# Dynamic Runtime Methods to Enhance Private Key Blinding

Karine Gandolfi-Villegas and Nabil Hamzi

Gemalto Security Labs
{nabil.hamzi,karine.villegas}@gemalto.com

**Abstract.** In this paper we propose new methods to blind exponents used in RSA and in elliptic curves based algorithms. Due to classical differential power analysis (DPA and CPA), a lot of countermeasures to protect exponents have been proposed since 1999 Kocher [20] and by Coron [13]. However, these blinding methods present some drawbacks regarding execution time and memory cost. It also got some weaknesses. Indeed they could also be targeted by some attacks such as The Carry Leakage on the Randomized Exponent proposed by P.A. Fouque *et al.* in [23] or inefficient against some others analysis such as Single Power Analysis. In this article, we explain how the most used method could be exploited when an attacker can access test samples. We target here new dynamic blinding methods in order to prevent from any learning phase and also to improve the resistance against the latest side channel analyses published.

**Keywords:** Reverse engineering, exponent blinding, side channel attacks, RSA, ECC

## 1 Introduction

In embedded systems, the usage of cryptographic algorithms may be the target of so-called physical attacks. In the past years, a lot of researchers have dedicated their studies around those subjects. In this paper, we focus on the randomization of a secret data used for exponentiation or scalar multiplication, as used in RSA and in Elliptic Curve based Cryptographic algorithms (for example ECDSA).

The need of randomization comes from attackers who use Side-Channel Analysis (SCA) in order to recover a secret data. Each devices, while running, leaks informations which could be retrieved through the power consumption measurements as first suggested by Kocher *et al.* in [21], by analyzing electromagnetic radiations - as proposed in the paper *Electromagnetic analysis: Concrete Results*, [15], radio-frequencies -in case of contact-less devices, or evaluating its thermal differential... Especially, when this device performs cryptographic algorithms and manipulates secret data, these leakages contain information about the key.

As we focus on exponentiation or scalar multiplication, SCA countermeasures regarding the implementation of these algorithms were designed to prevent such attacks. Kocher *et al.* [20] and Coron [13] first introduced ways to protect

against SCA attacks. Followed by Chevallier-Mames *et al.* [8] and Joye [16] who introduced secure implementations to performs exponentiation and scalar multiplications.

When considering attacks it often appears that countermeasures offer a protection but lend themselves to others attack paths. The randomization of exponents faces this problem. As suggested in *The Carry Leakage on the Randomized Exponent* [23], the operation which masked the secret is targeted and in this case, P.A Fouque *et al.* propose to use the side channel leakage to detect the presence of a carry during an addition with the secret value. This leads them to retrieve the secret with an attack complexity linked to the architecture of the processor used to compute the addition. In [25], W. Schindler for RSA an classical modular exponentiation algorithms and A. Kruger in [22] extends this approach for ECC and sliding window exponentiation algorithms, present attacks that could work even if the exponent, respectively the scalar, is masked. The success of their attacks is linked to the error rate got to find each secret bit or operation. In both cases the size of the random used for masking the secret is the determinant factor to enable those attacks.

S. Bauer [3] showed more recently how an attacker, given some information about each bit of a blinded exponent, can use redundancy in key material to correct observation errors and retrieve the random mask. By repeating this several times, he showed that an attacker could then find a number of blinding factors and then could combine the information of several power traces to determine the secret. In this last case, even large random factors could not be sufficient.

These results highlight the fact that according to the leakage observed, parameters size and application context, the randomization using additive mask could not suffice to protect the secret elements against physical attacks.

In this paper, we show that an attacker could keep the hand on the data he chooses even if there is an additive random mask. Indeed, it is the case as soon as he can access an open device on which he is able to load elements. As an example, we consider Java smart cards on which users can load applets or keys. In this context an attacker can call functions that were designed secure as long as nobody could access elements handled by them. Nevertheless, this last assumption is not respected in the case of open platforms, and the attacker makes functions leak by calling them with specific parameters. The information obtained might be exploitable in the future on unknown data to build attacks and retrieve secrets.

To avoid the described situation, this article is intended to present new runtime blinding methods in order to prevent from any learning phase and also to improve the resistance against the latest side channel analyses published.

The remaining of this paper is organized as follows. Section 2 exposes the weaknesses of the existing exponent blinding methods. In Section 3, we present a new way to protect against leakages whatever the attacker abilities are and at

any step when exponentiation or point multiplication have to be performed with secret element. Our conclusions are drawn in Section 4.

## 2 Weaknesses of exponent blinding methods

Since the first publications on side channel attacks [21], randomization is widely used to protect sensitive elements embedded into devices in order to guarantee physical security. Using algebraic properties a lot of solutions have been already proposed to protect the secret exponent or scalar against side channel attacks. In this section we explore the ones which concern the exponent blinding and we make the assumption that the base is masked to avoid any exploitable leakage or attacks using it such as SPA, DPA or CPA.

### 2.1 Coron's blinding method

A first way is to use the homomorphic property of the modular exponentiation as described in [13] for ECC or in [7] for the RSA. Indeed, we have $m^{a+b} = m^a.m^b \bmod n$.

Given $(n, e)$ the RSA public key and $(n, d)$ the private one (or in case of CRT $(p, q, d_p, d_q, i_q)$) and $\varphi(n)$, the Euler function of $n$ , such that:

$$n = p.q \text{ and } e.d \equiv 1 \bmod \varphi(n) \ , \tag{1}$$

we replace the computation

$$s = m^d \bmod n \ , \tag{2}$$

which is sensitive to physical attacks such as side channel analysis when it is performed in a physical accessible device, by two modular exponentiations

$$s_1 = m^{d^*} \bmod n \text{ s.t. } d^* = d - R, R \text{ a } |d|\text{-bit random} \ , \tag{3}$$

$$s = s_1.m^R \bmod n \ . \tag{4}$$

This method presents an important drawback: the performances. Indeed it doubles the exponentiation time and it needs a large random to be generated and stored.

### 2.2 Addition chain blinding method

Another approach which was presented in [10], is to consider addition chains to re-code the exponent or scalar value. More precisely, we remind that an addition chain of length $l$, for a positive integer $d$ is a sequence $\Gamma(d) = \{d^{(0)}, d^{(1)}, ..., d^{(l)}\}$ and satisfies:

1. $d^{(0)} = 1, d^{(l)} = d$

2. $\forall i, 1 \leq i \leq l, \exists j(i), \exists k(i) < i \mid d^{(i)} = d^{(j(i))} + d^{(k(i))}$.

This provides an easy mean to evaluate $s = m^d \mod n$. Indeed, for $i$ from 1 to $l$, we need to compute $m^{d^{(i)}} = m^{d^{(j(i))}}.m^{d^{(k(i))}}$ and then set $s = m^{d^{(l)}}$. So, from a $l$-length addition chain, $l$ multiplications are required to compute $s$.

Regarding side-channel analysis the main protection when using addition chain is to make it different at each construction. Indeed, if the addition chain is always the same for a given secret, classical SPA or DPA could be applied successively to retrieve each element $d^{(i)}$ and then the secret $d$ could be reconstructed using all values. The use of a random value is then required to generate the addition chain of a given secret if we want that method to be an efficient protection.

However, the use of addition chain to re-code secret presents some drawbacks. The first one is linked to the performances aspect. Thus, the memory storage of the addition chain itself in addition to the secret value must be considered. The second one is linked to side channel and fault injections that could be performed during the addition chain generation and usage.

In this paper, we will not investigate this last method any further and we will focus instead on the next one, Kocher's blinding method, because it is the most used and the least costly.

### 2.3 Kocher's blinding method

Another approach, in case of RSA, using Euler's theorem, is to add a random multiple of the Euler $\varphi$ function to the private exponent. This technique was detailed in the patent [20].

- ◇ For RSA standard mode, $d^* = d + r.\varphi(n)$ with $r$ a $k$-bit random
- ◇ For RSA-CRT mode, $d_p^* = d_p + r_p.(p-1)$ with $d_p$ and $p$ part of the RSA-CRT private key and $r_p$ a $k$-bit random (the same operation is performed on $d_q$ with $q$ and another random $r_q$)

This method is widely used compared to the previous ones as it only induces few additional modular operations. However it could be attacked as presented in [23]. Moreover, if randoms are not large enough it could not be efficient against side-channel analysis performed during modular exponentiation such those described in [25] and [22]. Then and due to the recent results from [3], it might be necessary to prevent an attacker from knowing the links between the parameters, the secret, the random and the values on which he has some knowledge.

In this paper, in addition of the named attacks, we consider the fact that it is possible to have specific parameters to perform or prepare an attack. Either in case of a specific application context which enables to load keys like in JavaCard API or in case of combined attack such as described in [1]. The primes $p$, or $q$ and so the Euler $\varphi$ function, or the curve order for ECC, could be loaded or transformed by a perturbation in a way that $p-1$, $q-1$ or $n$ have at least $\sigma$

bits set to 0. For example, for a prime $p$ with at least $\sigma$ least significant 0-bit, we will have:

$$(p-1) \leftarrow \sum_{i=k}^{i=|p|-1} p_i.2^i \text{ s.t. } k \in \mathbb{N}, k > \sigma \text{ and } p_i \in \{0,1\} \ . \tag{5}$$

$$\text{Given } d_p = \sum_{i=0}^{i=|p|-1} d_{p_i}.2^i, \text{ s.t. } d_{p_i} \in \{0,1\} \ . \tag{6}$$

Then $d_p^* = d_p + r(p-1) = \sum_{i=0}^{i=k-1} d_{p_i}.2^i + \sum_{i=k}^{i=|r.p|-1} d_{p_i}^*.2^i$ we have :

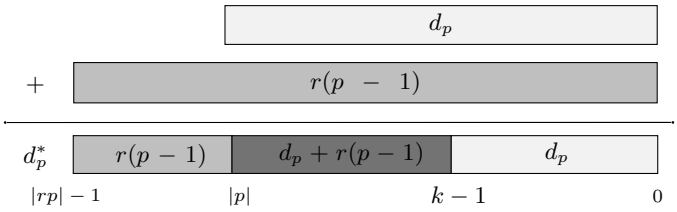


Fig. 1: Bit analysis of a random exponent

In this case it appears that the exponent randomization is partial and the randomized exponent could have at least $\sigma$ bits in common with the original value. Thus, it will enable to make easier the reverse engineering through side channel analysis. The signal processing such as synchronization, averaging, filtering ... or templates extractions will be easier on the traces obtained during the exponentiation with this kind of parameters.

In order to illustrate our comment we have conducted an analysis on several components which embed different crypto-coprocessors and observed the impact on the power or electromagnetic traces with specific parameters that we will name *hollow keys* in the rest of the paper.

The figure 2 represents the power consumption obtained during a square and multiply exponentiation $m^{d_p^*} \mod p$ which processes the exponent from the most significant bits to the least ones. The message is the same for both acquisitions while the modulus and thus $d_p^*$ differ. The component used is a smart card 8-bit micro controller with a public key cryptographic co-processor. The *hollow* parameters are used for the red trace (upper one) and classical RSA key is used for the blue one (lower one). The *hollow* parameters were chosen in order that $d_p$ has a lot of 0 in its least significant bits. At the end of the trace, when least significant bits are processed, we observe that the component does not behave exactly the same. Let's explain this phenomenon.

The first comment is that exponentiation time is not the same, red trace activity is shorter than blue one. It directly implies that less operations have been

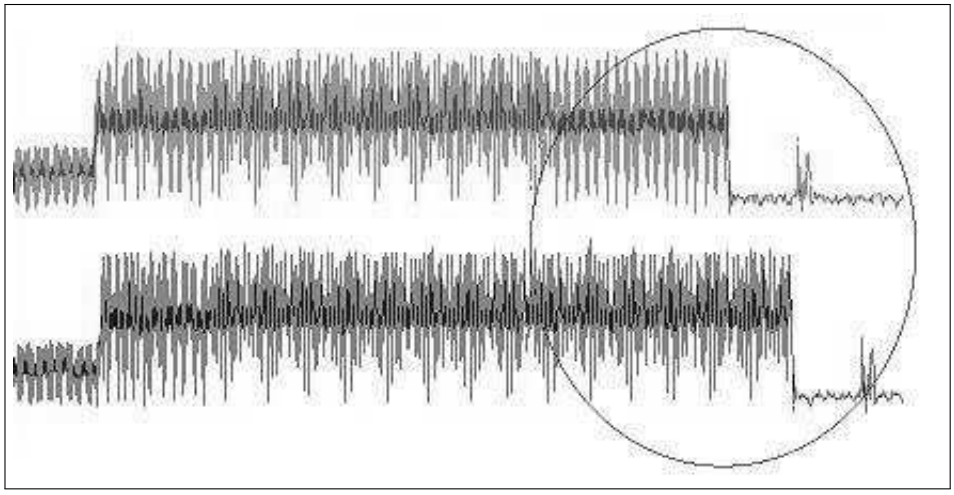

Fig. 2: Modular exponentiations power consumption traces with hollow parameters (red/upper) and classical parameters (blue/lower).

performed and thus the hamming weight of the red trace exponent is lower than the hamming weight of the blue trace exponent. Moreover when least significant bits are treated - we highlight this part by surrounding it on traces of figure 2 - we observed that the component behave differently. We can see that the series of 0 processed in the red trace changes the leakage while in the blue trace as we don't have the change of shape the randomization is still effective.

## 3 New methods to avoid leakages at any stage

In the remaining sections we will define algorithms under the assumption that the core exponentiation algorithm used is SPA-safe. More precisely: it is assumed to be designed according to the component leakage and using classical principles like constant time and code, whichever exponent bit is processed. The base $m$ is also supposed well masked to prevent from cryptanalysis and from physical attacks like DPA and chosen message SPA such as presented in the paper *Simple Power Exponentiation Revisited*, [14].

### 3.1 Extension of Kocher's randomization

We propose a new way to randomize exponents or scalar during the modular exponentiation or point multiplication execution. Even if the Euler $\varphi$ function of the numbers used for the randomization or the number itself is *hollow*, this method will control the link between the secret data and the randomized one. To do so, let's randomize a large number $x$ in $\mathbb{N}$ based on Kocher and Coron's randomization:

$$x^* = x + r.(\eta - u) \text{ s.t. } u, r \text{ are positive random integers and } \eta \text{ an integer } . \quad (7)$$

Let $\kappa$ be $x^*$ length in t-bit words. We pad $x$ with 0 until its bit length, $|x|$, is equal to $\kappa.t$.

The variable $\widetilde{x}$ such that $|\widetilde{x}| = \kappa$ is introduced to control the blinding quality. It is based on the distance $\delta$ between $x^*$ and $x$.

$$\widetilde{x}_i = \delta(x_i, x_i^*, \epsilon) \text{ for } 0 \leq i < \kappa \ . \quad (8)$$

In our case, as distance we consider the *hamming weight* $h$ of $x^* \oplus x$. It is defined as follow:

$$\begin{array}{rccc} h: & \{0,1\}^t & \to & [0,t] \\ & (x_{t-1}, \ldots, x_0) & \mapsto & \sum_{i=0}^{t-1} x_i \end{array} \ . \quad (9)$$

We determine $\widetilde{x}$ bits thanks to:

$$\begin{array}{rcl} \delta : \{0,1\}^t \times \{0,1\}^t \times \mathbb{N} & \to & \{0,1\} \\ (x_i, x_i^*, \epsilon) & \mapsto & \begin{cases} 0 \text{ if } & |\frac{t}{2} - h(x_i \oplus x_i^*)| < \epsilon \\ 1 \text{ else} & \end{cases} \end{array} \ . \quad (10)$$

This function $\delta$ is used to dynamically consolidate exponent randomization during the exponentiation. In case where the exponent is uniformly randomized, i.e. $\widetilde{x} = 0$, it is equivalent to a regular secure modular exponentiation.

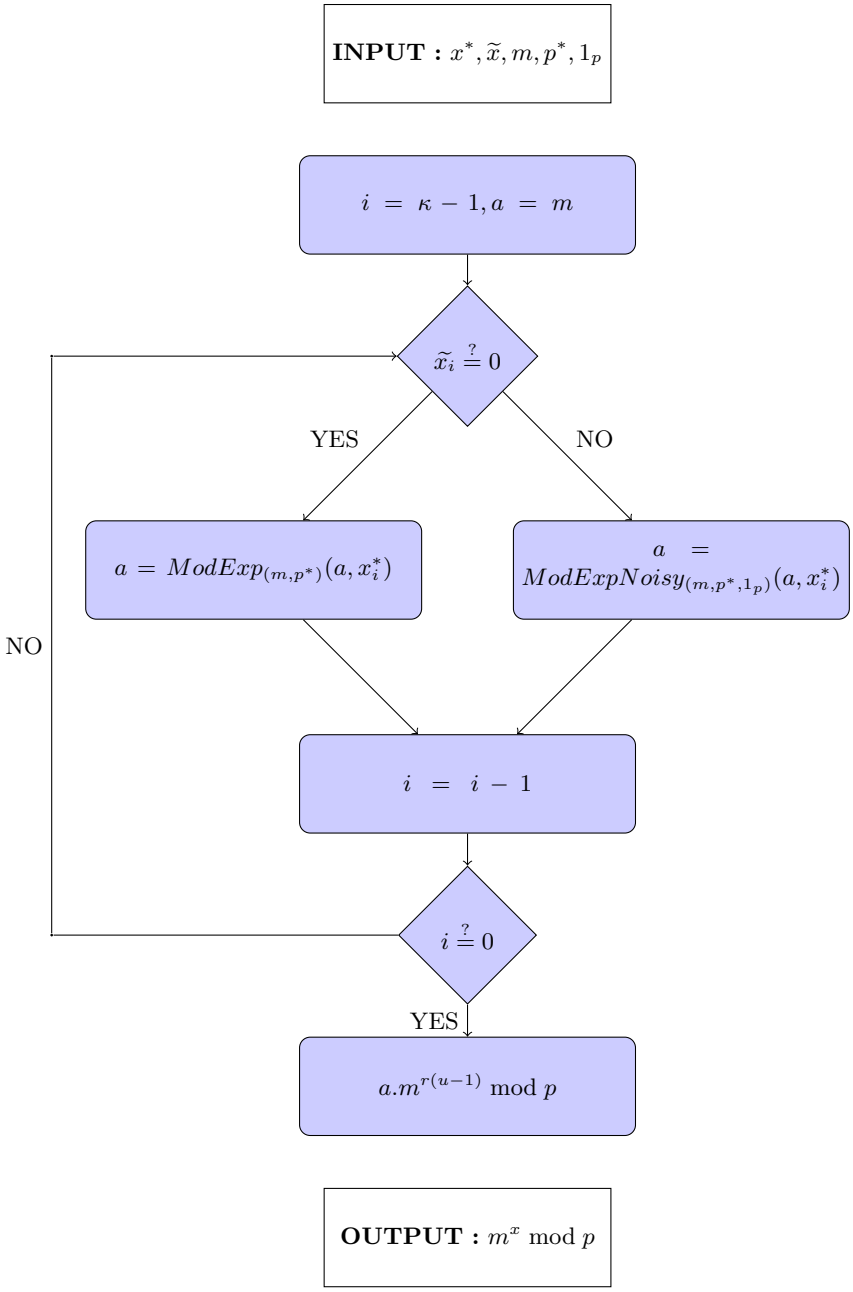


Fig. 3: Dynamic Blinding Exponentiation

As a decision function we take $|\frac{|x|}{2} - h| < \epsilon$ with $\epsilon$ the defined threshold. The purpose is to identify if the randomization is sufficient to cover all secret bytes. Input parameters $t$ and $\epsilon$ are linked to attacks performed with partial knowledge of the key. These attacks, such that the ones described in [4], [11], [12], to retrieve complete key, require that an attacker knows or retrieves a part of it thanks to side channel analysis.

To choose properly the parameters $t$ and $\epsilon$ we evaluate the impact of modulus randomness on decision function according to key and word's length. For each size $\{512, 1024, 2048\}$ we generated one hundred $\eta$ random and one hundred $\eta$ hollow. For each $\eta$ and thus each $x$, we compute $x^*$ using a 64-bit random and the average $|\frac{|x|}{2} - h|$. The results are summarized in table 1.
We remark that for small $t$ values, the average are too close to determine if $x$ is well randomized. This is due to our decision function that is not efficient for such word size. From figure 4, we observe that the choice $\epsilon$ bound depends strongly on key length and on word size $t$. Moreover for small key size such as 256 bits used for ECC, we note that the average are close for all word sizes.

| $t$ | 8 bits | 16 bits | 32 bits | 64 bits |
|---|---|---|---|---|
| $p$ 256-bit random | 0.94 | 1.91 | 3.99 | 7.66 |
| $p$ 512-bit random | 0.50 | 0.89 | 1.86 | 4.11 |
| $p$ 1024-bit random | 0.25 | 0.45 | 1.07 | 2.04 |
| $p$ 2048-bit random | 0.11 | 0.25 | 0.51 | 1.31 |
| $p$ 256-bit hollow | 1.97 | 3.97 | 7.92 | 15.83 |
| $p$ 512-bit hollow | 2.98 | 5.96 | 11.83 | 23.74 |
| $p$ 1024-bit hollow | 3.22 | 6.43 | 12.82 | 25.74 |
| $p$ 2048-bit hollow | 3.71 | 7.41 | 14.85 | 29.73 |

Table 1: Average of $|\frac{t}{2} - h(x_i \oplus x_i^*)|$ depending on $t$ and $p$ randomness

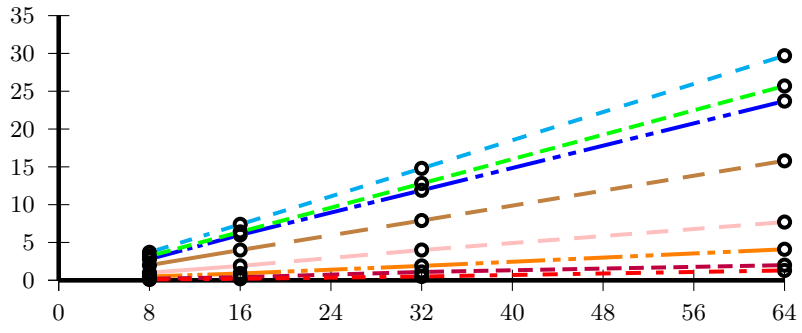


Fig. 4: Decision function bounds behavior

*Dynamic Blinding Exponentiation* algorithm represented by figure 3 illustrates our method applied to a modular exponentiation $m^x \bmod p$ with $x$ secret, $m$ a message correctly padded and $p$ a prime integer. Given $r_0, r_1, r_2$ random integers, let's consider:

$$\begin{aligned} p^* &= r_0.p \\ 1_p &= 1 + r_1.p \text{ s.t. } \mathrm{GCD}(r_0, r_1) = 1 \ . \\ x^* &= x + r_3.(p-u) \end{aligned} \tag{11}$$

We remark that:

$$\begin{aligned} 1_p &\equiv 1 \bmod p \\ 1_p &\equiv \widetilde{m} \bmod p^* \text{ s.t. } \widetilde{m} \text{ random } \ . \\ (m^{x^*}).(1_p)^{x^*} &\equiv m^{x^*} \bmod p \end{aligned} \tag{12}$$

In figure 3, we note by $ModExp_{(m,p)}(a, e)$ the exponentiation of $m$ by $e$ modulo $p$ and initializing the accumulator to $a$ instead of 1 in regular modular exponentiation. $ModExpNoisy_{(m,p,1_p)}(a, e)$ is based on $ModExp$ and both should be SPA secure implementations such as suggested in [18], [8] or [9]. The difference that we introduce in *ModExpNoisy* is that whatever the exponentiation algorithm used each operation required is followed by a modular multiplication by $1_p$. This operation has not impact on the result but generates noise regarding side channel leakage and is not sensitive to safe-error attack. Thus this method fully controls masked data. Indeed, $\widetilde{x}$ computed thanks to the function $\delta$ ensures propagation of random in secret data, only if it is needed. Actually, an interesting approach of this countermeasure is that the attacker will be penalized regarding execution time compared to a normal user. Even if the method is known by the attacker and if he constructs $p$ such that $(p - u_{max})$ is *hollow*, the algorithm will enable to master the link between the secret and its randomized value.

Performances are well managed. In case of classical parameters the correction $m^{r.(u-1)} \bmod p$ will only cost in average $c.|r|.|u|$ modular multiplications where $c$ is the complexity of the modular exponentiation algorithm used. That is to compare with the cost of the correction with $m^R \bmod p$ in case of the first countermeasure mentioned in [7]: $c.|x|$, which doubles the execution time of a simple exponentiation.

In addition to all previous remarks, we also have to consider that the modular exponentiation (or the point multiplication) could present leakages, despite the exponent (respectively the scalar) blinding. As an example we detail the RSA case. In standard mode, if the classical counter measure $d^* = d + r.\varphi(n)$ is used, an attacker is able to generate a valid signature. In CRT mode, the knowledge of $d_p^*$ leads to a disclosure of the key. Indeed we could find $p$ using $d_p^*$, $s$, $m$ and $n$ and:

$$\mathrm{GCD}((s - m^{d_p^*}), n) = p \ . \tag{13}$$

With our countermeasure he will need to distinguish, by SPA or SEMA, the exponentiations $ModExp$ from $ModExpNoisy$ and $m^{x^*} \bmod p^*$ from $m^{r.(u-1)} \bmod$

$p$ to reach the same goal. In order to make reverse engineering more difficult and to avoid SPA, those two final exponentiations could be interleaved and implemented in a way that it is not possible to distinguish them. We will detail this in section 3.3.

Finally, we observe that equation 7 might be sensitive to attacks linked to [23] or to template attacks as mentioned in [6] or [2], and this leads to the next section.

### 3.2 Disturb side channel acquisition

In this section, we introduce a new algorithm to compute an additive masking operation such as presented in [20] with a controlled leakage. Indeed, we already explained that computing a masked value thanks to an addition can lead to a leakage as exploited in [23], or to more advanced attacks like template attacks. The main idea here is to compute $x^* = x + \mu$ without performing direct addition and with a control of the information that may leak from $x$ handling.

We denote $x$ the secret value to be masked and $\mu$ its additive mask.

The algorithm *Additive Masking Artefact* stands to protect a component from additive carry leakages leading to attacks. Indeed, algorithm 1 manipulates secret data with division operation and not with addition operation. The addition operations are only used at step 7 with randomized values.
However and as noticed in M. Joye *et al.* paper [17], a division operation can induce a variable and exploitable leakage linked to parameters, component but also to the size of divisor $\rho$. We can note that $\rho$ size is inversely proportional to the number of secret bits exposed to a possible leakage. With this words, a positive integer $\beta$ is introduced in order to master the amount of secret bits possibly disclosed. At step 6, $\beta$ controls also the division complexity. As this value defines $\rho$ size, it will directly affect division. If $\beta$ is small and as we have $\lfloor \frac{x}{\rho} \rfloor$, we will get a division with the size of numerator close to the size of denominator. In those terms, only most significant bits of $x$ could be disclosed by the division if a side channel analysis is performed during its execution. Moreover, it will result, first in a fast computation: only few bits of $x$ will be handled; secondly, few loops in division algorithm will have to be performed. Finally, and for the same reason, the leakage will be minimized. Note that if $\beta = 0$, it would be equivalent to classical additive making method. It is not considered here and $\beta$ is $> 0$.

In algorithm 1, $x_1^*$ stands for the reminder of $x$ by $\rho$ and $x_2^*$ . Regarding the parameters length, $x_1^*$ size is equal to $\rho$ one and $x_2^*$ has $x$ size. Thus, at most $\beta$ most significant bits of the sum of $x_1^*$ and $x_2^*$ will be shared with $x_2^*$. Among them, in case of RSA, as $\mu$ comes from multiplication of $\varphi$ by at least a 64-bit random $r$, $x^*$ will have $|r|$ bits in common with $\mu$. However, these exceeding bits cannot be controlled by the attacker for a given key length.

If we take a closer look at operations required by algorithm 1, we remark that this countermeasure includes a modular reduction followed by a division. Those two operations are seen as big number operations in terms of implementation.

**Algorithm 1** Additive Masking Artefact

**Input:** $x, \mu, \beta > 0$
**Output:** $x^* = x + \mu$
1. Pick a random integer $\rho$, $|x| - \beta < |\rho| < |x|$
2. $x_1^* \leftarrow x \mod \rho$
3. **if** $x_1^* \equiv 0 \mod \rho$ **then**
4. Go to step 1.
5. **end if**
6. $x^* \leftarrow x_1^* + \mu$
7. $x_2^* \leftarrow \rho \lfloor \frac{x}{\rho} \rfloor$
8. $x^* \leftarrow x^* + x_2^*$

At first glance in algorithm 1 step 2 and step 6 will take more calculation time. However, on second thought, cryptographic processor designer often provides both functions in a single command. So this cost is managed thanks to well designed dedicated processors.

To summarize, we replace an addition between the secret and a randomized value potentially mastered by an attacker by a randomized quotient/remainder evaluation and then two additions between randomized data.

### 3.3 Interleaved exponentiation

We already mentioned that the exponentiation of algorithm in figure 3 could be interleaved to make the reverse engineering more difficult and to improve the protection regarding SPA/SEMA. Let's present an algorithm doing this operation in a smart way. The aim of interleaving exponentiation is to mix both bits from first and second exponent. Let's suppose that we want to compute $m^{a+b} \mod n$. The homomorphic property of modular exponentiation gives:

$$m^{a+b} = m^a m^b \mod n \ . \tag{14}$$

The main idea is to split $a$ and $b$ in $l_a$ and $l_b$ slices, respectively, and process set of bits one after the other by switching set at each iteration. Algorithm 2 implements this idea.

Let's remark that the operation used in algorithm 2 is defined previously in figure 3. For this specific exponentiation, all bits are treated even most significant null bits. Indeed, as the accumulator could be different from the neutral element of multiplication, first squares are meaning. Moreover any operation might replace this *ModExp* operation, the only requirement comes from the homomorphic property.

Besides, we could insert an other random for driving the choice of the bits set. This random will determine if we choose bits set from $a$ or from $b$ in order to increase reverse engineering complexity.

*Interleaved Homomorphic Operation* algorithm takes parameters $l_a$ and $l_b$ as input to define in how many slices exponent a and exponent b, respectively, will

**Algorithm 2** Interleaved Homomorphic Operation

**Input:** $m, a, b, n, l_a, l_b$
**Output:** $s \leftarrow m^{a+b} \mod n$

1. $s_a \leftarrow 1$
2. $s_b \leftarrow 1$
3. **while** $|a| > 0$ or $|b| > 0$ **do**
4. **if** $|a| > l_a$ **then**
5. Pick a random integer $r_a$, $r_a \leq l_a$
6. **else**
7. $r_a \leftarrow |a|$
8. **end if**
9. $a_{low} \leftarrow a \mod 2^{|a|-r_a}$
10. $a_{high} \leftarrow \frac{a}{2^{|a|-r_a}}$
11. $s_a \leftarrow ModExp_{(m,n)}(s_a, a_{high})$
12. $a \leftarrow a_{low}$
13. **if** $|b| > l_b$ **then**
14. Pick a random integer $r_b$, $r_b \leq l_b$
15. **else**
16. $r_b \leftarrow |b|$
17. **end if**
18. $b_{low} \leftarrow b \mod 2^{|b|-r_b}$
19. $b_{high} \leftarrow \frac{b}{2^{|b|-r_b}}$
20. $s_b \leftarrow ModExp_{(m,n)}(s_b, b_{high})$
21. $b \leftarrow b_{low}$
22. **end while**
23. $s \leftarrow s_a.s_b$
24. **return** s

be split. At the end of the algorithm, both values $a$ and $b$ must be equal to 0, this asset might be used to verify the well execution of the implementation.

To conclude, this approach enables to break the link between an exponentiation execution or any other homomorphic operation and its side channel leakage.

## 4 Conclusion

We introduced a new way to mask secret exponent or scalar in order to control the link between the original value and the randomized one without inducing too heavy computation in case of normal execution and parameters. The attackers are penalized by this approach. Moreover, we not only modify the way to do the data masking but also the operation used for that. Indeed we replace an addition by a division and a remainder operation in a context where they are not costly and do not leak exploitable information. Finally, we propose a method to interleave operations so as to make reverse engineering and side channel analysis more complex. For future work, we could improve $\delta$ and $h$ functions used as

distance. These functions could be designed to enlarge their scope especially for small word sizes and small key length.

## References


1. F. Amiel, K. Villegas, B. Feix, and L. Marcel. Passive and active combined attacks: Combining fault attacks and side channel analysis. In Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Fourth International Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC 2007*, pages 92–102. IEEE Computer Society, 2007.
2. C. Archambeau, E. Peeters, F-X. Standaert, and J-J. Quisquater. Template attacks in principal subspaces. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - 8th International Workshop – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.
3. S. Bauer. Attacking exponent blinding in rsa without crt. In Schindler and Huss [24], pages 82–88.
4. D. Boneh, G. Durfee, and Y. Frankel. Exposing an rsa private key given a small fraction of its bits, 1998.
5. Çetin Kaya Koç, David Naccache, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems, Third International Workshop - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*. Springer, 2001.
6. S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In Jr. et al. [19], pages 13–28.
7. D. Chaum. Security without identification: transaction systems to make Big Brother obsolete. *Communications of the ACM*, 28:1030–1044, 1985.
8. B. Chevallier-Mames, M. Ciet, and M. Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004.
9. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Square always exponentiation. In *INDOCRYPT*, pages 40–57, 2011.
10. C. Clavier and M. Joye. Universal Exponentiation Algorithm. In Çetin Kaya Koç et al. [5], pages 300–308.
11. D. Coppersmith. Finding a Small Root of a Univariate Modular Equation. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1996.
12. D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.
13. J-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop – CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.
14. J-C Courrege, B. Feix, and M. Roussellet. Simple Power Analysis on Exponentiation Revisited. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application, 9th International Conference – CARDIS 2010*, volume 6035 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2010.
15. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç et al. [5], pages 251–261.
16. M. Joye. Elliptic Curves and Side-Channel Analysis, 2003.

17. M. Joye and K. Villegas. A protected division algorithm. In *Smart Card Research and Advanced Application Conference, 5th International Conference – CARDIS 2002*. USENIX, 2002.
18. M. Joye and S-M. Yen. The Montgomery Powering Ladder. In Jr. et al. [19], pages 291–302.
19. Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems, 4th International Workshop – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*. Springer, 2002.
20. P. Kocher and J. Jaffe. Leak-Resistant cryptographic method and apparatus. Patent WO99/35782, 1998.
21. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology, 19th Annual International Cryptology Conference – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
22. A. Krüger. The schindler-itoh-attack in case of partial information leakage. In Schindler and Huss [24], pages 199–214.
23. F. Valette P.A Fouque, D. Réal and M'hamed Drissi. The Carry Leakage on the Randomized Exponent Countermeasure. In In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems, 10th International Workshop – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 198–213. Springer, 2008.
24. Werner Schindler and Sorin A. Huss, editors. *Constructive Side-Channel Analysis and Secure Design - Third International Workshop – COSADE 2012*, volume 7275 of *Lecture Notes in Computer Science*. Springer, 2012.
25. Werner Schindler and Kouichi Itoh. Exponent Blinding Does Not Always Lift (Partial) Spa Resistance to Higher-Level Security. In Javier Lopez and Gene Tsudik, editors, *ACNS*, volume 6715 of *Lecture Notes in Computer Science*, pages 73–90. Springer, 2011.