Countermeasures Against High-Order Fault-Injection Attacks on CRT-RSA

Pablo Rauzy
rauzy@enst.fr
pablo.rauzy.name

Sylvain Guilley
guilley@enst.fr
perso.enst.fr/∼guilley

Telecom ParisTech
CNRS LTCI / COMELEC / SEN

FDTC 2014
Eleventh Workshop on Fault Diagnosis and Tolerance in Cryptography
September 23, 2014 @ Busan, Korea

IACR ePrint 2014/559

RSA (*Rivest, Shamir, Adleman*) **Definition**

RSA [RSA78] is an algorithm for public key cryptography. It can be used as both an encryption and a signature algorithm.

- Let M be the message, (N, e) the public key, and (N, d) the private key such that $d \cdot e \equiv 1 \mod \varphi(N)$.
- The signature S is computed by $S \equiv M^d \mod N$.
- The signature can be verified by checking that $M \equiv S^e \mod N$.

CRT (*Chinese Remainder Theorem*) — Definition

CRT-RSA [Koç94] is an optimization of the RSA computation which allows a fourfold speedup.

- Let p and q be the primes from the key generation ($N = p \cdot q$).
- These values are pre-computed (considered part of the private key):
  - $d_p \doteq d \mod (p-1)$
  - $d_q \doteq d \mod (q-1)$
  - $i_q \doteq q^{-1} \mod p$
- S is then computed as follows:
  - $S_p = M^{d_p} \mod p$
  - $S_q = M^{d_q} \mod q$
  - $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ (recombination method of [Gar65]).

BellCoRe (*Bell Communications Research*) **Definition**

The BellCoRe attack [BDL97] consists in revealing the secret primes p and q by faulting the computation. It is very powerful as it works even with very random faulting.

- The intermediate variable S_p (resp. S_q) is faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$).
- The attacker thus gets an erroneous signature $\widehat{S}$.
- The attacker can recover p (resp. q) as $\gcd(N, S - \widehat{S})$.

Why does it Work?

- For all integer x, $\gcd(N, x)$ can only take 4 values:
  - 1, if N and x are co-prime,
  - p, if x is a multiple of p,
  - q, if x is a multiple of q,
  - N, if x is a multiple of both p and q, *i.e.*, of N.

Why does it Work?

- If S_p is faulted (*i.e.*, replaced by $\widehat{S_p} \neq S_p$):
  - $S - \widehat{S} = q \cdot \Big((i_q \cdot (S_p - S_q) \mod p) - (i_q \cdot (\widehat{S_p} - S_q) \mod p)\Big)$

  $\Rightarrow \gcd(N, S - \widehat{S}) = q$

Why does it Work?

- If S_q is faulted (*i.e.*, replaced by $\widehat{S_q} \neq S_q$):
  - $S - \widehat{S} \equiv (S_q - \widehat{S_q}) - (q \mod p) \cdot i_q \cdot (S_q - \widehat{S_q}) \equiv 0 \mod p$ (because $(q \mod p) \cdot i_q \equiv 1 \mod p$)

$\Rightarrow$ $\gcd(N, S - \widehat{S}) = p$

Fault injection
Definition

During the execution of an algorithm, the attacker can:

- modify any intermediate value by setting it to either a random value (*randomizing fault*) or zero (*zeroing fault*), such a fault can be either *permanent* or *transient*;
- skip any number of consecutive instructions (*skipping fault*).

At the end of the computation the attacker can read the result returned by the algorithm.

Attack order
Definition

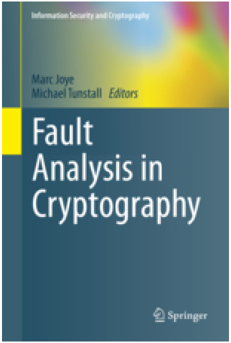
We call *order* of the attack the number of fault injections in the computation.
An attack is said to be *high-order* if its order is strictly more than 1.

Equivalence between faults on the code and on the data — **Lemma**

The effect of a skipping fault (*i.e.*, fault on the code) can be captured by considering only randomizing and zeroing faults (*i.e.*, fault on the data).

- If the skipped instructions are part of an arithmetic operation:
  - either the computation has not been done at all: its results becomes zero (if initialized) or random (if not),
  - or the computation has partly been done: its result is thus considered random at our modeling level.
- If the skipped instruction is a branching instruction, it is equivalent to fault the result of the branching condition:
  - at zero (*i.e.*, `false`), to avoid branching,
  - at random (*i.e.*, `true`), to force branching. □

- High-order attacks?
- High-order countermeasures?
- Proved high-order countermeasures?



FDTC Fault Diagnosis and Tolerance in Cryptography

- High-order attacks have been studied and shown practical:
  - *Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures* [KQ07],
    by C. H. Kim and J.-J. Quisquater at WISTP'07.
  - *Multi Fault Laser Attacks on Protected CRT-RSA* [TK10],
    by E. Trichina and R. Korkikyan at FDTC'10.

Existing High-Order Countermeasures?

- A few countermeasures claim to be second-order:
  - *Practical fault countermeasures for chinese remaindering based RSA* [CJ05],
    by M. Ciet and M. Joye at FDTC'05.
  - *On Second-Order Fault Analysis Resistance for CRT-RSA Implementations* [DGRS09],
    by E. Dottax, C. Giraud, M. Rivain, and Y. Sierra at WISTP'09.

  But they do not work in our more general fault model as our tool finja shows: `crt-rsa_ciet-joye.fia.zzt.html`, `crt-rsa_dottax-etal.fia.rzt.html`.

- We found no countermeasure claiming to resist > 2 faults.

- If we want a high-order countermeasure, we have to create it.
- What is a countermeasure?
- What makes a countermeasure work? What makes it fail?
- How do the existing first-order countermeasures work?

- The goal of a countermeasure against fault-injection attacks is to avoid returning a compromised value to the attacker.
- This is done by *verifying the integrity of the computation* before returning its result, and returning a random number or an error constant rather than the actual result if appropriate.

Computation Integrity Verification

- Obvious idea: repeat the computation and compare the results.
- But of course that costs too much.
- Existing countermeasures optimize this idea in many different ways.

- What are the different methods used by the existing countermeasures to verify the computation integrity faster than $(M^d)^e \stackrel{?}{\equiv} M \mod N$?
- We used 4 main parameters to classify countermeasures.

1. Shamir's or Giraud's Family of Countermeasures

- Two main families of countermeasures:
  - descendants of Giraud's countermeasure [Gir06],
  - descendants of Shamir's countermeasure [Sha99].

Giraud's Family

- Use particular exponentiation algorithms.
- Keep track of variables involved in intermediate steps.
- Consistency check of an invariant that is supposed to be spread till the last steps.
- Examples of countermeasures in this family include:
  - Boscher *et al.* [BNP07],
  - Rivain [Riv09] (and its recently improved version [LRT14]),
  - Kim *et al.* [KKHH11].
- The detailed study of the countermeasures in Giraud's family is left as future work.

Shamir's Family

- Rely on a kind of "checksum" of the computation using smaller numbers:
  - RSA computes in rings $\mathbb{Z}_a$ where a is either a large prime number (*e.g.*, $a = p$ or $a = q$) or the product of large prime numbers (*e.g.*, $a = pq$).
  - Any small number b is coprime with a.
  - We have an isomorphism between the *overring* $\mathbb{Z}_{ab}$ and $\mathbb{Z}_a \times \mathbb{Z}_b$.
  - The nominal computation and the checksum can be conducted in parallel in $\mathbb{Z}_{ab}$.
- Attempt to assert that some invariants on the computations and the checksums hold.
- Many different ways to use the checksums and to verify these invariants.

Notation: $\mathbb{Z}_n$ is a shorthand for $\mathbb{Z}/n\mathbb{Z}$.

2. Test-Based or Infective Countermeasures

- A first way to classify countermeasures is to separate:
  - those which consist in step-wise internal checks during the CRT computation,
  - and those which use an infective computation strategy to make the result unusable by the attacker in case of fault injection.

Test-Based Countermeasures

Test-based countermeasure — Definition

A countermeasure is said to be *test-based* if it attempts to detect fault injections by verifying that some arithmetic invariants are respected, and branch to return an error instead of the numerical result of the algorithm in case of invariant violation.

- Examples of test-based countermeasures:
  - Shamir [Sha99],
  - Aumüller *et al.* [ABF+02],
  - Vigilant [Vig08],
  - Joye *et al.* [JPY01].

Infective Countermeasures

Infective countermeasure — **Definition**

A countermeasure is said to be *infective* if rather than testing arithmetic invariants it uses them to compute a neutral element of some arithmetic operation in a way that would not result in this neutral element if the invariant is violated.
It then uses the results of these computations to infect the result of the algorithm before returning it to make it unusable by the attacker (thus, it does not need branching instructions).

- Examples of infective countermeasures:
  - Blömer *et al.* [BOS03],
  - Ciet & Joye [CJ05],
  - Kim *et al.* [KKHH11].

Infection-Test Equivalence Property

Equivalence between test-based and infective verification **Proposition**

Each test-based (resp. infective) countermeasure has a direct equivalent infective (resp. test-based) countermeasure.

- Invariants that must be verified by countermeasures are modular equality, *i.e.*, they are of the form $a \stackrel{?}{\equiv} b \mod m$.
- Test-based: `if a != b [mod m] then return` *error*.
- Infective: `c := a - b + 1 mod m; ... return` $\mathtt{S}^{\mathtt{c}}$. □

3. Intended Order

- In our fault model, both the countermeasures claiming to be first-order and the ones claiming to be second-order actually offer the same level of protection.
  That is, they resist any number of randomizing faults, but can be broken by a well targeted fault injection + a skipping (test-based) or zeroing (infective) fault to bypass the right verification.

⇒ The concept of integrity verification does not depend on the attack order.

- In most countermeasures, the computations of S_p and S_q take place in overrings $\mathbb{Z}_{pr_1}$ and $\mathbb{Z}_{qr_2}$ rather than in $\mathbb{Z}_p$ and $\mathbb{Z}_q$.
- This allows the retrieval of the results modulo p and q, and verifying the signature modulo r_1 and r_2 (aforementioned checksums).

- Are the smaller rings used to verify the intermediate signatures?
- Or are they used directly to compute checksums that are verified?
- Does CRT recombination takes place in an overring?
- If r_1 is equal to r_2, what is permitted by the resulting symmetry?

Recap

Countermeasure	Family	Verification method/count	Intended order	Order	Small subrings usage
Shamir [Sha99]	Shamir	test / 1	1	0	$r_1 = r_2$, consistency of intermediate signatures
Joye *et al.* [JPY01]	Shamir	test / 2	1	0	checksums of the intermediate CRT signatures
Aumüller *et al.* [ABF+02]	Shamir	test / 5	1	1	$r_1 = r_2$, consistency of the checksums of both intermediate signatures
Blömer *et al.* [BOS03]	Shamir	infection / 2	1	1	direct verification of the intermediate CRT signatures, CRT recombination happens in overring
Ciet & Joye [CJ05]	Shamir	infection / 2	2	1	checksums of the intermediate CRT signatures, CRT recombination happens in overring
Giraud [Gir06]	Giraud	test / 1	1	1	NA
Boscher *et al.* [BNP07]	Giraud	test / 1	1	1	NA
Vigilant [Vig08]	Shamir	test / 7	1	1	$r_1 = r_2$, embedded control values, CRT recombination happens in overring
Rivain [Riv09]	Giraud	test / 2	1	1	NA
Kim *et al.* [KKHH11]	Giraud	infection / 6	1	1	NA

Correctness of a countermeasure — Proposition

A countermeasure is correct if it verifies the integrity of

- the intermediate computation modulo p,
- the intermediate computation modulo q, and
- the CRT recombination (which can be subject to transient fault).

Additional verifications might be necessary if the computations needed for the countermeasure add new vulnerabilities.

- The straightforward countermeasure works at the arithmetic level.
- Any correct optimization of this algorithm is also a correct countermeasure.
- We saw that the countermeasures we studied are optimizations of the straightforward countermeasure. □

High-Order Countermeasures

Proposition

Against randomizing faults, all correct countermeasures are high-order.

However, there are no generic high-order countermeasures if the three types of faults in our attack model are taken into account, but it is possible to build nth-order countermeasures for any n.

- A random fault cannot induce a verification skip, whether test-based of infective.
- Repeating verifications n times can force the attacker to need $n + 1$ faults (one actually faulting the computation and the n others for bypassing the verifications). □

Correcting Shamir's Countermeasure

Algorithm: CRT-RSA with Shamir's countermeasure

Input: Message M, key (p, q, d, i_q) **Output:** Signature $M^d \mod N$, or error

Choose a small random integer r.

$p' = p \cdot r$

$q' = q \cdot r$

$S'_p = M^{d \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr}$

$S'_q = M^{d \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr}$

if $S'_p \not\equiv S'_q \mod r$ **then return** error

$S_p = S'_p \mod p$ // Retrieve intermediate signature in $\mathbb{Z}_p$

$S_q = S'_q \mod q$ // Retrieve intermediate signature in $\mathbb{Z}_q$

$S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ // Recombination in $\mathbb{Z}_N$

return S

Correcting Shamir's Countermeasure

Algorithm: CRT-RSA with Shamir's countermeasure

Input: Message M, key (p, q, d, i_q) **Output:** Signature $M^d \mod N$, or error

Choose a small random integer r.

$p' = p \cdot r$

$q' = q \cdot r$

if $p' \not\equiv 0 \mod p$ **or** $q' \not\equiv 0 \mod q$ **then return** error

$S'_p = M^{d \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr}$

$S'_q = M^{d \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr}$

if $S'_p \not\equiv S'_q \mod r$ **then return** error

$S_p = S'_p \mod p$ // Retrieve intermediate signature in $\mathbb{Z}_p$

$S_q = S'_q \mod q$ // Retrieve intermediate signature in $\mathbb{Z}_q$

$S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ // Recombination in $\mathbb{Z}_N$

if $S \not\equiv S'_p \mod p$ **or** $S \not\equiv S'_q \mod q$ **then return** error

return S

- We simplified Vigilant's countermeasure in 4 steps:
  - simplification of Coron *et al.*'s corrections [CGM+10] + our simplifications from our PPREW'14 paper [RG14];
  - remove additional computation with random numbers;
  - taking advantage of Vigilant's clever sub-CRT embedding technique to verify the 3 necessary invariants in one single step in the small subring;
  - Bonus: transform the countermeasure to it's infective variant.

Algorithm: CRT-RSA with Vigilant's countermeasure

Input: Message M, key (p, q, d_p, d_q, i_q) **Output:** Signature $M^d \mod N$, or error

Choose a small random integer r, R_1, R_2, R_3, R_4. $N = p \cdot q$

$p' = p \cdot r^2$

$i_{pr} = p^{-1} \mod r^2$

$M_p = M \mod p'$

$B_p = p \cdot i_{pr}$; $A_p = 1 - B_p \mod p'$

$M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \mod p'$ // CRT insertion of verification value in M'_p

$d'_p = d_p + R_3 \cdot (p - 1)$

$S'_p = {M'_p}^{d'_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr^2}$

if $M'_p \not\equiv M \mod p$ **or** $d'_p \not\equiv d_p \mod p - 1$ **or** $B_p \cdot S'_p \not\equiv B_p \cdot (1 + d'_p \cdot r) \mod p'$ **then return** error

$S_{pr} = S'_p - B_p \cdot (1 + d'_p \cdot r - R_1)$ // Verification value of S'_p swapped with R_1

$q' = q \cdot r^2$

$i_{qr} = q^{-1} \mod r^2$

$M_q = M \mod q'$

$B_q = q \cdot i_{qr}$; $A_q = 1 - B_q \mod q'$

$M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \mod q'$ // CRT insertion of verification value in M'_q

$d'_q = d_q + R_4 \cdot (q - 1)$

$S'_q = {M'_q}^{d'_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr^2}$

if $M'_q \not\equiv M \mod q$ **or** $d'_q \not\equiv d_q \mod q - 1$ **or** $B_q \cdot S'_q \not\equiv B_q \cdot (1 + d'_q \cdot r) \mod q'$ **then return** error

$S_{qr} = S'_q - B_q \cdot (1 + d'_q \cdot r - R_2)$ // Verification value of S'_q swapped with R_2

if $M_p \not\equiv M_q \mod r^2$ **then return** error

$S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \mod p')$ // Recombination checksum in $\mathbb{Z}_{Nr^2}$

if $N \cdot (S_r - R_2 - q \cdot i_q \cdot (R_1 - R_2)) \not\equiv 0 \mod Nr^2$ **then return** error

if $q \cdot i_q \not\equiv 1 \mod p$ **then return** error

return $S = S_r \mod N$ // Retrieve result in $\mathbb{Z}_N$

Algorithm: CRT-RSA with Vigilant's countermeasure

Input: Message M, key (p, q, d_p, d_q, i_q) **Output:** Signature $M^d \mod N$, or error

Choose a small random integer r, R_1, R_2, R_3, R_4. $N = p \cdot q$

$p' = p \cdot r^2$

$i_{pr} = p^{-1} \mod r^2$

$M_p = M \mod p'$

$B_p = p \cdot i_{pr}$; $A_p = 1 - B_p \mod p'$

$M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \mod p'$ // CRT insertion of verification value in M'_p

$d'_p = d_p + R_3 \cdot (p - 1)$

$S'_p = {M'_p}^{d'_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr^2}$

if $M'_p \not\equiv M \mod p$ **or** $d'_p \not\equiv d_p \mod p - 1$ **or** $B_p \cdot S'_p \not\equiv B_p \cdot (1 + d'_p \cdot r) \mod p'$ **then return** error

$S_{pr} = S'_p - B_p \cdot (1 + d'_p \cdot r - R_1)$ // Verification value of S'_p swapped with R_1

$q' = q \cdot r^2$

$i_{qr} = q^{-1} \mod r^2$

$M_q = M \mod q'$

$B_q = q \cdot i_{qr}$; $A_q = 1 - B_q \mod q'$

$M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \mod q'$ // CRT insertion of verification value in M'_q

$d'_q = d_q + R_4 \cdot (q - 1)$

$S'_q = {M'_q}^{d'_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr^2}$

if $M'_q \not\equiv M \mod q$ **or** $d'_q \not\equiv d_q \mod q - 1$ **or** $B_q \cdot S'_q \not\equiv B_q \cdot (1 + d'_q \cdot r) \mod q'$ **then return** error

$S_{qr} = S'_q - B_q \cdot (1 + d'_q \cdot r - R_2)$ // Verification value of S'_q swapped with R_2

$S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \mod p')$ // Recombination checksum in $\mathbb{Z}_{Nr^2}$

if $pq \cdot (S_r - R_2 - q \cdot i_q \cdot (R_1 - R_2)) \not\equiv 0 \mod Nr^2$ **then return** error

return $S = S_r \mod N$ // Retrieve result in $\mathbb{Z}_N$

Algorithm: CRT-RSA with Vigilant's countermeasure

Input: Message M, key (p, q, d_p, d_q, i_q) **Output:** Signature $M^d \mod N$, or error

Choose a small random integer r, R_1, R_2. $N = p \cdot q$

$p' = p \cdot r^2$

$i_{pr} = p^{-1} \mod r^2$

$M_p = M \mod p'$

$B_p = p \cdot i_{pr}$; $A_p = 1 - B_p \mod p'$

$M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \mod p'$ // CRT insertion of verification value in M'_p

$S'_p = M'^{\,d_p \mod \varphi(p')}_p \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr^2}$

if $M'_p \not\equiv M \mod p$ **or** $B_p \cdot S'_p \not\equiv B_p \cdot (1 + d_p \cdot r) \mod p'$ **then return** error

$S_{pr} = S'_p - B_p \cdot (1 + d_p \cdot r - R_1)$ // Verification value of S'_p swapped with R_1

$q' = q \cdot r^2$

$i_{qr} = q^{-1} \mod r^2$

$M_q = M \mod q'$

$B_q = q \cdot i_{qr}$; $A_q = 1 - B_q \mod q'$

$M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \mod q'$ // CRT insertion of verification value in M'_q

$S'_q = M'^{\,d_q \mod \varphi(q')}_q \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr^2}$

if $M'_q \not\equiv M \mod q$ **or** $B_q \cdot S'_q \not\equiv B_q \cdot (1 + d_q \cdot r) \mod q'$ **then return** error

$S_{qr} = S'_q - B_q \cdot (1 + d_q \cdot r - R_2)$ // Verification value of S'_q swapped with R_2

$S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \mod p')$ // Recombination checksum in $\mathbb{Z}_{Nr^2}$

if $pq \cdot (S_r - R_2 - q \cdot i_q \cdot (R_1 - R_2)) \not\equiv 0 \mod Nr^2$ **then return** error

return $S = S_r \mod N$ // Retrieve result in $\mathbb{Z}_N$

Algorithm: CRT-RSA with Vigilant's countermeasure

Input: Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or

Choose a small random integer r. $N = p \cdot q$
$p' = p \cdot r^2$
$i_{pr} = p^{-1} \mod r^2$
$M_p = M \mod p'$
$B_p = p \cdot i_{pr}$; $A_p = 1 - B_p \mod p'$
$M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \mod p'$ // CRT insertion of verification value in M'_p

$S'_p = {M'_p}^{d_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr^2}$
if $M'_p + N \not\equiv M \mod p$ **then return** error
$S_{pr} = 1 + d_p \cdot r$ // Checksum in $\mathbb{Z}_{r^2}$ for S'_p
$q' = q \cdot r^2$
$i_{qr} = q^{-1} \mod r^2$
$M_q = M \mod q'$
$B_q = q \cdot i_{qr}$; $A_q = 1 - B_q \mod q'$
$M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \mod q'$ // CRT insertion of verification value in M'_q

$S'_q = {M'_q}^{d_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr^2}$
if $M'_q + N \not\equiv M \mod q$ **then return** error
$S_{qr} = 1 + d_q \cdot r$ // Checksum in $\mathbb{Z}_{r^2}$ for S'_q

$S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \mod p')$ // Recombination checksum in $\mathbb{Z}_{r^2}$
$S' = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$ // Recombination in $\mathbb{Z}_{Nr^2}$
if $S' \not\equiv S_r \mod r^2$ **then return** error

return $S = S' \mod N$ // Retrieve result in $\mathbb{Z}_N$

Algorithm: CRT-RSA with Vigilant's countermeasure

Input: Message M, key (p, q, d_p, d_q, i_q) **Output:** Signature $M^d \mod N$, or a random value in $\mathbb{Z}_N$

Choose a small random integer r. $N = p \cdot q$

$p' = p \cdot r^2$

$i_{pr} = p^{-1} \mod r^2$

$M_p = M \mod p'$

$B_p = p \cdot i_{pr}$; $A_p = 1 - B_p \mod p'$

$M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \mod p'$ // CRT insertion of verification value in M'_p

$S'_p = {M'_p}^{d_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr^2}$

$c_p = M'_p + N - M + 1 \mod p$

$S_{pr} = 1 + d_p \cdot r$ // Checksum in $\mathbb{Z}_{r^2}$ for S'_p

$q' = q \cdot r^2$

$i_{qr} = q^{-1} \mod r^2$

$M_q = M \mod q'$

$B_q = q \cdot i_{qr}$; $A_q = 1 - B_q \mod q'$

$M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \mod q'$ // CRT insertion of verification value in M'_q

$S'_q = {M'_q}^{d_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr^2}$

$c_q = M'_q + N - M + 1 \mod q$

$S_{qr} = 1 + d_q \cdot r$ // Checksum in $\mathbb{Z}_{r^2}$ for S'_q

$S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \mod p')$ // Recombination checksum in $\mathbb{Z}_{r^2}$

$S' = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$ // Recombination in $\mathbb{Z}_{Nr^2}$

$c_S = S' - S_r + 1 \mod r^2$

return $S = S'^{c_p c_q c_S} \mod N$ // Retrieve result in $\mathbb{Z}_N$

Generating High-Order Countermeasures

Algorithm: Generation of CRT-RSA with Vigilant's countermeasure at order D

Input: order D **Output:** CRT-RSA algorithm protected against fault injection attack of order D

print Choose a small random integer r.
print $N = p \cdot q$
print $p' = p \cdot r^2$; $i_{pr} = p^{-1} \mod r^2$; $M_p = M \mod p'$; $B_p = p \cdot i_{pr}$; $A_p = 1 - B_p \mod p'$
print $M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \mod p'$
print $q' = q \cdot r^2$; $i_{qr} = q^{-1} \mod r^2$; $M_q = M \mod q'$; $B_q = q \cdot i_{qr}$; $A_q = 1 - B_q \mod q'$
print $M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \mod q'$
print $S'_p = {M'_p}^{d_p \mod \varphi(p')} \mod p'$
print $S'_q = {M'_q}^{d_q \mod \varphi(q')} \mod q'$
print $S_{pr} = 1 + d_p \cdot r$
print $S_{qr} = 1 + d_q \cdot r$
print $S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \mod p')$
print $S' = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
for $i \leftarrow 1$ **to** D **do**
    print c_p; **print** i; **print** $= M'_p + N - M + 1 \mod p$
    print c_q; **print** i; **print** $= M'_q + N - M + 1 \mod q$
    print c_S; **print** i; **print** $= S' - S_r + 1 \mod r^2$
end
print $c^* =$
for $i \leftarrow 1$ **to** D **do**
    print c_p; **print** i; **print** $\times$
    print c_q; **print** i; **print** $\times$
    print c_S; **print** i; **print** $\times$
end
print 1
print return $S = S^{c^*} \mod N$

- Better understanding of existing countermeasures.
- Unified algorithm representations with consistent naming of variables.
- Way to create high-order countermeasures.

- These countermeasures are not specific to CRT-RSA.
- Instead, they are generic ways to verify the integrity of any modular computations.
- Thus, their ideas can be reused...

[ABF+02] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert.
Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures.
In Burton S. Kaliski, Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton.
On the Importance of Checking Cryptographic Protocols for Faults.
In *Proceedings of Eurocrypt'97*, volume 1233 of *LNCS*, pages 37–51. Springer, May 11-15 1997. Konstanz, Germany. DOI: 10.1007/3-540-69053-0_4.

[BNP07] Arnaud Boscher, Robert Naciri, and Emmanuel Prouff.
CRT RSA Algorithm Protected Against Fault Attacks.
In Damien Sauveron, Constantinos Markantonakis, Angelos Bilas, and Jean-Jacques Quisquater, editors, *WISTP*, volume 4462 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2007.

[BOS03] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert.
A new CRT-RSA algorithm secure against bellcore attacks.
In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 311–320. ACM, 2003.

[CGM+10] Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant.
Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm.
In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC*, pages 89–96. IEEE Computer Society, 2010.

[CJ05] Mathieu Ciet and Marc Joye.
Practical fault countermeasures for chinese remaindering based RSA.
In *Fault Diagnosis and Tolerance in Cryptography*, 2005.

[DGRS09] Emmanuelle Dottax, Christophe Giraud, Matthieu Rivain, and Yannick Sierra.
On Second-Order Fault Analysis Resistance for CRT-RSA Implementations.
In Olivier Markowitch, Angelos Bilas, Jaap-Henk Hoepman, Chris J. Mitchell, and Jean-Jacques Quisquater, editors, *WISTP*, volume 5746 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2009.

[Gar65] Harvey L. Garner.
Number Systems and Arithmetic.
Advances in Computers, 6:131–194, 1965.

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

[JPY01] Marc Joye, Pascal Paillier, and Sung-Ming Yen.
Secure evaluation of modular functions, 2001.

[KKHH11] Sung-Kyoung Kim, Tae Hyun Kim, Dong-Guk Han, and Seokhie Hong.
An efficient CRT-RSA algorithm secure against power and fault attacks.
J. Syst. Softw., 84:1660–1669, October 2011.

[Koç94] Çetin Kaya Koç.
High-Speed RSA Implementation, November 1994.
Version 2, `ftp://ftp.rsasecurity.com/pub/pdfs/tr201.pdf`.

[KQ07] ChongHee Kim and Jean-Jacques Quisquater.
Fault attacks for crt based rsa: New attacks, new results, and new countermeasures.
In Damien Sauveron, Konstantinos Markantonakis, Angelos Bilas, and Jean-Jacques Quisquater, editors, *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228. Springer Berlin Heidelberg, 2007.

[LRT14] Duc-Phong Le, Matthieu Rivain, and Chik How Tan.
On double exponentiation for securing RSA against fault analysis.
In Josh Benaloh, editor, *CT-RSA*, volume 8366 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2014.

[RG14] Pablo Rauzy and Sylvain Guilley.
Formal Analysis of CRT-RSA Vigilant's Countermeasure Against the BellCoRe Attack.
In *3rd ACM SIGPLAN Program Protection and Reverse Engineering Workshop (PPREW 2014)*, January 25 2014. San Diego, CA, USA. ISBN: 978-1-4503-2649-0.

[Riv09] Matthieu Rivain.
Securing RSA against Fault Analysis by Double Addition Chain Exponentiation.
Cryptology ePrint Archive, Report 2009/165, 2009.
http://eprint.iacr.org/2009/165/.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman.
A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.
Communications of the ACM, 21(2):120–126, 1978.

[Sha99] Adi Shamir.
Method and apparatus for protecting public key schemes from timing and fault attacks, November 1999.
Patent Number 5,991,415; also presented at the rump session of EUROCRYPT '97.

[TK10] Elena Trichina and Roman Korkikyan.
Multi fault laser attacks on protected CRT-RSA.
In *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010*, pages 75–86, 2010.

[Vig08] David Vigilant.
RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks.
In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.

That was it. Questions?

`rauzy@enst.fr`

IACR ePrint 2014/559