Formal Analysis of CRT-RSA Vigilant's Countermeasure 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

January 25, 2014 — 16h30–17h
PPREW 2014 @ San Diego

ACM Digital Library
http://dx.doi.org/10.1145/2556464.2556466

IACR ePrint 2013/810

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 $\widehat{S_p} \neq S_p$):

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

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

Why does it 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 $(q \mod p) \cdot i_q \equiv 1 \mod p$)

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

Several protections against the BellCoRe attacks have been proposed.

Some of them are given below:

- Obvious countermeasures: no CRT, or with signature verification;
- 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;
- 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;
- 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].

- 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 $M^d \mod N$.

Choose random numbers r, R_1, R_2, R_3, and R_4.
$p' = pr^2$
$M_p = M \mod p'$
$i_{pr} = p^{-1} \mod r^2$
$B_p = p \cdot i_{pr}$
$A_p = 1 - B_p \mod p'$
$M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
if $M'_p \not\equiv M \mod p$ **then**
    return error
end
$d'_p = d_p + R_1 \cdot (p - 1)$
$S_{pr} = {M'_p}^{d'_p} \mod p'$
if $d'_p \not\equiv d_p \mod p - 1$ **then**
    return error
end
if $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then**
    return error
end
$S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
$q' = qr^2$
$M_q = M \mod q'$
$i_{qr} = q^{-1} \mod r^2$
$B_q = q \cdot i_{qr}$
$A_q = 1 - B_q \mod q'$
$M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
if $M'_q \not\equiv M \mod q$ **then**
    return error
end
if $M_p \not\equiv M_q \mod r^2$ **then**
    return error
end
$d'_q = dq + R_2 \cdot (q - 1)$
$S_{qr} = {M'_q}^{d'_q} \mod q'$
if $d'_q \not\equiv dq \mod q - 1$ **then**
    return error
end
if $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then**
    return error
end
$S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
$S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
$N = pq$
if $N \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then**
    return error
end
if $q \cdot i_q \not\equiv 1 \mod p$ **then**
    return error
end
return $S \mod N$

Algorithm

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

Choose random numbers r, R_1, R_2, R_3, and R_4.
$p' = pr^2$
$M_p = M \mod p'$
$i_{pr} = p^{-1} \mod r^2$
$B_p = p \cdot i_{pr}$
$A_p = 1 - B_p \mod p'$
$M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
if $M'_p \not\equiv M \mod p$ **then**
  return error
end
$d'_p = d_p + R_1 \cdot (p - 1)$
$S_{pr} = {M'_p}^{d'_p} \mod p'$
if $d'_p \not\equiv d_p \mod p - 1$ **then**
  return error
end
if $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then**
  return error
end
$S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
$q' = qr^2$
$M_q = M \mod q'$
$i_{qr} = q^{-1} \mod r^2$
$B_q = q \cdot i_{qr}$
$A_q = 1 - B_q \mod q'$
$M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
if $M'_q \not\equiv M \mod q$ **then**
  return error
end
if $M_p \not\equiv M_q \mod r^2$ **then**
  return error
end
$d'_q = dq + R_2 \cdot (q - 1)$
$S_{qr} = {M'_q}^{d'_q} \mod q'$
if $d'_q \not\equiv dq \mod q - 1$ **then**
  return error
end
if $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then**
  return error
end
$S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
$S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
$N = pq$
if $N \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then**
  return error
end
if $q \cdot i_q \not\equiv 1 \mod p$ **then**
  return error
end
return $S \mod N$

Algorithm

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

Choose random numbers r, R_1, R_2, R_3, and R_4.
$p' = pr^2$
$M_p = M \mod p'$
$i_{pr} = p^{-1} \mod r^2$
$B_p = p \cdot i_{pr}$
$A_p = 1 - B_p \mod p'$
$M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
if $M'_p \not\equiv M \mod p$ **then**
    return error
end
$d'_p = d_p + R_1 \cdot (p - 1)$
$S_{pr} = {M'_p}^{d'_p} \mod p'$
if $d'_p \not\equiv d_p \mod p - 1$ **then**
    return error
end
if $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then**
    return error
end
$S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
$q' = qr^2$
$M_q = M \mod q'$
$i_{qr} = q^{-1} \mod r^2$
$B_q = q \cdot i_{qr}$
$A_q = 1 - B_q \mod q'$
$M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
if $M'_q \not\equiv M \mod q$ **then**
    return error
end
if $M_p \not\equiv M_q \mod r^2$ **then**
    return error
end
$d'_q = dq + R_2 \cdot (q - 1)$
$S_{qr} = {M'_q}^{d'_q} \mod q'$
if $d'_q \not\equiv dq \mod q - 1$ **then**
    return error
end
if $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then**
    return error
end
$S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
$S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
$N = pq$
if $pq \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then**
    return error
end
if $q \cdot i_q \not\equiv 1 \mod p$ **then**
    return error
end
return $S \mod N$

Algorithm

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

Choose random numbers r, R_1, R_2, R_3, and R_4.
$p' = pr^2$
$M_p = M \mod p'$
$i_{pr} = p^{-1} \mod r^2$
$B_p = p \cdot i_{pr}$
$A_p = 1 - B_p \mod p'$
$M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
if $M'_p \not\equiv M \mod p$ **then**
    return error
end
$d'_p = d_p + R_1 \cdot (p - 1)$
$S_{pr} = {M'_p}^{d'_p} \mod p'$
if $d'_p \not\equiv d_p \mod p - 1$ **then**
    return error
end
if $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then**
    return error
end
$S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
$q' = qr^2$
$M_q = M \mod q'$
$i_{qr} = q^{-1} \mod r^2$
$B_q = q \cdot i_{qr}$
$A_q = 1 - B_q \mod q'$
$M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
if $M'_q \not\equiv M \mod q$ **then**
    return error
end
if $M_p \not\equiv M_q \mod r^2$ **then**
    return error
end
$d'_q = dq + R_2 \cdot (q - 1)$
$S_{qr} = {M'_q}^{d'_q} \mod q'$
if $d'_q \not\equiv dq \mod q - 1$ **then**
    return error
end
if $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then**
    return error
end
$S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
$S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
$N = pq$
if $pq \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then**
    return error
end
if $q \cdot i_q \not\equiv 1 \mod p$ **then**
    return error
end
return $S \mod N$

- 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.

⇒ 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`

- `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`

"Formal proof of the FA-resistance of Vigilant's scheme including our countermeasures is still an open (and challenging) issue."

Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant

- 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, thereby simplifying the protected computation of CRT-RSA while keeping it formally proved.

⇒ 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 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, thereby simplifying the protected computation of CRT-RSA while keeping it formally proved.

⇒ 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 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, thereby simplifying the protected computation of CRT-RSA while keeping it formally proved.

⇒ 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:
  - 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);
  - parallelizing computations. . .
- 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.

- We would like to continue improving finja:
  - 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);
  - parallelizing computations. . .
- 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.

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.

References

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.

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.

References

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?

rauzy@enst.fr