Attacking RSA–CRT Signatures with Faults on Montgomery Multiplication

Pierre-Alain Fouque[1,2], Nicolas Guillermin[3], Delphine Leresteux[3], Mehdi Tibouchi[4], and 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, 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 et al. in [3], and later refined in multiple publications such as [31]: 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

E. Prouff and P. Schaumont (Eds.): CHES 2012, LNCS 7428, pp. 447–462, 2012.

taxonomy of these countermeasures is given in [24]. 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 [22]. Another approach is to use an extended modulus. Shamir's trick [25] 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 [2, 7, 30, 9]. Finally, yet another way to protect RSA–CRT signatures against faults is to use redundant exponentiation algorithms, such as the Montgomery Ladder. Papers including [14, 24] propose such countermeasures. Regardless of the approach, RSA–CRT fault countermeasures tend to be rather costly: for example, Rivain's countermeasure [24] has a stated overhead of 10% compared to an unprotected implementation, and is purportedly more efficient than previous works including [14, 30].

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 [10, 12]. However, Coron and Mandal [11] were able to prove that Bellare and Rogaway's padding scheme RSA–PSS [1] 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 [20], 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 [15, 5, 8].

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 [11], 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

Proposed by Montgomery in [20], 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 $\mathrm{CIOS}(x, y) < q$.

1: **function** SIGN$_{\text{RSA–CRT}}(m)$
2: $M \leftarrow \mu(m) \in \mathbb{Z}_N$ ▷ message encoding
3: $M_p \leftarrow M \bmod p$
4: $M_q \leftarrow M \bmod q$
5: $S_p \leftarrow M_p^{d_p} \bmod p$
6: $S_q \leftarrow M_q^{d_q} \bmod q$
7: $t \leftarrow S_p - S_q$
8: **if** $t < 0$ **then** $t \leftarrow t + p$
9: $S \leftarrow S_q + \big((t \cdot \pi) \bmod p\big) \cdot q$
10: **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 $\pi = q^{-1} \bmod p$.

1: **function** CIOS(x, y)
2: $a \leftarrow 0$
3: $y_0 \leftarrow y \bmod b$
4: **for** $j = 0$ to $k-1$ **do**
5: $a_0 \leftarrow a \bmod b$
6: $u_j \leftarrow (a_0 + x_j \cdot y_0) \cdot q' \bmod b$
7: $a \leftarrow \left\lfloor \frac{a + x_j \cdot y + u_j \cdot q}{b} \right\rfloor$
8: **if** $a \geq q$ **then** $a \leftarrow a - q$
9: **return** a

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

Figure 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 [17] does not protect against any of our attacks.

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. The first three exponentiation algorithms will be considered in this paper, and two of those are detailed 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: $CIOS(\bar{A}, \bar{B}) = (AR)(BR)R^{-1} \bmod q = \overline{AB}$.

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

Square-and-Multiply LSB

```
1: function ExpLSB(x, e, q)
2:     x̄ ← CIOS(x, R^2 mod q)
3:     A ← R mod q
4:     for i = 0 to t do
5:         if e_i = 1 then
6:             A ← CIOS(A, x̄)
7:         x̄ ← CIOS(x̄, x̄)
8:     A ← CIOS(A, 1)
9:     return A
```

Montgomery Ladder

```
1: function ExpLadder(x, e, q)
2:     x̄ ← CIOS(x, R^2 mod q)
3:     A ← R mod q
4:     for i = t down to 0 do
5:         if e_i = 0 then
6:             x̄ ← CIOS(A, x̄)
7:             A ← CIOS(A, A)
8:         else if e_i = 1 then
9:             A ← CIOS(A, x̄)
10:            x̄ ← CIOS(x̄, x̄)
11:    A ← CIOS(A, 1)
12:    return A
```

Fig. 3. Two of the 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 $R = b^k$.

$S = m^d \bmod N$ 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 as well as the JavaCard API [22] 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:

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

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

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 [4]. 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 $q^{-1} \bmod p$. 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 register containing the precomputed value $q' = (-q \bmod b)$ 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 $j = 0, \ldots, k-1$, 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 CIOS$(1, A)$ instead of 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 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 the full version of this paper [13]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.

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.

	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

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.

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, $j = 0, \ldots, k-1$, 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 (the proof can be found in the full version [13]).

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:*

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

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 [5], 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 [15] based on Coppersmith techniques. It is the fastest option unless r is very small (a simple implementation in Sageruns 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 $j = j_0, j_0 + 1, \ldots, k - 1$ for some j_0.

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

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

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 [15] 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.

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 the full version [13].

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. 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 $\gcd(N, \widetilde{S} - \widetilde{S}') = q$.

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 $r = 16$

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

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. More precisely, we prove the following theorem in the full version of this paper [13]:

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 $|\widetilde{S} - qT| \leq 2^{-h} \cdot q + 1$.*

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 [8] 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 $N^{(1/2)^{1+1/\ell}}$. 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 [8]), 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. We examine this method in more details in the full version of this paper [13], and find that it makes it possible to factor N provided that:

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

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 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 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.

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

which is always a worse bound than (3) but usually not by a very large margin. Table 3 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 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 tools needed to get the desired effects, we focus on several implementation proposals [29, 18, 16, 21, 28, 19, 6], chosen for their relevance.

6.1 Characteristics of the Perturbation Tool

First all the perturbations needed to carry out our attacks need to be controlled and local to some gates of the chip. Therefore, the attacker needs 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 an isolated modular arithmetic acceleration coprocessor. A large proportion of them specifically use the Montgomery multiplication CIOS algorithm (or one of its described variants [17]). 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 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 [26].

6.2 Analysis of Classical Implementations of the Montgomery Multiplication

The public Montgomery architectures can be divided in 3 different categories :

- the first one [29, 18, 16] contains variations on the Tenca and Koç *Multiple Word Radix-2 Montgomery Multiplication* algorithm (MWR2MM) [29], which can be seen as a CIOS algorithm with $r = 1$. The characteristic of these implementations is that they use no multiplier. They are then suited for constrained area.
- the second category [28, 19] is an intermediate where r is a classical size for embedded architecture, such as 8,16 or 32 bits. They can be used for intermediate area/latency trade-offs.
- the last category [21, 6] 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 . For that purpose they use interpolation techniques, like Karatsuba in [6] or RNS in [21]. These implementations are designed to achieve the shortest latency.

Architectures Based on MWR2MM ($r = 1$). In this kind of architecture, q' cannot be manipulated, 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 on the logic driving the register 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).

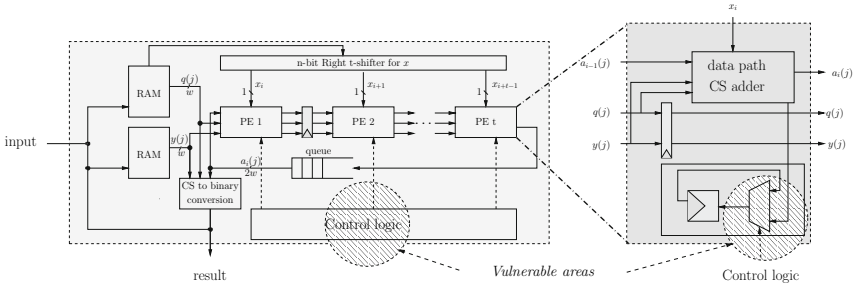

Fig. 4. Systolic Montgomery Multiplier of [29] and potential target of the fault

The first proposal [29] is a fully systolic [1] 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.

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. This architecture has the great advantage of being 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 [18], 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. [16] proposed a new version of the data dependency in the MWR2MM algorithm and rearranged the architecture of [29], 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 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 $a_i(j)$.

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

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

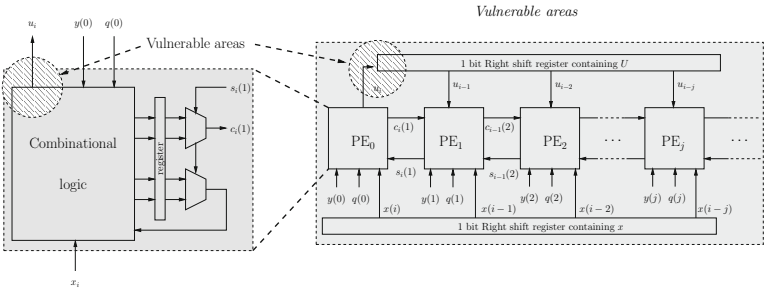

Fig. 5. Overview of the [16] architecture and potential target of the fault

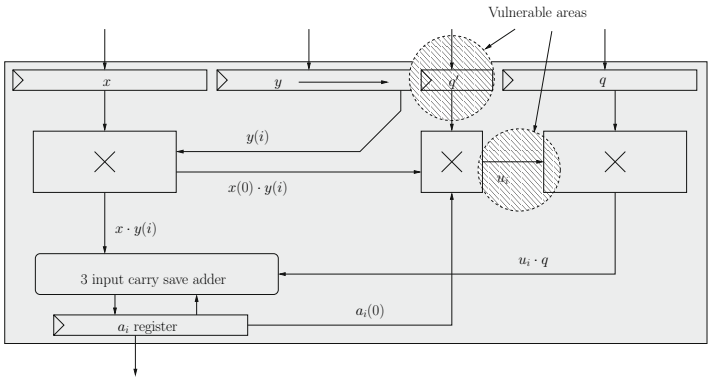

Fig. 6. Overview of the [19] architecture and potential target of the fault

used (using the attack from §3.1 or §4.1). If classical CRT reconstruction is used, according to Table 1, 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 the value $q' = -q^{-1} \bmod 2^r$ is computed in a r-bit register, unless the quotient pipelining approach [23] is used.

For example, the implementation of [19] 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 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 [19] with a four-deep pipeline, but it suffers from the same vulnerability.

The attack may be more difficult to achieve on the architecture of [28, Figure 4]. First, it uses quotient determination [23], 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. To reduce the full complexity of the big multiplication, interpolation techniques are used. In [6], a classical nested Karatsuba multiplication is used, whereas [21] proposes RNS.

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 [27]), 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 also an option. In the architecture of [6], the most significant bits of u_0 can be set to 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. Bellare, M., Rogaway, P.: Probabilistic signature scheme. US Patent 6266771 (2001)
2. Blömer, J., Otto, M., Seifert, J.-P.: A new CRT-RSA algorithm secure against Bellcore attacks. In: Jajodia, S., Atluri, V., Jaeger, T. (eds.) ACM Conference on Computer and Communications Security, pp. 311–320. ACM (2003)
3. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
4. Brier, É., Naccache, D., Nguyen, P.Q., Tibouchi, M.: Modulus Fault Attacks against RSA-CRT Signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 192–206. Springer, Heidelberg (2011)
5. Chen, Y., Nguyen, P.Q.: Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 502–519. Springer, Heidelberg (2012)

6. Chow, G.C.T., Eguro, K., Luk, W., Leong, P.: A Karatsuba-based Montgomery multiplier. In: FPL 2010, pp. 434–437 (2010)
7. Ciet, M., Joye, M.: Practical fault countermeasures for Chinese remaindering based cryptosystems. In: Breveglieri, L., Koren, I. (eds.) FDTC, pp. 124–131 (2005)
8. Cohn, H., Heninger, N.: Approximate common divisors via lattices. Cryptology ePrint Archive, Report 2011/437, (2011), http://eprint.iacr.org/ (to appear at ANTS-X)
9. Coron, J.-S., Giraud, C., Morin, N., Piret, G., Vigilant, D.: Fault attacks and countermeasures on Vigilant's RSA-CRT algorithm. In: Breveglieri et al. [4], pp. 89–96
10. Coron, J.-S., Joux, A., Kizhvatov, I., Naccache, D., Paillier, P.: Fault Attacks on RSA Signatures with Partially Unknown Messages. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 444–456. Springer, Heidelberg (2009)
11. Coron, J.-S., Mandal, A.: PSS Is Secure against Random Fault Attacks. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 653–666. Springer, Heidelberg (2009)
12. Coron, J.-S., Naccache, D., Tibouchi, M.: Fault Attacks Against EMV Signatures. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 208–220. Springer, Heidelberg (2010)
13. Fouque, P.-A., Guillermin, N., Leresteux, D., Tibouchi, M., Zapalowicz, J.-C.: Attacking RSA–CRT signatures with faults on Montgomery multiplication. Cryptology ePrint Archive, Report 2012/172 (2012), http://eprint.iacr.org/ (Full version of this paper)
14. Giraud, C.: An RSA implementation resistant to fault attacks and to simple power analysis. IEEE Trans. Computers 55(9), 1116–1120 (2006)
15. Howgrave-Graham, N.: Approximate Integer Common Divisors. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 51–66. Springer, Heidelberg (2001)
16. Huang, M., Gaj, K., Kwon, S., El-Ghazawi, T.: An Optimized Hardware Architecture for the Montgomery Multiplication Algorithm. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 214–228. Springer, Heidelberg (2008)
17. Koç, Ç.K., Acar, T.: Analyzing and comparing Montgomery multiplication algorithms. IEEE Micro 16(3), 26–33 (1996)
18. McIvor, C., McLoone, M., McCanny, J.: Modified Montgomery modular multiplication and RSA exponentiation techniques. IEE Proceedings - Computers and Digital Techniques 151(6), 402–408 (2004)
19. Mentens, N., Sakiyama, K., Preneel, B., Verbauwhede, I.: Efficient pipelining for modular multiplication architectures in prime fields. In: Proceedings of the 17th ACM Great Lakes Symposium on VLSI, GLSVLSI 2007, pp. 534–539. ACM, New York (2007)
20. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44, 519–521 (1985)
21. Nozaki, H., Motoyama, M., Shimbo, A., Kawamura, S.-I.: Implementation of RSA Algorithm Based on RNS Montgomery Multiplication. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 364–376. Springer, Heidelberg (2001)
22. Oracle. JavaCard 3.0.1 Platform Specification, http://www.oracle.com/technetwork/java/javacard/overview/
23. Orup, H.: Simplifying quotient determination in high-radix modular multiplication. In: IEEE Symposium on Computer Arithmetic 1995, pp. 193–193 (1995)

24. Rivain, M.: Securing RSA against Fault Analysis by Double Addition Chain Exponentiation. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 459–480. Springer, Heidelberg (2009)
25. Shamir, A.: Improved method and apparatus for protecting public key schemes from timing and fault attacks. Patent Application, WO 1998/052319 A1 (1998)
26. Skorobogatov, S.: Optical fault masking attacks. In: Breveglieri et al. [4], pp. 23–29
27. Skorobogatov, S.P., Anderson, R.J.: Optical Fault Induction Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003)
28. Suzuki, D.: How to Maximize the Potential of FPGA Resources for Modular Exponentiation. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 272–288. Springer, Heidelberg (2007)
29. Tenca, A.F., Koç, Ç.K.: A Scalable Architecture for Montgomery Multiplication. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 94–108. Springer, Heidelberg (1999)
30. Vigilant, D.: RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 130–145. Springer, Heidelberg (2008)
31. Yen, S.-M., Moon, S.-J., Ha, J.C.: Hardware Fault Attackon RSA with CRT Revisited. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 374–388. Springer, Heidelberg (2003)