A Method for Preventing "Skipping" Attacks

Marc Joye
Technicolor, Security & Content Protection Labs
1 avenue de Belle Fontaine, 35576 Cesson-Sévigné Cedex, France
Email: marc.joye@technicolor.com

***Abstract*—Until recently, known fault attacks against (non-CRT) exponentiation-based cryptosystems were supposed to be of rather theoretical nature, as they require a precise fault injection, e.g., a bit flip. However, Schmidt and Herbst (FDTC 2008) reported practical fault-attacks against RSA in standard mode using low-cost equipment. Although their attacks were described against RSA, they readily extend to any other exponentiation-based cryptosystem. This paper describes an efficient method to prevent those new attacks.**

***Keywords*-RSA cryptosystem; exponentiation-based cryptosystems; fault attacks; skipping attacks; countermeasures.**

I. Introduction

It is well understood that the use of a strong cryptosystem is not enough to guarantee the security. Indeed, if not properly implemented, secret information can be recovered. In particular, cryptographic implementations should resist against fault attacks [1]. Among them, we quote a special type of attacks against RSA [2], which we refer to as *skipping attacks*, that were reported by Herbst and Schmidt [3].

The basic operation in RSA is the evaluation of a modular exponentiation, say $y = x^d \bmod N$. A faster way to compute this exponentiation is to rely on the Chinese remainder theorem (CRT) [4]. However, our main focus will be on the secure evaluation of *non-CRT* (a.k.a. standard) exponentiations. There are several reasons not to consider CRT implementations. One of them is the key management or infrastructure: the format and the size are different. Another reason is security: CRT implementations of RSA are very sensitive to fault attacks [1], [5].

Shamir provided an elegant countermeasure against fault attacks [6] (see also [7]). When applied to RSA in standard mode for the evaluation of $y = x^d \bmod N$, the countermeasure is:

1) Compute $y' = x^d \bmod rN$ for a (small) random integer r;
2) Compute $z = x^d \bmod r$;
3) Check whether $y' \equiv z \pmod r$, and
   - if so, output $y = y' \bmod N$;
   - if not, return `error`.

Typically, r is a 64-bit integer. The correctness of Shamir's method follows from the Chinese remainder theorem. When the calculations are correct, it is obvious that $y' \equiv y \pmod N$ and $y' \equiv z \pmod r$. In the presence of random faults, the probability that $y' \equiv z \pmod r$ is about $1/r$. When r is a 64-bit value, this means that a random fault is undetected with probability of roughly 2^{-64}. Larger values for r imply a higher detection probability at the expense of more demanding computations.

Vigilant [8] proposed an alternative solution. For the evaluation of $y = x^d \bmod N$ in standard mode, it goes as follows:

1) Form $X = \mathrm{CRT}(x \pmod N, (1+r) \pmod{r^2})$ for a (small) random integer r;
2) Compute $y' = X^d \bmod r^2N$;
3) Check whether $y' \equiv 1 + dr \pmod{r^2}$, and
   - if so, output $y = y' \bmod N$;
   - if not, return `error`.

In Step 1), CRT denotes an application of the Chinese remainder theorem; namely the so-obtained X satisfies $X \equiv x \pmod N$ and $X \equiv 1 + r \pmod{r^2}$. Hence, we have $y' \equiv x^d \pmod N$ and $y' \equiv (1+r)^d \pmod{r^2}$ when the computations are not faulty. The correctness of Step 3) stems from the binomial theorem. We have $(1+r)^d = \sum_{0 \le k \le d} \binom{d}{k} 1^{d-k} r^k$, where $\binom{d}{k}$ denotes the binomial coefficient. Reducing this identity modulo r^2 gives $(1+r)^d \equiv 1 + dr \pmod{r^2}$ and thus $y' \equiv 1 + dr \pmod{r^2}$ when the computations are not faulty. The probability that a random fault is undetected is expected to be about $1/r^2$. As a result, a 32-bit value for r in Vigilant's method should provide the same security level as a 64-bit value for r in Shamir's method.

Vigilant's method presents a couple of advantages over Shamir's method. Specifically, it trades the exponentiation $z = x^d \bmod r$ against the multiplication $1 + dr \bmod r^2$, which is much faster. We note however that the evaluation of z in Shamir's method can be sped up as $x^{d \bmod \mathrm{ord}_r(x)} \bmod r$ (where $\mathrm{ord}_r(x)$ denotes the order of x as an element in $(\mathbb{Z}/r\mathbb{Z})^\times$), provided that this value (or a multiple thereof) is known.

Although offering protection against fault attacks — and so against skipping attacks, Shamir's method and its variants when applied to RSA result in larger moduli for the computation. More generally, when applied to a group exponentiation, Shamir's method and its variants imply handling larger elements and somewhat expensive operations (see for example [9] for an application to elliptic curve

DOI 10.1109/SPW.2012.14

groups). This may in turn incur important performance losses. Other methods are known to protect RSA (numerous countermeasures are reviewed in [10]), but they all come with shortcomings or limitations.

There is therefore a need for an improved solution that provides protection against skipping attacks. This paper provides such a solution. We give a generic description so that it can be applied to any exponentiation-based cryptosystem in any algebraic group, including on elliptic curves, regardless of the underlying exponentiation algorithm.

The rest of this paper is organized as follows. In the next section, we review skipping attacks. In Section III, we present an efficient method to thwart the attacks and detail some implementations. Finally, we conclude in Section IV.

II. Skipping Attacks

Known fault attacks against exponentiation-based cryptosystems assume pretty strong fault models (see [11], [12] for recent surveys). One exception are the skipping attacks due to Schmidt and Herbst against RSA. In this section, we describe their attacks when the exponentiation is performed with the square-and-multiply algorithm. We however stress that the attacks are applicable also to other exponentiation algorithms.

Let $\mathbb{G}$ denote a multiplicatively written group with identity element 1. For RSA, $\mathbb{G}$ is the multiplicative group of integers modulo N; i.e., $(\mathbb{Z}/N\mathbb{Z})^\times$. The square-and-multiply algorithm proceeds as follows:

Algorithm 1 Square-and-multiply

Require: $x \in \mathbb{G}$, $d = (d_{t-1}, \ldots, d_0)_2$
Ensure: $y = x^d$
1: $R_0 \leftarrow 1$; $R_1 \leftarrow x$
2: **for** $i = t-1$ down to 0 **do**
3: $\quad R_0 \leftarrow {R_0}^2$
4: $\quad$ **if** $d_i = 1$ **then**
5: $\quad\quad R_0 \leftarrow R_0 \cdot R_1$
6: **return** R_0

The attack assumes that the adversary manages to skip a squaring operation. This fault model is motivated by the possibilities of glitch and spike attacks, as used for example in [13]. This fault model is also validated in [3].

Suppose for example that the squaring at iteration j in the **for**-loop of Algorithm 1 is skipped. As a result, the output, denoted by $\hat{y}_j$, will be faulty and given by:

$$\hat{y}_j = \prod_{i=j+1}^{t-1} x^{d_i\, 2^{i-1}} \cdot \prod_{i=0}^{j} x^{d_i\, 2^i} \quad .$$

The attack then retrieves the value of exponent d in a bit-by-bit fashion, starting from the least significant bit, as:

$$\hat{y}_j = \begin{cases} \hat{y}_{j-1} & \text{for } d_j = 0 \\ x^{2^{j-1}}\, \hat{y}_{j-1} & \text{for } d_j = 1 \end{cases} \quad .$$

It is straightforward to adapt the described skipping attack to make it work against other exponentiation algorithms.

A more sophisticated skipping attack was mounted against ECDSA in [14]. We present hereafter a slight variant that applies to any DSA-like signature scheme. Let $\mathbb{G} = \langle g \rangle$ denote a (large) group of prime order n, generated by an element g. Let also $y = g^d$ for some secret exponent $d \in \mathbb{Z}/n\mathbb{Z}$. Finally, let $F : \mathbb{G} \to \mathbb{Z}$ be some public function mapping elements from $\mathbb{G}$ to integers and $h : \{0,1\}^* \to \mathbb{Z}/n\mathbb{Z}$ be a hash function. The signature σ on a message $m \in \{0,1\}^*$ is given by the pair (r, s) with

- $r = F(z) \bmod n$ where $z = g^k$ with k is chosen at random in $(\mathbb{Z}/n\mathbb{Z})^\times$, and
- $s = k^{-1}\big(h(m) + r \cdot d\big) \bmod n$.

The validity of signature $\sigma = (r, s)$ is checked by verifying whether $F(g^{u_1}\, y^{u_2}) \equiv r \pmod n$ where $u_1 = h(m)/s \bmod n$ and $u_2 = r/s \bmod n$.

As an illustration, suppose again that the square-and-multiply (Alg. 1) is used for exponentiation. Let $k = (k_{t-1}, \ldots, k_0)_2$ denote the binary expansion of k and $\tilde{k}$ its least significant part (i.e., $\tilde{k} = (k_j, \ldots, k_0)_2$ for some $j \ll t-1$). If the squaring at iteration j is skipped during the computation of $z = g^k$ then the corresponding faulty signature will be given by $\hat{\sigma} = (\hat{r}, \hat{s})$ where $\hat{r} = F(\hat{z}) \bmod n$ with

$$\hat{z} = \prod_{i=j+1}^{t-1} g^{k_i\, 2^{i-1}} \cdot \prod_{i=0}^{j} g^{k_i\, 2^i} = \big(z \cdot g^{\tilde{k}}\big)^{1/2}$$

and $\hat{s} = k^{-1}\big(h(m) + \hat{r} \cdot d\big) \bmod n$. Note that since n is a large prime (and thus is odd), square roots exist and are unique in $\mathbb{G}$. The main observation is that, letting $\hat{u}_1 = h(m)/\hat{s} \bmod n$ and $\hat{u}_2 = \hat{r}/\hat{s} \bmod n$, one recovers $z = g^k$ as

$$g^{\hat{u}_1}\, y^{\hat{u}_2} = g^{\frac{h(m)}{\hat{s}}}\, y^{\frac{\hat{r}}{\hat{s}}} = g^{\frac{h(m) + d\hat{r}}{\hat{s}}} = g^k \quad .$$

The attack therefore consists in testing for all possible $\tilde{k} \in \{0,1\}^j$ whether

$$\hat{r} \equiv F\big((z \cdot g^{\tilde{k}})^{1/2}\big) \pmod n \quad \text{with } z = g^{\hat{u}_1}\, y^{\hat{u}_2}$$

holds. If so, the corresponding $\tilde{k}$ is a candidate value for the least significant part of k. Collecting sufficiently many such values from multiple faulty signatures then allows one to recover the private signing exponent d through lattice reduction [15], [16].

III. An Efficient Prevention Method

As described in the previous section, an exponentiation algorithm has to evaluate, on input an element x in a group $\mathbb{G}$ and an exponent d, $y = x^d$.

The idea consists in evaluating, in parallel with $y = x^d$, the value of $f = d \cdot 1$ or a derived value thereof. The evaluations are performed using the same exponentiation algorithm by "gluing" together the group operations underlying the computation of y and f. In this context, "gluing" means that the two group operations appear as an *atomic* operation, so as to ensure that a perturbation to one operation also perturbs the other. Further, we note that such a behavior can be emulated; see an example at the end of this section. This models the fact that a perturbation will likely affect operations that are performed close in time.

The computation of f may be carried out over the integers, or for better efficiency, over the integers modulo Ω (where typically Ω is a 64-bit value). The computation is assumed to be error-free if f is equal to d (when calculated over the integers) or if f is equal to d modulo Ω. More generally, the computation of f may be computed in any group $\mathbb{G}'$ where the computations are fast. The first case corresponds to $\mathbb{G}' = \mathbb{Z}^+$ and the second case to $\mathbb{G}' = (\mathbb{Z}/\Omega\mathbb{Z})^+$ (namely, the additive group of integers and the additive group of integers modulo Ω). The correctness can also be checked "on-the-fly", e.g., after the computation of one or several words of exponent d.

We describe below several applications of the method when applied to the square-and-multiply algorithm.

Algorithm 2 Square-and-multiply protected against skipping attacks (I)

Require: $x \in \mathbb{G}$, $d = (d_{t-1}, \ldots, d_0)_2$
Ensure: $y = x^d$
1: $R_0 \leftarrow 1$; $R_1 \leftarrow x$
2: $T_0 \leftarrow 0$; $T_1 \leftarrow 1$
3: **for** $i = t - 1$ down to 0 **do**
4: $\quad (R_0, T_0) \leftarrow ({R_0}^2, 2 \cdot T_0)$
5: $\quad$ **if** $(d_i = 1)$ **then**
6: $\quad\quad (R_0, T_0) \leftarrow (R_0 \cdot R_1, T_0 + T_1)$
7: **if** $(T_0 \neq d)$ **then**
8: $\quad$ **return** `error`
9: **return** R_0

The implementation given in Algorithm 2 evaluates f over the integers; R_0 is the temporary variable used to compute $y = x^d$ while T_0 is used to compute $f = d \cdot 1$. Algorithm 3 shows how to evaluate f over the integers modulo Ω.

A variant of Algorithm 3 is to compute $\bar{d} = d \bmod \Omega$ at the beginning of the algorithm and then to replace the final check with $T_0 \neq \bar{d}$, i.e., without the $(\bmod \Omega)$. More generally, it is also possible to pre-compute some value depending on d, say $\bar{d} = G(d)$, and then check whether $G(T_0) \neq \bar{d}$ for some function G.

Algorithm 3 Square-and-multiply protected against skipping attacks (II)

Require: $x \in \mathbb{G}$, $d = (d_{t-1}, \ldots, d_0)_2$
Ensure: $y = x^d$
1: $R_0 \leftarrow 1$; $R_1 \leftarrow x$
2: $T_0 \leftarrow 0$; $T_1 \leftarrow 1$
3: **for** $i = t - 1$ down to 0 **do**
4: $\quad (R_0, T_0) \leftarrow ({R_0}^2, 2 \cdot T_0 \pmod \Omega)$
5: $\quad$ **if** $(d_i = 1)$ **then**
6: $\quad\quad (R_0, T_0) \leftarrow (R_0 \cdot R_1, T_0 + T_1 \pmod \Omega)$
7: **if** $(T_0 \not\equiv d \pmod \Omega)$ **then**
8: $\quad$ **return** `error`
9: **return** R_0

It is worthwhile noting that the operations on (R_0, T_0) are performed in an atomic way. Such a behaviour can be emulated by "gluing" together the operations. This is important as otherwise a skipping attack on R_0 may remain undetected. In order to glue the operations, an additional register A is used together with two random elements $r, r' \in \{0,1\}^{|A|}$. Random elements may be chosen once for all at the beginning of the exponentiation or dynamically whenever a glued multiplication is evaluated. Furthermore, for improved efficiency, random elements r and r' can be derived from shorter random seeds. In the following algorithm, an overlined element viewed as a bitstring (e.g., $\overline{T_0}$) means the two's complement and $\oplus$ denotes the exclusive-OR operator applied to two elements viewed as bitstrings.

Algorithm 4 Example of glued multiplication

Require: $R_0, R_1, T_0, T_1, r, r'$
Ensure: $(R_0 \cdot R_1, T_0 + T_1)$
1: $A \leftarrow r'$
2: $T_0 \leftarrow \overline{T_0}$
3: $A \leftarrow R_0 \cdot R_1$
4: $A \leftarrow A \oplus r$
5: $T_0 \leftarrow \overline{T_0} + T_1$
6: $R_0 \leftarrow A \oplus r$
7: **return** (R_0, T_0)

It is easily verified that if one of the above instructions in the glued multiplication is skipped this will result in a random value for R_0 or in an incorrect value for T_0. In the first case, the returned faulty value for R_0 will appear random and so the final output of the exponentiation algorithm will be of no use for the attacker. In the second case, the incorrect value for T_0 will be detected at the end of the exponentiation.

A similar algorithm can be designed for the squaring. Likewise, we can design similar algorithms wherein $T_0 + T_1$ (resp. $2 \cdot T_0$) is computed modulo Ω. There are of course numerous possible ways of implementing the gluing as long as the skipping of one or several continuous operations results in a loss of consistency.

IV. Conclusion

This paper presented an efficient method for preventing skipping attacks. The overhead induced by the countermeasure is minimal and does not impact the overall performance of the computation. It is generic and applies to any exponentiation algorithm.

References

[1] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of eliminating errors in cryptographic computations," *Journal of Cryptology*, vol. 14, no. 2, pp. 101–119, 2001, extended abstract in Proc. of EUROCRYPT '97.

[2] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] J.-M. Schmidt and C. Herbst, "A practical fault attack on square and multiply," in *5th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2008)*, L. Breveglieri *et al.*, Eds. IEEE Press, 2008, pp. 53–58.

[4] J.-J. Quisquater and C. Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," *Electronics Letters*, vol. 18, no. 21, pp. 120–126, 1982.

[5] M. Joye, A. K. Lenstra, and J.-J. Quisquater, "Chinese remaindering based cryptosystems in the presence of faults," *Journal of Cryptology*, vol. 12, no. 4, pp. 241–245, 1999.

[6] A. Shamir, "How to check modular exponentiation," Presented at the rump session of EUROCRYPT'97, Konstanz, Germany, May 13, 1997.

[7] M. Joye, P. Paillier, and S.-M. Yen, "Secure evaluation of modular functions," in *2001 International Workshop on Cryptology and Network Security*, R. J. Hwang and C. K. Wu, Eds., Taipei, Taiwan, Sep. 2001, pp. 227–229.

[8] D. Vigilant, "RSA with CRT: A new cost-effective solution to thwart fault attacks," in *Cryptographic Hardware and Embedded Systems – CHES 2008*, ser. Lecture Notes in Computer Science, E. Oswald and P. Rohatgi, Eds., vol. 5154. Springer, 2008, pp. 230–145.

[9] J. Blömer, M. Otto, and Jean-Pierre-Seifert, "Sign change fault attacks on elliptic curve cryptosystems," in *Fault Diagnosis and Tolerance in Cryptography*, ser. Lecture Notes in Computer Science, L. Breveglieri *et al.*, Eds., vol. 4236. Springer, 2006, pp. 36–52.

[10] M. Joye, "Protecting RSA against faults: The embedding method," in *6th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2009)*, L. Breveglieri *et al.*, Eds. IEEE Press, 2009, pp. 41–45.

[11] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings the IEEE*, vol. 94, no. 2, pp. 370–382, 2006, earlier version in Proc. of FDTC 2004.

[12] C. Giraud and H. Thiebeauld, "A survey on fault attacks," in *Smart Card Research and Advanced Applications VI (CARDIS 2004)*, J.-J. Quisquater *et al.*, Eds. Kluwer, 2004, pp. 159–176.

[13] C. H. Kim and J.-J. Quisquater, "Fault attacks for CRT-based RSA: New attacks, new results, and new countermeasures," in *Information Security Theory and Practices*, ser. Lecture Notes in Computer Science, D. Sauveron *et al.*, Eds., vol. 4462. Springer, 2007, pp. 215–228.

[14] J.-M. Schmidt and M. Medwed, "A fault attack on ECDSA," in *6th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2009)*, L. Breveglieri *et al.*, Eds. IEEE Press, 2009, pp. 93–99.

[15] N. A. Howgrave-Graham and N. P. Smart, "Lattice attacks on digital signature schemes," *Designs, Codes and Cryptography*, vol. 23, no. 3, pp. 283–290, 2001.

[16] P. Q. Nguyen and I. E. Shparlinski, "The insecurity of the Digital Signature Algorithm with partially known nonces," *Journal of Cryptology*, vol. 15, no. 3, pp. 151–176, 2002.