Published in L. Breveglieri *et al.*, Eds, *Fault Diagnosis and Tolerance in Cryptography (FDTC 2009)*, IEEE Computer Society, pp. 41–45, 2009.

Protecting RSA Against Fault Attacks: The Embedding Method

Marc Joye
Thomson R&D, Security Competence Center
Cesson-Sévigné, France
Email: marc.joye@thomson.net

Abstract—**Fault attacks constitute a major threat toward cryptographic products supporting RSA-based technologies. Most often, the public exponent is unknown, turning resistance to fault attacks into an intricate problem. Over the past few years, several techniques for secure implementations have been published, but none of them is fully satisfactory. We propose a completely different approach by embedding the public exponent into [the description of] the private key. As a result, we obtain a very efficient countermeasure with a 100% fault detection.**

Keywords- **RSA cryptosystem; fault attacks; countermeasures.**

I. Introduction

This paper deals with secure implementations, and more specifically, with the development of an efficient detection method against fault attacks (or errors) for RSA-based cryptosystems.

1) RSA cryptosystem: Let $N = pq$ be the product of two large (equal-size) prime integers. Let also a public exponent e coprime to $\phi(N)$ and corresponding private exponent $d = e^{-1} \bmod \phi(N)$, where $\phi(N) = (p-1)(q-1)$. The public key is $\mathsf{pk} = \{N, d\}$ while the private key is $\mathsf{sk} = \{d\}$ [31]. To compute the signature S on a message m, the legitimate user possessing sk computes $S = \mu(m)^d \bmod N$ for some appropriate padding function μ. Examples for function μ include FDH [2] and PSS [4]. The correctness of signature S can be publicly verified from pk by checking whether $S^e \equiv \mu(m) \pmod{N}$.

The computation of an RSA signature can be sped up using Chinese remaindering (a.k.a. CRT mode). The private key is then given by $\mathsf{sk} = \{p, q, d_p, d_q, i_q\}$ with $d_p = d \bmod (p-1)$, $d_q = d \bmod (q-1)$ and $i_q = q^{-1} \bmod p$. Letting $x = \mu(m)$, signature S is evaluated as $S_p = x^{d_p} \bmod p$, $S_q = x^{d_q} \bmod q$, and $S = \mathrm{CRT}(S_p, S_q) := S_q + q\big(i_q(S_p - S_q) \bmod p\big)$. This yields an expected speed-up factor of 4 [28].

RSA can also be used for encryption by "exchanging" the roles of e and d. The encryption of a message m is given by $C = \mu(m)^e \bmod N$ for some [probabilistic] padding function μ [17] (e.g., [3]). Decryption of ciphertext C is given by $\mu(m) = C^d \bmod N$, from which m is recovered.

2) Fault attacks: A fault attack disturbs the expected behavior of a security device and makes it work abnormally so as to infer sensitive data. Excellent surveys can be found in [7], [16] (see also [24]). Such attacks were introduced by Boneh *et al.* in 1997 [6].

Fault attacks can be very powerful. For example, given a faulty RSA signature evaluated using Chinese remaindering (as is customary for efficiency purposes), a *single* random fault can allow an attacker to recover the whole secret key from the faulty signature [19]. Suppose that the computation of $S_p = \mu(m)^{d_p} \bmod p$ is faulty. We let $\hat{S}_p$ denote the faulty value and $\hat{S} = \mathrm{CRT}(\hat{S}_p, S_q)$. Hence, it follows that $\gcd(\hat{S}^e - \mu(m) \pmod{N}, N) = q$. This attack was confirmed experimentally in [1]. It is thus clear that countermeasures must be taken.

The rest of this paper is organized as follows. In the next section, we review known (software) countermeasures aiming at preventing fault attacks. Section 3 is the core of the paper. We present our new approach for detecting faults. Finally, we conclude in Section 4.

II. Overcoming Fault Attacks

Since the discovery of fault attacks, several [software] countermeasures were proposed. The key principle consists in computing the exponentiation with some redundancy or in exploiting some redundancy already present. Three categories along these lines can be distinguished.

A. Shamir's method and variants

Most known countermeasures rely on an elegant method first suggested by Shamir [32] for RSA using Chinese remaindering. In chronological order, these include [22], [37], [1], [9], [11], [25], [34]. We follow the general presentation of [22]. The computation of $S = x^d \bmod N$ is carried out in three steps:

1) Choose a (small) random integer r;
2) Compute $S^* = x^d \bmod rN$ and $Z = x^d \bmod r$;
3) If $S^* \equiv Z \pmod{r}$ then output $S = S^* \bmod N$; otherwise return `error`.

In order to avoid the `if` branching (cf. Step 3), Yen *et al.* introduced the concept of *infective computation* [37]. For example, applied to the previous algorithm, Step 3 can be replaced with

3'. Choose a random integer $\rho > r$, compute $c = [\rho(S^* - Z) + 1] \bmod r$, and output $S = (S^*)^c \bmod N$.

(Observe that $c = 1$ when there is no error.)

Shamir's method can be adapted to CRT mode, as originally proposed in [32]. However, it cannot detect an error in the CRT recombination (e.g., a corrupted value for i_q). Other methods were subsequently proposed. Some of them have been shown not to offer full tamper-resistance, including the methods of [1], [9], [11] cryptanalyzed respectively in [38], [35], [5] (see also [8]). Note that [5] also suggests a modification of [11] so as to make the scheme immune against their attack. Nevertheless, all methods based on Shamir's method cannot guarantee to detect all faults with a 100% detection. Furthermore, they all impact the performance (running time and memory requirements) and, in certain cases, the personalization process as well.

B. *Giraud's method and variants*

The idea behind Giraud's method [15] is to perform a consistency check directly from the exponentiation algorithm itself. He observes that using the Montgomery powering ladder [27] (see also [23]) for evaluating $x^d \bmod N$, both the values of $x^{d-1} \bmod N$ and $x^d \bmod N$ are available at the end of the computation. Letting $x = \mu(m)$, the method for evaluating $S = x^d \bmod N$ proceeds as follows:

1) Compute $x^d \bmod N$ using Montgomery ladder and obtain the pair $(Z, S) = (x^{d-1} \bmod N, x^d \bmod N)$;
2) If $Z\,x \equiv S \pmod N$ then output S; otherwise return `error`.

Infective computation can be used here as well so as to avoid an explicit check (e.g., $S \leftarrow S^c \bmod N$ with $c = [\rho(S - Zx) + 1] \bmod N$ for a random integer $\rho > N$). Another version is presented in [14]. This countermeasure can also be adapted to CRT mode.

Giraud's method was later extended in [10] to a right-to-left exponentiation algorithm [12].

More recently, Rivain devised a double exponentiation algorithm [30], that is, an algorithm taking on input two exponents a and b and returning $(x^a \bmod N, x^b \bmod N)$. He used this algorithm as a means to detect faults by setting $a = d$ and $b = \phi(N) - d$ and checking that $x^d\, x^{\phi(N)-d} \equiv 1 \pmod N$.

The main disadvantage of Giraud's countermeasure and of its variants is that they impose the exponentiation algorithm. We note that Rivain's countermeasure, as presented in [30], is more suited to CRT mode as the value of $\phi(N)$ is usually unknown in standard mode. But the classical trick of replacing $\phi(N)$ with $e\,d - 1$ (which is a multiple of $\phi(N)$) can be used, provided of course that the value of public exponent e is available.

C. *Signature verification*

Maybe the most natural way to protect any signature scheme is to check its correctness before outputting it [24]. For RSA-type signatures, this requires little overhead as public exponent e is typically small in practice. However, this again assumes that the value of exponent e is available.

Remark 1. The standard security notion for encryption schemes is indistinguishability against adaptive chosen-ciphertext attacks [29]. To achieve indistinguishability, public-key encryption schemes must be probabilistic [17]. To resist chosen-ciphertext attacks, encryption schemes used in practice introduce redundancy so that a random ciphertext will be valid with negligible probability. This includes the widely-used RSA-OAEP encryption scheme [3]. There is therefore no need to add a fault detection mechanism: The correctness of the plaintext is explicitly checked by the decryption algorithm. This explains why the focus is put on RSA signature schemes in this paper.

III. The Embedding Method

In the context of existing APIs, an RSA key object is initialized from only $\{p, q, d_p, d_q, i_q\}$ in CRT mode and from only $\{N, d\}$ in standard mode. This is for example the case for Java Cards$^{\text{TM}}$ [33]. What is worth noting is that, although public, corresponding exponent e is *not* available. As shown in Sections II-A and II-B, this has led implementers to imagine various countermeasures to check whether the computation of $S := x^d \bmod N$ is faulty or not.

Given the value of e — which is in nearly all cases chosen as a small value (a typical value for e is $2^{16}+1$), checking the correctness of signature S is very fast by verifying whether

$$S^e \equiv x \pmod N$$

or equivalently whether $S^e \equiv x \pmod{\{p, q\}}$ in CRT mode. To make the value of e available (and hence allowing the use of the above method), we suggest to *embed* the value of e in the RSA key object. This will detect *all* errors.

One may argue that the value of public exponent e could be recovered. Before detailing our method, we first discuss how this can be performed and the limitations of such an approach.

A. *Recovering public exponent*

We analyze the cases of RSA used in standard mode and in CRT mode separately.

1) CRT mode: In CRT mode, one has to recover the value of e from $\{p, q, d_p, d_q, i_q\}$.

We can first compute $e_0 := {d_p}^{-1} \bmod (p-1)$. Since e is typically small, we should have $e = e_0$. This can be checked by verifying if $e_0\, d_q \equiv 1 \pmod{(q-1)}$. If not, e has to be computed from d_p and d_q [13]. Note that either solution involves the computation of (at least) one modular inversion.

Alternatively, noting that the values of 3, 17 and $2^{16} + 1$ cover the vast majority of use cases, we can *test* if e is one of these values. This can be done with a single modular *multiplication* as follows:

1) Define $\vec{\mathcal{E}} = (3, 17, 2^{16} + 1)$ and $\vec{\mathcal{T}} = (\frac{E}{3}, \frac{E}{17}, \frac{E}{2^{16}+1})$ with $E = \prod_{i=1}^{3} \vec{\mathcal{E}}[i]$;

2) Compute $t = E\, d_p \bmod (p-1)$;

3) If $t = \vec{\mathcal{T}}[i]$ for some $1 \le i \le 3$ then return $\vec{\mathcal{E}}[i]$.

(Optionally, one can further test that $\vec{\mathcal{E}}[i]\, d_q \equiv 1 \pmod{(q-1)}$.)

2) Standard mode: In standard mode, the value of e has to be recovered from $\{N, d\}$. As the value of $\phi(N)$ is unknown, only candidate values for e can be tested.

The first idea that comes to mind is to compute $y := x^d \bmod N$ for some integer x and check whether $y^{e_0} \equiv x \pmod N$ for a candidate value e_0. If so, we deduce that $e = e_0$. To reduce the computational burden, we can can choose the value of 2 for x in the computation of y. Assuming again that the tested value e_0 is small, this countermeasure then roughly amounts to $\log_2 N$ modular squarings. Still this can be prohibitively too expensive.

A more efficient way for testing the value of e is to rely on the observation that if $e = d^{-1} \bmod \phi(N)$ then

$$\begin{aligned}1 - e\,d &= -k\phi(N) = -k(N-(p+q-1))\\ &\equiv k(p+q-1) \pmod N\end{aligned}$$

for some $1 < k < e$. Hence, we can test a candidate value e_0 as follows.

1) Compute $T := (1 - e_0\, d) \bmod N$;
2) If $T < e_0 \cdot 2^{\lceil (\log_2 N)/2 \rceil + \epsilon}$ then return e_0.

In the above method, ϵ denotes a bound on the imbalance between the bit-lengths of p and q. In practice, viewing T as a $\log_2(e_0 N)$-bit string, we can simply test whether its most significant bits (say, its first 128 bits) are zero.

From the above discussion, we see that, when possible, the recovery of public exponent e is somewhat computationally expensive. A better option is to *test* a candidate value for e. However, if e is not one of the "usual" values, this method is of no use. In the next section, we present a different approach covering all practical cases.

B. Embedding public exponent

Our idea to get an efficient fault countermeasure is to embed a representation of public RSA exponent e in the [representation of] the RSA key object.

As aforementioned, such a key object is obtained from $\{p, q, d_p, d_q, i_q\}$ in CRT mode and from $\{N, d\}$ in standard mode. When computing an RSA signature $S = x^d \bmod N$, it is checked whether the public exponent e is embedded in the representation of the RSA key. If this is the case, public exponent e is recovered. Then it is verified if $S^e \equiv x \pmod N$ in standard mode or if $S^e \equiv x \pmod{\{p, q\}}$ in CRT mode. Only upon successful verification, the value of S is returned. Again, infective computation should be preferred to implement the verification step. This can for example be done as $c = [\rho(S^e - x) + 1] \bmod N$ for a random integer $\rho > N$ and $S \leftarrow S^c \bmod N$.

We choose to embed the value of e in RSA modulus N, which is shared between the standard and CRT RSA key objects (notice that N can be obtained as $p \cdot q$ in CRT mode).

An RSA modulus appears as a random string of bits. It is thus preferable to allow an application to distinguish between "regular" RSA moduli, i.e. moduli that do not embed public exponent e, from moduli generated according to our method. To do this, we insert a magical string Σ of, say, 128 bits (for example, $\Sigma =$ `A5A5A5A5A5A5A5A5 A5A5A5A5A5A5A5A5`) followed by public exponent exponent e. Σ serves as an indicator.

To easily identify exponent e, we represent it in $L|V$ format (i.e., length-value format) where L denotes the length (e.g., in words or bits) of the value V of exponent e. It remains to explain how to insert the string $\Sigma \| L \| V$ in the representation of N.

On input exponent e and bit-length ℓ, we have to generate an ℓ-bit RSA modulus $N = pq$ with a predetermined portion comprising Σ, L, and V. Such a modulus can be generated by any suitable method, such as the ones described in [26] or [18]. We refer the reader to [26] for security considerations.

We follow the first algorithm in [18] because the two primes to be generated then lie in a prescribed interval and can therefore benefit from the fast prime generation techniques in [20] (see also [21]). Letting ℓ_0 the bit-length of q (for a balanced modulus, $\ell_0 = \lceil \ell/2 \rceil$), the key generation goes as follows:

1) Generate a random prime $p \in [2^{\ell-\ell_0-1}, 2^{\ell-\ell_0} - 1]$ such that $\gcd(e, p-1) = 1$;
2) Define $N_H := \Sigma \| L \| V$ and let κ denote the bit-length of N_H;
3) If $\kappa < \ell_0$, generate a random prime
$$q \in \left[\left\lfloor \frac{2^{\ell-\kappa} N_H}{p} \right\rfloor + 1, \left\lfloor \frac{2^{\ell-\kappa}(N_H+1) - 1}{p} \right\rfloor\right]$$
such that $\gcd(e, q-1) = 1$;
[Otherwise (i.e., if $\kappa \ge \ell_0$), generate a random prime $q \in [2^{\ell_0-1}, 2^{\ell_0} - 1]$ such that $\gcd(e, q-1) = 1$]*
4) Compute and return
   [standard mode] $N = pq$ and $d = e^{-1} \bmod (p-1)(q-1)$;
   [CRT mode] p, q, $d_p = e^{-1} \bmod (p-1)$, $d_q = e^{-1} \bmod (q-1)$, and $i_q = q^{-1} \bmod p$.

From such a modulus N, it is easy to recover the value of public exponent e: left-right scan the representation of N; if it starts with Σ then exponent e follows in $L|V$ format. Likewise, in CRT mode, the value of public exponent e can be recovered from the product pq.

*[18] presents also an algorithm allowing to embed longer strings but it is not suitable to smart-card implementations. Once more, we insist that there are no reasons to use long public exponents: they are never used in practice.

Indeed, letting $q_{\min} = \left\lfloor \frac{2^{\ell-\kappa} N_H}{p} \right\rfloor + 1$ and $q_{\max} = \left\lfloor \frac{2^{\ell-\kappa}(N_H+1)-1}{p} \right\rfloor$, it can be verified that $[pq_{\min}, pq_{\max}] \subset [2^{\ell-\kappa} N_H, 2^{\ell-\kappa} N_H + 2^{\ell-\kappa} - 1]$. RSA moduli generated as above have therefore their most significant bits matching $N_H = \Sigma \| L \| V$. The condition $\kappa < \ell_0$ ensures that the interval q is searched for (cf. Step 3 above) is non-empty: From $q_{\max} = \left\lfloor \frac{2^{\ell-\kappa}(N_H+1)-1}{p} \right\rfloor \geq \left\lfloor \frac{2^{\ell-\kappa} N_H}{p} \right\rfloor + \left\lfloor \frac{2^{\ell-\kappa}-1}{p} \right\rfloor = q_{\min} + \left\lfloor \frac{2^{\ell-\kappa}-1}{p} \right\rfloor - 1$, it follows that $q_{\max} > q_{\min}$ when $\kappa < \ell_0$.

IV. Conclusion

This paper discussed various methods for preventing fault attacks in RSA-based cryptosystems, and more particularly, in RSA signatures. We proposed different approaches so as to accommodate the most natural yet efficient countermeasure: verifying a signature before outputting it. Interestingly, the proposed countermeasures are fully compliant with existing infrastructures and applications and have little impact on the overall performance. They are particularly well suited to Java Cards$^{\text{TM}}$ offering an on-board RSA key generation.

References

[1] C. Aumüler, P. Bier, W. Fischer, P. Hofreiter, and J-P. Seifert. Fault attack on RSA with CRT: Concrete results and practical countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of Lecture Notes in Computer Science, pages 260–275. Springer-Verlag, 2002.

[2] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.

[3] M. Bellare and P. Rogaway. Optimal asymmetric encryption – How to encrypt with RSA. In *Advances in Cryptology – EUROCRYPT '94*, volume 950 of Lecture Notes in Computer Science, pages 92–111. Springer-Verlag, 1995.

[4] M. Bellare and P. Rogaway. The exact security of digital signatures – How to sign with RSA and Rabin. In *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of Lecture Notes in Computer Science, pages 399–416. Springer-Verlag, 1996.

[5] A. Berzati, C. Canovas, and L. Goubin. (In)security against fault injection attacks for CRT-RSA implementations. *5th Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC 2008*, pages 101–107. IEEE Computer Society Press, 2008.

[6] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology* 14(2):101–119, 2001. Extended abstract in Proc. of EUROCRYPT '97.

[7] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings the IEEE* 94(2):370–382, 2006. Earlier version in Proc. of FDTC 2004.

[8] J. Blömer and M. Otto. Wagner's Attack on a secure CRT-RSA algorithm reconsidered. In *Fault Diagnosis and Tolerance in Cryptography (FDTC 2006)*, volume 4236 of Lecture Notes in Computer Science, pages 13–23. Springer-Verlag, 2006.

[9] J. Blömer, M. Otto, and J-P. Seifert. A new CRT-RSA algorithm secure against Bellcore attack. In *10th ACM Conference on Computer and Communications Security (CCS 2003)*, pages 311–320. ACM Press, 2003.

[10] A. Boscher, R. Naciri, and E. Prouff. CRT RSA algorithm protected against fault attacks. In Information Security Theory and Practices. *Smart Cards, Mobile and Ubiquitous Computing Systems (WISTP 2007)*, volume 4462 of Lecture Notes in Computer Science, pages 229–243. Springer-Verlag, 2007.

[11] M. Ciet and M. Joye. Practical fault countermeasures for Chinese remaindering based RSA. In *2nd Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC 2005)*, pages 124–132, 2005.

[12] J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of Lecture Notes in Computer Science, pages 292–302. Springer-Verlag, 1999.

[13] W. Fischer and J.-P. Seifert. Note on fast computation of secret RSA exponents. In *Information Security and Privacy (ACISP 2002)*, volume 2384 of Lecture Notes in Computer Science, pages 136–143. Springer-Verlag, 2002.

[14] G. Fumaroli and D. Vigilant. Blinded fault resistant exponentiation. In *Fault Diagnosis and Tolerance in Cryptography (FDTC 2006)*, volume 4236 of Lecture Notes in Computer Science, pages 62–70. Springer-Verlag, 2006.

[15] C. Giraud. An RSA implementation resistant to fault attacks and simple power analysis. *IEEE Transactions on Computers* 55(9):1116–1120, 2006. Extended abstract in Proc. of FDTC 2005.

[16] C. Giraud and H. Thiebeauld. A survey on fault attacks. In J.-J. Quisquater et al., editors, *Smart Card Research and Advanced Applications VI (CARDIS 2004)*, pages 159–176. Kluwer Academic Publishers, 2004.

[17] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and Systems Sciences* 28(2):270–299, 1984.

[18] M. Joye. RSA moduli with a predetermined portion: Techniques and applications. In *Information Security Practice and Experience (ISPEC 2008)*, volume 4991 of Lecture Notes in Computer Science, pages 116–130, Springer-Verlag, 2008.

[19] M. Joye, A. Lenstra, and J-J. Quisquater. Chinese remaindering based cryptosystems in the presence of faults. *Journal of Cryptology* 12(4):241–245, 1999.

[20] M. Joye and P. Paillier. Efficient generation of prime numbers on portable devices: An update. In *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of Lecture Notes in Computer Science, pages 160–173. Springer-Verlag, 2006.

[21] M. Joye, P. Paillier, and S. Vaudenay. Efficient generation of prime numbers. In *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of Lecture Notes in Computer Science, pages 340–354. Springer-Verlag, 2000.

[22] M. Joye, P. Paillier, and S-M. Yen. Secure evaluation of modular functions. In *2001 International Workshop on Cryptology and Network Security*, pages 227–229, Taipei, Taiwan, 2001.

[23] M. Joye and S.-M. Yen. The Montgomery powering ladder. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of Lecture Notes in Computer Science, pages 291–302. Springer-Verlag, 2002.

[24] B.S. Kaliski Jr. and M.J.B. Robshaw. Comments on some new attacks on cryptographic devices. *RSA Laboratories' Bulletin*, no. 5, July 14, 1997.

[25] C.H. Kim and J.-J. Quisquater. How can we overcome both side channel analysis and fault attacks on RSA-CRT? In *4th Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC 2007*, pages 21–29. IEEE Computer Society Press, 2007.

[26] A. Lenstra. Generating RSA moduli with a predetermined portion. In *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of Lecture Notes in Computer Science, pages 1–10. Springer-Verlag, 1998.

[27] P.L. Montgomery. Speeding up the Pollard and elliptic curve methods of factorization. *Mathematics of Computation* 48(177):243–264, 1987.

[28] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for public-key RSA cryptosystem. *Electronics Letters* 18(21):905–907, 1982.

[29] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology – CRYPTO '91*, volume 576 of Lecture Notes in Computer Science, pages 433–444. Springer-Verlag, 1992.

[30] M. Rivain. Securing RSA against fault analysis by double addition chain exponentiation. *Topics in Cryptology – CT-RSA 2009*, volume 5473 of Lecture Notes in Computer Science, pages 459–480. Springer-Verlag, 2009.

[31] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the ACM* 21(2):120–126, 1978.

[32] A. Shamir. Improved method and apparatus for protecting public key schemes from timing and fault attacks. US Patent, November 1999. Also presented at the Rump Session of EUROCRYPT '97, 1997.

[33] Sun Microsystems Inc. Application Programming Interface, Java Card$^{\mathrm{TM}}$ Platform, Version 2.2.2, March 2006. Available at URL http://java.sun.com/products/javacard/specs.html.

[34] D. Vigilant. RSA with CRT: A new cost-effective solution to thwart fault attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of Lecture Notes in Computer Science, pages 130–145. Springer-Verlag, 2008.

[35] D. Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In *11th ACM Conference on Computer and Communications Security (CCS 2004)*, pages 92–97. ACM Press, 2004.

[36] S-M. Yen and D. Kim. Cryptanalysis of two protocols for RSA with CRT based on fault infection. In *Fault Diagnosis and Tolerance in Cryptography (FDTC 2004)*, pages 381–385. IEEE Computer Society, 2006.

[37] S-M. Yen, D. Kim, S. Lim, and S. Moon. RSA speedup with residue number system immune against hardware fault cryptanalysis. In *Information Security and Cryptology (ICISC 2001)*, volume 2288 of Lecture Notes in Computer Science, pages 397–413. Springer-Verlag, 2001.

[38] S-M. Yen, S. Moon, and J-C. Ha. Hardware fault attack on RSA with CRT revisited. In *Information Security and Cryptology (ICISC 2002)*, volume 2587 of Lecture Notes in Computer Science, pages 374–388. Springer-Verlag, 2002.