Formal Proofs of CRT-RSA Countermeasures Against the BellCoRe Attack

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

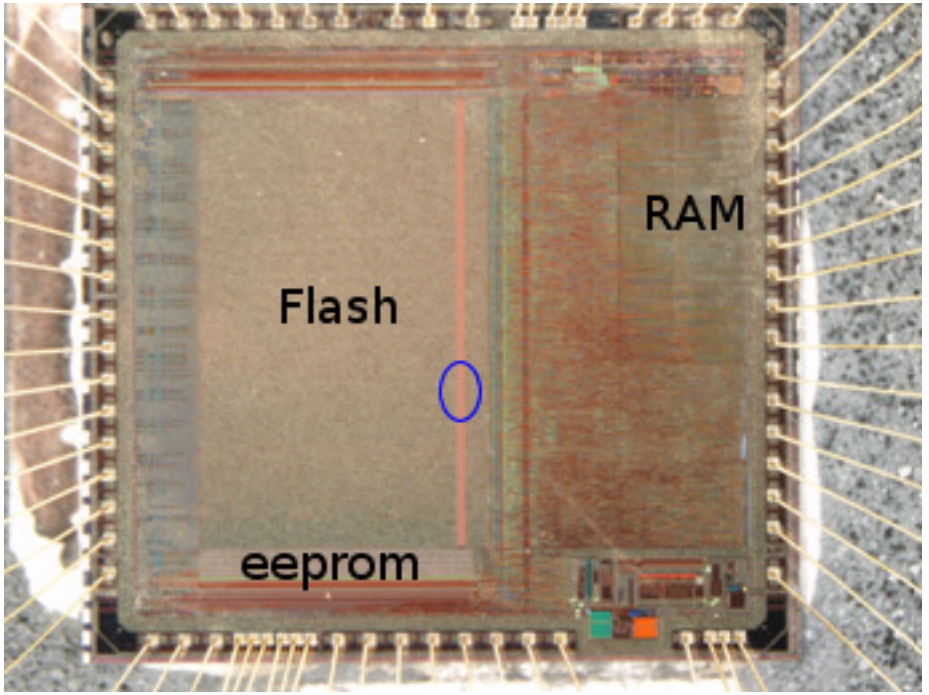
Sylvain Guilley
sylvain.guilley@enst.fr
perso.enst.fr/~guilley

Telecom ParisTech
LTCI / COMELEC / SEN

Séminaire "méthodes formelles et sécurité" Inria–DGA

March 14, 2014
Rennes, France

Fault Injection



RSA
CRT-RSA
The BellCoRe Attack
Countermeasures
Shamir Countermeasure
Aumüller *et al.* Countermeasure
Vigilant's Countermeasure
Shortcomings
Formal Analysis
Results
Conclusions and Perspectives

RSA (*Rivest, Shamir, Adleman*)

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

It works as follows (for simplicity we omit the padding operations):

- 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 $M \equiv S^e \mod N$.

RSA (*Rivest, Shamir, Adleman*)

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

It works as follows (for simplicity we omit the padding operations):

- 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*)

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

It works as follows:

- 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]).

CRT (*Chinese Remainder Theorem*)

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

It works as follows:

- 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]).

CRT (*Chinese Remainder Theorem*)

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

It works as follows:

- 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*)

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.

It works as follows:

- 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 $\gcd(N, S - \widehat{S})$.

BellCoRe (*Bell Communications Research*)

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.

It works as follows:

- 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 Works?

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 Works?

If S_p is faulted (*i.e.*, replaced by $\widehat{S_p} \neq S_p$):

- $S - \widehat{S} = q \cdot \left((i_q \cdot (S_p - S_q) \mod p) - (i_q \cdot (\widehat{S_p} - S_q) \mod p) \right)$

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

Why does it Works?

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 $(q \mod p) \cdot i_q \equiv 1 \mod p$)

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

Why does it Works?

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 $(q \mod p) \cdot i_q \equiv 1 \mod p$)

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

Why does it Works?

- `finja crt-rsa.fia -r`
- `finja crt-rsa.fia -z`

Several protections against the BellCoRe attacks have been proposed.

Some of them are given below:

- Obvious countermeasures: no CRT, or with signature verification [Gir14];
- Shamir [Sha99];
- Aumüller *et al.* [ABF+02];
- Vigilant, original [Vig08] and with some corrections by Coron *et al.* [CGM+10];
- Rivain [Riv09];
- Blömer *et al.* [BOS03];
- Kim *et al.* [KKHH11].

Several protections against the BellCoRe attacks have been proposed.

Some of them are given below:

- Obvious countermeasures: no CRT, or with signature verification [Gir14];
- Shamir [Sha99];
- Aumüller *et al.* [ABF+02];
- Vigilant, original [Vig08] and with some corrections by Coron *et al.* [CGM+10];
- Rivain [Riv09];
- Blömer *et al.* [BOS03];
- Kim *et al.* [KKHH11].

See our paper in the Journal of Cryptographic Engineering:
`http://dx.doi.org/10.1007/s13389-013-0065-3`

`http://eprint.iacr.org/2013/506`

Several protections against the BellCoRe attacks have been proposed.

Some of them are given below:

- Obvious countermeasures: no CRT, or with signature verification [Gir14];
- Shamir [Sha99];
- Aumüller *et al.* [ABF+02];
- Vigilant, original [Vig08] and with some corrections by Coron *et al.* [CGM+10];
- Rivain [Riv09];
- Blömer *et al.* [BOS03];
- Kim *et al.* [KKHH11].

See our paper at PPREW'14:
`http://dx.doi.org/10.1145/2556464.2556466`
`http://eprint.iacr.org/2013/810`

- Introduces a small random number r, co-prime with p and q.
- Carries out computations modulo $p' = p \cdot r$ and $q' = q \cdot r$.

$\Rightarrow$ Allows retrieval of the results by reduction modulo p and modulo q.

$\Rightarrow$ Enables verification by reduction modulo r.

Algorithm

Input : Message m, key (p, q, d, i_q), 32-bit random prime r
Output: Signature $m^d \mod N$, or error if some fault injection is detected.

1 $p' = p \cdot r$
2 $d_p = d \mod (p-1) \cdot (r-1)$
3 $S'_p = m^{d_p} \mod p'$
4 $q' = q \cdot r$
5 $d_q = d \mod (q-1) \cdot (r-1)$
6 $S'_q = m^{d_q} \mod q'$
7 $S_p = S'_p \mod p$
8 $S_q = S'_q \mod q$
9 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$
10 **if** $S'_p \not\equiv S'_q \mod r$ **then**
11 | **return** error
12 **else**
13 | **return** S
14 **end**

Algorithm

Input : Message m, key (p, q, d, i_q), 32-bit random prime r
Output: Signature $m^d \mod N$, or error if some fault injection is detected.

1 $p' = p \cdot r$
2 $d_p = d \mod (p-1) \cdot (r-1)$
3 $S'_p = m^{d_p} \mod p'$
4 $q' = q \cdot r$
5 $d_q = d \mod (q-1) \cdot (r-1)$
6 $S'_q = m^{d_q} \mod q'$
7 $S_p = S'_p \mod p$
8 $S_q = S'_q \mod q$
9 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$
10 **if** $S'_p \not\equiv S'_q \mod r$ **then**
11 | **return** error
12 **else**
13 | **return** S
14 **end**

- Variation of Shamir countermeasure primarily intended to fix two shortcomings:
  - removes the need for d during the computation;
  - checks the CRT recombination step.
- Uses *asymmetrical* verification (computations modulo p' and q' operate on two different objects).
- Also adds some verifications of the intermediate computations.

Algorithm

Input : Message m, key (p, q, d_p, d_q, i_q), 32-bit random prime t
Output : Signature $m^d \mod N$, or error if some fault injection is detected.

1 $p' = p \cdot t$
2 $d'_p = d_p + \text{random}_1 \cdot (p - 1)$
3 $S'_p = m^{d'_p} \mod p'$
4 **if** $(p' \mod p \neq 0)$ **or** $(d'_p \not\equiv d_p \mod (p - 1))$ **then**
5 | **return** error
6 **end**
7 $q' = q \cdot t$
8 $d'_q = d_q + \text{random}_2 \cdot (q - 1)$
9 $S'_q = m^{d'_q} \mod q'$
10 **if** $(q' \mod q \neq 0)$ **or** $(d'_q \not\equiv d_q \mod (q - 1))$ **then**
11 | **return** error
12 **end**
13 $S_p = S'_p \mod p$
14 $S_q = S'_q \mod q$
15 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$
16 **if** $(S - S'_p \not\equiv 0 \mod p)$ **or** $(S - S'_q \not\equiv 0 \mod q)$ **then**
17 | **return** error
18 **end**
19 $S_{pt} = S'_p \mod t$
20 $S_{qt} = S'_q \mod t$
21 $d_{pt} = d'_p \mod (t - 1)$
22 $d_{qt} = d'_q \mod (t - 1)$
23 **if** $S_{pt}^{d_{qt}} \not\equiv S_{qt}^{d_{pt}} \mod t$ **then**
24 | **return** error
25 **else**
26 | **return** S
27 **end**

- All the CRT computations (even the recombination) is carried out in an overring $\mathbb{Z}_{Nr^2}$ of $\mathbb{Z}_N$, where r is a small random number (coprime with N).
- M is transformed into M^* such that
  - $M^* \equiv M \mod N$, and
  - $M^* \equiv 1 + r \mod r^2$.
- Let $S^* = M^{*d} \mod Nr^2$, then
  - $S^* \equiv M^d \mod N$, and
  - $S^* \equiv 1 + dr \mod r^2$,
    using of the binomial theorem in the $\mathbb{Z}_{r^2}$ subring.
- If the verification $S^* \stackrel{?}{=} 1 + dr \mod r^2$ succeeds, then the final result $S = S^* \mod N$ is returned.

- All the CRT computations (even the recombination) is carried out in an overring $\mathbb{Z}_{Nr^2}$ of $\mathbb{Z}_N$, where r is a small random number (coprime with N).
- M is transformed into M^* such that
  - $M^* \equiv M \mod N$, and
  - $M^* \equiv 1 + r \mod r^2$.
- Let $S^* = M^{*d} \mod Nr^2$, then
  - $S^* \equiv M^d \mod N$, and
  - $S^* \equiv 1 + dr \mod r^2$,
    using of the binomial theorem in the $\mathbb{Z}_{r^2}$ subring.
- If the verification $S^* \stackrel{?}{=} 1 + dr \mod r^2$ succeeds, then the final result $S = S^* \mod N$ is returned.

- All the CRT computations (even the recombination) is carried out in an overring $\mathbb{Z}_{Nr^2}$ of $\mathbb{Z}_N$, where r is a small random number (coprime with N).
- M is transformed into M^* such that
  - $M^* \equiv M \mod N$, and
  - $M^* \equiv 1 + r \mod r^2$.
- Let $S^* = M^{*d} \mod Nr^2$, then
  - $S^* \equiv M^d \mod N$, and
  - $S^* \equiv 1 + dr \mod r^2$,
    using of the binomial theorem in the $\mathbb{Z}_{r^2}$ subring.
- If the verification $S^* \stackrel{?}{=} 1 + dr \mod r^2$ succeeds, then the final result $S = S^* \mod N$ is returned.

- All the CRT computations (even the recombination) is carried out in an overring $\mathbb{Z}_{Nr^2}$ of $\mathbb{Z}_N$, where r is a small random number (coprime with N).
- M is transformed into M^* such that
  - $M^* \equiv M \mod N$, and
  - $M^* \equiv 1 + r \mod r^2$.
- Let $S^* = M^{*d} \mod Nr^2$, then
  - $S^* \equiv M^d \mod N$, and
  - $S^* \equiv 1 + dr \mod r^2$,
    using of the binomial theorem in the $\mathbb{Z}_{r^2}$ subring.
- If the verification $S^* \stackrel{?}{=} 1 + dr \mod r^2$ succeeds, then the final result $S = S^* \mod N$ is returned.

Corrections by Coron *et al.*

- Three small modifications are proposed by the authors.
- After that, a safety-claim is made, however
- *"Formal proof of the FA-resistance of Vigilant's scheme including our countermeasures is still an open (and challenging) issue."*

Algorithm

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

1 Choose random numbers r, R_1, R_2, R_3, and R_4.
2 $p' = pr^2$
3 $M_p = M \mod p'$
4 $i_{pr} = p^{-1} \mod r^2$
5 $B_p = p \cdot i_{pr}$
6 $A_p = 1 - B_p \mod p'$
7 $M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
8 **if** $M'_p \not\equiv M \mod p$ **then**
9 | **return** error
10 **end**
11 $d'_p = d_p + R_1 \cdot (p - 1)$
12 $S_{pr} = {M'_p}^{d'_p} \mod p'$
13 **if** $d'_p \not\equiv d_p \mod p - 1$ **then**
14 | **return** error
15 **end**
16 **if** $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then**
17 | **return** error
18 **end**
19 $S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
20 $q' = qr^2$
21 $M_q = M \mod q'$
22 $i_{qr} = q^{-1} \mod r^2$
23 $B_q = q \cdot i_{qr}$
24 $A_q = 1 - B_q \mod q'$
25 $M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
26 **if** $M'_q \not\equiv M \mod q$ **then**
27 | **return** error
28 **end**
29 **if** $M_p \not\equiv M_q \mod r^2$ **then**
30 | **return** error
31 **end**
32 $d'_q = d_q + R_2 \cdot (q - 1)$
33 $S_{qr} = {M'_q}^{d'_q} \mod q'$
34 **if** $d'_q \not\equiv d_q \mod q - 1$ **then**
35 | **return** error
36 **end**
37 **if** $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then**
38 | **return** error
39 **end**
40 $S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
41 $S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
42 $N = pq$
43 **if** $N \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then**
44 | **return** error
45 **end**
46 **if** $q \cdot i_q \not\equiv 1 \mod p$ **then**
47 | **return** error
48 **end**
49 **return** $S \mod N$

Algorithm

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

1 Choose random numbers r, R_1, R_2, R_3, and R_4.
2 $p' = pr^2$
3 $M_p = M \mod p'$
4 $i_{pr} = p^{-1} \mod r^2$
5 $B_p = p \cdot i_{pr}$
6 $A_p = 1 - B_p \mod p'$
7 $M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
8 **if** $M'_p \not\equiv M \mod p$ **then**
9 | **return** error
10 **end**
11 $d'_p = d_p + R_1 \cdot (p - 1)$
12 $S_{pr} = {M'_p}^{d'_p} \mod p'$
13 **if** $d'_p \not\equiv d_p \mod p - 1$ **then**
14 | **return** error
15 **end**
16 **if** $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then**
17 | **return** error
18 **end**
19 $S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
20 $q' = qr^2$
21 $M_q = M \mod q'$
22 $i_{qr} = q^{-1} \mod r^2$
23 $B_q = q \cdot i_{qr}$
24 $A_q = 1 - B_q \mod q'$
25 $M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
26 **if** $M'_q \not\equiv M \mod q$ **then**
27 | **return** error
28 **end**
29 **if** $M_p \not\equiv M_q \mod r^2$ **then**
30 | **return** error
31 **end**
32 $d'_q = d_q + R_2 \cdot (q - 1)$
33 $S_{qr} = {M'_q}^{d'_q} \mod q'$
34 **if** $d'_q \not\equiv d_q \mod q - 1$ **then**
35 | **return** error
36 **end**
37 **if** $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then**
38 | **return** error
39 **end**
40 $S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
41 $S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
42 $N = pq$
43 **if** $N \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then**
44 | **return** error
45 **end**
46 **if** $q \cdot i_q \not\equiv 1 \mod p$ **then**
47 | **return** error
48 **end**
49 **return** $S \mod N$

Algorithm

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

1 Choose random numbers r, R_1, R_2, R_3, and R_4.
2 $p' = pr^2$
3 $M_p = M \mod p'$
4 $i_{pr} = p^{-1} \mod r^2$
5 $B_p = p \cdot i_{pr}$
6 $A_p = 1 - B_p \mod p'$
7 $M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
8 **if** $M'_p \not\equiv M \mod p$ **then**
9 | **return** error
10 **end**
11 $d'_p = d_p + R_1 \cdot (p - 1)$
12 $S_{pr} = {M'_p}^{d'_p} \mod p'$
13 **if** $d'_p \not\equiv d_p \mod p - 1$ **then**
14 | **return** error
15 **end**
16 **if** $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then**
17 | **return** error
18 **end**
19 $S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
20 $q' = qr^2$
21 $M_q = M \mod q'$
22 $i_{qr} = q^{-1} \mod r^2$
23 $B_q = q \cdot i_{qr}$
24 $A_q = 1 - B_q \mod q'$
25 $M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
26 **if** $M'_q \not\equiv M \mod q$ **then**
27 | **return** error
28 **end**
29 **if** $M_p \not\equiv M_q \mod r^2$ **then**
30 | **return** error
31 **end**
32 $d'_q = d_q + R_2 \cdot (q - 1)$
33 $S_{qr} = {M'_q}^{d'_q} \mod q'$
34 **if** $d'_q \not\equiv d_q \mod q - 1$ **then**
35 | **return** error
36 **end**
37 **if** $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then**
38 | **return** error
39 **end**
40 $S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
41 $S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
42 $N = pq$
43 **if** $pq \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then**
44 | **return** error
45 **end**
46 **if** $q \cdot i_q \not\equiv 1 \mod p$ **then**
47 | **return** error
48 **end**
49 **return** $S \mod N$

Algorithm

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

1 Choose random numbers r, R_1, R_2, R_3, and R_4.
2 $p' = pr^2$
3 $M_p = M \mod p'$
4 $i_{pr} = p^{-1} \mod r^2$
5 $B_p = p \cdot i_{pr}$
6 $A_p = 1 - B_p \mod p'$
7 $M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
8 **if** $M'_p \not\equiv M \mod p$ **then**
9 | **return** error
10 **end**
11 $d'_p = d_p + R_1 \cdot (p - 1)$
12 $S_{pr} = {M'_p}^{d'_p} \mod p'$
13 **if** $d'_p \not\equiv d_p \mod p - 1$ **then**
14 | **return** error
15 **end**
16 **if** $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then**
17 | **return** error
18 **end**
19 $S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
20 $q' = qr^2$
21 $M_q = M \mod q'$
22 $i_{qr} = q^{-1} \mod r^2$
23 $B_q = q \cdot i_{qr}$
24 $A_q = 1 - B_q \mod q'$
25 $M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
26 **if** $M'_q \not\equiv M \mod q$ **then**
27 | **return** error
28 **end**
29 **if** $M_p \not\equiv M_q \mod r^2$ **then**
30 | **return** error
31 **end**
32 $d'_q = d_q + R_2 \cdot (q - 1)$
33 $S_{qr} = {M'_q}^{d'_q} \mod q'$
34 **if** $d'_q \not\equiv d_q \mod q - 1$ **then**
35 | **return** error
36 **end**
37 **if** $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then**
38 | **return** error
39 **end**
40 $S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
41 $S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
42 $N = pq$
43 **if** $pq \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then**
44 | **return** error
45 **end**
46 **if** $q \cdot i_q \not\equiv 1 \mod p$ **then**
47 | **return** error
48 **end**
49 **return** $S \mod N$

Shortcomings

- All these countermeasures are hand crafted iteratively, by trial-and-error.
- No proof of their efficiency is given.

- The goal is making sure countermeasures are trustworthy.
- We want to cover a very general attacker model.
- We want our proof to apply to any implementation that is a refinement of the abstract algorithm.

$\Rightarrow$ We want our tool to offer a full fault coverage of CRT-RSA algorithm, thereby keeping the proof valid even if the code is transformed (*e.g.*, optimized, compiled, partitioned in software/hardware, or equipped with dedicated countermeasures).

- The goal is making sure countermeasures are trustworthy.
- We want to cover a very general attacker model.
- We want our proof to apply to any implementation that is a refinement of the abstract algorithm.

$\Rightarrow$ We want our tool to offer a full fault coverage of CRT-RSA algorithm, thereby keeping the proof valid even if the code is transformed (*e.g.*, optimized, compiled, partitioned in software/hardware, or equipped with dedicated countermeasures).

Attacker Model

- An attacker can request a CRT-RSA computation.
- During the computation, the attacker can fault any intermediate value.
- A faulted value can be zero or random.
- The attacker can read the final result of the computation.

- Faulting can occur in the global memory (*permanent fault*) or in a local register or bus (*transient fault*).
- The control flow graph is untouched (however, our fault model covers some types of CFG modifications).

Attacker Model

- An attacker can request a CRT-RSA computation.
- During the computation, the attacker can fault any intermediate value.
- A faulted value can be zero or random.
- The attacker can read the final result of the computation.

- Faulting can occur in the global memory (*permanent fault*) or in a local register or bus (*transient fault*).
- The control flow graph is untouched (however, our fault model covers some types of CFG modifications).

Algorithm Description

- Low level enough for the attack to work if protections are not implemented.
- Intermediate variable that would appear during refinement could be the target of an attack, but such a fault would propagate to an intermediate variable of the high level description.

finja

- Input:
  - A high level description of the computation, and
  - an attack success condition.
- Output:
  - Either the list of possible attacks, or
  - a proof that the computation is resistant to fault injections.

⇒ `http://pablo.rauzy.name/sensi/finja.html`

How does it Works?

- The description of the computation is transformed into a *term*.
- The term is a tree which encodes:
  - dependencies between the intermediate values, and
  - properties of the intermediate values (such as being null, being null modulo another term, or being a multiple of another term).
- Each intermediate value (subterms of the tree) can be faulted, in such case its properties become:
  - nothing, in the case of a randomizing fault, or
  - being null, in the case of a zeroing fault.
- Symbolic computation by term rewriting is used to simplify the term and the attack success condition.

How does it Works?

- The description of the computation is transformed into a *term*.
- The term is a tree which encodes:
  - dependencies between the intermediate values, and
  - properties of the intermediate values (such as being null, being null modulo another term, or being a multiple of another term).
- Each intermediate value (subterms of the tree) can be faulted, in such case its properties become:
  - nothing, in the case of a randomizing fault, or
  - being null, in the case of a zeroing fault.
- Symbolic computation by term rewriting is used to simplify the term and the attack success condition.

How does it Works?

- The description of the computation is transformed into a *term*.
- The term is a tree which encodes:
  - dependencies between the intermediate values, and
  - properties of the intermediate values (such as being null, being null modulo another term, or being a multiple of another term).
- Each intermediate value (subterms of the tree) can be faulted, in such case its properties become:
  - nothing, in the case of a randomizing fault, or
  - being null, in the case of a zeroing fault.
- Symbolic computation by term rewriting is used to simplify the term and the attack success condition.

Rewriting System

- Most of the $\mathbb{Z}$ ring axioms,
- $\mathbb{Z}_n$ subrings,
- And a few theorems.

Rewriting System

- Most of the $\mathbb{Z}$ ring axioms:
  - neutral elements (0 for sums, 1 for products);
  - absorbing element (0, for products);
  - inverses and opposites;
  - associativity and commutativity;
  - but no distributivity (not confluent).
- $\mathbb{Z}_n$ subrings,
- And a few theorems.

Rewriting System

- Most of the $\mathbb{Z}$ ring axioms,
- $\mathbb{Z}_n$ subrings:
  - identity:
    - $(a \mod n) \mod n = a \mod n$,
    - $N^k \mod n = 0$;
  - inverse:
    - $(a \mod n) \times (a^{-1} \mod n) \mod n = 1$,
    - $(a \mod n) + (-a \mod n) \mod n = 0$;
  - associativity and commutativity:
    - $(b \mod n) + (a \mod n) \mod n = a + b \mod n$,
    - $(a \mod n) \times (b \mod n) \mod n = a \times b \mod n$;
  - subrings: $(a \mod n \times m) \mod n = a \mod n$.
- And a few theorems.

Rewriting System

- Most of the $\mathbb{Z}$ ring axioms,
- $\mathbb{Z}_n$ subrings,
- And a few theorems:
  - Fermat's little theorem;
  - its generalization, Euler's theorem;
  - Chinese remainder theorem;
  - Binomial theorem in $\mathbb{Z}_{r^2}$ rings
    $(1 + r)^d \equiv 1 + dr \mod r^2$.

Testing Attacks

For each possible fault attack:

- the faulted term is simplified to propagate to modified properties;
- simplified terms (faulted and original) are then fed into the attack success condition;
- the attack success condition itself is then simplified to either true (the attack works) or false (it doesn't).

Minimal Example of Usage

- Computation: $t = a + b \times c$.
- Let's say the "attack" works if $t \not\equiv a \mod b$.

minimal-example.fia

```
noprop a, b, c ;

t := a + b * c ;

return t ;

%%

@ !=[b] a
```

- `finja minimal-example.fia -r`
- `finja minimal-example.fia -z`

Shamir

- `finja crt-rsa_shamir.fia -r`
- `finja crt-rsa_shamir.fia -z`
- `finja crt-rsa_shamir.fia -t -r`
- `finja crt-rsa_shamir.fia -t -z`

Aumüller *et al.*

- `finja crt-rsa_aumuller.fia -r`
- `finja crt-rsa_aumuller.fia -z`
- `finja crt-rsa_aumuller.fia -t -r`
- `finja crt-rsa_aumuller.fia -t -z`
- `finja crt-rsa_aumuller.fia -s -t -n 2 -r -r`
- `finja crt-rsa_aumuller.fia -s -t -n 2 -r -z`
- `finja crt-rsa_aumuller.fia -s -t -n 2 -z -r`
- `finja crt-rsa_aumuller.fia -s -t -n 2 -z -z`
- `finja crt-rsa_aumuller_pc.fia -s -t -n 2 -r -r`
- `finja crt-rsa_aumuller_pc.fia -s -t -n 2 -r -z`
- `finja crt-rsa_aumuller_pc.fia -s -t -n 2 -z -r`
- `finja crt-rsa_aumuller_pc.fia -s -t -n 2 -z -z`

Vigilant + Coron *et al.*

- `finja crt-rsa_vigilant.fia -t -r`
- `finja crt-rsa_vigilant.fia -t -z`
- `finja crt-rsa_vigilant-fixed.fia -t -r`
- `finja crt-rsa_vigilant-fixed.fia -t -z`
- `finja crt-rsa_vigilant-fixed.fia -s -t -n 2 -r -r`
- `finja crt-rsa_vigilant-fixed.fia -s -t -n 2 -r -z`
- `finja crt-rsa_vigilant-fixed.fia -s -t -n 2 -z -r`
- `finja crt-rsa_vigilant-fixed.fia -s -t -n 2 -z -z`
- `finja crt-rsa_vigilant-fixed_pc.fia -s -t -n 2 -r -r`
- `finja crt-rsa_vigilant-fixed_pc.fia -s -t -n 2 -r -z`
- `finja crt-rsa_vigilant-fixed_pc.fia -s -t -n 2 -z -r`
- `finja crt-rsa_vigilant-fixed_pc.fia -s -t -n 2 -z -z`

- We have formally proven the resistance of the Aumüller *et al.* countermeasure against the BellCoRe attack by fault injection on CRT-RSA.
- We have formally proven the resistance of a fixed version of Vigilant's CRT-RSA countermeasure against the BellCoRe fault injection attack.
- Our research allowed us to safely remove two out of nine verifications in Vigilant's countermeasure, thereby simplifying the protected computation of CRT-RSA while keeping it formally proved.

$\Rightarrow$ We have shown the **importance** of **formal analysis** in the field of **implementation security**.
Not only for the development of trustable devices, but also as an *optimization enabler*, both *for speed and security*.

- We have formally proven the resistance of the Aumüller *et al.* countermeasure against the BellCoRe attack by fault injection on CRT-RSA.
- We have formally proven the resistance of a fixed version of Vigilant's CRT-RSA countermeasure against the BellCoRe fault injection attack.
- Our research allowed us to safely remove two out of nine verifications in Vigilant's countermeasure, thereby simplifying the protected computation of CRT-RSA while keeping it formally proved.

$\Rightarrow$ We have shown the **importance** of **formal analysis** in the field of **implementation security**.
Not only for the development of trustable devices, but also as an *optimization enabler*, both *for speed and security*.

- We have formally proven the resistance of the Aumüller *et al.* countermeasure against the BellCoRe attack by fault injection on CRT-RSA.
- We have formally proven the resistance of a fixed version of Vigilant's CRT-RSA countermeasure against the BellCoRe fault injection attack.
- Our research allowed us to safely remove two out of nine verifications in Vigilant's countermeasure, thereby simplifying the protected computation of CRT-RSA while keeping it formally proved.

$\Rightarrow$ We have shown the **importance** of **formal analysis** in the field of **implementation security**.

Not only for the development of trustable devices, but also as an *optimization enabler*, both *for speed and security*.

- We have formally proven the resistance of the Aumüller *et al.* countermeasure against the BellCoRe attack by fault injection on CRT-RSA.
- We have formally proven the resistance of a fixed version of Vigilant's CRT-RSA countermeasure against the BellCoRe fault injection attack.
- Our research allowed us to safely remove two out of nine verifications in Vigilant's countermeasure, thereby simplifying the protected computation of CRT-RSA while keeping it formally proved.

$\Rightarrow$ We have shown the **importance** of **formal analysis** in the field of **implementation security**.
Not only for the development of trustable devices, but also as an *optimization enabler*, both *for speed and security*.

- We would like to continue improving finja:
  - parallelizing computations;
  - take into account fault injection in the control flow as studied by Heydemann *et al.* [HMER13];
  - automatic variable properties refinement (for attacks by chosen message);
- CRT-RSA resistant against multiple faults;
- Formalization of the proof: "Formal system for the rewritting of terms in an equational theory".
- It would also be interesting to see if general purpose tool such as EasyCrypt [BGZB09] could be a good fit for this kind of work.

References

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.

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.

Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin.
Formal certification of code-based cryptographic proofs.
In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101. ACM, 2009.

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.

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.

Harvey L. Garner.

Number Systems and Arithmetic.

Advances in Computers, 6:131–194, 1965.

Giraud, Christophe.

On the Use of RSA Public Exponent to Improve Implementation Efficiency and Side-Channel Resistance.

COSADE (to appear), 2014.

`http://cosade.org/accepted.html`.

Karine Heydemann, Nicolas Moro, Emmanuelle Encrenaz, and Bruno Robisson.

Formal Verification of a Software Countermeasure Against Instruction Skip Attacks.

Cryptology ePrint Archive, Report 2013/679, 2013.

`http://eprint.iacr.org/`.

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.

Çetin Kaya Koç.

High-Speed RSA Implementation, November 1994.

Version 2, ftp://ftp.rsasecurity.com/pub/pdfs/tr201.pdf.

Matthieu Rivain.

Securing rsa against fault analysis by double addition chain exponentiation.

Cryptology ePrint Archive, Report 2009/165, 2009.

http://eprint.iacr.org/.

Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman.

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.

Commun. ACM, 21(2):120–126, 1978.

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.

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's it. Questions?

RSA
CRT-RSA
The BellCoRe Attack
    Why does it Works?
Countermeasures
Shamir Countermeasure
    Algorithm
Aumüller *et al.* Countermeasure
    Algorithm
Vigilant's Countermeasure
    Corrections by Coron *et al.*
    Algorithm
Shortcomings
Formal Analysis
    Attacker Model
    Algorithm Description
    finja
    Analysis Results
Results
Conclusions and Perspectives

rauzy@enst.fr