Defeating with Fault Injection a Combined Attack Resistant Exponentiation

Benoit Feix[1,⋆] and Alexandre Venelli[2]

[1] XLIM-CNRS, Université de Limoges, France
benoit.feix@xlim.fr
[2] INSIDE Secure, Aix-en-Provence, France
avenelli@insidefr.com

Abstract. Since the introduction of the side-channel and fault injection analysis late in the 90's, implementing cryptographic standards on embedded devices has become a difficult challenge. Developers were obliged to add new appropriate countermeasures into their code. To prevent those separate threats, they often implemented countermeasures separately. The side-channel dedicated countermeasures were added to the algorithm when on the other hand specific protections against fault injections, like computation verifications, were implemented. However in 2007 Amiel *et al.* demonstrated that a single fault injection combined with simple side-channel analysis can defeat such a classical implementation. Then it became obvious that side-channel and fault countermeasures had to be designed together. In that vein Schmidt *et al.* published at Latincrypt 2010 an efficient exponentiation algorithm supposedly resistant against this combined attack category. Despite the clever design of these algorithms, we present here two new attacks that can defeat its security. Our first attack is a single fault injection scheme requiring only few faulted ciphertexts. The second one requires the combination of a single fault injection with a differential treatment. We also propose a more secure version of this algorithm that thwarts our attacks.

Keywords: Embedded Exponentiation, Side-channel Analysis, Fault Analysis, Combined Attack, RSA, ECC.

1 Introduction

For years the development of secure embedded products, such as smartcards, has become more and more challenging for designers. In the middle of the 90's the security of the smartcards mainly consists in measuring the strength of the hardware mechanisms which protect the product from invasive attacks. But with the introduction of two new categories of attacks the task has become more difficult.

Side-Channel Analysis (SCA), also referred as *Passive Attacks*, is introduced in 1996 by Kocher [13]. He demonstrates that an embedded device supporting

⋆ This work was carried out when the author was with Inside Secure.

cryptographic calculations can reveal information on secrets manipulated when analyzing the physical interactions between the integrated circuit and its environment. An attacker can then observe the power consumption trace of the device [15] or its electromagnetic emanations [8]. SCA regroups two different techniques: the *Simple Side-Channel Analysis* (SSCA) and the *Differential Side-Channel Analysis* (DSCA). SSCA exploits only a single trace measurement of the targeted algorithm execution to recover the secret values. DSCA requires many execution of the targeted algorithm and applies statistical analysis on the corresponding side-channel traces to successfully validate guesses done on the secret.

Fault Analysis (FA), or *Active Attacks*, consists in perturbing the algorithm process to obtain an abnormal behavior. It can be done by injecting power glitches on the circuit pad or by precise laser light emissions on the device surface (front side or back side). An erroneous computation result is then obtained which can be exploited to recover entirely or partially the secrets. Different active attacks exist: the *Differential Fault Analysis* (DFA), the *Ineffective Fault Analysis* (IFA), the *Collision Fault Analysis* (CFA).

Most of the cryptosystems are nowadays threatened by both techniques like RSA [18] and ECC [12, 16] embedded implementations. We focus our study in this paper on those embedded implementations. In the last decade many countermeasures have been presented to design side-channel resistant algorithm on the first hand and fault injection countermeasures on the other hand. For years implementing those countermeasures separately has never been an issue. But in 1997 Amiel *et al.* [2] present a *combined passive and active attack* on an RSA implementation which is considered at this time resistant to both SCA and FA techniques separately. In 2010 Schmidt *et al.* [20] propose combined-attack resistant algorithms to compute exponentiation and scalar multiplication. Their implementations cleverly include tricks to thwart the Amiel *et al.* attack.

However in this paper we present new attacks on their algorithms. The first technique we introduce is a first order fault attack which can recover the whole secret exponent with a practical number of faulted results. Our fault injections benefit from a flaw in the infective computation design of the Schmidt *et al.* algorithms. The second threat on these algorithms is an attack combining fault injection with differential analysis on many executions. This analysis targets their use of a specific exponentiation technique, *i.e.* left-to-right multiply always, in order to thwart its supposedly resistance against combined attacks.

Roadmap. Section 2 reminds the reader the necessary background on side-channel and fault attacks, as well as on combined attack resistant implementation in order to understand the attacks presented in this paper. In Section 3 we introduce the first order fault attacks which can defeat the combined exponentiation from Schmidt *et al.* on a simplified and the complete versions of this algorithm. New combined attacks are presented in Section 4. Section 5 propose an improved version of the Schmidt *et al.* algorithm which counterfeit the new attacks presented. We conclude in Section 6.

2 Background

We present in this section the combined attack principle and the previous publications on the subject. We also remind the Schmidt *et al.* algorithms we are attacking in the rest of this paper.

2.1 Combined Attacks on Asymmetric Cryptosystems

Since the publication from Amiel *et al.*, combined attacks have been more and more investigated. This technique exploits leakage information from both a fault analysis (FA) and a classical side-channel attack like SSCA or DSCA. Both symmetric and asymmetric cryptosystems have been shown vulnerable to it. In this paragraph we briefly review the combined attacks proposed in the literature.

The first combined attack publication from Amiel *et al.* [2] combines a fault attack with an SSCA in order to break a modular exponentiation that is supposedly secure against DFA and SSCA. The authors attack a left-to-right multiply always algorithm implementing the *atomicity* principle from Chevallier-Mames *et al.* [4]. Additionally the message and the secret exponent values were randomized to counterfeit DSCA. The first step of the attack consists in injecting a fault in one of the registers (or in the RAM) before (or during) the beginning of the exponentiation. The fault aims at creating a modified message value that will leak in SSCA each time it is manipulated. For instance a low Hamming weight value has been introduced into a part of the message, or the message pointer has been modified to include an erased area of the RAM. This message modification renders the message manipulations visible into a side-channel trace. It becomes then possible to distinguish a squaring operation from a multiplication using SSCA as described in [5]. Hence, the FA protection that is present at the end of the algorithm cannot prevent the SSCA leakage that has already occurred during the computation. This attack is very efficient as a single fault applied successfully to the calculation execution will make the SSCA efficient. The principle of the attack of Amiel *et al.* seems to be applicable to any classic left-to-right atomic algorithm, either exponentiation or scalar multiplication. In [2], the authors propose a countermeasure called *Detect and Derive* based on the principle of infective computation. However, it was shown vulnerable in [20]. In this paper, Schmidt *et al.* introduce a new resistant exponentiation algorithm, as well as a scalar multiplication algorithm, also based on infective computation. The idea is to be able to detect a fault as soon as it happens and corrupt the data if necessary so that no relevant information is leaking anymore.

More recently, in [7], Fan *et al.* study the case of combined attacks specially targeting elliptic curve scalar multiplication. Using the properties of elliptic curves, they develop a powerful attack that can defeat atomic and regular algorithms. In order to perform the attack, one needs to choose a particular input point of the scalar multiplication. By injecting a fault after the initial point verification, the attacker is then able to obtain a point with a small order. During the scalar multiplication, computations with the faulted point will end up on the infinity point which is particularly visible by SSCA in most implementations. The attacker is then able to find information on the secret scalar.

2.2 Schmidt *et al.* Resistant Algorithm

We remind in the following the combined attack resistant implementation from Schmidt *et al.* [20] to give the reader the necessary notions to understand our attacks. We principally consider the exponentiation algorithm in this paper, however most of our attack paths can be directly applied to the scalar multiplication counterpart.

Fault model considered. In their paper [20], Schmidt *et al.* deal with the three following fault attack models. The attacker is able with fault injection to:

- randomize data to an unknown value,
- reset data to all zeros or all ones or any given fix value,
- modify opcodes, *i.e.* skip instructions, break loops, etc.

The authors only take into consideration first order fault injections, *i.e.* an attacker injects only one fault per execution of the algorithm. They present two algorithms protected against combined attacks under these fault models. Their first algorithm (Alg. 1) [20, Alg. 3] is a protected exponentiation, and their second one [20, Alg. 4] is a protected scalar multiplication. Both algorithms are based on the same principles of countermeasures.

We remind the reader through Algorithm 1 the detailed combined attack resistant algorithm for exponentiation from Schmidt *et al.* [20].

Notations. In the rest of the paper, we use the following notations:

- let W be the block length that is generally the size of a processor word, *i.e.* $W = 8$ (resp. $W = 16$ or $W = 32$) for an 8-bit (resp. for a 16-bit or a 32-bit) architecture,
- let d be the t-bit secret exponent and $d = (d_{t-1}, d_{t-2}, \dots d_1, d_0)_2$, with d_i the i-th bit of d, its binary representation,
- let $\bar{d} = (\bar{d}_{t+\lambda-1}, \bar{d}_{t+\lambda-2}, \dots \bar{d}_1, \bar{d}_0)_2$ be the blinded exponent,
- let $\tilde{d}$ be the blinded exponent encoded using the function ψ_α detailed below,
- let $\hat{d}$ be the exponent decoded using ψ_α^{-1},
- let $d^{(j)}$ be the j-th W-bit word of d.

The exponent is protected through an encoding function $\psi_\alpha : \mathbb{Z}_{r_2} \times \mathbb{Z}_{r_2} \rightarrow \mathbb{Z}_{r_2}$ which is an invertible function defined as:

$$\psi_\alpha(d^{(j)}) = (\alpha + N)^{-1} \cdot d^{(j)} \bmod r_2,$$
$$\psi_\alpha^{-1}(\tilde{d}^{(j)}) = (\alpha + N) \cdot \tilde{d}^{(j)} \bmod r_2,$$

Algorithm 1 Schmidt *et al.* [20, Alg. 3] left-to-right exponentiation.

Input: $d = (d_{t-1}, \ldots, d_0)_2, m \in \mathbb{Z}_N, N$ and block length W.
Output: $m^d \bmod N$

```
1: $r_1 \leftarrow \mathsf{random}(1, 2^\lambda - 1)$
2: $r_2 \leftarrow \mathsf{random}(1, 2^\lambda - 1)$
3: $i \leftarrow (r_2^{-1} \bmod N) \cdot r_2$
4: $R_0 \leftarrow i \cdot 1 \bmod Nr_2$
5: $R_1 \leftarrow i \cdot m \bmod Nr_2$
6: $\bar{d} \leftarrow d + r_1 \cdot \varphi(N)$
7: $[\tilde{d}^{(l-1)}, \ldots, \tilde{d}^{(0)}] \leftarrow [\psi_0(\bar{d}^{(l-1)}), \ldots, \psi_0(\bar{d}^{(0)})]$
8: $k \leftarrow 0$
9: $j \leftarrow \mathsf{bitlength}(\tilde{d}) - 1$
10: while $j \geq 0$ do
11:     $R_0 \leftarrow R_0 \cdot R_k \bmod Nr_2$
12:     if $(R_0 = 0)$ or $(R_1 = 0)$ then
13:         $[\tilde{d}^{(l-1)}, \ldots, \tilde{d}^{(0)}] \leftarrow [1, \ldots, 1]$
14:     end if
15:     $\hat{d} \leftarrow \psi^{-1}_{(R_0 + R_1 \bmod r_2)}(\tilde{d}^{(\lfloor j/W \rfloor)})$
16:     $k \leftarrow k \oplus \mathsf{bit}(\hat{d}, j \bmod W)$
17:     $j \leftarrow j - \neg k$
18: end while
19: $c \leftarrow R_0 \bmod N$
    return $c$
```

with $\alpha \in \mathbb{Z}_{r_2}$, N the modulus and r_2 a small random value such that $r_2 > 2^W$.

In the next section, we introduce single fault attacks on a simplified version (without exponent/scalar blinding) of Algorithm 1 and on the complete Algorithm 1.

3 Fault Attack on Schmidt *et al.* Algorithms

We show in this section that a classical single fault attack can still be applied to the exponentiation algorithm proposed by Schmidt *et al.* [20, Alg. 3]. We consider in this section fault attacks based on the modification of an opcode, *i.e.* skip of instruction. We first propose a fault attack on a simplified version of Alg. 1 where the blinding of the exponent is not present (Line 6). Then, based on the same fault attack principle, we propose an attack on the complete version of Alg. 1.

3.1 Fault Attack on a Simplified Algorithm

As we consider no exponent blinding in this section, we have that $\bar{d} = d$, hence the encoded exponent $\tilde{d}^{(k)} = \psi_0(d^{(k)})$ for $0 \leq k \leq l - 1$ where l is the length of d in W-bit words.

To prevent their implementation from the combined attack presented in [2], the authors introduced at Line 12 of the algorithm an infective operation. The purpose is to corrupt the secret exponent when a fault injection is detected in order to cancel the side-channel leakage that could reveal the secret. More precisely the purpose of the test Line 12 of Alg. 1 is to corrupt the exponent in case one of the registers R_0 or R_1 was erased by fault which could leak simple side-channel information. Hence, the exponentiation would continue its course but using false exponent bits. Schmidt *et al.* choose to affect the value 1 to all words of the encoded exponent $\tilde{d}$. The decoding of a word of exponent performed Line 15, assuming no faults in registers R_0 or R_1, computes for the k-th word of the exponent:

$$\hat{d} = \psi_0^{-1}(\tilde{d}^{(k)}) = N \cdot \psi_0(d^{(k)}) \bmod r_2 = d^{(k)} \bmod r_2.$$

It is very important for our attacks to notice that if the exponent has been corrupted in Line 13, all the decoded W-bit words of exponent until the end of the exponentiation are equal to the value:

$$\hat{d} = \psi_0^{-1}(1) = N \cdot 1 \bmod r_2 = N \bmod r_2.$$

Moreover, we note from Line 16 that only the W least significant bits of $\hat{d}$ are considered for the exponentiation. It signifies that from the moment a single fault is injected to skip the test at Line 12, all the remaining W-bit words $\hat{d}^{(i)}$ being used for the rest of the exponentiation are equal to this same and unique value $N \bmod r_2$.

We introduce for our analysis two additional notations. Let H be the value $(N \bmod r_2) \bmod 2^W$ and $\tilde{t} = l \cdot W$ be the bit length of $\tilde{d}$.

Now consider that an attacker already knows the v (can be zero) first bits of the exponent and skips Line 12 by fault injection u bits after in the loop of the algorithm. The algorithm outputs the faulted result $\check{S}_u$ that used the following exponent:

$$\check{d}_u = \underbrace{\sum_{i=\tilde{t}-v}^{\tilde{t}-1} 2^i \cdot \tilde{d}_i}_{\text{known part of the exponent}} + \sum_{i=\tilde{t}-v-u}^{\tilde{t}-v-1} 2^i \cdot \tilde{d}_i + \sum_{i=0}^{\tilde{t}-v-u-1} 2^i \cdot H_{(i \bmod W)}. \tag{1}$$

By doing a guess on the next u unknown bits of d and another guess on the value of H, an attacker can compute the guessed result of the exponentiation, denoted $S_g(u, H)$. Then by comparing this value $S_g(u, H)$ with $\check{S}_u$, he can decide if his guesses are correct or not. After an exhaustive calculation for all possible values, when $S_g(u, H) = \check{S}_u$ the attacker recovers the right values $(d_{t-v-1}, \ldots, d_{t-v-u})$ and H.

Complexity. The computational complexity $\mathcal{C}$ of our fault attack to recover the exponent is:

$$\mathcal{C} = \mathcal{O}\left(\frac{2^{(u+W)} \cdot \tilde{t}}{u}\right) \quad \text{exponentiations.}$$

The number of faulty signatures $\mathcal{F}$ to collect is:

$$\mathcal{F} = \mathcal{O}\left(\frac{\tilde{t}}{u}\right).$$

We have validated our attack on a standard PC using the GMP library[3] for different RSA keys (values and bit-length) with success.

Table 1 gives examples of computational complexity of our attack for $u = 1$ and different values of W and t.

W — Bit-length t	512 bits	1024 bits	2048 bits
8	$\mathcal{C} = 2^{18}$	$\mathcal{C} = 2^{19}$	$\mathcal{C} = 2^{20}$
16	$\mathcal{C} = 2^{26}$	$\mathcal{C} = 2^{27}$	$\mathcal{C} = 2^{28}$
32	$\mathcal{C} = 2^{42}$	$\mathcal{C} = 2^{43}$	$\mathcal{C} = 2^{44}$

Table 1. Example of computational complexities for $u = 1$ to recover the exponent on the simplified algorithm.

This attack also applies to the simplified scalar multiplication algorithm of Schmidt *et al.* [20, Alg. 4], *i.e.* with no scalar blinding. However this analysis only works if the attacker can retrieve the exponent u bits at a time using different faulty results. Hence in the presence of exponent blinding, it cannot be applied directly. We present in the following an adaptation of the attack to the blinded exponentiation algorithm.

3.2 Fault Attack on the Complete Version of the Algorithm

Based on the attack presented previously, we propose in this section a variation in order to attack Alg. 1 considering the exponent blinding countermeasure. As previously observed by Berzati *et al.* in [3], the blinding using $\varphi(N)$ does not mask homogeneously the exponent. We propose here an attack which exploits this flaw. We do not include the processing of the exponent through the encoding function ψ for easier notation. As seen in the previous section, the output size of the encoding function, *i.e.* the size of the random r_2, has no effect on the attack because the algorithm only considers bits modulo W.

[3] The GNU Multiple Precision Arithmetic Library, available at http://gmplib.org/

Let $\bar{d}$ be the blinded exponent such that $\bar{d} = d + r_1\varphi(N)$ with r_1 a λ-bit random. Let $\bar{d} = \sum_{i=0}^{t+\lambda-1} 2^i \cdot \bar{d}_i$ be its binary decomposition. We can also write it as:

$$\bar{d} = \sum_{i=t}^{t+\lambda-1} 2^i \cdot (r_1 N)_i + \sum_{i=t/2+\lambda}^{t-1} 2^i \cdot (d + r_1 N)_i + \sum_{i=0}^{t/2+\lambda-1} 2^i \cdot (d + r_1\varphi(N))_i. \quad (2)$$

We observe that the least significant bits of the secret exponent d are randomized with the full mask $r_1\varphi(N)$. On the other hand, the most significant (half upper) bits of d are only masked with $r_1 N$. The attack consists in finding d from its most significant bits to its least significant ones.

We note $\check{S}_u$ the faulty result of an exponentiation where the test Line 12 of Alg. 1 has been skipped by fault after the u-th unknown bit of the exponent has been processed. The faulty exponent $\check{d}_u$ corresponding to $\check{S}_u$ is detailed in Eq. (1). We consider that the attacker has already retrieved the v most significant bits of d.

Retrieving the MSB part of *d*. We first consider a fault injected after the u-th unknown bit within the range of bits of d being $[(t/2 + \lambda), t]$. We consider then:

$$\bar{d} = \sum_{i=t}^{t+\lambda-1} 2^i \cdot (r_1 N)_i + \sum_{i=t-u}^{t-1} 2^i \cdot (d + r_1 N)_i + \sum_{i=t/2+\lambda}^{t-u-1} 2^i \cdot (d + r_1 N)_i + \sum_{i=0}^{t/2+\lambda-1} 2^i \cdot (d + r_1\varphi(N))_i. \quad (3)$$

Let $\bar{d}_{[u]} = \sum_{i=t-u}^{t+\lambda-1} 2^i \cdot \bar{d}_i$ and $\bar{d}_{<u>} = \sum_{i=0}^{t-u-1} 2^i \cdot \bar{d}_i$. The faulty exponent $\check{d}_u$ of the result $\check{S}_u$ can be approximated as $\check{d}_u \approx \bar{d}_{[u]} + \bar{d}_{<u>}$, not considering the carry propagation.

Once the fault has been injected, as we observed previously, the least significant part of the encoded exponent is fixed at 1 in Line 13 of Alg. 1 as an infective calculation countermeasure. Hence, we have that after the fault at the u-th bit, $\bar{d}_{<u>} = \sum_{i=0}^{t-u-1} 2^i \cdot H_{(i \bmod W)}$ with $H = (N \bmod r_2) \bmod 2^W$.

In order to find $\bar{d}_{<u>}$, the attacker only needs to guess W bits of H. We note $d_{\mathsf{known}} = \sum_{i=t-v}^{t-1} 2^i \cdot d_i$ the most significant v bits of d already retrieved by the attacker.

From Eq. (2) and (3), the most significant part of the exponent $\bar{d}_{[u]}$ can be approximated as:

$$\begin{aligned}\bar{d}_{[u]} &\approx \sum_{i=t-u}^{t+\lambda-1} 2^i \cdot (d + r_1 N)_i \\ &\approx d_{\mathsf{known}} + \sum_{i=t-v-u}^{t-v-1} 2^i \cdot d_i + \sum_{i=t-v-u}^{t+\lambda-1} 2^i \cdot (r_1 N)_i + \mathsf{carry}\end{aligned}$$

where carry is the possible carry bit resulting from the addition between the u first bits of r_1 and N. In order to find the value of $\bar{d}_{[u]}$, the attacker needs to guess u bits of d and λ bits of r_1. The possible carry bit only gives an uncertainty on the parity of the guessed value of d. By guessing $2^{(u+W+\lambda)}$ bits, the attacker can construct a guess of the full exponent $\check{d}_u$. He can then validate his guess by checking if the following relation is verified:

$$\check{S}_u \overset{?}{=} m^{\bar{d}_{[u]}+\bar{d}_{<u>}} \bmod N. \tag{4}$$

Retrieving the LSB part of d. Once we have recovered the MSB part of d, we now consider a fault injected after the u-th unknown bit within the range of bits of d being $[0,\ (t/2+\lambda)]$. Contrary to the MSB case, the bits of d will not be guessable directly as the full mask $r_1\varphi(N)$ is now applied.

We consider then:

$$\bar{d} = \sum_{i=t/2+\lambda}^{t+\lambda-1} 2^i \cdot (d + r_1 N)_i + \sum_{i=t/2+\lambda-u}^{t/2+\lambda-1} 2^i \cdot (d + r_1\varphi(N))_i + \sum_{i=0}^{t/2+\lambda-u-1} 2^i \cdot (d + r_1\varphi(N))_i. \tag{5}$$

The least significant part of the faulted exponent is still equal to $\bar{d}_{<u>} = \sum_{i=0}^{t/2+\lambda-u-1} 2^i \cdot H_{(i \bmod W)}$. As previously, in order to find $\bar{d}_{<u>}$, the attacker only needs to guess W bits of H.

We can write the most significant part of the exponent using Eq. (2) as:

$$\begin{aligned}
\bar{d}_{[u]} &= \sum_{i=t/2+\lambda-u}^{t+\lambda-1} 2^i \cdot (d + r_1\varphi(N))_i \\
&= \sum_{i=t/2+\lambda-u}^{t+\lambda-1} 2^i \cdot (d + r_1 N - r_1(p+q-1))_i \\
&\approx d_{\mathsf{known}} + \sum_{i=t/2+\lambda-v-u}^{t/2+\lambda-v-1} 2^i \cdot \delta_i + \sum_{i=t/2+\lambda-v-u}^{t+\lambda-1} 2^i \cdot (r_1 N)_i + \mathsf{carry}
\end{aligned}$$

where $\delta_i = (d - r_1(p+q-1))_i$ and carry is the possible carry due to the addition of the u bits of r_1N with $(d - r_1(p+q-1))$.

As previously, the possible carry bit is not taken into account in the analysis as it only affects the parity of the final guess and is easily checkable. In order to find the value of $\check{d}_u$, the attacker needs to guess $2^{(u+W+\lambda)}$ bits: u bits of δ, λ bits of r_1 and W bits of H. The attacker can then construct a guess of the full exponent and validate this guess by checking if $\check{S}_u \overset{?}{=} m^{\bar{d}_{[u]}+\bar{d}_{<u>}} \bmod N$. Contrarily to the MSB case we described previously, recovered bits are not bits of d but u bits of δ_i. This can be solved by using many faulted executions instead of one.

Indeed as the values of d and $(p+q-1)$ are fixed between different exponentiations, by faulting at the same time u, the attacker can obtain an additional guess for δ with a different r_1. With two or more faulted exponentiations, he will be able to determine the u bits of d and the u bits of $(p+q-1)$. The validation of the guesses are made, similarly to the MSB case, by comparing the faulted result of exponentiation to the exponentiation with our entire guessed exponent (see Eq. (4)).

Complexity. The computational complexity $\mathcal{C}$ of our fault attack to recover the exponent is:

$$\mathcal{C} = \mathcal{O}\left(\frac{2^{(u+W+\lambda)} \cdot t}{u}\right) \quad \text{exponentiations.}$$

The number of faulty signatures $\mathcal{F}$ to collect is:

$$\mathcal{F} = \mathcal{O}\left(\frac{t}{u}\right).$$

We can note that our fault attack does not require non-faulted results of exponentiations. The complexity of our attack is not impacted by the size of r_2 used in the encoding function ψ but by the size of the window W as only W bits of the output of the encoding are used to perform the exponentiation. This undesirable effect of Alg. 1 implies that the smaller processor words, the easier this fault attack is to perform. As previously, this attack has been validated on a standard PC using the GMP library.

We have presented first order (single) fault injections that defeat the combined resistant implementation with few faulted executions and a reasonable complexity that render this attack practical. Our attacks use a flaw in the design of the infective computation in Schmidt *et al.* algorithms. In the next section we discuss the resistance of Algorithm 1 against combined attacks and particularly with regards to the combined attacks we introduce.

4 Combined Attacks on Schmidt *et al.* Algorithms

Although the algorithms proposed by Schmidt *et al.* [20] are supposedly resistant to the combined attack published by Amiel *et al.* [2], we explain in the following that Alg. 1 can be threatened by more advanced combined attacks.

Combining Fault Injection with Differential Analysis. We consider the exponentiation algorithm (Alg. 1) for the description of this attack, however it directly applies to the scalar multiplication algorithm [20, Alg. 4]. Note that the internal registers R_0 and R_1 are randomized at the beginning of the algorithm with a random idempotent element i (Line 6 Alg. 1). Hence, we can only use attacks that consider unknown plaintexts as the randomization by i cannot be easily removed.

A combined attack that uses an instruction skip fault combined with one of the differential attack using unknown plaintext can be mounted on Schmidt *et al.* algorithms. If the attacker can skip Line 6 in Alg. 1 by fault injection, then the exponentiation is performed without exponent blinding, *i.e.* $\bar{d} = d$. In case a bit of d is $d_j = 0$, the multiplication Line 11 becomes $R_0 \cdot R_0$, whereas if a bit equals 1, it computes $R_0 \cdot R_1$. More precisely, if $d_j = 0$ the output of the multiplication will have the expected Hamming weight of a squaring which is distinct from the expected Hamming weight of a multiplication output as demonstrated in [1, 22]. Hence the attack of Amiel *et al.* [1] can be applied. However it requires few thousand curves in order to distinguish correctly squaring from multiplication operations. The fault attack on Line 6 then needs to be repeatable which is demonstrated realistic from recent fault injection techniques [17, 6]. Note that the fault repeatability does not need to be perfect as failed faults are considered as noise in the differential analysis treatment. Hence, it only affects the number of curves necessary to recover the secret.

Combining Fault Injection with Template Analysis. A template attack[4] using the same principle as Amiel *et al.* was proposed by Hanley *et al.* [10]. With very few curves, the attacker can recover the full exponent in a template matching phase. If the exponent blinding of Line 6 is removed, this attack can also be applied with less faults and less traces compared to the previous one. Note that without the fault injection, this template attack can be mounted using only one curve. Hence, the recovery of the exponent will most certainly not be complete. Depending on the size of the blinding factor r_1 (Line 1), the size of the modulus N and the success rate of the template attack, the methodology of Schindler and Itoh [19] can be applied to recover the full exponent.

5 Improved Combined Attack Resistant Algorithms

We propose in this section improvements on the exponentiation algorithm (Alg. 1) to prevent the attacks presented previously. Our proposed improvements also applies to the scalar multiplication variant.

The fault attack presented in Section 3 exploits a skip of instruction on the conditional test in Line 12 where the infective calculation replaced the entire encoded exponent by 1. A simple and efficient countermeasure consists in replacing this fixed value by random values for each words of the exponent. Another protection could be offered through the classical DFA countermeasure consisting in verifying the calculation with the public exponent e when possible.

[4] As the plaintext can be unknown to construct these templates, an open device is not mandatory contrary to the usual definition of a template attack. The attacker only needs to record the power consumption of multiplications and squarings with random inputs.

Algorithm 2 Improved Schmidt *et al.* left-to-right exponentiation.

Input: $d = (d_{t-1}, \ldots, d_0)_2, m \in \mathbb{Z}_N, N$ and block length W.
Output: $m^d \bmod N$

```
1: $r_1 \leftarrow \mathsf{random}(1, 2^\lambda - 1)$
2: $r_2 \leftarrow \mathsf{random}(1, 2^\lambda - 1)$
3: $i \leftarrow (r_2^{-1} \bmod N) \cdot r_2$
4: $R_0 \leftarrow i \cdot 1 \bmod Nr_2$
5: $R_1 \leftarrow i \cdot m \bmod Nr_2$
6: $\bar{d} \leftarrow d + r_1 \cdot \varphi(N)$                              (optional)
7: $[\tilde{d}^{(l-1)}, \ldots, \tilde{d}^{(0)}] \leftarrow [\psi_0(\bar{d}^{(l-1)}), \ldots, \psi_0(\bar{d}^{(0)})]$
8: for $i = 0$ to $l - 1$ do
9:     $w_i \leftarrow \mathsf{random}(1, 2^W - 1)$
10: end for
11: $k \leftarrow 0$
12: $j \leftarrow \mathsf{bitlength}(\tilde{d}) - 1$
13: while $j \geq 0$ do
14:     $r_3 \leftarrow \mathsf{random}(1, 2^\lambda - 1)$
15:     $R_2 \leftarrow R_k + r_3 \cdot N \bmod Nr_2$
16:     $R_0 \leftarrow R_0 \cdot R_2 \bmod Nr_2$
17:     if $(R_0 = 0)$ or $(R_1 = 0)$ then
18:         $[\tilde{d}^{(l-1)}, \ldots, \tilde{d}^{(0)}] \leftarrow [w_{l-1}, \ldots, w_0]$
19:     end if
20:     $\hat{d} \leftarrow \psi^{-1}_{(R_0 + R_1 \bmod r_2)}(\tilde{d}^{(\lfloor j/W \rfloor)})$
21:     $k \leftarrow k \oplus \mathsf{bit}(\hat{d}, j \bmod W)$
22:     $j \leftarrow j - \neg k$
23: end while
24: $c \leftarrow R_0 \bmod N$
    return $c$
```

To prevent the combined attacks we introduced, it becomes necessary to prevent template and differential side-channel techniques. A possible fix consists in randomizing the internal registers R_0 and R_1 before the multiplication so that even if we have to compute $R_0 \cdot R_0$ the representation of the two operands will be different. The Line 11 of Alg. 1 can be replaced by the following:

```
1: $r_3 \leftarrow \mathsf{random}(1, 2^\lambda - 1)$
2: $R_2 \leftarrow R_k + r_3 \cdot N \bmod Nr_2$
3: $R_0 \leftarrow R_0 \cdot R_2 \bmod Nr_2$
```

This modification adds to the cost of Alg. 1 one more register R_2, one modular multiplication with addition and the selection of a random value r_3 at each turn of the loop. Even if the exponent blinding is removed by fault, none of the attacks presented before can be applied now as multiplication and squaring operations are no more distinguishable. A similar modification can be applied to the scalar multiplication algorithm [20, Alg. 4] but at a higher cost. One needs to randomize each coordinates of the elliptic curve point which means, in the case of classical projective coordinates, an overhead of 3 modular multiplications, 3 random values and a point buffer. Moreover, this technique might not be sufficient on most normalized curves, *i.e.* NIST curves, as their modulus have very particular forms that can still allow for side-channel leakage on randomized coordinates. A more costly alternative solution consists in using a randomized multi-precision multiplication as proposed in [14] and [21, Sec. 2.7].

It is important also to notice that in practice the public exponent and the value $\varphi(N)$ can be unknown when computing an exponentiation. In that case, the exponent cannot be blinded and the calculation verified. Although there are alternative solutions, as for instance those proposed by Joye in [11], it only applies to particular cases. Hence it could be sometimes impossible to apply the blinding on the exponent. However our improved Algorithm 2 is resistant to combined attacks even when those values are unknown.

To the best of our knowledge, the only other exponentiation algorithm resistant against combined attacks is the algorithm proposed by Giraud [9] based on the Montgomery ladder. However it only protects from a corruption of the data registers, the integrity of the exponent is not assured contrary to Schmidt *et al.* algorithm.

6 Conclusion

We have presented in this paper two new attacks which threatens the combined attack resistant implementations Schmidt *et al.* published in [20]. Our first technique is a single fault injection technique which can recover with few faulted ciphertexts the secret exponent. This attack was possible due to a flaw in the infective computation countermeasure proposed by the original authors. The second method combines fault injection with differential analysis to reach the same objective. Introducing those new vulnerabilities lead us to propose an improved version of this algorithm which offer better protection against the different attacks based on side channel analysis and fault injection techniques.

References

1. Amiel, F., Feix, B., Tunstall, M., Whelan, C., Marnane, W.: Distinguishing multiplications from squaring operations. Selected Areas in Cryptography, LNCS 5381, 346–360 (2008)
2. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and active combined attacks: combining fault attacks and side channel analysis. In: Breveglieri, I., Gueron, S., Koren, I., Naccache, D., Seifert, J. (eds.) FDTC. pp. 92–102. IEEE Computer Society, Washington, DC, USA (2007)
3. Berzati, A., Canovas-Dumas, C., Goubin, L.: Public key perturbation of randomized RSA implementations. In: Mangard, S., Standaert, F.X. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010, Lecture Notes in Computer Science, vol. 6225, pp. 306–319. Springer Berlin / Heidelberg (2010)
4. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. IEEE Transactions on Computers 53, 760–768 (2004)

5. Courrège, J., Feix, B., Roussellet, M.: Simple power analysis on exponentiation revisited. Smart Card Research and Advanced Application 6035, 65–79 (2010)
6. Dehbaoui, A., Dutertre, J., Robisson, B., Orsatelli, P., Maurine, P., Tria, A.: Injection of transient faults using electromagnetic pulses -practical results on a cryptographic system-. Cryptology ePrint Archive, Report 2012/123 (2012)
7. Fan, J., Gierlichs, B., Vercauteren, F.: To infinity and beyond: Combined attack on ECC using points of low order. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011, Lecture Notes in Computer Science, vol. 6917, pp. 143–159. Springer Berlin / Heidelberg (2011)
8. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Ç., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2001, Lecture Notes in Computer Science, vol. 2162, pp. 251–261. Springer Berlin / Heidelberg (2001)
9. Giraud, C.: An RSA implementation resistant to fault attacks and to simple power analysis. Computers, IEEE Transactions on 55(9), 1116–1120 (2006)
10. Hanley, N., Tunstall, M., Marnane, W.: Using templates to distinguish multiplications from squaring operations. International Journal of Information Security 10, 255–266 (2011)
11. Joye, M.: Protecting RSA against fault attacks: The embedding method. In: Breveglieri, L., Koren, I., Naccache, D., Oswald, E., Seifert, J.P. (eds.) Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, pp. 41–45. IEEE Computer Society (2009)
12. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of computation 48, 203–209 (1987)
13. Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO '96, Lecture Notes in Computer Science, vol. 1109, pp. 104–113. Springer Berlin / Heidelberg (1996)
14. Medwed, M., Herbst, C.: Randomizing the Montgomery multiplication to repel template attacks on multiplicative masking. COSADE 2010 (2010)
15. Messerges, T., Dabbish, E., Sloan, R.: Investigations of power analysis attacks on smartcards. In: USENIX workshop on Smartcard Technology. pp. 151–161 (1999)
16. Miller, V.: Use of elliptic curves in cryptography. In: Advances in Cryptology – CRYPTO'85 Proceedings. pp. 417–426 (1986)
17. Poucheret, F., Tobich, K., Lisart, M., Chusseau, L., Robisson, B., Maurine, P.: Local and direct EM injection of power into CMOS integrated circuits. In: FDTC. pp. 100–104 (2011)
18. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21, 120–126 (1978)
19. Schindler, W., Itoh, K.: Exponent blinding does not always lift (partial) SPA resistance to higher-level security. In: Lopez, J., Tsudik, G. (eds.) Applied Cryptography and Network Security, Lecture Notes in Computer Science, vol. 6715, pp. 73–90. Springer Berlin / Heidelberg (2011)
20. Schmidt, J.M., Tunstall, M., Avanzi, R., Kizhvatov, I., Kasper, T., Oswald, D.: Combined implementation attack resistant exponentiation. In: Abdalla, M., Barreto, P. (eds.) Progress in Cryptology - LATINCRYPT 2010, Lecture Notes in Computer Science, vol. 6212, pp. 305–322. Springer Berlin / Heidelberg (2010)
21. Verneuil, V.: Elliptic Curve Cryptography and Security of Embedded Devices. Ph.D. thesis, Université de Bordeaux (2012)

22. Witteman, M., van Woudenberg, J., Menarini, F.: Defeating RSA multiply-always and message blinding countermeasures. In: Kiayias, A. (ed.) Topics in Cryptology – CT-RSA 2011, Lecture Notes in Computer Science, vol. 6558, pp. 77–88. Springer Berlin / Heidelberg (2011)