Attacking RSA–CRT Signatures with Faults on Montgomery Multiplication

Pierre-Alain Fouque[1,2] Nicolas Guillermin[3] Delphine Leresteux[3] Mehdi Tibouchi[4] Jean-Christophe Zapalowicz[2]

[1] École normale supérieure
pierre-alain.fouque@ens.fr
[2] INRIA Rennes
jean-christophe.zapalowicz@inria.fr
[3] DGA IS
nicolas.guillermin@m4x.org
delphine.leresteux@dga.defense.gouv.fr
[4] NTT Secure Platform Laboratories
tibouchi.mehdi@lab.ntt.co.jp

Abstract. In this paper, we present several efficient fault attacks against implementations of RSA–CRT signatures that use modular exponentiation algorithms based on Montgomery multiplication. They apply to any padding function, including randomized paddings, and as such are the first fault attacks effective against RSA–PSS.

The new attacks work provided that a small register can be forced to either zero, or a constant value, or a value with zero high-order bits. We show that these models are quite realistic, as such faults can be achieved against many proposed hardware designs for RSA signatures.

Keywords: Fault Attacks, Montgomery Multiplication, RSA–CRT, RSA–PSS

1 Introduction

The RSA signature scheme is one of the most used schemes nowadays. An RSA signature is computed by applying some encoding function to the message, and raising the result to d-th power modulo N, where d and N are the private exponent and the public modulus respectively. This modular exponentiation is the costlier part of signature generation, so it is important to implement it efficiently. A very commonly used speed-up is the RSA–CRT signature generation, where the exponentiation is carried out separately modulo the two factors of N, and the results are then recombined using the Chinese Remainder Theorem. However, when unprotected, RSA–CRT signatures are vulnerable to the so-called Bellcore attack first introduced by Boneh, DeMillo and Lipton in [6], and later refined in a number of subsequent publications [7,2,43]: an attacker who knows the padded message and is able to inject a fault in one of the two half-exponentiations can factor the public modulus using a faulty signature with a simple GCD computation.

Many workarounds have been proposed to patch this vulnerability, including extra computations and sanity checks of intermediate and final results. A recent taxonomy of these countermeasures is given in [32]. The simplest countermeasure may be to verify the signature before releasing it. This is reasonably cheap if the public exponent e is small and available in the signing device. In some cases, however, e is not small, or even not given—e.g. the JavaCard API does not provide it [29]. Another approach is to use an extended modulus. Shamir's trick [34] was the first such technique to be proposed; later refinements were suggested that also protect CRT recombination when it is computed using Garner's formula [5,12,41,14]. Finally, yet another way to protect RSA–CRT signatures against faults is to use redundant exponentiation algorithms, such as the Montgomery Ladder. Papers including [19,32] propose such countermeasures. Regardless of the approach, RSA–CRT fault countermeasures tend to be rather costly: for example, Rivain's countermeasure [32] has a stated overhead of 10% compared to an unprotected implementation, and is purportedly more efficient than previous works including [19,41].

Relatedly, while Boneh et al.'s original fault attack does not apply to RSA signatures with probabilistic encoding functions, some extensions of it were proposed to attack randomized ad hoc padding schemes such as ISO 9796-2 and EMV [15,17]. However, Coron and Mandal [16] were able to prove that Bellare and Rogaway's padding scheme RSA–PSS [3,4,22] is secure against *random* faults in the random oracle model. In other words, if injecting a fault on the half-exponentiation modulo the second factor q of N produces a result that can be modeled as uniformly distributed modulo q, then the result of such a fault cannot be used to break RSA–PSS signatures. It is tempting to conclude that using RSA–PSS should enable signers to dispense with costly RSA–CRT countermeasures. In this paper, we argue that this is not necessarily the case.

Our contributions. The RSA–CRT implementations targeted in this paper use the state-of-the-art modular multiplication algorithm due to Montgomery [27], which avoids the need to compute actual divisions on large integers, replacing them with only multiplications and bit shifts. A typical implementation of the Montgomery multiplication algorithm will use small registers to store precomputed values or short integer variables throughout the computation. The size of these registers varies with the architecture, from a single bit in certain hardware implementations to 16 bits, 32 bits or more in software. This paper presents several fault attacks on these small registers during Montgomery multiplication, that cause the result of one of the half-exponentiations to be unusually small. The factorization of N can then be recovered using a GCD, or an approximate common divisor algorithm such as [20,10,13].

We consider three models of faults on the small registers. In the first model, one register can be forced to zero. In that case, we show that causing such a fault in the inverse Montgomery transformation of the result of a half-exponentiation, or a few earlier consecutive Montgomery multiplications, yields a faulty signature which is a multiple of the corresponding factor q of N. Hence, we can factor N by taking a simple GCD. In the second model, another register can be forced to some (possibly unknown) constant value throughout the inverse Montgomery transformation of the result of a half-exponentiation, or a few earlier consecutive Montgomery multiplications. A faulty signature in this model is a close multiple of the corresponding factor q of N, and we can thus factor N using an approximate common divisor algorithm. Finally, the third model makes it possible to force some of the higher-order bits of one register to zero. We show that, while injecting one such fault at the end of the inverse Montgomery transformation results in a faulty signature that isn't usually close enough to a multiple of q to reveal the factorization of N on its own, a moderate number of faulty signatures (a dozen or so) obtained using that process are enough to factor N.

The RSA padding scheme used for signing, whether deterministic or probabilistic, is irrelevant in our attacks. In particular, RSA–PSS implementations are also vulnerable. Of course, this does not contradict the security result due to Coron and Mandal [16], as the faults we consider are strongly non-random. Our results do suggest, however, that exponentiation algorithms based on Montgomery multiplication are quite sensitive to a very realistic type of fault attacks and that using RSA–CRT countermeasures is advisable even for RSA–PSS.

Organization of the paper. In §2, we recall some background material on the Montgomery multiplication algorithm, on modular exponentiation techniques, and on RSA–CRT signatures. Our new attacks are then described in §§3–5, corresponding to three different fault models: null faults, constant faults, and zero high-order bits faults. Finally, in §6, we discuss the applicability of our fault models to concrete hardware implementations of RSA–CRT signatures, and find that many proposed designs are vulnerable.

2 Preliminaries

2.1 Montgomery multiplication

First proposed by Montgomery in [27], the Montgomery multiplication algorithm provides a fast way method for computing modular multiplications and squarings. Indeed, the Montgomery multiplication algorithm only uses multiplications, additions and shifts, and its cost is about twice that of a simple multiplication (compared to 2.5 times for a multiplication and a Barett reduction), without imposing any constraint on the modulus.

Usually, one of two different techniques is used to compute Montgomery multiplication: either Separate Operand Scanning (SOS), or Coarsely Integrated Operand Scanning (CIOS). Consider a device whose processor or coprocessor architecture has r-bit registers (typically $r = 1, 8, 16, 32$ or 64 bits). Let $b = 2^r$, q be the (odd) modulus with respect to which multiplications are carried out, k the number of r-bit registers used to store q, and $R = b^k$, so that $q < R$ and $\gcd(q, R) = 1$. The SOS variant consists in using the Montgomery reduction after the multiplication: for an input A such that $A < Rq$, it computes $\mathrm{Mgt}(A) \equiv AR^{-1} \pmod{q}$, with $0 \leq \mathrm{Mgt}(A) < q$. The CIOS mixes the reduction algorithm with the previous multiplication step: considering x and y with $xy < Rq$, it computes $\mathrm{CIOS}(x, y) = xyR^{-1} \bmod q$ with $\mathrm{CIOS}(x, y) < q$.

Algorithm 2 presents the main steps of the CIOS variant, which will be used thereafter. However, replacing the CIOS by the SOS or any other variant proposed in [24] does not protect against any of our attacks.

Among all the variants proposed for this algorithm, the optimization of [42] is well-known: if $Rq > xy$, then the result of algorithm 2 without the final reduction (line 8) is between 0 and $2q$. Therefore for an exponentiation algorithm, there is no need to execute this final reduction if $R > 4q$. Besides its efficiency, this variant has the advantage of thwarting timing attacks [33,9,1], which essentially rely on detecting whether the reduction is executed or not. Nevertheless, these attacks do not work with randomized paddings, since the attacker needs to carefully choose the message. On the contrary, our attacks work on any padding, with or without this reduction.

```
function $\textsc{Sign}_{\text{RSA–CRT}}(m)$
    $M \leftarrow \mu(m) \in \mathbb{Z}_N$    ▷ message encoding
    $M_p \leftarrow M \bmod p$
    $M_q \leftarrow M \bmod q$
    $S_p \leftarrow M_p^{d_p} \bmod p$
    $S_q \leftarrow M_q^{d_q} \bmod q$
    $t \leftarrow S_p - S_q$
    if $t < 0$ then $t \leftarrow t + p$
    $S \leftarrow S_q + \big((t \cdot \pi) \bmod p\big) \cdot q$
    return $S$
```

Fig. 1. RSA–CRT signature generation with Garner's recombination. The reductions d_p, d_q modulo $p-1, q-1$ of the private exponent are precomputed, as is $\pi = q^{-1} \bmod p$.

```
function CIOS$(x, y)$
    $a \leftarrow 0$
    $y_0 \leftarrow y \bmod b$
    for $j = 0$ to $k-1$ do
        $a_0 \leftarrow a \bmod b$
        $u_j \leftarrow (a_0 + x_j \cdot y_0) \cdot q' \bmod b$
        $a \leftarrow \left\lfloor \frac{a + x_j \cdot y + u_j \cdot q}{b} \right\rfloor$
    if $a \geq q$ then $a \leftarrow a - q$
    return $a$
```

Fig. 2. The Montgomery multiplication algorithm. The x_i's and y_i's, $i = 0, \ldots, k-1$, are the digits of x and y in base b; $q' = q^{-1} \bmod b$ is precomputed. The returned value is $(xy \cdot b^{-k} \bmod q)$.

2.2 Exponentiation algorithms using Montgomery multiplication

Montgomery reduction is especially interesting when used as part of a modular exponentiation algorithm. A large number of such exponentiation algorithms are known, including the Square-and-Multiply algorithm from either the least or the most significant bit of the exponent, the Montgomery Ladder (used as a side-channel countermeasure against cache analysis, branch analysis, timing analysis and power analysis), the Square-and-Multiply k-ary algorithm (which boasts greater efficiency thanks to fewer multiplications), etc. Detailed descriptions of the first three of those are given in Figure 3.

Note that using the Montgomery multiplications inside any exponentiation algorithm requires all variables to be in Montgomery representation ($\bar{x} = xR \bmod q$ is the Montgomery representation of x) before applying the exponentiation process. In line 2 of each algorithm from Figure 3, the message is transformed into Montgomery representation by computing $CIOS(x, R^2) = xR^2R^{-1} \bmod q = \bar{x}$. At the end, the very last CIOS call allows to revert to the classical representation by performing a Montgomery reduction: $CIOS(\bar{A}, 1) = (\bar{A} \cdot 1)R^{-1} \bmod q = ARR^{-1} \bmod q = A$. Finally the other CIOS steps compute the product in Montgomery representation: $CIOS(\bar{A}, \bar{B}) = (AR)(BR)R^{-1} \bmod q = \overline{AB}$.

Square-and-Multiply MSB

```
function $\textsc{Exp}_{\mathrm{MSB}}(x, e, q)$
    $\bar{x} \leftarrow \mathrm{CIOS}(x, R^2 \bmod q)$
    $A \leftarrow R \bmod q$
    for $i = t$ down to $0$ do
        $A \leftarrow \mathrm{CIOS}(A, A)$
        if $e_i = 1$ then
            $A \leftarrow \mathrm{CIOS}(A, \bar{x})$
    $A \leftarrow \mathrm{CIOS}(A, 1)$
    return $A$
```

Square-and-Multiply LSB

```
function $\textsc{Exp}_{\mathrm{LSB}}(x, e, q)$
    $\bar{x} \leftarrow \mathrm{CIOS}(x, R^2 \bmod q)$
    $A \leftarrow R \bmod q$
    for $i = 0$ to $t$ do
        if $e_i = 1$ then
            $A \leftarrow \mathrm{CIOS}(A, \bar{x})$
        $\bar{x} \leftarrow \mathrm{CIOS}(\bar{x}, \bar{x})$
    $A \leftarrow \mathrm{CIOS}(A, 1)$
    return $A$
```

Montgomery Ladder

```
function $\textsc{Exp}_{\mathrm{Ladder}}(x, e, q)$
    $\bar{x} \leftarrow \mathrm{CIOS}(x, R^2 \bmod q)$
    $A \leftarrow R \bmod q$
    for $i = t$ down to $0$ do
        if $e_i = 0$ then
            $\bar{x} \leftarrow \mathrm{CIOS}(A, \bar{x})$
            $A \leftarrow \mathrm{CIOS}(A, A)$
        else if $e_i = 1$ then
            $A \leftarrow \mathrm{CIOS}(A, \bar{x})$
            $\bar{x} \leftarrow \mathrm{CIOS}(\bar{x}, \bar{x})$
    $A \leftarrow \mathrm{CIOS}(A, 1)$
    return $A$
```

Fig. 3. The three exponentiation algorithms considered in this paper. In each case, $e_0, \ldots, e_t$ are the bits of the exponent e (from the least to the most significant), b is the base in which computations are carried out ($\gcd(b, q) = 1$) and $R = b^k$.

2.3 RSA–CRT signature generation

Let $N = pq$ be a n-bit RSA modulus. The public key is denoted by (N, e) and the associated private key by (p, q, d). For a message M to be signed, we note $S = m^d \bmod N$ the corresponding signature, where m is deduced from M by an encoding function, possibly randomized. A well-known optimization of this operation is the RSA–CRT which takes advantage of the decomposition of N into prime factors. By replacing a full exponentiation of size n by two $n/2$, it divides the computational cost by a factor of around 4. Therefore RSA–CRT is almost always employed: for example, OpenSSL [40] as well as the JavaCard API [29] use it.

Recovering S from its reductions S_p and S_q modulo p and q can be done either by the usual CRT reconstruction formula (1) below, or using the recombination technique (2) due to Garner [18]:

$$S = (S_q \cdot p^{-1} \mod q) \cdot p + (S_p \cdot q^{-1} \mod p) \cdot q \mod N \tag{1}$$

$$S = S_q + q \cdot (q^{-1} \cdot (S_p - S_q) \mod p) \tag{2}$$

Garner's formula (2) does not require a reduction modulo N, which is interesting for efficiency reasons and also because it prevents certain fault attacks [8]. On the other hand, it does require an inverse Montgomery transformation $S_q = CIOS(\bar{S}_q, 1)$, whereas that step is not necessary for formula (1), as it can be mixed with the multiplication with $q^{-1} \mod p$. This is an important point, as some of our attacks specifically target the inverse Montgomery transformation. The main steps of the RSA–CRT signature generation with Garner's recombination are recalled in Figure 1.

3 Null Faults

We first consider a fault model in which the attacker can force the precomputed value q' to zero in certain calls to the CIOS algorithm during the computation of S_q.

Under suitable conditions, we will see that such faults can cause the q-part of the signature to be erroneously evaluated as $\widetilde{S}_q = 0$, which makes it possible to retrieve the factor q of N from one such faulty signature $\widetilde{S}$, as $q = \gcd(\widetilde{S}, N)$.

3.1 Attacking CIOS$(A, 1)$

Suppose first that the fault attacker can force q' to zero in the very last CIOS computation during the evaluation of S_q, namely the computation of CIOS$(A, 1)$. In that case, the situation is quite simple.

Theorem 1. *A faulty signature $\widetilde{S}$ generated in this fault model is a multiple of q (for any of the exponentiation algorithms considered herein and regardless of the encoding function involved, probabilistic or not).*

Proof. The faulty value $\widetilde{q'} = 0$ causes all of the variables u in the CIOS loop to vanish; indeed, for $j = 0, \dots, k-1$, they evaluate to:

$$\widetilde{u_j} = (a_0 + A_j \cdot 1) \cdot \widetilde{q'} \bmod 2^r = 0.$$

As a result, the value $\widetilde{S_q}$ computed by this CIOS loop can be written as:

$$\widetilde{S_q} = \left\lfloor \left(\left\lfloor \cdots \left\lfloor \left(\lfloor A_0 \cdot 2^{-r} \rfloor + A_1 \right) \cdot 2^{-r} \right\rfloor + \cdots \right\rfloor + A_{k-1} \right) \cdot 2^{-r} \right\rfloor .$$

Now, the values A_j are r-words, i.e. $0 \leq A_j \leq 2^r - 1$. It follows that each of the integer divisions by 2^r evaluate to zero, and hence $\widetilde{S_q} = 0$. As a result, the faulty signature $\widetilde{S}$ is a multiple of q as stated. □

It is thus easy to factor N with a single faulty signature $\widetilde{S}$, by computing $\gcd(\widetilde{S}, N)$. Note also that if this last CIOS step is computed as $\mathrm{CIOS}(1, A)$ instead of $\mathrm{CIOS}(A, 1)$, the formulas are slightly different but the result still holds.

3.2 Attacking consecutive CIOS steps

If Garner recombination is not used or the computation of $\mathrm{CIOS}(A, 1)$ is somehow protected against faults, a similar result can be achieved by forcing q' to zero in earlier calls to CIOS, provided that a certain number of successive CIOS executions are faulty.

Assuming that the values $\bar{x}$ and A in Montgomery representation are uniformly distributed modulo q before the first faulty CIOS, we show in Appendix A that faults across $\ell = \lceil \log_2 \lceil \log_2 q \rceil \rceil$ iterations in the loop of the exponentiation algorithm are enough to ensure that $\widetilde{S_q}$ will evaluate to zero with probability at least 1/2. For example, if q is a 512-bit prime, we have $\ell = 9$. This means that forcing q' to zero in 9 iterations (from 9 to 18 calls to CIOS depending on the exponentiation algorithm under consideration and on the input bits) is enough to factor the modulus at least 50% of the time—and more faulty iterations translate to higher success rates.

The k-ary Square-and-Multiply exponentiation process is vulnerable as well whatever k is, although details are omitted for lack of space.

Simulation results. We have carried out a simulation of null faults on consecutive CIOS steps for each of the three exponentiation process algorithms, with varying numbers of faulty iterations; for the Square-and-Multiply MSB and the Montgomery Ladder algorithms, two sets of experiments have been conducted for each parameter set: one with faults starting from the first iteration, and another one with faults starting from a random iteration somewhere in the exponentiation loop. Results are collected in Table 1 in Appendix C. As we can see, success rates are noticeably higher than the lower bounds obtained in Appendix A.

4 Constant Faults

In this section, we consider a different fault model, in which the fault attacker can force the variables u_j in the CIOS algorithm to some (possibly unknown) constant value $\widetilde{u}$.

Just as with null faults, we consider two scenarios: one in which the last CIOS computation is attacked, and another in which several inner consecutive CIOS computations in the exponentiation algorithm are targeted.

4.1 Attacking CIOS$(A, 1)$

Faults on all iterations. Consider first the case when faults are injected in all iterations of the very last CIOS computation. In other words, the device computes CIOS$(A, 1)$, except that the variables u_j, $j = 0, \dots, k-1$, are replaced by a fixed, possibly unknown value $\widetilde{u}$. In that case, we show that a single faulty signature is enough to factor N and recover the secret key. The key result is as follows.

Theorem 2. *Let $\widetilde{S}$ be a faulty signature obtained in the fault model described above. Then, $(2^r - 1) \cdot \widetilde{S}$ is a close multiple of q with error size at most 2^{r+1}, i.e. there exists an integer T such that:*

$$\left|(2^r - 1) \cdot (\widetilde{S} + 1) - qT\right| \leq 2^{r+1}.$$

Proof. Up to the possible subtraction of q, which clearly doesn't affect our result, the value $\widetilde{S}_q$ computed in the faulty execution of CIOS$(A, 1)$ can be written as:

$$\widetilde{S}_q = \left\lfloor \left(\left\lfloor \cdots \left\lfloor \left(\lfloor (A_0 + \widetilde{u} \cdot q) \cdot 2^{-r} \rfloor + A_1 + \widetilde{u} \cdot q \right) \cdot 2^{-r} \right\rfloor + \cdots \right\rfloor + A_{k-1} + \widetilde{u} \cdot q \right) \cdot 2^{-r} \right\rfloor.$$

We claim that this value $\widetilde{S}_q$ is close to the real number $\widetilde{u} \cdot q/(2^r - 1)$. Indeed, on the one hand, by using the fact that $\lfloor x \rfloor \leq x$ for all x and $A_j \leq 2^r - 1$ for $j = 0, \dots, k-1$, we obtain:

$$\widetilde{S}_q \leq \frac{A_0 + \widetilde{u} \cdot q}{2^{rk}} + \cdots + \frac{A_{k-1} + \widetilde{u} \cdot q}{2^r} \leq \frac{1}{2^r - 1}(2^r - 1 + \widetilde{u} \cdot q) \leq \frac{\widetilde{u} \cdot q}{2^r - 1} + 1.$$

On the other hand, since $\lfloor x \rfloor > x - 1$ and $A_j \geq 0$, we get:

$$\begin{aligned} \widetilde{S}_q &> \frac{\widetilde{u} \cdot q}{2^{rk}} - \frac{1}{2^{r(k-1)}} + \cdots + \frac{\widetilde{u} \cdot q}{2^r} - 1 = \frac{1 - 2^{-rk}}{2^r - 1} \cdot (\widetilde{u} \cdot q - 2^r) \\ &> \frac{\widetilde{u} \cdot q}{2^r - 1} - \frac{\widetilde{u}}{2^r - 1} \cdot \frac{q}{2^{rk}} - \frac{2^r}{2^r - 1} > \frac{\widetilde{u} \cdot q}{2^r - 1} - 3 \end{aligned}$$

as we have both $\widetilde{u} \leq 2^r - 1$ and $q < 2^{rk}$. As a result, we obtain:

$$\left| (\widetilde{S}_q + 1) - \frac{\widetilde{u} \cdot q}{2^r - 1} \right| \leq 2$$

and hence:

$$\left| (2^r - 1) \cdot (\widetilde{S}_q + 1) - \widetilde{u} \cdot q \right| \leq 2^{r+1}.$$

Since $\widetilde{S} = \widetilde{S}_q + ((\tilde{t} \cdot \pi) \bmod p) \cdot q$, the stated result follows, with $T = \widetilde{u} + (2^r - 1) \cdot ((\tilde{t} \cdot \pi) \bmod p)$. □

Thus, a single faulty signature yields a value $V = (2^r - 1) \cdot (\widetilde{S} + 1) \bmod N$ which is very close to a multiple of q. It is easy to use this value to recover q itself. Several methods are available:

- If r is small (say 8 or 16), it may be easiest to just use exhaustive search: q is found among the values $\gcd(V + X, N)$ for $|X| \leq 2^{r+1}$, and hence can be retrieved using around 2^{r+2} GCD computations.
- A more sophisticated option, which may be interesting for $r = 32$, is the baby step, giant step-like algorithm by Chen and Nguyen [10], which runs in time $\widetilde{O}(2^{r/2})$.
- Alternatively, for any r up to half of the size of q, one can use Howgrave-Graham's algorithm [20] based on Coppersmith techniques. It is the fastest option unless r is very small (a simple implementation in Sage [36] runs in about 1.5 ms on our standard desktop PC with a 512-bit prime q for a any r up to ≈ 160 bits, whereas exhaustive search already takes over one second for $r = 16$).

Faults on most iterations. Howgrave-Graham's algorithm is especially relevant if the constant faults do not start at the very first iteration in the CIOS loop. More precisely, suppose that the fault attacker can force the variables u_j to a constant value $\widetilde{u}$ not for all j but for $j = j_0, j_0+1, \dots, k-1$ for some j_0.

Then, the same computation as in the proof of Theorem 2 yields the following bound on $\widetilde{S}_q$:

$$\frac{\widetilde{u} \cdot q}{2^r - 1} - 2^{rj_0} - 2 < \widetilde{S}_q \le \frac{\widetilde{u} \cdot q}{2^r - 1} + 2^{rj_0} + 1.$$

It follows that $(2^r-1)\cdot\widetilde{S}$ is a close multiple of q with error size $\lesssim 2^{r(j_0+1)}$.

Now note that Howgrave-Graham's algorithm [20] will recover q given N and a close multiple with error size at most $q^{1/2-\varepsilon}$. This means that one faulty signature $\widetilde{S}$ is enough to factor N as long as $j_0 + 1 < k/2$, i.e. the constant faults start in the first half of the CIOS loop.

Moreover, if the faults start even later, a single signature will no longer suffice, but q can be recovered if several faults are available, using the generalization of Howgrave-Graham's algorithm due to Cohn and Heninger [13]. That algorithm is discussed in a different context in §5.

4.2 Attacking other CIOS steps

As in §3.2, if Garner recombination is not used or CIOS$(A, 1)$ is protected against faults, we can adapt the previous attack to target earlier calls to CIOS and still reveal the factorization of N. However, the attack requires two faulty signatures with the same constant fault $\widetilde{u}$. Details are given in Appendix B.

In short, depending on the ratios $q/2^{\lceil \log_2 q \rceil}$ and $\widetilde{u}/(2^r-1)$, two faulty signatures $\widetilde{S}$, $\widetilde{S}'$ with the same faulty value $\widetilde{u}$ have a certain probability of being equal modulo q. Thus, we recover q as $\gcd(N, \widetilde{S} - \widetilde{S}')$. This attack works with the Square-and-Multiply LSB and Montgomery Ladder algorithms, but not with Square-and-Multiply MSB exponentiation.

Simulation results are presented in Table 2 in Appendix C. For various 512-bit primes q, the attack has been carried out for 1000 pairs of random messages, with a random constant fault $\widetilde{u}$ for each pair. It is successful if the two resulting faulty signatures $\widetilde{S}$, $\widetilde{S}'$ satisfy $\gcd(N, \widetilde{S} - \widetilde{S}') = q$.

5 Zero High-Order Bits Faults

In this section, we consider yet another fault model, in which the fault attacker targets the very last iteration in the evaluation of CIOS$(A, 1)$

during the computation of S_q. We assume that the attacker is able to force a certain number h of the highest-order bits of u_{k-1} to zero, possibly but not necessarily all of them (i.e. $1 \leq h \leq r$). Then, while a single faulty signature is typically not sufficient to factor the modulus, multiple such signatures will be enough if h is not too small.

Theorem 3. *Let $\widetilde{S}$ be a faulty signature obtained in this fault model. Then, $\widetilde{S}$ is a close multiple of q with error size at most $2^{-h} \cdot q + 1$, i.e. there exists an integer T such that $|\widetilde{S} - qT| \leq 2^{-h} \cdot q + 1$.*

Proof. The iterations numbered $0, 1, \ldots, k-2$ in the evaluation of $\mathrm{CIOS}(A, 1)$ are all computed correctly. Let $a_1, a_2, \ldots, a_{k-1}$ be the values of the variable a at the end of these respective iterations. We have:

$$a_1 = \left\lfloor \frac{0 + A_0 + u_0 \cdot q}{2^r} \right\rfloor \leq \frac{0 + (2^r - 1) + (2^r - 1) \cdot q}{2^r} \leq q + 1$$
$$a_2 = \left\lfloor \frac{a_1 + A_1 + u_1 \cdot q}{2^r} \right\rfloor \leq \frac{(q+1) + (2^r - 1) + (2^r - 1) \cdot q}{2^r} \leq q + 1$$

and it is then easy to see by induction that $0 \leq a_{k-1} \leq q + 1$. Then, the computation of the last iteration is attacked, with a value $\widetilde{u}_{k-1}$ of u satisfying $0 \leq \widetilde{u}_{k-1} \leq 2^{r-h} - 1$. Thus, the value of a after that iteration becomes:

$$a_k = \left\lfloor \frac{a_{k-1} + A_{k-1} + \widetilde{u}_{k-1} \cdot q}{2^r} \right\rfloor \leq \frac{(q+1) + (2^r - 1) + (2^{r-h} - 1) \cdot q}{2^r} \leq \frac{q}{2^h} + 1.$$

In particular, $a_k < q$, so that the q-part of the signature $\widetilde{S}_q$ is a_k, and hence $|\widetilde{S}_q| \leq 2^{-h} \cdot q + 1$. Since $\widetilde{S}_q = \widetilde{S} - qT$ for $T = (t \cdot p) \bmod p$, this concludes the proof. □

Note that exactly the same result still holds if, in addition to u_{k-1}, previous values of u are attacked in the same fashion as well, so there is no need to synchronize the attack extremely precisely so as to target only the last iteration.

Now, recovering q from faulty signatures of the form $\widetilde{S}$ is a partial approximate common divisor (PACD) problem, as we know one exact multiple of q, namely N, and several close multiples, namely the faulty signatures. Since the error size $\approx q/2^h$ is rather large relative to q, the state-of-the-art algorithm to recover q in that case is the one proposed by Cohn and Heninger [13] using multivariate Coppersmith techniques.

The algorithm by Cohn and Heninger is likely to recover the common divisor $q \approx N^{1/2}$ given ℓ close multiples $\widetilde{S}^{(1)}, \ldots, \widetilde{S}^{(\ell)}$ provided that the error size is significantly less than $N^{(1/2)^{1+1/\ell}}$. Thus, we should have:

$$\log_2 q - h \lesssim \left(\frac{1}{2}\right)^{1+1/\ell} \log_2 N \approx 2^{1/\ell} \cdot \log_2 q.$$

Hence, if the faults cancel the top h bits of u_{k-1}, we need ℓ of them to factor the modulus, where:

$$\ell \gtrsim -\frac{1}{\log_2\left(1 - \frac{h}{\log_2 q}\right)}. \tag{3}$$

In practice, if a few more faults can be collected, it is probably preferable to simply use the linear case of the Cohn-Heninger attack (the case $t = k = 1$ in their paper [13]), since it is much easier to implement (as it requires only linear algebra rather than Gröbner bases) and involves lattice reduction in a lattice of small dimension that is straightforward to construct. More precisely, reducing the lattice L generated by the rows of the following matrix:

$$\begin{pmatrix} B & & & -\widetilde{S}^{(1)} \\ & \ddots & & \vdots \\ & & B & -\widetilde{S}^{(\ell)} \\ & & & N \end{pmatrix}$$

where $B = 2^{kr-h}$, gives a lattice basis consisting of affine forms the first ℓ of which vanish on the vector of "error values" $\big(\widetilde{S}_q^{(1)}/B, \ldots, \widetilde{S}_q^{(\ell)}/B\big)$ if ℓ is large enough. More precisely, they vanish on this vector modulo q, and also do over the integers provided that their coefficients are much smaller than q. If we assume that L behaves like a random lattice, the length of vectors in the reduced basis should be roughly $\det(L)^{1/\dim(L)} = (B^\ell \cdot N)^{1/(\ell+1)}$. This gives the condition:

$$B^\ell \cdot N \ll q^{\ell+1}$$
$$\ell(\log_2 q - h) + \log_2 N \lesssim (\ell + 1)\log_2 q.$$

Hence, this method should recover the error vector and thus make it possible to factor N provided that:

$$\ell \gtrsim \frac{\log_2 q}{h} \tag{4}$$

which is always a worse bound than (3) but usually not by a very large margin. Table 3 in Appendix C gives the theoretical number of faulty signatures required to factor N for various values of h, both in the general attack by Cohn and Heninger and in the simplified linear case.

We carried out a simulation of the linear version of the attack on a 1024-bit modulus N with various values of h, and found that it works very well in practice with a number of faulty signatures consistent with the theoretical minimum. The results are collected in Table 4. The attack is also quite fast: a naive implementation in Sage [36] runs in a fraction of a second on a standard PC.

6 Fault Models

In this section we discuss how realistic the setup of the attacks described above can be. In principle, all the RSA–CRT implementations using Montgomery multiplication may be vulnerable, but we have to note that the fault setup (and how realistic it is) depends heavily on implementation choices, since many variations around the algorithm from Figure 2 have been proposed in recent literature. After a discussion about the characteristics of the tools needed to get the desired effects, we focus on several implementation proposals [39,25,21,28,37,26,11], chosen for their relevance, and discuss whether our fault model is realistic in those settings.

6.1 Characteristics of the perturbation tool

First all the perturbations needed to carry out our attacks need to be well controlled and local to some gates of the chip. Therefore, before the attacker implements the fault, they need to identify the localization of the vulnerable gates and registers. The null fault attacks described in §3 need either a q' value set to 0, or multiple consecutive faults in line 6 of the main loop of $CIOS(A, 1)$ or during multiple consecutive $CIOS$. The attacks described in §4 also need these multiple consecutive faults. Considering that state-of-art secure micro-controllers embed desynchronization countermeasures such as clock jitters and idle cycles, if the target of the perturbation is some shared logic with other treatments (like in the ALU of a CPU), the fault must be accurately space and time controlled, and the effects must be repeatable as well. Identification of the good cycles to inject the perturbation may be a very difficult task, and our attacks seem to be irrelevant. The only exception may be the null fault of §3, if the fault is injected when the q' register is loaded.

Nevertheless, many secure microcontrollers embed a modular arithmetic acceleration coprocessor, which is specifically designed to implement the modular operations. A large proportion of them specifically use the Montgomery multiplication CIOS algorithm (or one of its described variants [24]). Therefore, if the q' or the u_j value is isolated in a specific small size register, a unique long duration perturbation can be sufficient for our attack to succeed. The duration of the perturbation varies of course with the implementation choices and can vary from one cycle to $\log_2 q$, which does not exceed a hundred microseconds on actual chips. To get this kind of effect, laser diodes are the best-suited tool, since the duration of the spot is completely controlled by the attacker.

6.2 Analysis of classical implementations of the Montgomery multiplication

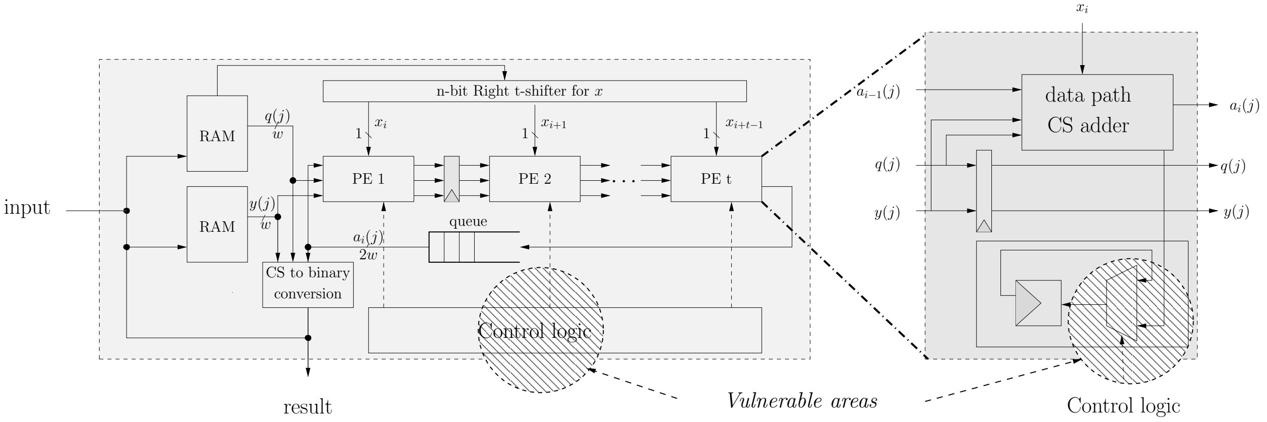
The Montgomery coprocessors proposed in the literature can be divided in 3 different categories :

- the first category [39,25,21] contains variations on the Tenca and Koç *Multiple Word Radix-2 Montgomery Multiplication* algorithm (MWR2MM)[38], which can be seen as a CIOS algorithm with $r = 1$. The characteristic of these implementations is that they use no multiplier architecture, and are therefore really suitable for constrained ASIC implementations.
- the second category [37,26] is an intermediate where r is a classical size for embedded architecture, such as 8,16 or 32 bits, or even 36 bits if we consider FPGA architectures. These designs are more suitable for FPGA targets, since these technologies embed very powerful built-in multipliers blocks. Nevertheless they can also be used in ASIC for intermediate area/latency trade-offs.
- finally, the last category [28,11] propose a version of CIOS/SOS with only one loop, implying that $r \geq \lceil \log_2 q \rceil$. The main difficulty of these implementation techniques is to deal with the very large multiplications they require (one $r \times r$ and two half $r \times r$ multiplications per CIOS). For that purpose they use interpolation techniques, like Karatsuba in [11] or the *Residue Number System* (RNS) in [28]. These implementations are designed to achieve the shortest latency, and are therefore area consuming.

Architectures based on MWR2MM ($r = 1$). In this kind of architecture, it is not feasible to manipulate the value of q', since it is always equal to 1, so no wire or register carries its value. On the other hand, the value of u_j is computed at every loop of the CIOS, and since it is only one bit, a simple shot anywhere on any part of the logic level during the final multiplication $CIOS(A,1)$ is sufficient to get an exploitable result ($u_j = 0$ corresponds to the null fault of §3, and $u_j = 1$ to the constant fault of §4).

The first proposal [39] is a fully systolic [5] array of processing elements (PE) executing consecutively line 6 of the CIOS algorithm in one cycle, and line 7 in k cycles from LSB to MSB. Figure 4 proposes an overview of the architecture. Each PE consists of a w-word carry save adder, able to compute a w word addition and to keep the carry for the next cycle. In the figure, $T(j)$ stands for the j-th least significant w word of T.

Fig. 4. Systolic Montgomery Multiplier of [39] and potential target of the fault



At each clock cycle, the PE presents the computed result $a_i(j)$ to the next one, and the value u_i is kept in the PE for the computation of the next word $a_i(j+1)$. The value of u_i is computed before the word $a_i(0)$ is presented, and then is kept in each PE during the whole computation of a_i in a register. After a complete multiplication, the result a_n is transformed from a carry save representation to binary thanks to the CS to binary converter. This architecture has the great advantage of being

[5] Meaning that all the PEs are the same.

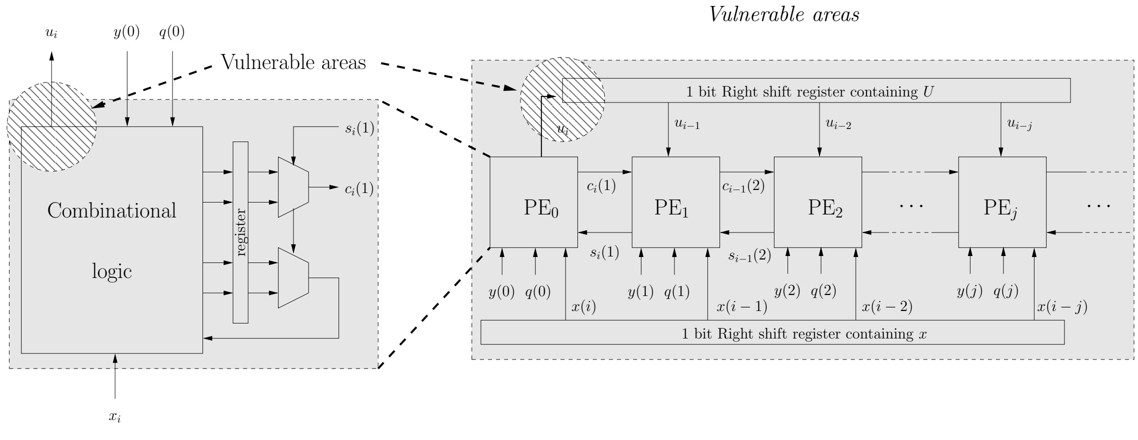
completely scalable (whatever the number of PEs and the size of M, this architecture can compute the expected result as long as the RAM are correctly dimensioned).

To achieve our attack, the register keeping u_i can be the targeted, but every PE must be targeted simultaneously in order to get the correct result. Therefore it is more interesting to target the control logic responsible for the sequencing of the register loading, since all the PEs are connected.

In [25], the authors manage to get rid of the CS to binary converter by redesigning the CS adder of every PE. The vulnerability to our attack is therefore the same, since the redesign does not affect the targeted area.

Huang et al. [21] proposed a new version of the data dependency in the MWR2MM algorithm and rearranged the architecture of [39], in a semi systolic form. Figure 5 gives an overview of the architecture. In this architecture, the intermediate value a_i is manipulated in carry save format ($a_i = c_i + s_i$). A specific PE, PE$_0$ is specialized in generating the u_i values at each cycle, while the j-th PE is in charge of computing the sequence $a_i(j)$. The scalability is lost in exchange for a better time/area trade-off.

Fig. 5. Overview of the [21] architecture and potential target of the fault



This architecture is very vulnerable to our attacks, since a simple n-cycle long shot on the right logic in the PE$_0$ (see Figure 5) is sufficient to get the expected result.

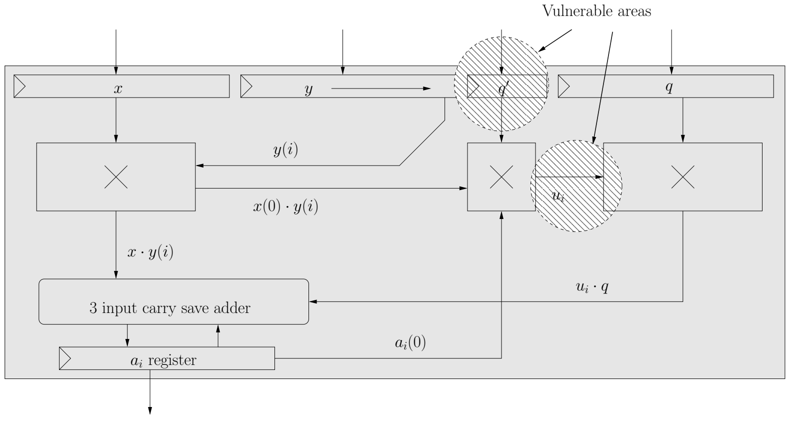
According to the authors, the design works at 100 MHz on their target platform (a Xilinx Virtex II FPGA), therefore the duration of the perturbation is at least 10 μs for a 1024 bits multiplication (2048 bits RSA) if the Garner recombination is used (using the attack from §3.1 or §4.1). If classical CRT reconstruction is used, according to Table 1 in Appendix C, 200 μs will be enough for a null fault.

As a conclusion we can see that this kind of implementation is very vulnerable, since the setup of the attack is quite simple.

High radix architecture ($1 < r < \lceil \log_2 q \rceil$). In this type of implementation choice the value $q' = -p^{-1} \mod 2^r$ is computed in a register, unless the quotient pipelining approach [30] is used. In all the implementations, the value q' is an r-bit register and can be the target of the attack.

For example, the implementation of [26] is described in Figure 6. It relies on the coordinated usage of multiplier blocks of the Xilinx Virtex II together with specifically designed carry save adders. The CIOS algorithm from Figure 2 is completely respected in this implementation. The values u_j can be the target of any fault described in this paper, but it may be easier to put once for all the q' register to 0, with a 100% success rate for the attack if properly carried out. Another implementation is mentioned in [26] with a four-deep pipeline, but it suffers from the same vulnerability.

Fig. 6. Overview of the [26] architecture and potential target of the fault



On the contrary, the attack may be more difficult to achieve on the architecture of [37, Figure 4]. First, it uses quotient determination [30], and therefore does not need to store q' anywhere. Second, the multiplier in charge of computing u_j is shared for all the Montgomery computation. In order to carry out the attack of §4 on this architecture, the attacker has to determine the specific cycles where u_j is computed to generate a perturbation. For that particular design, the attacks seem out of reach.

Full radix architecture ($r \geq \lceil \log_2 q \rceil$). In this kind of implementation, a single round is enough to compute the Montgomery algorithm. This implementation choice reports all the complexity on the design of a $\log_2 q \times \log_2 q$ multiplier, fully used once during the multiplication process and partially twice during the Montgomery reduction. To reduce the full complexity of the big multiplication, interpolation techniques are used. In [11], a classical nested Karatsuba multiplication is used, whereas [28] proposes RNS. Both can be seen as derived from the Lagrange interpolation, with different bases.

In these architectures, a specific laser shot must swap all the u_0 or q' at the same time to produce a null fault. To have a chance, a better solution is to use non invasive attacks (in the sense of [35]), such as power or clock glitches. Indeed u_0 or q' are fully manipulated on the same clock cycle (or in very few), therefore it may be more practical to make the sequencer miss an instruction instead of aiming directly at the registers.

The zero high-order bits fault attack from §5 is more feasible. In the architecture of [11], the most significant bits of u_0 can be set to 0. On the other hand, the architecture of [28] is more immune to this attack, since the RNS representation makes it impractical to modify the significant bits of u_0.

7 Conclusion

In this paper, we have shown that specific realistic faults can defeat unprotected RSA–CRT signatures with any padding scheme, probabilistic or not. While it is not difficult to devise suitable countermeasures (for example, checking that S_q is not too small before outputting a signature is enough to thwart all of our attacks), this underscores the fact that relying on probabilistic signature schemes does not, in itself, protect against faults.

References

1. O. Aciiçmez, W. Schindler, and Ç. K. Koç. Improving Brumley and Boneh timing attack on unprotected SSL implementations. In V. Atluri, C. Meadows, and

A. Juels, editors, *ACM Conference on Computer and Communications Security*, pages 139–146. ACM, 2005.

2. C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In Kaliski et al. [23], pages 260–275.
3. M. Bellare and P. Rogaway. PSS: Provably secure encoding method for digital signatures. Submission to IEEE P1363, 1998.
4. M. Bellare and P. Rogaway. Probabilistic signature scheme. Patent, July 2001. US 6266771.
5. J. Blömer, M. Otto, and J.-P. Seifert. A new CRT-RSA algorithm secure against Bellcore attacks. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 311–320. ACM, 2003.
6. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT*, pages 37–51, 1997.
7. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.
8. E. Brier, D. Naccache, P. Q. Nguyen, and M. Tibouchi. Modulus fault attacks against RSA-CRT signatures. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2011.
9. D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
10. Y. Chen and P. Q. Nguyen. Faster algorithms for approximate common divisors: Breaking fully homomorphic encryption challenges over the integers. In T. Johansson and D. Pointcheval, editors, *EUROCRYPT*, volume 7237, 2012. To appear.
11. G. C. T. Chow, K. Eguro, W. Luk, and P. Leong. A Karatsuba-based Montgomery multiplier. In *FPL'10*, pages 434–437, 2010.
12. M. Ciet and M. Joye. Practical fault countermeasures for Chinese remaindering based cryptosystems. In L. Breveglieri and I. Koren, editors, *FDTC*, pages 124–131, 2005.
13. H. Cohn and N. Heninger. Approximate common divisors via lattices. Cryptology ePrint Archive, Report 2011/437, 2011. `http://eprint.iacr.org/`.
14. J.-S. Coron, C. Giraud, N. Morin, G. Piret, and D. Vigilant. Fault attacks and countermeasures on Vigilant's RSA-CRT algorithm. In L. Breveglieri, M. Joye, I. Koren, D. Naccache, and I. Verbauwhede, editors, *FDTC*, pages 89–96. IEEE Computer Society, 2010.
15. J.-S. Coron, A. Joux, I. Kizhvatov, D. Naccache, and P. Paillier. Fault attacks on RSA signatures with partially unknown messages. In C. Clavier and K. Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 444–456. Springer, 2009.
16. J.-S. Coron and A. Mandal. PSS is secure against random fault attacks. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 653–666. Springer, 2009.
17. J.-S. Coron, D. Naccache, and M. Tibouchi. Fault attacks against EMV signatures. In J. Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 2010.
18. H. L. Garner. The residue number system. In *IRE-AIEE-ACM '59 (Western)*, pages 146–153. ACM, 1959.
19. C. Giraud. An RSA implementation resistant to fault attacks and to simple power analysis. *IEEE Trans. Computers*, 55(9):1116–1120, 2006.

20. N. Howgrave-Graham. Approximate integer common divisors. In J. H. Silverman, editor, *CaLC*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001.
21. M. Huang, K. Gaj, S. Kwon, and T. A. El-Ghazawi. An optimized hardware architecture for the Montgomery multiplication algorithm. In R. Cramer, editor, *Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2008.
22. B. S. Kaliski. Raising the standard for RSA signatures: RSA-PSS. CryptoBytes Technical Newsletter, February 2003. `http://www.rsa.com/rsalabs/node.asp?id=2005`.
23. B. S. Kaliski, Ç. K. Koç, and C. Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*. Springer, 2003.
24. Ç. K. Koç and T. Acar. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.
25. C. McIvor, M. McLoone, and J. McCanny. Modified Montgomery modular multiplication and RSA exponentiation techniques. *IEE Proceedings - Computers and Digital Techniques*, 151(6):402–408, 2004.
26. N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede. Efficient pipelining for modular multiplication architectures in prime fields. In *Proceedings of the 17th ACM Great Lakes symposium on VLSI*, GLSVLSI '07, pages 534–539, New York, NY, USA, 2007. ACM.
27. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.
28. H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura. Implementation of RSA algorithm based on RNS Montgomery multiplication. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 364–376. Springer, 2001.
29. Oracle. JavaCard 3.0.1 Platform Specification. `http://www.oracle.com/technetwork/java/javacard/overview/`.
30. H. Orup. Simplifying quotient determination in high-radix modular multiplication. In *IEEE Symposium on Computer Arithmetic'95*, pages 193–193, 1995.
31. E. Oswald and P. Rohatgi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*. Springer, 2008.
32. M. Rivain. Securing RSA against fault analysis by double addition chain exponentiation. In M. Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 459–480. Springer, 2009.
33. W. Schindler. A timing attack against RSA with the Chinese remainder theorem. In Ç. K. Koç and C. Paar, editors, *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 2000.
34. A. Shamir. Improved method and apparatus for protecting public key schemes from timing and fault attacks. Patent Application, November 1998. WO 1998/052319 A1.
35. S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In Kaliski et al. [23], pages 2–12.
36. W. Stein et al. *Sage Mathematics Software (Version 4.8)*. The Sage Development Team, 2012. `http://www.sagemath.org`.

37. D. Suzuki. How to maximize the potential of FPGA resources for modular exponentiation. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2007.
38. A. F. Tenca and Ç. K. Koç. A scalable architecture for Montgomery multiplication. In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '99, pages 94–108, London, UK, UK, 1999. Springer-Verlag.
39. A. F. Tenca and Ç. K. Koç. A scalable architecture for modular multiplication based on Montgomery's algorithm. *IEEE Trans. Comput.*, 52:1215–1221, September 2003.
40. The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. `http://www.openssl.org/`.
41. D. Vigilant. RSA with CRT: A new cost-effective solution to thwart fault attacks. In Oswald and Rohatgi [31], pages 130–145.
42. C. D. Walter. Montgomery's multiplication technique: How to make it smaller and faster. In Ç. K. Koç and C. Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 80–93. Springer, 1999.
43. S.-M. Yen, S.-J. Moon, and J. Ha. Hardware fault attack on RSA with CRT revisited. In P. J. Lee and C. H. Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2002.

A Null Faults in Successive CIOS

We consider here the fault model where we force q' to zero on consecutive CIOS steps. We will examine how this plays out in each on the three exponentiation algorithms in turn. Throughout this appendix, we let $\ell = \lceil \log_2 \lceil \log_2 q \rceil \rceil$.

In all cases, we also assume, heuristically, that the values $\bar{x}, A$ in Montgomery representation involved in our computation are uniformly distributed modulo q before the first fault is injected; this means, in particular, that they are smaller than $2^{\lceil \log_2 q \rceil} - 1$ with probability at least $1/2$.

Square-and-Multiply LSB. We first consider a fault model in which the attacker can force the precomputed value q' to zero during ℓ calls of $CIOS(\bar{x}, \bar{x})$ in the Square-and-Multiply LSB algorithm (line 7), during the computation of S_q.

Theorem 4. *With probability at least* $1/2$*, a faulty signature* $\widetilde{S}$ *generated in this fault model, using Square-and-Multiply LSB, is a multiple of* q *(regardless of the encoding function involved, probabilistic or not).*

Proof. Suppose faults are injected starting from iteration $i = \alpha$ in the loop of the exponentiation algorithm, and that before then, $|\bar{x}| \leq \lceil \log_2 q \rceil - 1$:

this happens with probability at least $1/2$. The fault $q' = 0$ has to occur in $\mathrm{CIOS}(\bar{x}, \bar{x})$ (the CIOS at line 6 does not modify $\bar{x}$ and then is ignored in this case). Then, the output $\bar{x}$ of this faulty CIOS is, up to rounding errors:

$$\widetilde{x} = \left\lfloor \frac{\bar{x}_0 \bar{x}}{2^{rk}} \right\rfloor + \cdots + \left\lfloor \frac{\bar{x}_{k-1} \bar{x}}{2^r} \right\rfloor = \left\lfloor \frac{\bar{x}_{k-1} \bar{x}}{2^r} \right\rfloor + o(2^{r(k-1)})$$

With our assumption on the size of $\bar{x}$, we obtain $|\widetilde{x}| \leq \lceil \log_2 q \rceil - 2$. Therefore, for $i = \alpha + 1$ and with $q' = 0$, the size of the output of $\mathrm{CIOS}(\widetilde{x}, \widetilde{x})$ will be reduced to at most $\lceil \log_2 q \rceil - 4$, and so on. By induction, keeping the fault $q' = 0$ up to iteration $i = \alpha + \ell - 1$, i.e. through ℓ executions of CIOS, brings the value $\bar{x}$ down to 0. Clearly, the faulty half-exponentiation thus outputs $\widetilde{S}_q = 0$, hence the stated result. □

Square-and-Multiply MSB. For now, we force q' to zero during ℓ consecutive steps of the loop of the Square-and-Multiply MSB. That represents at worst 2ℓ faulty calls to CIOS.

Theorem 5. *With probability at least* $1/2$*, a faulty signature* $\widetilde{S}$ *generated in this fault model, using Square-and-Multiply MSB, is a multiple of* q *(regardless of the encoding function involved, probabilistic or not).*

Proof. The main difference in the case of the Square-and-Multiply MSB is that the CIOS at line 7 affects the same value A as $\mathrm{CIOS}(A, A)$. Consequently, the fault has to be injected in execution of $\mathrm{CIOS}(A, A)$ as well, and, when it occurs, $\mathrm{CIOS}(A, \bar{x})$ to reduce the size of A. The details are the same as for the Square-and-Multiply LSB algorithm and the required number of consecutive faulty CIOS is 2ℓ in the worst case, still with probability $\geq 1/2$. □

Remark 1. While the occurrence of $\mathrm{CIOS}(A, \bar{x})$ demands one fault, if the size of $\bar{x}$ is less than $\lceil \log_2 q \rceil$, it induces a reduction of the size of A too, decreasing the required number of faults to have $A = 0$. Moreover, the probability $1/2$ can be removed if the faults are initiated at the begin of the Square-and-Multiply MSB algorithm. Indeed, the initial value of A is equal to $R \bmod q$, and so $A < 2^{\lceil \log_2 q \rceil}$.

Montgomery Ladder. Finally, we consider the case when q' can be forced to zero in $2\ell - 1$ suitable consecutive CIOS steps of the Montgomery Ladder.

Theorem 6. *With probability at least $1/2$, a faulty signature $\widetilde{S}$ generated in this fault model, using Montgomery Ladder, is a multiple of q (regardless of the encoding function involved, probabilistic or not).*

Proof. The principle is to cancel A or $\bar{x}$, depending on the values of e_i during the attack. For instance, we cause a fault in the CIOS that affect A (line 7 and 10). With probability at least $1/2$, $A < 2^{\lceil \log_2 q \rceil}$ and we assume that ℓ $CIOS(A, A)$ are computed before ℓ $CIOS(\bar{x}, \bar{x})$ (otherwise, the choice $\bar{x}$ is more efficient). Hence, by faulting the CIOS at line 7 when $e_i = 0$ and the CIOS at line 10 else, the worst case requires at worst $2\ell - 1$ consecutive faults to bring A to zero. □

Remark 2. In practice, it may not be possible to decide which CIOS should be attacked since that depends on the secret exponent bits. So one can instead cause faults throughout $2\ell - 1$ iterations of the exponentiation process, amounting to $4\ell - 2$ consecutive faulty CIOS, to ensure that $S_q = 0$ (always with the probability $\geq 1/2$). Note that this worst case is rarely reached since the size of the non chosen value ($\bar{x}$ in our example) has an active role in the reduction of the size of the chosen value (here A) during the computation of CIOS$(A, \bar{x})$. Moreover, the probability $1/2$ can be lifted if the faults are injected from the start of the Montgomery Ladder algorithm, in view of the initial value of A.

B Constant Faults in Successive CIOS

We focus on the Square-and-Multiply LSB algorithm and assume that constant faults are injected in the evaluations of CIOS$(\bar{x}, \bar{x})$ and CIOS$(A, \bar{x})$ during the exponentiation process computing S_q. More precisely, suppose that $u_i = \widetilde{u}$ $(i = 0, \ldots, k-1)$ for these particular CIOS, and write:

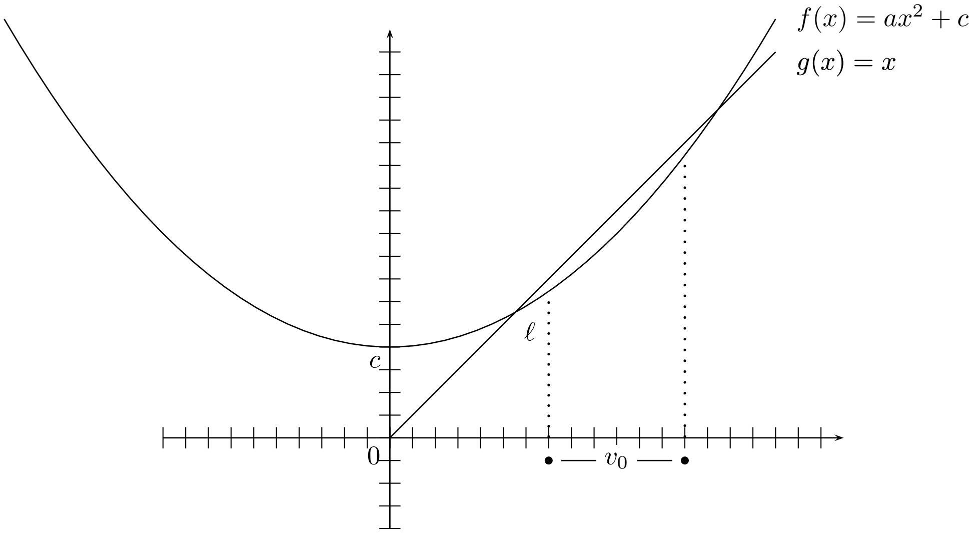
$$\ell = 2^{\lceil \log_2 q \rceil - 1} \sqrt{1 - \frac{\widetilde{u}}{(2^r - 1) 2^{\lceil \log_2 q \rceil - 2}}}.$$

We claim that if the initial value of $\bar{x}$ is such that $\ell < \bar{x} < 2^{\lceil \log_2 q \rceil - 1}$, then the computed value A of the Square-and-Multiply LSB approaches ℓ.

Indeed, one can see that the output $\widetilde{x}$ of each faulty CIOS$(\bar{x}, \bar{x})$ is roughly:

$$\widetilde{x} \approx \frac{\bar{x}^2}{2^{\lceil \log_2 q \rceil}} + \frac{\widetilde{u} q}{2^r - 1} - \varepsilon \cdot q \quad (\varepsilon \in \{0, 1\})$$

Looking at the sequence $v_{n+1} = f(v_n) = av_n^2 + c$ with $a = \frac{1}{2^{\lceil \log_2 q \rceil}}$, $c = \frac{\widetilde{u}q}{2^r-1}$ and v_0 represents the message in Montgomery representation. This sequence will tend to a limit ℓ if $\mathit{\Delta} = 1 - 4ac > 0$, i.e. $\frac{\widetilde{u}}{2^r-1} < \frac{2^{\lceil \log_2 q \rceil - 2}}{q}$. Our assumption on the value of v_0 implies that $f(I) \in I$. Referring to the graph below, it appears then that the sequence will tend to $\ell = \min(\ell_1, \ell_2)$ where ℓ_1 and ℓ_2 denote the two roots of $f(\ell) = \ell$.



However, we want that this limit be reached before the end of the exponentiation process. Let us determine the convergence speed of this sequence:

$$|v_{n+1} - \ell| = |f(v_n) - f(\ell)| \leq |f'(\ell)| \cdot |v_n - l| \leq |f'(\ell)|^{n+1} \cdot |v_0 - \ell|$$

Since ℓ and v_0 are integer values, we look for the condition $|f'(\ell)|^{n+1} \cdot |v_0 - \ell| \leq 1$. Hence, the limit is reached for n such as:

$$n + 1 \geq -\frac{\log_2(|v_0 - \ell|)}{\log_2(|f'(\ell)|)}$$

For example, we search a condition in order to have $\mathrm{CIOS}(\bar{x}, \bar{x}) = \ell$ before the half ($|q|/2$) of the exponentiation process. Since $\log_2(|v_0 - \ell|) \approx |q|$, this condition is $\frac{-1}{\log_2(|f'(\ell)|)} < 1/2$, i.e. $|f'(\ell)| < 1/4$. Moreover, $f'(\ell) = 2a\ell = 1 - \sqrt{\mathit{\Delta}} > 0$ leads to $9/16 < \mathit{\Delta}$ and then to $\frac{\widetilde{u}}{2^r-1} < \frac{9}{16}\frac{2^{\lceil \log_2 q \rceil - 2}}{q}$. We see on this example that the success of the attack will depend on the ratio $q/2^{\lceil \log_2 q \rceil}$ and on the ratio $\widetilde{u}/(2^r - 1)$.

Looking at CIOS$(A, \bar{x})$, the output $\widetilde{A}$ is roughly:

$$\widetilde{A} \approx \frac{A\bar{x}}{2^{\lceil \log_2 q \rceil}} + \frac{\widetilde{u}}{2^r - 1} - \varepsilon \cdot q \quad (\varepsilon \in \{0, 1\})$$

and the associated sequence is a little more complicated:

$$g(w_n) = w_{n+1} = \begin{cases} w_n & \text{if } e_{n+1} = 0 \\ a w_n v_n + c & \text{else} \end{cases}$$

It is clear that if $v_n = \ell$, w_n will tend to this limit too. We just have to verify that this sequence reaches ℓ before the end of the exponentiation process. In fact, both sequences are linked by the following relation:

$$\frac{w_{n+1} - v_{n+1}}{w_n - v_n} = \frac{v_n}{2^{\lceil \log_2 q \rceil}}$$

With our assumption, the sequence (v_n) is decreasing, and we have:

$$\frac{w_{\lceil \log_2 q \rceil} - v_{\lceil \log_2 q \rceil}}{w_0 - v_0} < \frac{1}{2^{\lceil \log_2 q \rceil}}$$

Hence the range between the two sequences constantly decreases during the exponentiation process and if (v_n) tends to ℓ before the end of the exponentiation process, then (w_n) will reach this value too.

The attack consists on computing two signatures $\tilde{S}, \tilde{S}'$ by faulting them with the same fault $\widetilde{u}$. In consequence, with a certain probability depending on the ratios $q/2^{\lceil \log_2 q \rceil}$ and $\widetilde{u}/(2^r - 1)$, these two signatures will be equal modulo q. Thus, we recover q as $\gcd(N, \tilde{S} - \tilde{S}')$.

Remark 3. In Table 2 below, the success rates are even better than expected. In fact, if $\Delta < 0$, the value $\bar{x}$ can enter in a cycle of a few different values. As a consequence, with some probability, two messages can have the same value S_q. Geometrically, that can be explained by the representation of the function $f \circ \cdots \circ f$ which is flatter and can intersect the line representing $g(x) = x$.

C Simulation Results

	S&M LSB	S&M MSB		Montgomery Ladder	
Faulty iterations	(%)	Start (%)	Anywhere (%)	Start (%)	Anywhere (%)
8	31	93	62	45	30
9	65	100	93	87	76
10	89	100	100	99	93

Table 1. Success rate of the null fault attack on consecutive CIOS steps, for a 512-bit prime q and $r = 16$. 100 faulty signatures were computed for each parameter set. For the Square-and-Multiply MSB and Montgomery Ladder algorithms, we compare success rates when faults start at the beginning of the loop vs. at a random iteration.

$q/2^{\lceil \log_2 q \rceil}$	0.666	0.696	0.846	0.957
Success rate (%)	36	34.4	26.7	20.4

Table 2. Success rate of the constant fault attack on successive CIOS steps, when using Square-and-Multiply LSB exponentiation with random 512-bit primes q and $r = 16$.

Number h of zero top bits	48	40	32	24	16
Minimum ℓ with the general attack	8	9	11	15	22
Minimum ℓ with the linear attack	11	13	16	22	32

Table 3. Theoretical minimum number ℓ of zero higher-order h-bit faulty signatures required to factor a balanced 1024-bit RSA modulus N using the general Cohn-Heninger attack or the simplified linear one.

Number ℓ of faulty signatures	11	12	13	14	15	16	17	18
Success rate with $h = 48$ (%)	23	100	100	100	100	100	100	100
Success rate with $h = 40$ (%)	0	0	2	100	100	100	100	100
Success rate with $h = 32$ (%)	0	0	0	0	0	0	99	100
Average CPU time (ms)	33	35	38	41	45	49	54	59

Table 4. Experimental success rate of the simplified (linear) Cohn-Heninger attack with ℓ faulty signatures when N is a balanced 1024-bit RSA modulus. Timings are given for our Sage implementation on a single core of a Core 2 CPU at 3 GHz.