EM Campaign – SNR result for the seg_{free} value *versus* the squaring initialization (bottom) and original EM trace (top).

The SNR is around 55.10^{-2} for the best points of interest. The corresponding points will be used in our deep learning attacks reported in Subsection 5.2. A similar leakage assessment has also been performed on another card. The SNR results are very similar between the two cards. However, a temporal shift between SNR peaks has been observed (see Figure 9).

This shift is due to a clock frequency variation between cards. This difference has to be compensated in order to be able to utilize traces from different cards during the side-channel analysis. It can be done by detecting every electromagnetic peak of every original trace and recreating a new trace by concatenating same length intervals around the peaks.

4.2 Operands' Manipulations

In order to apply the attack path described in Subsubsection 2.3.2 on EM leakages, one must be able to recover enough information on the left-hand side operand value of each `MMM` operation from the corresponding sub-traces $\mathbf{L}_i^{\text{mult.}}$ and $\mathbf{L}_i^{\text{sq.}}$ pre-treated as explained in Subsection 3.2 ($50,000$ traces have been used, each composed of around $300,000$ time samples).

Since the co-processor performs elementary operations on 32-bits words, it seemed to be sound to focus on a specific 32-bit (sub)-word of each target operand (whose bit-length was 1088 for our experiments). In Figure 10 (top), it may *first* be observed that the leakage of the 32 bits of a single word of the left-hand side operand during a `MMM` processing

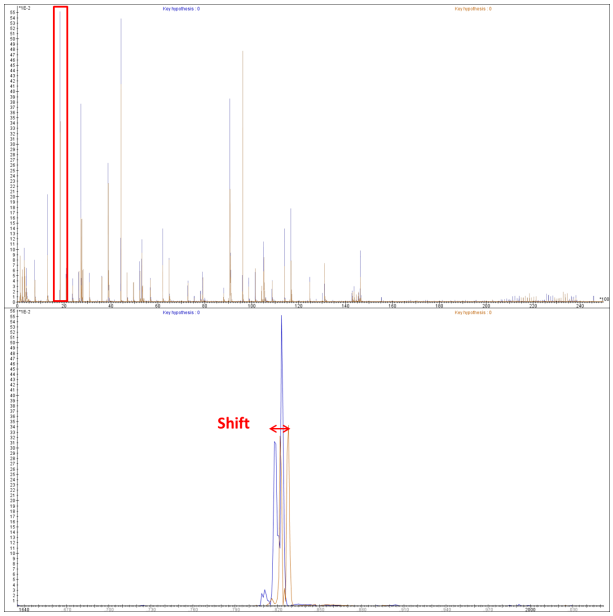


Figure 9: EM Campaign – SNR results on MMM initialization traces using seg_{free} value on 2 different cards.

is unbalanced. Indeed the amplitude of the SNR for the 12th less significant bit of the word is almost hundred times greater than that of the others. Moreover, and as expected, this bit leaks at many different locations. Figure 10 (bottom) *secondly* shows that these characteristics are true for all the 32-bits words composing the `MMM` operand. It also shows that the SNRs corresponding to the 12th bit of the 34 words are not exactly located at the same time samples.

Due to our first observation, we decided to restrict the recovery to the most leaking bits, *i.e.* all the 12th bits of the 32-bits words composing the operand.[4] As a consequence of our second observation, a different set of points of interest has been recorded for the 12th bit of each operand word.

[4]Since this operand is assumed to have size $1088 = 34 \times 32$, the number of 32-bits words per 1088-bits operand, and hence the number of guessed bits, is 34.

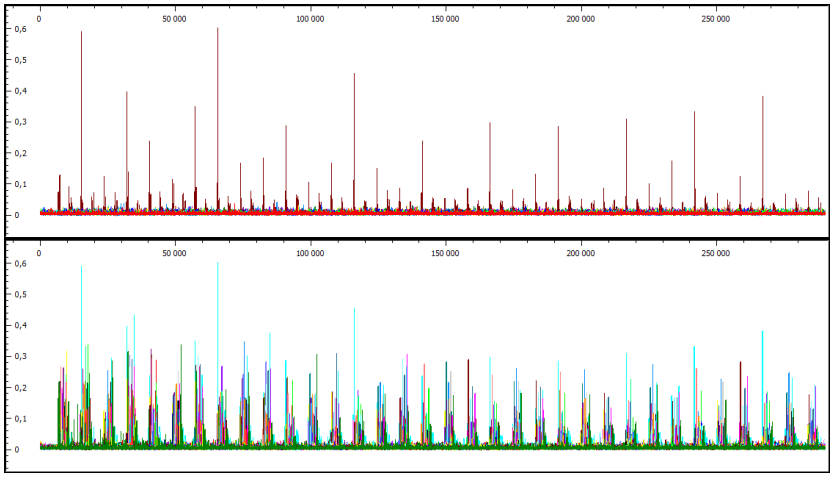


Figure 10: Monobit SNRs (on 50,000 traces) for the first operand of the `MMM`. Top: 32 SNR traces for each bit of *the least significant* 32-bit word. Bottom: 34 SNR traces for *all* the 12th bit of each 32-bit sub-word of the operand.

5 Attacks Description and Results

5.1 Registers Manipulation (Power Consumption)

To exploit the information revealed by the characterization described in Subsubsection 4.1.1, several acquisition campaigns with a RSA masked modulus of bit-length $n = 1088$ have been launched with a sampling rate equal to 50 MS/s. Three different samples of the target smart-card have been used, which led to split the acquisitions sets as follows:

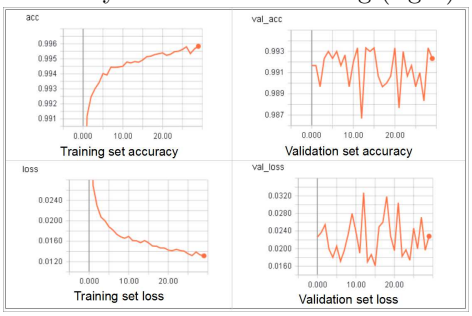
- Set C0: the card #0 is used for the profiling/training stage over the 100 first multiplication patterns from the modular exponentiation. The power consumption has been measured for 2,016 RSA execution. Hence, the set is composed of $201,600 = 2,016 \times 100$ multiplication patterns.
- Set C1: the card #1 is used as an evaluation/validation set for the training of deep learning algorithms. The 100 first multiplication patterns of 30 traces have been recorded. The set is hence composed of 3,000 multiplication patterns.
- Set C2: the power consumption of a full randomized modular exponentiation has been measured on card #2. It is the attacked set for the exploitation/testing stage. The set is composed of 1088 multiplication patterns.

For comparisons, several profiled approaches have been tested on the campaigns. They have been performed on a 256 Gb RAM calculation server with NVIDIA Tesla P100-PCIE-16Gb. The scores listed in the following table for all the attacks correspond to the percentage of all the 1088 exponent bits which are revealed once the attack has been performed against seg_{free} for a single trace in our set C2.

Some of the attacks listed above are very impacted by the size of the sub-traces on which they are applied (e.g. TA or KNN). To get a fair comparison, all the profiled attacks have hence been performed after reducing the sub-traces $\mathbf{L}_i^{\text{mult.}}$ to 9 Points of Interest (POI for short) selected thanks to the characterization described in Subsubsection 4.1.1 (see Figure 7). We insist here on the fact that, after resynchronization, this is exactly the same points which are kept for all the sub-traces (used for profiling or for attack).

Table 2: Attacks success rates (left), loss and accuracy for the CNN training (right).

Attack type	*Best score*
Template attack (TA) [CJRR99]	83.4%
Random Forest [MLB11]	93.1%
Support Vector Machine [HGM+11, ZGLG14]	97.1%
MLP [HGM+11, MDM16]	98.38%
K-Nearest Neighbors (KNN) [Bis07]	98.7%
Extreme Gradient Boosting [Bis07]	98.7%
Convolutional Neural Network [Bis07]	99.31%



For template attacks, and following the analysis done in [FR14], different considerations regarding the covariance matrix have been tested: no covariance matrix (*i.e.* replaced by identity matrix), a covariance matrix for each class, common "pooled" covariance matrix (*i.e.* average of the covariance matrices over all the classes). Neural networks and machine learning parameters have been optimized using various approaches such as *grid search* and *Bayesian optimization* (see e.g. [BB12] for an argumentation of this approach). More specifically hyperas/hyperopt library [Pum18, Ber18] was used for CNN.

The CNN has been trained with the acquisitions in set C0. The acquisitions in C1 have been used as an evaluation set. The obtained training accuracy is 99.64% while the accuracy for the evaluation set is 99.27%. The figure in Table 2 gives the evolution of the *accuracy* and the *loss* as a function of the number of *epochs*, for both the training and validation sets. We recall that the accuracy evaluates how accurate is the model prediction compared to the given (true) labelling, while the loss (which is the value that aims to be minimized during the training) is a summation of the errors made when comparing the trained algorithms outputs and the (true) labels. Remarkably, the results immediately converge around 99% and *over-fitting* on the validation set quickly arises; it is thus decided to stop the learning very quickly, at epoch 7.

After the recovery of 99.31% of the full randomized exponent, the wrong guesses can be corrected thanks to [SW14b]. To apply the latter correction, the attack has to be repeated against 15 full randomized exponents (*i.e.* 15 full RSA processings) to reveal the secret exponent in clear.

The architectures of the trained MLP and CNN models are given in Figure 11 and Figure 12.

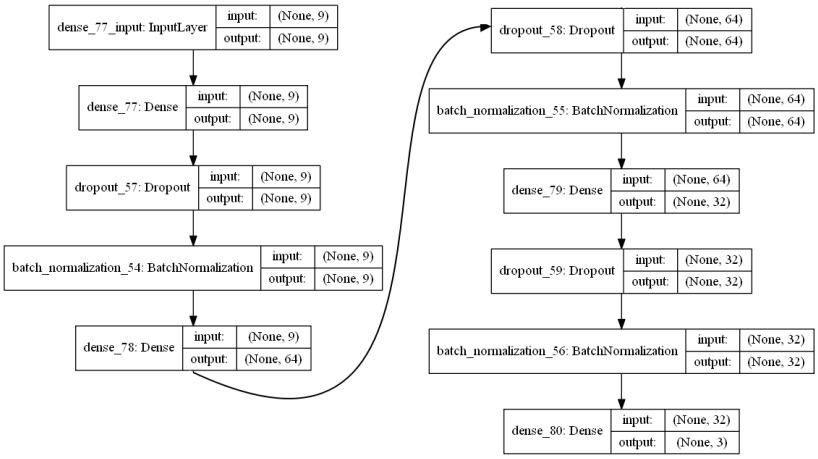


Figure 11: Architecture of the best trained MLP

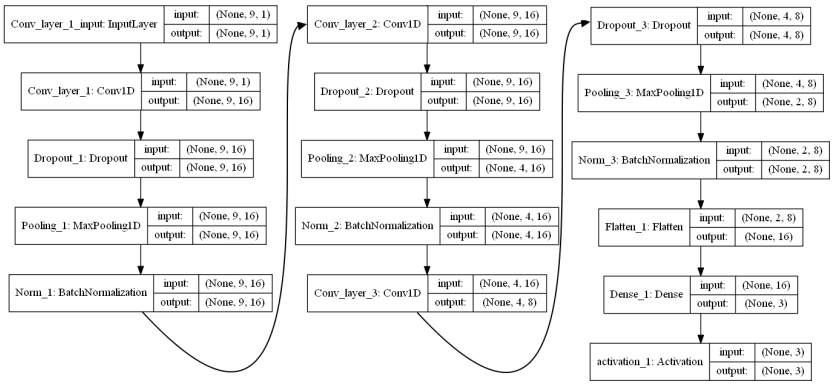


Figure 12: Architecture of the best trained CNN

5.2 Registers Manipulation (EM)

To exploit the information revealed by the characterization described in Subsubsection 4.1.2, deep learning attacks targeting the value of the register seg_{free} have then been applied on the electromagnetic measurements. For this attacks' campaign, taking into account the comparisons' results presented in previous section, we chose to only test MLP and CNN architectures. Since it was no longer needed to scale the dimension of the exploited traces to comply with the capacity of other profiling techniques, this choice allowed us to test the efficiency of the latter models almost directly on the squaring initialization sub-traces identified in Subsection 3.2. In other words we did not select a small number of POI and directly took the $13,000$ first time samples among $30,000$ in the `MMM` initialization pattern of each sub-trace (the SNR characterization reported in Subsection 4.2 indeed shows that the amount of useful information in the second half of the sub-traces is much less than that in the first half). The training and testing have been done on a high-end desktop PC, composed notably of 128 Gb RAM and an NVIDIA TITAN Xp graphics card.

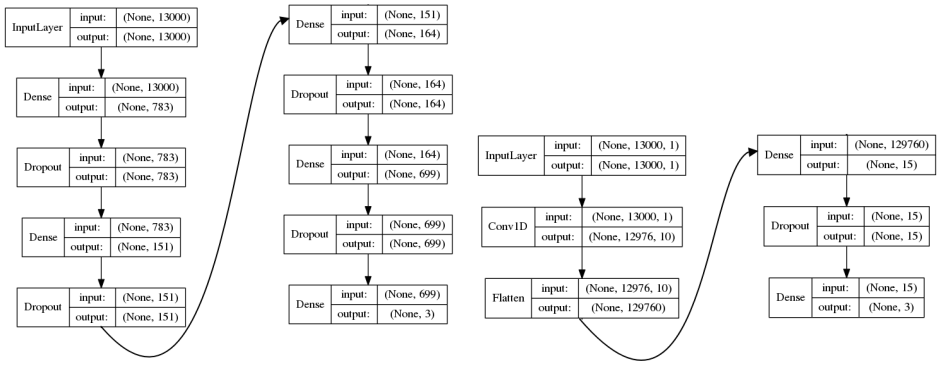
Similarly as reported in Subsection 5.1, the architectures and the hyper-parameters have been searched by exhaustively testing all values in specific intervals carefully chosen to bound the overall computational effort. First, the profiled attacks have been tested on the sub-traces re-aligned as described in Subsection 3.2. Then, to validate the robustness of CNN models to de-synchronization effects, they have also been tested on *raw* sub-traces without specific re-alignment.

Aligned traces. For the training phase, datasets of $750,000$ aligned traces acquired on the 3 different cards have been grouped to form the training dataset C0 and the validation dataset C1 (10% of the training dataset). For the testing phase, a dataset C2 of $10,000$ aligned traces acquired on a fourth card has been built. The whole $13,000$ first time samples of the sub-traces described in Subsection 3.2 are used to input the models.

A first *standardization step* [GBCB16, Section 12.2.1] has been applied on the datasets in order to help the training by cancelling the mean of each sample and by equalizing the covariances (see also [LBOM12]). To get a rough idea of the impact of the different architectures and hyper-parameters tuning on the model accuracy, the generation of ten random models has been launched. Each model has been trained over 3 epochs and this process has been done for both MLP and CNN architectures. Then, different metrics (as e.g. the accuracy and the loss) have been involved to compare them and to eventually select the best parameters. For each of the ten networks, these metrics have been analysed in order to assess the performance of the networks with regards to their architecture (number of layers, number of neurons per layers, etc.) and their hyper-parameters (learning

rate, regularization rate, etc.). This analysis has allowed us to identify the best network parameters and hyper-parameters amongst the randomly generated ones.

In a second step, the selected MLP and CNN models have been selected and directly applied to the 10,000 sub-traces in dataset C1. The accuracy of the best MLP model and CNN model was respectively 99.87%, and 99.7%. The architecture of the best MLP model is given in Figure 13a and the best CNN model is given in Figure 13b.



(a) Best MLP architecture **(b)** Best CNN architecture

Figure 13: Best neural networks architectures on aligned traces.

A more in-depth tuning of the hyper-parameters has been launched on the best CNN model in order to improve it. In particular, the number of epochs, the batch size, the optimizer algorithms have been tested thoroughly. Finally, different initialization weights have been set for the architectures, hence resulting in different initial conditions for the networks. An *ensemble learning technique* [Zho12, Section 4.2] has then been applied on the different initialized networks. Several copies of the same network with different initializations have eventually been trained and their results have been averaged (aka cross-validated). The best CNN model has then been repeated 3 times using this ensemble learning method. This technique gave the best accuracy for our CNN model, namely 99.91%.

Raw traces. One of the main advantages of the CNN models is the temporal invariance of the convolution layers. This theoretically allows neural networks to be immune against desynchronization. Therefore, CNN models have also been trained on 750,000 electromagnetic traces without the alignment steps (corresponding to Figure 5b) and applied to 10,000 traces from another card.

Contrary to the aligned traces case, the standardization pre-processing technique has no sense on desynchronized traces. Hence, the datasets values have only been scaled in the range $[0, 1]$. Then, a random search over 20 networks has been launched. The training has been performed over 10 epochs (instead of 3 due to the greater difficulty for the networks in finding relevant patterns in desynchronized data). On aligned traces, most of randomly generated architectures gave significant results. Here, only a few of the 20 generated architectures gave high accuracies ($> 90\%$). The best model accuracy was *only* 93.3%. To improve it, a new layer called *batch normalization* [IS15] has been integrated. This additional layer has been included in the random architecture search and 10 new neural networks have been generated. Using the *batch normalization* layers in CNN models, the best model accuracy was 97.7%.

The architecture of the best CNN model with *batch normalization* is given in Figure 14.

Table 3 summarizes the accuracy of the deep learning attacks targeting the register

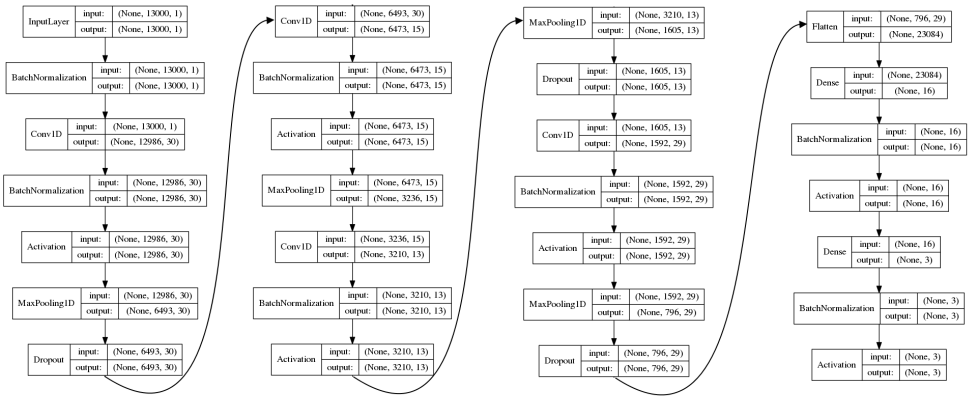


Figure 14: Best CNN with *batch normalization* architecture on raw traces.

manipulation on the electromagnetic traces. The percentage corresponds to the ability of the trained model to recover a bit of the masked exponent. A success rate of 97% (or greater) is considered to be sufficient to recover the full exponent with non-negligible probability (possibly combined with techniques as in [SW14a]).

Table 3: Attacks summary.

Attack type	*Best score*
Multi-Layer Perceptron on aligned traces	99.7%
Convolutional Neural Network on aligned traces	99.91%
Convolutional Neural Network on raw traces	97.7%

5.3 Values Manipulation (EM)

Based on the analysis recalled in Subsection 4.2, the goal is to recover, from two EM leakages $\mathbf{L}_i^{\text{mult.}}$ and $\mathbf{L}_{i+1}^{\text{sq.}}$, the twelve bit of each of the 34 words composing the two left-hand side operands $\mathbf{a}_i^{\text{mult.}}$ and $\mathbf{a}_{i+1}^{\text{sq.}}$ of consecutive `MMM` processings, before performing a comparison between the two bit sets and decide whether $\mathbf{a}_i^{\text{mult.}}$ equals $\mathbf{a}_{i+1}^{\text{sq.}}$ or not (which directly gives the corresponding exponent bit, see Subsubsection 2.3.2).

First stage: training. For the reasons discussed in Subsection 4.2, for each of the 34 words, a new model has been trained to recover the 12th bit from a single observation. The database C0 for the training was composed of $10,000$ traces, each of size of $300,000$ time samples corresponding to a single `MMM` processing. Since the latter size was unnecessarily too high, the SNR characterization reported in Subsection 4.2 has been priorly used to reduce them to $5,000$ points of interest, which were different for each word. This led to the definition of 34 training databases of $10,000$ traces composed of $5,000$ samples: the first $8,000$ traces were dedicated to the training, while the remaining $2,000$ traces were used for validation. The traces in the i-th database are labelled by the 12th bit of the i-th word of the left-hand side operand of the corresponding `MMM` processing. Then, the CNN model depicted on Figure 15 has been trained onto each database. As for previous trainings, we observed that the validation accuracy quickly converged to the training accuracy (after only 2 epochs).

Second stage: guessing values. The 34 trained models have then then been tested on a new test database C2 composed of $1,400$ traces corresponding to different single `MMM`

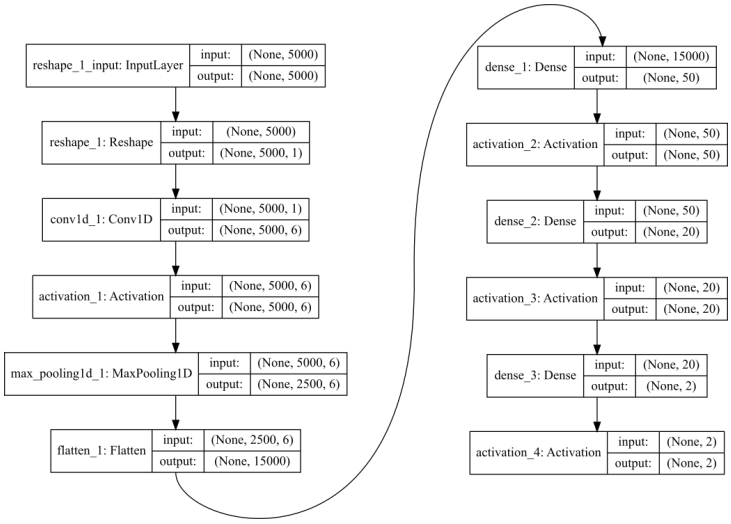


Figure 15: CNN architecture.

processings. Before each of the 34 tests, the traces have been reduced to $5,000$ samples by selecting the corresponding points of interest identified during the signal characterization. For the $1,400$ traces, the success rate of our 34 models ranges from 88% to 99.5%, with an average of 95%.

Third stage: recovering the exponent bit. Eventually, the attack is performed on a single trace. For each exponent bit $d'[n-1-i]$, $i \in [1..n-1]$, two values have to be recovered, so the 34 trained models are applied twice, on $\mathbf{L}_i^{\text{mult.}}$ and on $\mathbf{L}_{i+1}^{\text{sq.}}$. Before each of the 2×34 attacks, the two targeted sub-traces are reduced to $5,000$ samples by selecting the same points of interest. Based on the 2×34 guessed bits, our purpose was to decide if the left-hand side operands collide or not.

Given the average accuracy of our 34 models (to correctly recover the twelve bit of a given 32-bits word), it may be checked that only 26 bits (or more) among the 34 ones have to be equal in order to accurately decide that there is a collision or a non-collision, and hence that the corresponding exponent bit is 1 or 0 (a detailed argumentation of this point is given in Appendix A). Eventually, with a recovering of 34 bits of each operand with an average success rate of 95%, we were able to recover all the exponent bits with a success rate close to 100% (except for the least significant one which was recovered by exhaustive testing).

6 Conclusion

In this paper, the results of several profiled/supervised side-channel attacks against a secure implementation of the RSA algorithm have been described and discussed. The implementation was running on a ARM Core SC 100 with a secure arithmetic co-processor. The work has been co-jointly done by three different teams, each working on a specific attack path. For completeness, and because this information is often missing in the literature while being of important practical interest, we have detailed all the attack steps (identification of the attack paths, acquisitions, pre-processing, attacks) that are usually followed to evaluate the security of an implementation in a security laboratory. The attacks results, which exploit different types of leakages measured either through the power consumption or the electromagnetic emanation of the devices, show the high potential of

deep learning attacks (and in particular CNN models) against secure implementations of RSA. The architectures of the best trained models are given for information. They however strongly depend on the device, the targeted algorithm and the measurements campaigns, and even if some general design principles may be reused, it is very likely that they cannot be directly applied on a context differing on one of the latter points. Usually, a EAL4+ certified arithmetic co-processor leaks much less information than what was observed in this paper. We hence expect the attack to be more difficult to directly apply on devices certified in the SOG-IS scheme, especially if the security guidelines of the chip are correctly followed (which is verified by the evaluation laboratory during the testing of the software part). To remove the first attack path exploiting the dependence between the secret exponent bits and the variable addresses, countermeasures exist like e.g. the randomization of the roles of the registers used during the exponentiation [IIT03]. Dealing with the second attack path is more tricky. Replacing the Square & Multiply Always exponentiation algorithm by another one, like for instance the Montgomery Ladder [JY02, Mon87], is not a solution since collisions are still exploitable (as argued in [BJPW13]). A possible approach is to frequently re-randomize the internal state as proposed in [BJPW13] or to re-randomize the output of each modular operations as proposed in [DV11]. Another possibility is to mix (in a random order) several exponentiation routines. A careful analysis of the security and efficiency of these countermeasures and/or the design of new ones are open avenues for further research.

Acknowledgements

The target implementation used in this paper to present the different attacks is part of a challenge organized by ANSSI's laboratory of hardware security for industrial partners. It has been developed by CryptoExperts (https://www.cryptoexperts.com/) who deliberately did not include countermeasures against horizontal and address-bit attacks. We would like to thank them, and also people who were involved in the project at ANSSI: Soline Renner, Manuel San-Pedro, Adrian Thillard and Jérôme Vidal. Eventually, we thank Victor Lomné from Ninjalab (https://ninjalab.io/) who performed preliminary tests to validate the challenge (and hence the target security).

References

[Arr18] Common Criteria Recognition Arrangement. Common Criteria Portal. https://www.commoncriteriaportal.org/, 2018.

[BB12] James Bergsta and Yoshua Bengio. Random search for hyper-parameter optimization. *Jounal of Machine Learning Research*, 13(Feb):281–305, 2012.

[BDGN14] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Side-channel leakage and trace compression using normalized inter-class variance. In Ruby B. Lee and Weidong Shi, editors, *HASP 2014, Hardware and Architectural Support for Security and Privacy, Minneapolis, MN, USA, June 15, 2014*, pages 7:1–7:9. ACM, 2014.

[Ber18] James Bergstra. Hyperopt: Distributed Asynchronous Hyperparameter Optimization in Python. https://github.com/hyperopt/hyperopt, 2018.

[Bis07] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.

[BJPW13] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and vertical side-channel attacks against secure RSA implementations. In Ed Dawson, editor, *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[BLR13] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient Template Attacks Based on Probabilistic Multi-class Support Vector Machines. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications CARDIS*, volume 7771 of *Lecture Notes in Computer Science*, pages 263–276. Springer Berlin Heidelberg, 2013.

[BMV05] Lejla Batina, Nele Mentens, and Ingrid Verbauwhede. Side-channel issues for designing secure hardware implementations. In *11th IEEE International On-Line Testing Symposium (IOLTS 2005), 6-8 July 2005, Saint Raphael, France*, pages 118–121. IEEE Computer Society, 2005.

[BÖPV03] Lejla Batina, Siddika Berna Örs, Bart Preneel, and Joos Vandewalle. Hardware architectures for public key cryptography. *Integration*, 34(1-2):1–64, 2003.

[BSS05] Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart, editors. *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2005.

[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*, volume 6476 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2010.

[CJRR99] S. Chari, C.S. Jutla, J.R. Rao, and P. Rohatgi. A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. In *Second AES Candidate Conference – AES 2*, March 1999.

[Cor99] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Koç and Paar [KP99], pages 292–302.

[CRR02] S. Chari, J.R. Rao, and P. Rohatgi. Template Attacks. In Kaliski Jr. et al. [KJKP02], pages 13–29.

[DPRS11] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate Side Channel Attacks and Leakage Modeling. *Journal of Cryptographic Engineering*, 1(2):123–144, 2011.

[DV11] Vincent Dupaquis and Alexandre Venelli. Redundant modular reduction algorithms. In *International Conference on Smart Card Research and Advanced Applications*, pages 102–114. Springer, 2011.

[fIS18] Federal Office for Information Security. *Minimum Requirements for Evaluating Side-Channel Attack Resistance of RSA, DSA and Diffie-Hellman Key Exchange Implementations*. BSI, 2018.

[FR14] 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*. Springer, 2014.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GHO15] Richard Gilmore, Neil Hanley, and Máire O'Neill. Neural network based attack on a masked implementation of AES. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015*, pages 106–111. IEEE Computer Society, 2015.

[Gir06] C. Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, September 2006.

[HGM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptographic Engineering*, 1(4):293–302, 2011.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2012.

[IIT02] K. Itoh, T. Izu, and M. Takenak. Address-bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In Kaliski Jr. et al. [KJKP02], pages 129–143.

[IIT03] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. A Practical Countermeasure against Address-Bit Differential Power Analysis. In C.D. Walter, Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 382–396. Springer, 2003.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[JY02] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In Kaliski Jr. et al. [KJKP02], pages 291–302.

[KJKP02] B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors. *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*. Springer, 2002.

[KP99] Ç.K. Koç and C. Paar, editors. *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*. Springer, 1999.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack: an approach based on machine learning. *IJACT*, 3(2):97–115, 2014.

[LBOM12] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer, 2012.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES. In Francillon and Rohatgi [FR14], pages 61–75.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 20–33. Springer, 2015.

[MAB+18] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Muneeb Yousaf, Umer Farooq, Vianney Lapotre, and Guy Gogniat. Machine Learning For Security: The Case of Side-Channel Attack Detection at Runtime. In *ICECS-2018*, Bordeaux, France, December 2018.

[MDM16] Zdenek Martinasek, Petr Dzurenda, and Lukas Malina. Profiling power analysis attack based on MLP in DPA contest V4.2. In *39th International Conference on Telecommunications and Signal Processing, TSP 2016, Vienna, Austria, June 27-29, 2016*, pages 223–226. IEEE, 2016.

[MDS99a] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *the USENIX Workshop on Smartcard Technology (Smartcard '99)*, pages 151–161, 1999.

[MDS99b] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcard. In Koç and Paar [KP99], pages 144–157.

[Mem18] SOG-IS Members. SOG-IS Portal, 2018.

[MLB11] Olivier Markowitch, Liran Lerman, and Gianluca Bontempi. Side channel attack: An approach based on machine learning. In *2nd International Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE 2011*, February 2011.

[Mon87] P.L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48:243–264, 1987.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks – Revealing the Secrets of Smartcards*. Springer, 2007.

[PSB+18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.

[Pum18] Max Pumperla. Keras + Hyperopt: A very simple wrapper for convenient hyperparameter optimization. https://github.com/maxpumperla/hyperas, 2018.

[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, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 534–554. Springer, 2017.

[Sch08] Werner Schindler. Advanced Stochastic Methods in Side Channel Analysis on Block Ciphers in the Presence of Masking. *Journal of Mathematical Cryptology*, 2:291–310, 2008.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.

[SW14a] Werner Schindler and Andreas Wiemers. Power attacks in the presence of exponent blinding. *J. Cryptographic Engineering*, 4(4):213–236, 2014.

[SW14b] Werner Schindler and Andreas Wiemers. Power attacks in the presence of exponent blinding. *J. Cryptographic Engineering*, 4(4):213–236, 2014.

[VKMJ10] B. V. K. Vijaya Kumar, Abhijit Mahalanobis, and Richard D. Juday. *Correlation Pattern Recognition*. Cambridge University Press, New York, NY, USA, 2010.

[vWWB11] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 104–119. Springer, 2011.

[Wal01] Colin D. Walter. Sliding windows succumbs to big mac attack. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 286–299. Springer, 2001.

[ZGLG14] Zhong Zeng, Dawu Gu, Junrong Liu, and Zheng Guo. An improved side-channel attack based on support vector machine. In *CIS*, pages 676–680. IEEE Computer Society, 2014.

[Zho12] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

A Optimal Number of Binary Collisions to Decide on the Operands' Equality

Assuming that 34 bits of two binary values have been correctly recovered with some probability, our goal is to optimize our ability to correctly assess that the two values are equal each other.

A direct approach, named T, is to simply test whether the two returned sets of 34 bits equal or not and conclude about the collison. The success rate of T will increase with the success rates of the involved models. As all the 2×1088 bits are not known, we are not sure of success, but a complete recovering of each operand is not necessary to suspect a collision. The success rate of T depends on the probability to correctly guess the collision but also to correctly guess the non-collision. From 6 we remark the probability of false negative should decrease by considering less bits. This phenomenon is advantageous in a certain limit as the probability of false positive increases in the same time.

A good compromise leads to the second strategy, named S, that consists in deciding the collision from the equality of possibly less bits among the 34 bits. Equation 7 shows that this number of bits which must match also depends on the success rates of the involved models. For instance, if this success rate is 95% (which is almost case for our experiments), then only 26 bits (or more) among the 34 ones have to be equal in order to accurately decide that there is a collision or a non-collision (and hence that the the corresponding exponent bit is 1 or 0).

Proposition 1. *We suppose the distribution of the values and the masked exponent is almost uniform, and the* 68 *attacks are independent. We denote* p *the probability to correctly predict a bit and* $\alpha = p^2 + (1-p)^2$.
The probability to correctly guess that $\mathbf{a}_i^{\text{mult.}} = \mathbf{a}_{i+1}^{\text{sq.}}$ *and* $\mathbf{a}_i^{\text{mult.}} \neq \mathbf{a}_{i+1}^{\text{sq.}}$ *from the equality of the* 34 *targeted bits (event named* T*) is estimated by:*

$$\tau(T) \approx \frac{1}{2}\left(\alpha^{34} + 1 - \frac{1}{2^{34}}\right) \tag{6}$$

The probability to correctly guess that $\mathbf{a}_i^{\text{mult.}} = \mathbf{a}_{i+1}^{\text{sq.}}$ *and* $\mathbf{a}_i^{\text{mult.}} \neq \mathbf{a}_{i+1}^{\text{sq.}}$ *from the equality of only* c *bits among the* 34 *targeted bits (event named* $S(c)$ $c \in [0..34]$*) is estimated by:*

$$\tau(S(c)) \approx \frac{1}{2}\left(\sum_{c \leq j \leq 34} \binom{34}{j} \alpha^j (1-\alpha)^{34-j} + 1 - \frac{\sum_{c \leq j \leq 34} \binom{34}{j}}{2^{34}}\right) \tag{7}$$

The success rates are drawn for several guessing probabilities p in Figure 16. We remark that guessing a operand bit with more than 83% success is enough to achieve at least 90% success to recover the bit exponent.

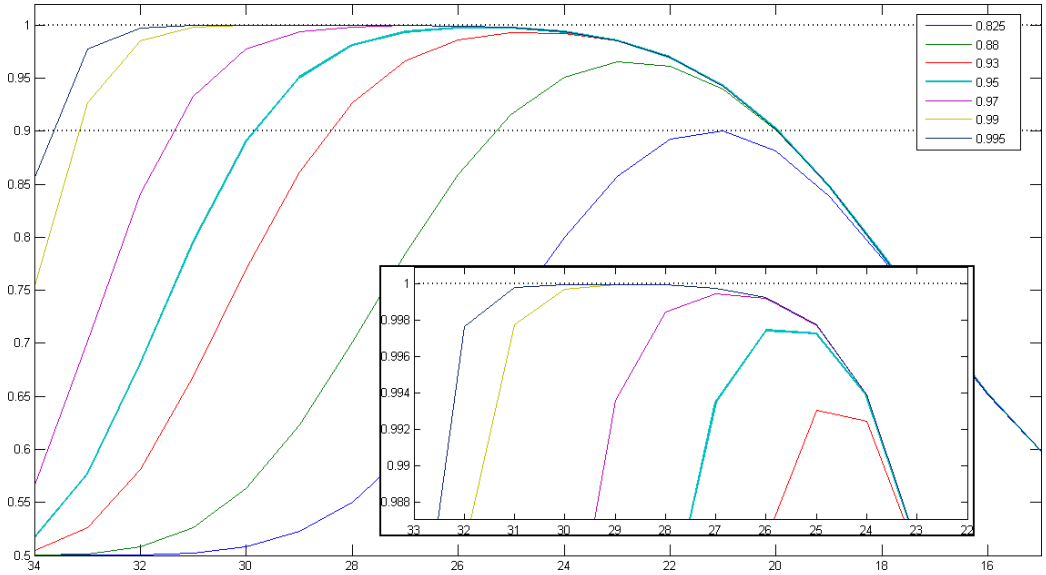


Figure 16: Evolution of the success rate $\tau(S(c))$ (ordinate) in function of c (abscissa) for different probabilities p (colors). A zoom is drawn in the box.

Proof. The two values that collide or not are represented by the random variables $X \in \mathcal{M}$ and $Y \in \mathcal{M}$ where $\mathcal{M} = \{0, \ldots, 2^n - 1\}$. The values of the corresponding bits are represented by the random variables X_i and Y_i where $i \in \mathcal{B} = \{0, \ldots, n-1\}$. We suppose that the distribution of the masked exponent bits is almost uniform, so the collision (bit equal to 0) has same probability than the non-collision (bit equal to 1).

$$\mathsf{Pr}\left[X = Y\right] = \mathsf{Pr}\left[X \neq Y\right] = \frac{1}{2}. \tag{8}$$

We also suppose that the realizations of X and Y are uniformly distributed between 0 and $2^n - 1$. This is not exactly correct because they are modular values. So $\forall x \in \mathcal{M},\ \mathsf{Pr}\left[X = x\right] = \frac{1}{2^n}$ and $\forall y \in \mathcal{M},\ \mathsf{Pr}\left[Y = y\right] = \frac{1}{2^n}$. This implies:

$$\mathsf{Pr}\left[X = x \text{ and } Y = x | X = Y\right] = \frac{1}{2^n}, \tag{9}$$

and

$$\mathsf{Pr}\left[X = x \text{ and } Y = y | X \neq Y\right] = \frac{1}{2^{2n} - 2^n} = \frac{1}{2^n(2^n - 1)}. \tag{10}$$

The two guessed values are represented by the random variables $G_X \in \mathcal{M}$ and $G_Y \in \mathcal{M}$. The values of the corresponding bits are represented by the random variables $G_{X\,i}$ and $G_{Y\,i}$ where $i \in \mathcal{B}$. Here we only guess the bit number 12 of the 32-bit words, so we define the set of these bit indexes $\mathcal{B}_{12} = \{j | j \in \mathcal{B} \text{ and } j \mod \frac{n}{32} = 12\}$.

We denote $p_i(a, b)$ the probability to predict that the bit $G_{X\,i}$ (resp. $G_{Y\,i}$) equals to b knowing that it equals to a in reality. So $\forall i \in \mathcal{B}$ and $\forall (a, b) \in \{0, 1\}^2$:

$$p_i(a, b) = \mathsf{Pr}\left[G_{X\,i} = b | X_i = a\right] = \mathsf{Pr}\left[G_{Y\,i} = b | Y_i = a\right] \tag{11}$$

The following probabilities are defined $\forall i \in \mathcal{B}$ and will be used later:

$$\alpha_i = \mathsf{Pr}\left[G_{X\,i} = G_{Y\,i} | X_i = Y_i\right] = \frac{1}{2} \sum_{(a,b)\in\{0,1\}^2} p_i(a,b)^2 \tag{12}$$

$$\gamma_i = \mathsf{Pr}\left[G_{X\,i} \neq G_{Y\,i} | X_i = Y_i\right] = \sum_{a\in\{0,1\}} p_i(a,0)p_i(a,1) \tag{13}$$

$$\beta_i = \mathsf{Pr}\left[G_{X\,i} = G_{Y\,i}\right] = \frac{1}{4} \sum_{(a,a',b)\in\{0,1\}^3} p_i(a,b)p_i(a',b) \tag{14}$$

$$\delta_i = \mathsf{Pr}\left[G_{X\,i} \neq G_{Y\,i}\right] = \frac{1}{2} \sum_{(a,a')\in\{0,1\}^2} p_i(a,0)p_i(a',1) \tag{15}$$

First we compute the probability of predicting a collision knowing that the two values really collide. We suppose that the 68 attacks are independent, so the probability of guessing one bit $G_{X\,i}$ or $G_{Y\,i}$ only depends on the true value, *i.e.* it does not depend on the others bits.

For a subset of bits $\mathcal{B}' \subseteq \mathcal{B}_{12}$ and for a set of correct bit values $(b_i)_{\forall i\in\mathcal{B}_{12}}$, we define $G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}})$ the fact of correctly guessing the bits of the subset and of being wrong about the others.

$$\begin{aligned} G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}}) \Leftrightarrow \quad & \forall i \in \mathcal{B}', G_{X\,i} = G_{Y\,i} = b_i \\ \text{and} \quad & \forall j \notin \mathcal{B}', G_{X\,j} = b_j, G_{Y\,j} = 1 - b_j \end{aligned} \tag{16}$$

Given a subset $\mathcal{B}' \subseteq \mathcal{B}_{12}$ of correct guessed bits, the probability of predicting a collision knowing this is true is:

$$P_{ok}(\mathcal{B}') = \mathsf{Pr}\left[G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}}) | X = Y\right] \tag{17}$$

$$= \sum_{\substack{b_i\in\{0,1\} \\ \forall i\in\mathcal{B}_{12}}} \sum_{x\in\mathcal{M}} \mathsf{Pr}\left[X = x | X = Y\right] \mathsf{Pr}\left[G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}}) | X = Y = x\right] \tag{18}$$

The sum on all possible values $x \in \mathcal{M}$ can be rewritten as multiple sums on each bit $x_i \in \{0,1\}$ of x.

$$P_{ok}(\mathcal{B}') = \frac{1}{2^n} \sum_{\substack{b_i\in\{0,1\} \\ \forall i\in\mathcal{B}_{12}}} 2^{n-|\mathcal{B}_{12}|} \sum_{\substack{x_i\in\{0,1\} \\ \forall i\in\mathcal{B}_{12}}} \mathsf{Pr}\left[G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}}) | X = Y = x\right] \tag{19}$$

$$= \frac{1}{2^{|\mathcal{B}_{12}|}} \sum_{\substack{b_i\in\{0,1\} \\ \forall i\in\mathcal{B}_{12}}} \sum_{\substack{x_i\in\{0,1\} \\ \forall i\in\mathcal{B}'}} \sum_{\substack{x_j\in\{0,1\} \\ \forall j\notin\mathcal{B}'}} \mathsf{Pr}\left[\forall i \in \mathcal{B}', G_{X\,i} = G_{Y\,i} = b_i | X = Y = x\right] \times \mathsf{Pr}\left[\forall j \notin \mathcal{B}', G_{X\,j} = b_j, G_{Y\,j} = 1 - b_j | X = Y = x\right] \tag{20}$$

We separate the terms related to bits in $\mathcal{B}'$:

$$P_{ok}(\mathcal{B}') = \frac{1}{2^{|\mathcal{B}_{12}|}} \sum_{\substack{b_i\in\{0,1\} \\ \forall i\in\mathcal{B}'}} \sum_{\substack{x_i\in\{0,1\} \\ \forall i\in\mathcal{B}'}} \prod_{i\in\mathcal{B}'} \mathsf{Pr}\left[G_{X\,i} = b_i | X_i = x_i\right]^2 \times \sum_{\substack{b_j\in\{0,1\} \\ \forall j\notin\mathcal{B}'}} \sum_{\substack{x_j\in\{0,1\} \\ \forall j\notin\mathcal{B}'}} \prod_{j\notin\mathcal{B}'} \mathsf{Pr}\left[G_{X\,j} = b_j | X_j = x_j\right] \times \mathsf{Pr}\left[G_{Y\,j} = 1 - b_j | Y_j = x_j\right] \tag{21}$$

After factorization, we have:

$$\begin{aligned} P_{ok}(\mathcal{B}') &= \frac{1}{2^{|\mathcal{B}_{12}|}} \prod_{i\in\mathcal{B}'} \sum_{b_i\in\{0,1\}} \sum_{x_i\in\{0,1\}} p_i(x_i,b_i)^2 \\ &\quad \times \prod_{j\notin\mathcal{B}'} \sum_{b_j\in\{0,1\}} \sum_{x_j\in\{0,1\}} p_j(x_j,b_j)p_j(x_j,1-b_j) \qquad (22) \\ &= \frac{1}{2^{|\mathcal{B}_{12}|}} \prod_{i\in\mathcal{B}'} 2\alpha_i \cdot \prod_{j\notin\mathcal{B}'} 2\gamma_j \qquad (23) \\ P_{ok}(\mathcal{B}') &= \prod_{i\in\mathcal{B}'} \alpha_i \prod_{j\notin\mathcal{B}'} \gamma_j \qquad (24) \end{aligned}$$

Given a subset $\mathcal{B}' \subseteq \mathcal{B}_{12}$ of correct guessed bits, the probability of predicting a collision knowing this is false is:

$$\begin{aligned} P_{ko}(\mathcal{B}') &= \mathsf{Pr}\left[G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}})|X\neq Y\right] \qquad (25) \\ &= \sum_{\substack{b_i\in\{0,1\}\\ \forall i\in\mathcal{B}_{12}}} \sum_{\substack{(x,y)\in\mathcal{M}^2\\ x\neq y}} \mathsf{Pr}\left[X=x \text{ and } Y=y|X\neq Y\right] \\ &\quad \times \mathsf{Pr}\left[G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}})|X=x \text{ and } Y=y\right] \qquad (26) \end{aligned}$$

We separate the collisions:

$$\begin{aligned} P_{ko}(\mathcal{B}') &= \frac{1}{2^n(2^n-1)} \sum_{\substack{b_i\in\{0,1\}\\ \forall i\in\mathcal{B}_{12}}} \left(\sum_{(x,y)\in\mathcal{M}^2} \mathsf{Pr}\left[G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}})|X=x \text{ and } Y=y\right] \right. \\ &\quad \left. - \sum_{x\in\mathcal{M}} \mathsf{Pr}\left[G_{\mathcal{B}'}((b_i)_{i\in\mathcal{B}_{12}})|X=x \text{ and } Y=x\right] \right) \qquad (27) \end{aligned}$$

After binary rewritting, we have:

$$\begin{aligned} P_{ko}(\mathcal{B}') = \frac{2^{2\cdot n-2|\mathcal{B}_{12}|}}{2^n(2^n-1)} & \sum_{\substack{(b_i,x_i,y_i)\in\{0,1\}^3\\ \forall i\in\mathcal{B}'}} \prod_{\forall i\in\mathcal{B}'} p_i(x_i,b_i)p_i(y_i,b_i) \\ & \times \sum_{\substack{(b_j,x_j,y_j)\in\{0,1\}^3\\ \forall j\notin\mathcal{B}'}} \prod_{\forall j\notin\mathcal{B}'} p_j(x_j,b_j)p_j(y_j,1-b_j) \\ - \quad \frac{2^{n-|\mathcal{B}_{12}|}}{2^n(2^n-1)} & \sum_{\substack{(b_i,x_i)\in\{0,1\}^2\\ \forall i\in\mathcal{B}'}} \prod_{\forall i\in\mathcal{B}'} p_i(x_i,b_i)^2 \\ & \times \sum_{\substack{(b_j,x_j)\in\{0,1\}^2\\ \forall j\notin\mathcal{B}'}} \prod_{\forall j\notin\mathcal{B}'} p_j(x_j,b_j)p_j(x_j,1-b_j) \qquad (28) \end{aligned}$$

After simplification with the above defined probabilities, we have:

$$P_{ko}(\mathcal{B}') = \frac{2^{2\cdot n-2|\mathcal{B}_{12}|}}{2^n(2^n-1)} \left(\prod_{\forall i\in\mathcal{B}'} 4\beta_i\right)\left(\prod_{\forall j\notin\mathcal{B}'} 4\delta_j\right) - \frac{2^{n-|\mathcal{B}_{12}|}}{2^n(2^n-1)} \left(\prod_{\forall i\in\mathcal{B}'} 2\alpha_i\right)\left(\prod_{\forall j\notin\mathcal{B}'} 2\gamma_j\right) \tag{29}$$

$$= \frac{2^n \prod_{i\in\mathcal{B}'} \beta_i \prod_{j\notin\mathcal{B}'} \delta_j - \prod_{i\in\mathcal{B}'} \alpha_i \prod_{j\notin\mathcal{B}'} \gamma_j}{2^n-1} \tag{30}$$

If the size n is such that $2^n \gg 1$, we approximate:

$$P_{ko}(\mathcal{B}') \approx \prod_{i\in\mathcal{B}'} \beta_i \prod_{j\notin\mathcal{B}'} \delta_j \tag{31}$$

So we deduce the probabilities of prediction with all bits in case of collision or not:

$$\mathsf{Pr}\left[T|X=Y\right] = P_{ok}(\mathcal{B}_{12}) \tag{32}$$

$$\mathsf{Pr}\left[T|X\neq Y\right] = P_{ko}(\mathcal{B}_{12}) \tag{33}$$

and the probabilities of prediction with a minimum number of identical bits in case of collision or not:

$$\mathsf{Pr}\left[S(c)|X=Y\right] = \sum_{s\in S_c} P_{ok}(s) \tag{34}$$

$$\mathsf{Pr}\left[S(c)|X\neq Y\right] = \sum_{s\in S_c} P_{ko}(s) \tag{35}$$

where $\mathcal{P}(\mathcal{B}_{12})$ is the set of subsets of $\mathcal{B}_{12}$, $c \in \{1\ldots|\mathcal{B}_{12}|\}$ and $S_c = \{s|s\in\mathcal{P}(\mathcal{B}_{12}) \text{ and } |s|\geq c\}$.

These computations can be simplified if the probabilities p_i are the same for all the bits. If for all $i\in\mathcal{B}_{12},\ p_i(0,0) = p_i(1,1) = p$ then $\alpha_i = \alpha = p^2+(1-p)^2$ and $\gamma_i = \gamma = 2p(1-p)$ and $\beta_i = \beta = \delta_i = \delta = \frac{1}{2}$. The above probabilities become:

$$\mathsf{Pr}\left[T|X=Y\right] = \alpha^{34} \tag{36}$$

$$\mathsf{Pr}\left[T|X\neq Y\right] \approx \frac{1}{2^{34}} \tag{37}$$

$$\mathsf{Pr}\left[S(c)|X=Y\right] = \sum_{c\leq j\leq 34} \binom{34}{j} \alpha^j (1-\alpha)^{34-j} \tag{38}$$

$$\mathsf{Pr}\left[S(c)|X\neq Y\right] \approx \sum_{c\leq j\leq 34} \binom{34}{j} \frac{1}{2^{34}} = \frac{\sum_{c\leq j\leq 34}\binom{34}{j}}{2^{34}} \tag{39}$$

At last, the success rates $\tau(T)$ and $\tau(S(c))$, representing the probabilities that the predictions T and $S(c)$ are correct, are estimated by:

$$\tau(T) = \mathsf{Pr}\left[X=Y\right]\mathsf{Pr}\left[T|X=Y\right] + \mathsf{Pr}\left[X\neq Y\right]\mathsf{Pr}\left[\text{not } T|X\neq Y\right] \approx \frac{1}{2}\left(\alpha^{34}+1-\frac{1}{2^{34}}\right) \tag{40}$$

$$\tau(S(c)) \approx \frac{1}{2}\left(\sum_{c\leq j\leq 34}\binom{34}{j}\alpha^j(1-\alpha)^{34-j} + 1 - \frac{\sum_{c\leq j\leq 34}\binom{34}{j}}{2^{34}}\right) \tag{41}$$

□