Noname manuscript No.
(will be inserted by the editor)

A Formal Proof of Countermeasures Against Fault Injection Attacks on CRT-RSA

Pablo Rauzy · **Sylvain Guilley**

the date of receipt and acceptance should be inserted later

Abstract In this article, we describe a methodology that aims at either breaking or proving the security of CRT-RSA implementations against fault injection attacks. In the specific case-study of the BellCoRe attack, our work bridges a gap between formal proofs and implementation-level attacks. We apply our results to three implementations of CRT-RSA, namely the unprotected one, that of Shamir, and that of Aumüller *et al.* Our findings are that many attacks are possible on both the unprotected and the Shamir implementations, while the implementation of Aumüller *et al.* is resistant to all single-fault attacks. It is also resistant to double-fault attacks if we consider the less powerful threat-model of its authors.

Keywords RSA (*Rivest, Shamir, Adleman* [RSA78]) · CRT (*Chinese Remainder Theorem*) · fault injection · BellCoRe (*Bell Communications Research*) attack · formal proof · OCaml

1 Introduction

It is known since 1997 that injecting faults during the computation of CRT-RSA could yield to malformed signatures that expose the prime factors (p and q) of the public modulus ($N = p \cdot q$). Notwithstanding, computing without the fourfold acceleration conveyed by the CRT is definitely not an option in practical applications. Therefore, many countermeasures have appeared that consist in step-wise internal checks during the CRT computation. To our best knowledge, none of these countermeasures have been proven formally. Thus without surprise, some of them have been broken, and then patched. The current state-of-the-art in computing CRT-RSA without exposing p and q relies thus on algorithms that have been carefully scrutinized by cryptographers. Nonetheless, neither the hypotheses of the fault attack nor the security itself have been unambiguously modeled.

This is the purpose of this paper. The difficulties are *a priori* multiple: in fault injection attacks, the attacker has an extremely high power because he can fault any variable. Traditional approaches thus seem to fall short in handling this problem. Indeed, there are two canonical methodologies: *formal* and *computational* proofs. Formal proofs (*e.g.*, in the so-called Dolev-Yao model) do not capture the requirement for faults to preserve some information about one of the two moduli; indeed, it considers the RSA as a black-box with a key pair. Computational proofs are way too complicated (in terms of computational complexity) since the handled numbers are typically $2,048$ bit long.

The state-of-the-art contains one reference related to the formal proof of a CRT-RSA implementation: it is the work of Christofi, Chetali, Goubin and Vigilant [CCGV13]. For tractability purposes, the proof is conducted on reduced versions of the algorithms parameters. One fault model is chosen authoritatively (the zeroization of a complete intermediate data), which is a strong assumption. In addition, the verification is conducted on a pseudo-code, hence concerns about its portability after its compilation into machine-level code. Another reference related to formal proofs and fault injection attacks is the work of Guo, Mukhopadhyay and Karri. In [GMK12], they explicit an AES implementation that is provably protected against differential fault analyses [BS97]. The approach is purely combinational, because the faults propagation in AES concerns 32-bit words called columns; consequently, all fatal faults (and thus all innocuous faults) can be enumerated.

Contributions. Our contribution is to reach a full fault coverage of the CRT-RSA algorithm, thereby keeping the proof valid even if the code is transformed (*e.g.*, compiled or par-

Institut Mines-Télécom ; Télécom ParisTech ; CNRS LTCI, E-mail: {*firstname.lastname*}@telecom-paristech.fr

titioned in software/hardware). To this end we developed a tool called finja[1] based on symbolic computation in the framework of modular arithmetic, which enables formal analysis of CRT-RSA and its countermeasures against fault injection attacks. We apply our methods on three implementations: the unprotected one, the one protected by Shamir's countermeasure, and the one protected by Aumüller *et al.*'s countermeasure. We find many possible fault injections that enable a BellCoRe attack on the unprotected implementation of the CRT-RSA computation, as well as on the one protected by Shamir's countermeasure. We formally prove the security of the Aumüller *et al.*'s countermeasure against the BellCoRe attack, under a fault model that considers *permanent faults* (in memory) and *transient faults* (one-time faults, even on copies of the secret key parts), with or without forcing at zero, and with possibly faults at various locations.

Organization of the paper. We recall the CRT-RSA cryptosystem and the BellCoRe attack in Sec. 2; still from an historical perspective, we explain how the CRT-RSA implementation has been amended to withstand more or less efficiently the BellCoRe attack. Then, in Sec. 3, we define our approach. Sec. 4, Sec. 5, and Sec. 6 are case studies using the methods developed in Sec. 3 of respectively an unprotected version of the CRT-RSA computation, a version protected by Shamir's countermeasure, and a version protected by Aumüller *et al.*'s countermeasure. Conclusions and perspectives are in Sec. 7.

2 CRT-RSA and the BellCoRe Attack

This section recaps known results about fault injection attacks on CRT-RSA (see also [Koç94] and [TW12, Chap. 3]). Its purpose is to settle the notions and the associated notations that will be used in the later sections (that contain novel contributions).

2.1 CRT-RSA

RSA is both an *encryption* and a *signature* scheme. It relies on the identity that for all message $0 \leq m < N$, $(m^d)^e \equiv m \mod N$, where $d \equiv e^{-1} \mod \varphi(N)$, by Euler's theorem. In this equation, φ is Euler's totient function, equal to $\varphi(N) = (p-1)\cdot(q-1)$ when $N = p \cdot q$ is a composite number, product of two primes p and q. For example, if Alice generates the signature $S = m^d \mod N$, then Bob can verify it by computing $S^e \mod N$, which must be equal to m unless Alice is pretending to know d although she does not. Therefore the pair (N,d) is called the private key, while the pair (N,e) is called the public key. In this paper, we are not concerned about the key generation step of RSA, and simply assume that d is an unknown number in $\llbracket 1, \varphi(N) = (p-1)\cdot(q-1) \llbracket$. Actually, d can also be chosen equal to the smallest value $e^{-1} \mod \lambda(n)$, where $\lambda(n) = \frac{(p-1)\cdot(q-1)}{\gcd(p-1,q-1)}$ is the Carmichael function. The computation of $m^d \mod N$ can be speeded-up by a factor four by using the Chinese Remainder Theorem (CRT). Indeed, figures modulo p and q are twice as short as those modulo N. For example, for $2,048$ bit RSA, p and q are $1,024$ bit long. The CRT-RSA consists in computing $S_p = m^d \mod p$ and $S_q = m^d \mod q$, which can be recombined into S with a limited overhead. Due to the little Fermat theorem (special case of the Euler theorem when the modulus is a prime), $S_p = (m \mod p)^{d \mod (p-1)} \mod p$. This means that in the computation of S_p, the processed data have $1,024$ bit, and the exponent itself has $1,024$ bits (instead of $2,048$ bits). Thus the multiplication is four times faster and the exponentiation eight times faster. However, as there are two such exponentiations (modulo p and q), the overall CRT-RSA is roughly speaking four times faster than RSA computed modulo N.

This acceleration justifies that CRT-RSA is always used if the factorization of N as $p \cdot q$ is known. In CRT-RSA, the private key is a more rich structure than simply (N,d): it is actually comprised of the 5-tuple (p,q,d_p,d_q,i_q), where:

- $d_p \doteq d \mod (p-1)$,
- $d_q \doteq d \mod (q-1)$,
- $i_q \doteq q^{-1} \mod p$.

The unprotected CRT-RSA algorithm is presented in Alg. 1. It takes advantage of the CRT recombination proposed by Garner in [Gar65]. It is straightforward to check that the signature computed at line 3 belongs to $\llbracket 0, p \cdot q - 1 \rrbracket$. Consequently, no reduction modulo N is necessary before returning S.

Algorithm 1: Unprotected CRT-RSA

```
  Input  : Message $m$, key $(p,q,d_p,d_q,i_q)$
  Output: Signature $m^d \mod N$
1 $S_p = m^{d_p} \mod p$                          /* Signature modulo $p$ */
2 $S_q = m^{d_q} \mod q$                          /* Signature modulo $q$ */
3 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$   /* Recombination */
4 return $S$
```

2.2 BellCoRe Attack on CRT-RSA

In 1997, an dreadful remark has been made by Boneh, DeMillo and Lipton [BDL97], three staff members of BellCoRe: Alg. 1 could reveal the secret primes p and q if the computation is faulted, even in a very random way. The attack can be expressed as the following proposition.

Proposition 1 (Orignal BellCoRe attack) *If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp.*

[1] http://pablo.rauzy.name/sensi/finja.html

$\widehat{S_q}$)[2]*, then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover p (resp. q) as* $\gcd(N, S-\widehat{S})$ *(*with an overwhelming probability*).*

Proof For all integer x, $\gcd(N,x)$ can only take 4 values:

- 1, if N and x are coprime,
- p, if x is a multiple of p but not of q,
- q, if x is a multiple of q but not of p,
- N, if x is a multiple of both p and q, *i.e.*, of N.

In Alg. 1, if S_p is faulted (*i.e.*, replaced by $\widehat{S_p} \neq S_p$), then $S-\widehat{S} = q \cdot ((i_q \cdot (S_p - S_q) \mod p) - (i_q \cdot (\widehat{S_p} - S_q) \mod p))$, and thus $\gcd(N, S-\widehat{S}) = q$.
If S_q is faulted (*i.e.*, replaced by $\widehat{S_q} \neq S_q$), then $S-\widehat{S} \equiv (S_q - \widehat{S_q}) - (q \mod p) \cdot i_q \cdot (S_q - \widehat{S_q}) \equiv 0 \mod p$ because $(q \mod p) \cdot i_q \equiv 1 \mod p$. Thus $S-\widehat{S}$ is a multiple of p. Additionally, $S-\widehat{S}$ is not a multiple of q.
So $\gcd(N, S-\widehat{S}) = p$.

In both cases, the greatest common divisor could yield N. However, $(S-\widehat{S})/q$ in the first case (resp. $(S-\widehat{S})/p$ in the second case) is very unlikely to be a multiple of p (resp. q). Indeed, if the random fault is uniformly distributed, the probability that $\gcd(N, S-\widehat{S})$ is equal to p (resp. q) is negligible[3]. □

This version of the BellCoRe attack requires that two identical messages with the same key can be signed; indeed, one signature that yields the genuine S and an other that is perturbed and thus returns $\widehat{S}$ are needed. Little later, the BellCoRe attack has been improved by Joye, Lenstra and Quisquater [JLQ99]. This time, the attacker can recover p or q with one only faulty signature, provided the input m of RSA is known.

Proposition 2 (One faulty signature BellCoRe attack) *If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$), then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover p (resp. q) as* $\gcd(N, m-\widehat{S}^e)$ *(*with an overwhelming probability*).*

Proof By proposition 1, if a fault occurs during the computation of S_p, then $\gcd(N, S-\widehat{S}) = q$ (*most likely*). This means that:

- $S \not\equiv \widehat{S} \mod p$, and thus $S^e \not\equiv \widehat{S}^e \mod p$ (*indeed, if the congruence was true, we would have $e|p-1$, which is very unlikely*);
- $S \equiv \widehat{S} \mod q$, and thus $S^e \equiv \widehat{S}^e \mod q$;

As $S^e \equiv m \mod N$, this proves the result. A symmetrical reasoning can be done if the fault occurs during the computation of S_q. □

2.3 Protection of CRT-RSA Against the BellCoRe Attack

Several protections against the BellCoRe attack have been proposed. A non-exhaustive list is given below, and then, the most salient features of these countermeasures are described:

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

2.3.1 Obvious Countermeasures

Fault attacks on RSA can be thwarted simply by refraining from implementing the CRT.

If this is not affordable, then the signature can be verified before being outputted. If $S = m^d \mod N$ is the signature, this straightforward countermeasure consists in testing $S^e \overset{?}{\equiv} m \mod N$. Such protection is efficient in practice, but is criticized for three reasons. First of all, it requires an access to e, which is not always present in the secret key structure, as in the 5-tuple example given in Sec. 2.1. Nonetheless, we attract the author's attention on paper [Joy09] for a clever embedding of e into [the representation of] d. Second, the performances are incurred by the extra exponentiation needed for the verification. In some applications, the public exponent can be chosen small (for instance e can be equal to a number such as 3, 17 or 65537), and then d is computed as $e^{-1} \mod \lambda(N)$ using the extended Euclidean algorithm or better alternatives [JP03]. But in general, e is a number possibly as large as d (both are as large as N), thus the obvious countermeasure doubles the computation time (which is really non-negligible, despite the CRT fourfold acceleration). Third, this protection is not immune to a fault injection that would target the comparison. Overall, this explains why other countermeasures have been devised.

2.3.2 Shamir

The CRT-RSA algorithm of Shamir builds on top of the CRT and introduces, in addition to the two primes p and q, a third factor r. This factor r is random[4] and small (less than 64 bit

[2] In other papers related to faults, the faulted variables (such as X) are noted either with a star (X^*) or a tilde ($\tilde{X}$); in this paper, we use a hat, as it can stretch, hence cover the adequate portion of the variable. For instance, it allows to make an unambiguous difference between a faulted data raised at some power and a fault on a data raised at a given power (contrast $\widehat{X}^e$ with $\widehat{X^e}$).

[3] If it nonetheless happens that $\gcd(N, S-\widehat{S}) = N$, then the attacker can simply retry another fault injection, for which the probability that $\gcd(N, S-\widehat{S}) \in \{p, q\}$ increases.

[4] The authors notice that in Shamir's countermeasure, r is *a priori* not a secret, hence can be static and safely divulged.

long), and thus co-prime with p and q. The computations are carried out modulo $p' = p \cdot r$ (resp. modulo $q' = q \cdot r$), which allows for a retrieval of the intended results by reducing them modulo p (resp. modulo q), and for a verification by a reduction modulo r. Alg. 2 describes one version of Shamir's countermeasure. This algorithm is aware of possible fault injections, and thus can raise an *exception* if an incoherence is detected. In this case, the output is not the (purported faulted) signature, but a specific message "error".

Algorithm 2: Shamir CRT-RSA

```
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 has been detected.
$p' = p \cdot r$
$d_p = d \mod (p-1) \cdot (r-1)$
$S'_p = m^{d_p} \mod p'$                // Signature modulo $p'$
$q' = q \cdot r$
$d_q = d \mod (q-1) \cdot (r-1)$
$S'_q = m^{d_q} \mod q'$                // Signature modulo $q'$
$S_p = S'_p \mod p$
$S_q = S'_q \mod q$
 // Same as in line 3 of Alg. 1
$S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$
if $S'_p \not\equiv S'_q \mod r$ then
    return error
else
    return $S$
end
```

2.3.3 Aumüller

The CRT-RSA algorithm of Aumüller *et al.* is a variation of that of Shamir, that is primarily intended to fix two shortcomings. First it removes the need for d in the signature process, and second, it also checks the recombination step. The countermeasure, given in Alg. 3, introduces, in addition to p and q, a third prime t. The computations are done modulo $p' = p \cdot t$ (resp. modulo $q' = q \cdot t$), which allows for a retrieval of the intended results by reducing them modulo p (resp. modulo q), and for a verification by a reduction modulo t. However, the verification is more subtle than for the case of Shamir. In Shamir's CRT-RSA (Alg. 2), the verification is *symmetrical*, in that the computations modulo $p \cdot r$ and $q \cdot r$ operate on the same object, namely m^d. In Aumüller *et al.*'s CRT-RSA (Alg. 3), the verification is *asymmetrical*, since the computations modulo $p \cdot t$ and $q \cdot t$ operate on two different objects, namely $m^{d_p \mod (t-1)}$ and $m^{d_q \mod (t-1)}$. The verification consists in an identity that resembles that of ElGamal for instance: is $(m^{d_p \mod (t-1)})^{d_q \mod (t-1)}$ equivalent to $(m^{d_q \mod (t-1)})^{d_p \mod (t-1)}$ modulo t? Specifically, if we note S'_p the signature modulo p', then $S_p = S \mod p$ is equal to $S'_p \mod p$. Furthermore, let us denote

- $S_{pt} = S'_p \mod t$,
- $S_{qt} = S'_q \mod t$,
- $d_{pt} = d_p \mod (t-1)$, and
- $d_{qt} = d_q \mod (t-1)$.

It can be verified that those figures satisfy the identity: $S_{pt}^{d_{qt}} \equiv S_{qt}^{d_{pt}} \mod t$, because both terms are equal to $m^{d_{pt} \cdot d_{qt}} \mod t$. The prime t is referred to as a security parameter, as the probability to pass the test (at line 23 of Alg. 3) is equal to $1/t$ (*i.e.*, about 2^{-32}), assuming a uniform distribution of the faults. Indeed, this is the probability to find a large number that, once reduced modulo t, matches a predefined value.

Algorithm 3: Aumüller CRT-RSA

```
   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 has been detected.
1  $p' = p \cdot t$
2  $d'_p = d_p + \mathsf{random}_1 \cdot (p-1)$     // Against SPA, not fault
   attacks
3  $S'_p = m^{d'_p} \mod p'$                  // Signature modulo $p'$
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  $d'_q = d_q + \mathsf{random}_2 \cdot (q-1)$     // Against SPA, not fault
   attacks
9  $S'_q = m^{d'_q} \mod q'$                  // Signature modulo $q'$
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 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$     // Same as in line 3
   of Alg. 1
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
```

Alg. 3 does some verifications during the computations, and reports an error in case a fault injection can cause a malformed signature susceptible of unveiling p and q. More precisely, an error is returned in either of these seven cases:

1. p' is not a multiple of p (*because this would amount to faulting p in the unprotected algorithm*)
2. $d'_p = d_p + \mathsf{random}_1 \cdot (p-1)$ is not equal to $d_p \mod (p-1)$ (*because this would amount to faulting d_p in the unprotected algorithm*)
3. q' is not a multiple of q (*because this would amount to faulting q in the unprotected algorithm*)
4. $d'_q = d_q + \mathsf{random}_2 \cdot (q-1)$ is not equal to $d_q \mod (q-1)$ (*because this would amount to faulting d_q in the unprotected algorithm*)
5. $S - S'_p \mod p$ is nonzero (*because this would amount to faulting the recombination modulo p in the unprotected algorithm*)
6. $S - S'_q \mod q$ is nonzero (*because this would amount to faulting the recombination modulo q in the unprotected algorithm*)
7. $S_{pt}^{d_q} \mod t$ is not equal to $S_{qt}^{d_p} \mod t$ (*this checks simultaneously for the integrity of S'_p and S'_q*)

Notice that the last verification could not have been done on the unprotected algorithm, it constitutes the added value of Aumüller *et al.*'s algorithm. These seven cases are *informally* assumed to protect the algorithm against the BellCoRe attack. The criteria for fault detection is not to detect all faults; for instance, a fault on the final return of S (line 26) is not detected. However, of course, such a fault is not exploitable by a BellCoRe attack.

Remark 1 Some parts of the Aumüller algorithm are actually not intended to protect against fault injection attacks, but against side-channel analysis, such as the simple power analysis (SPA). This is the case of lines 2 and 8 in Alg. 3. These SPA attacks consist in monitoring via a side-channel the activity of the chip, in a view to extract the secret exponent, using *generic* methods described in [KJJ99] or more *accurate* techniques such as wavelet transforms [SED$^+$11, DSE$^+$12]. They can be removed if a minimalist protection against only fault injection attacks is looked for; but as they do not introduce weaknesses (in this very specific case), they are simply kept as such.

2.3.4 Vigilant

The CRT-RSA algorithm of Vigilant [Vig08] also considers computations in a larger ring than $\mathbb{Z}_p$ (abbreviation for $\mathbb{Z}/p\mathbb{Z}$) and $\mathbb{Z}_q$, to enable verifications. In this case, a small random number r is cast, and computations are carried out in $\mathbb{Z}_{p \cdot r^2}$ and $\mathbb{Z}_{q \cdot r^2}$. In addition, the computations are now conducted not on the plain message m, but on an encoded message m', built using the CRT as the solution of those two requirements:
i: $m' \equiv m \mod N$, and
ii: $m' \equiv 1 + r \mod r^2$.

This system of equations has a single solution modulo $N \times r^2$, because N and r^2 are coprime. Such a representation allows to conduct in parallel the functional CRT-RSA (line *i*) and a verification (line *ii*). The verification is elegant, as it leverages this remarkable equality: $(1+r)^{d_p} = \sum_{i=0}^{d_p} \binom{d_p}{i} \cdot r^i \equiv 1 + d_p \cdot r \mod r^2$. Thus, as opposed to Aumüller *et al.*'s CRT-RSA, which requires one exponentiation (line 23 of Alg. 3), the verification of Vigilant's algorithm adds only one affine computation (namely $1 + d_p \mod r^2$).

The original description of Vigilant's algorithm involves some trivial computations on p and q, such as $p-1$, $q-1$ and $p \times q$. Those can be faulted, in such a way the BellCoRe attack becomes possible despite all the tests. Thus, a patch by Coron *et al.* has been released in [CGM$^+$10] to avoid the reuse of $\widehat{p-1}$, $\widehat{q-1}$ and $\widehat{p \times q}$ in the algorithm.

2.3.5 Kim

Kim, Kim, Han and Hong propose in [KKHH11] a CRT-RSA algorithm that is based on a collaboration between a customized modular exponentiation and verifications at the recombination level based on Boolean operations. The underlying protection concepts being radically different from the algorithms of Shamir, Aumüller and Vigilant, we choose not to detail this interesting countermeasure.

2.3.6 Other Miscellaneous Fault Injections Attacks

When the attacker has the power to focus its fault injections on *specific bits* of *sensitive resources*, then more challenging security issues arise [BCDG12]. These threats require a highly qualified expertise level, and are thus considered out of the scope of this paper.

Besides, for completeness, we mention that other fault injections mitigating techniques have been promoted, such as the *infective computation scheme* (refer to the seminal paper [BOS03]). This family of protections, although interesting, is neither covered by this article.

In this paper, we will focus on three implementations, namely the unprotected one (Sec. 4), the one protected by Shamir's countermeasure (Sec. 5), and the one protected by Aumüller *et al.*'s countermeasure (Sec. 6).

3 Formal Methods

For all the countermeasures presented in the previous section (Sec. 2), we can see that no formal proof of resistance against attacks is claimed. Informal arguments are given, that convince that for some attack scenarii, the attack attempts are detected hence harmless. Also, an analysis of the probability that an attack succeeds by chance (with a low probability of $1/t$) is carried out, however, this analysis strongly relies on assumptions on the faults distribution. Last but not least, the algorithms include protections against both passive side-channel attacks (typically SPA) and active side-channel attacks, which makes it difficult to analyze for instance the minimal code to be added for the countermeasure to be correct.

3.1 CRT-RSA and Fault Injections

Our goal is to prove that a given countermeasure works, *i.e.*, that it delivers a result which does leak information about neither p nor q even when the implementation is subject to fault injections and to a BellCoRe attack. In addition, we wish to reach this goal with the two following assumptions:

- our proof applies to a very general attacker model, and
- our proof applies to any implementation that is a (strict) refinement of the abstract algorithm.

First, we must define what computation is done, and what is our threat model.

Definition 1 (CRT-RSA) The CRT-RSA computation takes as input a message m, assumed known by the attacker, and a secret key (p,q,d_p,d_q,i_q). Then, the implementation is free to instantiate any variable, but must return a result equal to $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$, where:

- $S_p = m^{d_p} \mod p$, and
- $S_q = m^{d_q} \mod q$.

Definition 2 (fault injection) An attacker is able to request RSA computations, as per Def. 1. During the computation, the attacker can modify any intermediate value by setting it to either a random value or zero. At the end of the computation the attacker can read the result.

Of course, the attacker cannot read the intermediate values used during the computation, since the secret key and potentially the modulus factors are used. Such “whitebox” attack would be too powerful; nonetheless, it is very hard in practice for an attacker to be able to access intermediate variables, due to specific protections (*e.g.*, blinding) and noise in the side-channel leakage (*e.g.*, power consumption, electromagnetic emanation). Remark that our model only takes into account fault injection on data; the control flow is supposed not to be mutable.

As a side remark, we notice that the fault injection model of Def. 2 corresponds to that of Vigilant ([Vig08]), with the exception that the conditional tests can also be faulted. To summarize, an attacker can:

- modify a value in memory (*permanent fault*), and
- modify a value in a local register, cache, or bus (*transient fault*),

but cannot

- inject a permanent fault in the input data (message and secret key), nor
- modify the algorithm control flow graph.

The independence of the proofs on the algorithm implementation demands that the algorithm is described at a high level. The two properties that characterize the relevant level are as follows:

1. The description should be low level enough for the attack to work if protections are not implemented.
2. Any additional 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, thereby having the same effect.

From those requirements, we deduce that:

1. The RSA description must exhibit the computation modulo p and q and the CRT recombination; typically, a completely blackbox description, where the computations would be realized in one go without intermediate variables, is not conceivable.
2. However, it can remain abstract, especially for the computational parts[5].

In our approach, the protections must thus be considered as an augmentation of the unprotected code, *i.e.*, a derived version of the code where additional variables are used. The possibility of an attack on the unprotected code attests that the algorithm is described at the adequate level, while the impossibility of an attack (to be proven) on the protected code shows that added protections are useful in terms of resistance to attacks.

Remark 2 The algorithm only exhibits evidence of safety. If after a fault injection, the algorithm does not simplify to an error detection, then it might only reveal that some simplification is missing. However, if it does not claim safety, it produces a *simplified* occurrence of a possible weakness to be investigated further.

3.2 finja

Several tools are *a priori* suitable for a formal analysis of CRT-RSA. PARI/GP is a specialized computer algebra system, primarily aimed at solving number theory problems. Although PARI/GP can do a fair amount of symbolic manipulation, it remains limited compared to systems like Axiom, Magma, Maple, Mathematica, Maxima, or Reduce. Those last software also fall short to implement automatically number theoretic results like Euler’s theorem. This explains why we developed from scratch a system to reason on modular

[5] For example, a fault in the implementation of the multiplication is either inoffensive, and we do not need to care about it, or it affects the result of the multiplication, and our model take it into account without going into the details of how the multiplication’s is computed

```
1  term    ::= ( stmt )* 'return' mp_expr ';'
2  stmt    ::= ( decl | assign | verif ) ';'
3  decl    ::= 'noprop' mp_var ( ',' mp_var )*
4            | 'prime' mp_var ( ',' mp_var )*
5  assign  ::= var ':=' mp_expr
6  verif   ::= 'if' mp_cond 'abort with' mp_expr
7  mp_expr ::= '{' expr '}' | expr
8  expr    ::= '(' mp_expr ')'
9            | '0' | '1' | var
10           | '-' mp_expr
11           | mp_expr '+' mp_expr
12           | mp_expr '-' mp_expr
13           | mp_expr '*' mp_expr
14           | mp_expr '^' mp_expr
15           | mp_expr 'mod' mp_expr
16 mp_cond ::= '{' cond '}' | cond
17 cond    ::= '(' mp_cond ')'
18           | mp_expr '=' mp_expr
19           | mp_expr '!=' mp_expr
20           | mp_expr '=[' mp_expr ']' mp_expr
21           | mp_expr '!=[' mp_expr ']' mp_expr
22           | mp_cond '/\' mp_cond
23           | mp_cond '\/' mp_cond
24 mp_var  ::= '{' var '}' | var
25 var     ::= [a-zA-Z][a-zA-Z0-9_']*
```

Fig. 1 BNF of finja's input language.

numbers from a formal point of view. Our system is not general, in that it cannot for instance factorize terms in an expression. However, it is able to simplify recursively what is simplifiable from a set of unambiguous rules. This behavior is suitable to the problem of resistance to fault attacks, because the redundancy that is added in the computation is meant to be simplified at the end (if no faults happened).

Our tool finja works within the framework of modular arithmetic, which is the mathematical framework of CRT-RSA computations. The general idea is to represent the computation term as a tree which encodes the computation properties. This term can be simplified by finja, using rules from arithmetic and the properties encoded in the tree. Fault injections in the computation term are simulated by changing the properties of a subterm, thus impacting the simplification process. An attack success condition is also given and used on the term resulting from the simplification to check whether the corresponding attack works on it. The outputs of finja are in HTML form: easily readable reports are produced, which contains all the information about the possible fault injections and their outcome.

3.2.1 Computation Term

The computation is expressed in a convenient statement-based input language. This language's Backus Normal Form (BNF) is given in Fig. 1.

A computation term is defined by a list of statements finished by a `return` statement. Each statement can either:

- declare a variable with no properties (line 3);
- declare a variable which is a prime number (line 4);
- declare a variable by assigning it a value (line 5), in this case the properties of the variable are the properties of the assigned expression;
- perform a verification (line 6).

As can be seen in lines 9 to 15, an expression can be:

- zero, one, or an already declared variable;
- the sum (or difference) of two expressions;
- the product of two expressions;
- the exponentiation of an expression by another;
- the modulus of an expression by another.

The condition in a verification can be (lines 18 to 23):

- the equality or inequality of two expressions;
- the equivalence or non-equivalence of two expressions modulo another (lines 20 and 21);
- the conjunction or disjunction of two conditions.

Optionally, variables (when declared using the `prime` or `noprop` keywords), expressions, and conditions can be protected (lines 3, 4, 7 and 16, `mp` stands for "maybe protected") from fault injection by surrounding them with curly braces. This is useful for instance when it is needed to express the properties of a variable which cannot be faulted in the studied attack model. For example, in CRT-RSA, the definitions of variables d_p, d_q, and i_q are protected because they are seen as input of the computation.

Finally, line 25 gives the regular expression that variable names must match (they start with a letter and then can contain letters, numbers, underscore, and simple quote).

After it is read by finja, the computation expressed in this input language is transformed into a tree (just like the abstract syntax tree in a compiler). This tree encodes the arithmetical properties of each of the intermediate variable, and thus its dependencies on previous variables. The properties of intermediate variables can be everything that is expressible in the input language. For instance, being null or being the product of other terms (and thus, being a multiple of each of them), are possible properties.

3.2.2 Fault Injection

A fault injection on an intermediate variable is represented by changing the properties of the subterm (a node and its whole subtree in the tree representing the computation term) that represent it. In the case of a fault which forces at zero, then the whole subterm is replaced by a term which only has the property of being null. In the case of a randomizing fault, by a term which have no properties.

finja simulates *all the possible fault injections* of the attack model it is launched with. The parameters allow to choose:

- *how many faults* have to be injected (however, the number of tests to be done is impacted by a factorial growth with this parameter, as is the time needed to finish the computation of the proof);
- *the type* of each fault (*randomizing* or *zeroing*);
- if *transient* faults are possible or if only *permanent* faults should be performed.

3.2.3 Attack Success Condition

The attack success condition is expressed using the same condition language as presented in Sec. 3.2.1. It can use any variable introduced in the computation term, plus two special variables `_` and `@` which are respectively bound to the expression returned by the computation term as given by the user and to the expression returned by the computation with the fault injections. This success condition is checked for each possible faulted computation term.

3.2.4 Simplification Process

The simplification is implemented as a recursive traversal of the term tree, based on pattern-matching. It works just like a naive interpreter would, except it does symbolic computation only, and reduces the term based on rules from arithmetic. Simplifications are carried out in $\mathbb{Z}$ ring, and its $\mathbb{Z}_N$ subrings. The tool knows how to deal with most of the $\mathbb{Z}$ ring axioms:

- the neutral elements (0 for sums, 1 for products);
- the absorbing element (0, for products);
- inverses and opposites (only if N is prime);
- associativity and commutativity.

However, it does not implement distributivity as it is not confluent. Associativity is implemented by flattening as much as possible (“removing” all unnecessary parentheses), and commutativity is implemented by applying a stable sorting algorithm on the terms of products or sums.

The tool also knows about most of the properties that are particular to $\mathbb{Z}_N$ rings and applies them when simplifying a term modulo N:

- 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: $(a \mod N \times m) \mod N = a \mod N$.

In addition to those properties a few theorems are implemented to manage more complicated cases where the properties are not enough when conducting symbolic computations:

- Fermat’s little theorem;
- its generalization, Euler’s theorem.

Example. If we have the term `t := a + b * c`, it can be faulted in five different ways (using the randomizing fault):

1. `t := Random`, the final result is faulted;
2. `t := Random + b * c`, a is faulted;
3. `t := a + Random`, the result of $b \times c$ is faulted;
4. `t := a + Random * c`, b is faulted;
5. `t := a + b * Random`, c is faulted.

If the properties that interest us is to know whether t is congruent with a modulo b, we can use `t =[b] a` as the attack success condition. Of course it will be true for t, but it will only be true for the fifth version of faulted t. If we had used the zeroing fault, it would also have been true for the third and fourth versions.

4 Study of an Unprotected CRT-RSA Computation

The description of the unprotected CRT-RSA computation in finja code is given in Fig. 2 (note the similarity of finja’s input code with Alg. 1).

```
noprop m, e ;
prime  {p}, {q} ;

dp := { e^-1 mod (p-1) } ;
dq := { e^-1 mod (q-1) } ;
iq := { q^-1 mod p } ;

Sp := m^dp mod p ;
Sq := m^dq mod q ;

S := Sq + (q * (iq * (Sp - Sq) mod p)) ;

return S ;

%%

_ != @ /\ ( _ =[p] @ \/ _ =[q] @ )
```

Fig. 2 finja code for the unprotected CRT-RSA computation.

As we can see, the definitions of d_p, d_q, and i_q are protected so the computation of the values of these variables cannot be faulted (since they are seen as inputs of the algorithm). After that, S_p and S_q are computed and then recombined in the last expression, as in Def. 1.

To test whether the BellCoRe attack works on a faulted version $\widehat{S}$, we perform the following tests (we note $|S|$ for the simplified version of S):

1. is $|S|$ equal to $|\widehat{S}|$?
2. is $|S \mod p|$ equal to $|\widehat{S} \mod p|$?
3. is $|S \mod q|$ equal to $|\widehat{S} \mod q|$?

If the first test is false and at least one of the second and third is true, we have a BellCoRe attack, as seen in Sec. 2. This is what is described in the attack success condition (after the %% line).

Without transient faults enabled, and in a single fault model, there are 12 different fault injections of which 8 enable a BellCoRe attack with a randomizing fault, and 9 with a zeroing fault. As an example, replacing the intermediate variable holding the value of $i_q \cdot (S_p - S_q) \mod p$ in the final expression with zero or a random value makes the first and second tests false, and the last one true, and thus allows a BellCoRe attack.

5 Study of Shamir's Countermeasure

The description, using finja's formalism, of the CRT-RSA computation allegedly protected by Shamir's countermeasure is given in Fig. 3 (again, note the similarity with Alg. 2).

```
1  noprop error, m, d ;
2  prime {p}, {q}, r ;
3  iq := { q^-1 mod p } ;
4
5  p' := p * r ;
6  dp := d mod ((p-1) * (r-1)) ;
7  Sp' := m^dp mod p' ;
8
9  q' := q * r ;
10 dq := d mod ((q-1) * (r-1)) ;
11 Sq' := m^dq mod q' ;
12
13 Sp := Sp' mod p ;
14 Sq := Sq' mod q ;
15
16 S := Sq + (q * (iq * (Sp - Sq) mod p)) ;
17
18 if Sp' !=[r] Sq' abort with error ;
19
20 return S ;
21
22 %%
23
24 _ != @ /\ ( _ =[p] @ \/ _ =[q] @ )
```

Fig. 3 finja code for the Shamir CRT-RSA computation.

Using the same settings as for the unprotected implementation of CRT-RSA, we find that among the 31 different fault injections, 10 enable a BellCoRe attack with a randomizing fault, and 9 with a zeroing fault. This is not really surprising, as the test which is done on line 18 does not verify if a fault is injected during the computations of S_p or S_q, nor during their recombination in S. For instance zeroing or randomizing the intermediate variable holding the result of $S_p - S_q$ during the computation of S (line 16) results in a BellCoRe attack. To explain why there is this problem in Shamir's countermeasure, some context might be necessary. It can be noted that the fault to inject in the countermeasure must be more accurate in timing (since it targets an intermediate variable obtained by a *subtraction*) than the faults to achieve a BellCoRe attack on the unprotected CRT-RSA (since a fault during an *exponentiation* suffices). However, there is today a consensus to believe that it is very easy to pinpoint in time any single operation of a CRT-RSA algorithm, using a simple power analysis method [KJJ99]. Besides, timely fault injection benches exist. Therefore, the weaknesses in Shamir's countermeasure can indeed be practically exploited.

If the attacker can do *transient faults*, there are a lot more attacks: 66 different possible fault injections of which 24 enable a BellCoRe attack with a randomizing fault and 22 with a zeroing fault. In practice, a transient faults would translate into faulting the variable when it is read (*e.g.*, in a register or on a bus), rather than in (persistent) memory. This behavior could also be the effect of a fault injection in cache, which is later replaced with the good value when it is read from memory again. To the authors knowledge, these are not impossible situations. Nonetheless, growing the power of the attacker to take that into account break some very important assumptions that are classical (sometimes even implicit) in the literature. It does not matter that the parts of the secret key are stored in a secure "key container" if their values can be a faulted at read time. Indeed, we just saw that allowing this kind of fault enable even more possibilities to carry out a BellCoRe attack successfully on a CRT-RSA computation protected by the Shamir's countermeasure. For instance, if the value of p is randomized for the computation of the value of S_p (line 13), then we have $S \neq \widehat{S}$, but also $S \equiv \widehat{S} \mod q$, which enables a BellCoRe attack, as seen in Sec. 2.

It is often asserted that the countermeasure of Shamir is unpractical due to its need for d (as mentioned in [ABF$^+$02] and [Vig08]), and because there is a possible fault attack on the recombination, *i.e.*, line 16 (as mentioned in [Vig08]). However, the attack on the recombination can be checked easily, by testing that $S - S_p \not\equiv 0 \mod p$ and $S - S_q \not\equiv 0 \mod q$ before returning the result. Notwithstanding, to our best knowledge, it is difficult to detect all the attacks our tool found, and so the existence of these attacks (new, in the sense they have not all been described previously) is a compelling reason for not implementing Shamir's CRT-RSA.

6 Study of Aumüller *et al.*'s Countermeasure

The description of the CRT-RSA computation protected by Aumüller *et al.*'s countermeasure is given in Fig. 4 (here too, note the similarity with Alg. 3)

```
noprop error, m, e, r1, r2 ;
prime {p}, {q}, t ;

dp := { e^-1 mod (p-1) } ;
dq := { e^-1 mod (q-1) } ;
iq := { q^-1 mod p } ;

p' := p * t ;
dp' := dp + r1 * (p-1) ;
Sp' := m^dp' mod p' ;

if p' !=[p] 0 \/ dp' !=[p-1] dp abort with error ;

q' := q * t ;
dq' := dq + r2 * (q-1) ;
Sq' := m^dq' mod q' ;

if q' !=[q] 0 \/ dq' !=[q-1] dq abort with error ;

Sp := Sp' mod p ;
Sq := Sq' mod q ;

S := Sq + (q * (iq * (Sp - Sq) mod p)) ;

if S !=[p] Sp' \/ S !=[q] Sq' abort with error ;

Spt := Sp' mod t ;
Sqt := Sq' mod t ;
dpt := dp' mod (t-1) ;
dqt := dq' mod (t-1) ;

if Spt^dqt !=[t] Sqt^dpt abort with error ;

return S ;

%%

_ != @ /\ ( _ =[p] @ \/ _ =[q] @ )
```

Fig. 4 finja code for the Aumüller *et al.* CRT-RSA computation.

Using the same method as before, we can prove that on the 52 different possible faults, *none* of which allow a BellCoRe attack, whether the fault is zero or random. This is a proof that the Aumüller *et al.*'s countermeasure works when there is one fault[6].

Since it allowed more attacks on the Shamir's countermeasure, we also tested the Aumüller *et al.*'s countermeasure against *transient faults* such as described in Sec. 5. There are 120 different possible fault injections when transient faults are activated, and Aumüller *et al.*'s countermeasure is resistant against such fault injections too.

We also used finja to confirm that the computation of d_p, d_q, and i_q (in terms of p, q, and d) must not be part of the algorithm. The countermeasure effectively needs these three variables to be inputs of the algorithm to work properly. For instance there is a BellCoRe attack if d_q happens to be zeroed. However, even with d_p, d_q, and i_q as inputs, we can still attempt to attack a CRT-RSA implementation protected by the Aumüller *et al.*'s countermeasure by doing more than one fault.

We then used finja to verify whether Aumüller *et al.*'s countermeasure would be resistant against *high order* attacks, starting with two faults. We were able to break it if at least one of the two faults was a zeroing fault. We found that this zeroing fault was used to falsify the condition of a verification, which is possible in our threat-model, but which was not in the one of the authors of the countermeasure. If we protect the conditions against fault injection, then the computation is immune two double-fault attacks too. However, even in this less powerful threat-model, a CRT-RSA computation protected by Aumüller *et al.*'s countermeasure is breakable using 3 faults, two of which must be zeroing the computations of d_{pt} and d_{qt}.

7 Conclusions and Perspectives

We have formally proven the resistance of the Aumüller *et al.*'s countermeasure against the BellCoRe attack by fault injection on CRT-RSA. To our knowledge, it is the first time that a formal proof of security is done for a BellCoRe countermeasure.

During our research, we have raised several questions about the assumptions traditionally made by countermeasures. The possibility of fault at read time is, in particular, responsible for many vulnerabilities. The possibility of such fault means that part of the secret key can be faulted (even if only for one computation). It allows an interesting BellCoRe attack on a computation of CRT-RSA protected by Shamir's countermeasure. We also saw that the assumption that the result of conditional expression cannot be faulted, which is widespread in the literature, is a dangerous one as it increased the number of fault necessary to break Aumüller *et al.*'s countermeasure from 2 to 3.

The first of these two points demonstrates the lack of formal studies of fault injection attack and their countermeasures, while the second one shows the importance of formal methods in the field of implementation security.

As a first perspective, we would like to address the hardening of software codes of CRT-RSA under the threat of a bug attack. This attack has been introduced by Biham, Carmeli and Shamir [BCS08] at CRYPTO 2008. It assumes that a hardware has been trapped in such a way that there exists two integers a and b, for which the multiplication is incorrect. In this situation, Biham, Carmeli and Shamir mount an explicit attack scenario where the knowledge of a and b is leveraged to produce a faulted result, that can lead to a remote BellCoRe attack. For sure, testing for the correct

[6] This result is worthwhile some emphasis: the genuine algorithm of Aumüller is thus *proved* resistant against single-fault attacks. At the opposite, the CRT-RSA algorithm of Vigilant is not immune to single fault attacks (refer to [CGM$^+$10]), and the corrections suggested in the same paper by Coron *et al.* have not been proved yet.

functionality of the multiplication operation is impractical (it would amount to an exhaustive verification of 2^{128} multiplications on 64 bit computer architectures). Thus, it can be imagined to use a countermeasure, like that of Aumüller, to detect a fault (caused logically). Our aim would be to assess in which respect our fault analysis formal framework allows to validate the security of the protection. Indeed, a fundamental difference is that the fault is not necessarily injected at *one* random place, but can potentially show up at *several* places.

As another perspective, we would like to handle the repaired countermeasure of Vigilant [CGM+10] and the countermeasure of Kim [KKHH11]. Regarding Vigilant, the difficulty that our verification framework in OCaml [INR] shall overcome is to decide how to inject the remarkable identity $(1+r)^{d_p} \equiv 1 + d_p \cdot r \mod r^2$: either it is kept as such such, like an *ad hoc* theorem (but we need to make sure it is called only at relevant places, since it is not confluent), or it is made more general (but we must ascertain that the verification remains tractable). However, this effort is worthwhile[7], because the authors themselves say in the conclusion of their article [CGM+10] that:

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

Regarding the CRT-RSA algorithm from Kim, the computation is very detailed (it goes down to the multiplication level), and involves Boolean operations (`and`, `xor`, *etc.*). To manage that, more expertise about both arithmetic and logic must be added to our software.

Eventually, we wish to answer a question raised by Vigilant [Vig08] about the prime t involved in Aumüller *et al.*'s countermeasure:

> *"Is it fixed or picked at random in a fixed table?"*

The underlying issue is that of *replay* attacks on CRT-RSA, that are more complicated to handle; indeed, they would require a formal system such as ProVerif [Bla], that is able to prove interactive protocols.

Concerning the tools we developed during our research, they currently only allow to study fault injection in the data, and not in the control flow, it would be interesting to enable formal study of fault injections affecting the control flow.

Eventually, we would like to define and then implement an automatic code mutation algorithm that could transform an unprotected CRT-RSA into a protected one. We know that with a few alterations (see that the differences between Alg. 1 and Alg. 3 are enumerable), this is possible. Such promising approach, if successful, would uncover the *smallest possible* countermeasure of CRT-RSA against fault injection attacks.

Acknowledgements

The authors wish to thank Jean-Pierre Seifert and Wieland Fischer for insightful comments and pieces of advice. We are also grateful to the anonymous reviewers of PROOFS 2013 (UCSB, USA), who helped improve the preliminary version of this paper. Eventually, we acknowledge precious suggestions contributed by Jean-Luc Danger, Jean Goubault-Larrecq and Karine Heydemann.

References

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.

BCDG12. Alexandre Berzati, Cécile Canovas-Dumas, and Louis Goubin. A Survey of Differential Fault Analysis Against Classical RSA Implementations. In Marc Joye and Michael Tunstall, editors, *Fault Analysis in Cryptography*, Information Security and Cryptography, pages 111–124. Springer, 2012.

BCS08. Eli Biham, Yaniv Carmeli, and Adi Shamir. Bug attacks. In *CRYPTO*, volume 5157 of *LNCS*, pages 221–240. Springer, 2008. Santa Barbara, CA, USA.

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.

Bla. Bruno Blanchet. ProVerif: Cryptographic protocol verifier in the formal model. `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`.

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.

BS97. Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO*, volume 1294 of *LNCS*, pages 513–525. Springer, August 1997. Santa Barbara, California, USA. DOI: 10.1007/BFb0052259.

CCGV13. Maria Christofi, Boutheina Chetali, Louis Goubin, and David Vigilant. Formal verification of an implementation of CRT-RSA Vigilant's algorithm. *Journal of Cryptographic Engineering*, 3(3), 2013. DOI: 10.1007/s13389-013-0049-3.

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.

[7] Some results will appear in the proceedings of the 3rd ACM SIGPLAN Program Protection and Reverse Engineering Workshop (PPREW 2014) [RG14], collocated with POPL 2014.

DSE$^+$12. Nicolas Debande, Youssef Souissi, Moulay Abdelaziz Elaabid, Sylvain Guilley, and Jean-Luc Danger. Wavelet Transform Based Pre-processing for Side Channel Analysis. In *HASP*, pages 32–38. IEEE, December 2nd 2012. Vancouver, British Columbia, Canada. DOI: 10.1109/MICROW.2012.15.

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

GMK12. Xiaofei Guo, Debdeep Mukhopadhyay, and Ramesh Karri. Provably secure concurrent error detection against differential fault analysis. Cryptology ePrint Archive, Report 2012/552, 2012. `http://eprint.iacr.org/2012/552/`.

INR. INRIA. OCaml, a variant of the Caml language. `http://caml.inria.fr/ocaml/index.en.html`.

JLQ99. Marc Joye, Arjen K. Lenstra, and Jean-Jacques Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *J. Cryptology*, 12(4):241–245, 1999.

Joy09. Marc Joye. Protecting RSA against Fault Attacks: The Embedding Method. In Luca Breveglieri, Israel Koren, David Naccache, Elisabeth Oswald, and Jean-Pierre Seifert, editors, *FDTC*, pages 41–45. IEEE Computer Society, 2009.

JP03. Marc Joye and Pascal Paillier. GCD-Free Algorithms for Computing Modular Inverses. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 243–253. Springer, 2003.

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Proceedings of CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, 1999.

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`.

RG14. Pablo Rauzy and Sylvain Guilley. Formal Analysis of CRT-RSA Vigilant's Countermeasure Against the BellCoRe Attack — A Pledge for Formal Methods in the Field of Implementation Security. In *3rd ACM SIGPLAN Program Protection and Reverse Engineering Workshop (PPREW 2014)*, 2014. ISBN: 978-1-4503-2649-0.

RSA78. 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.

SED$^+$11. Youssef Souissi, Moulay Aziz Elaabid, Jean-Luc Danger, Sylvain Guilley, and Nicolas Debande. Novel Applications of Wavelet Transforms based Side-Channel Analysis, September 26-27 2011. Non-Invasive Attack Testing Workshop (NIAT 2011), co-organized by NIST & AIST. Todai-ji Cultural Center, Nara, Japan. (PDF).

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.

TW12. Mohammad Tehranipoor and Cliff Wang, editors. *Introduction to Hardware Security and Trust*. Springer, 2012. ISBN 978-1-4419-8079-3.

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.