Attacking Right-to-Left Modular Exponentiation with Timely Random Faults*

Michele Boreale

Dipartimento di Sistemi e Informatica
Università di Firenze

Abstract. We show that timely induction of random failures can potentially be used to mount very cost effective attacks against smartcards deploying cryptographic schemes based on (right-to-left) modular exponentiation. We introduce a model where an external perturbation, or *glitch*, may cause a single modular multiplication to produce a truly random result. Based on this assumption, we present a probabilistic attack against the implemented cryptosystem. Under reasonable assumptions, we prove that using a single faulty signature the attack recovers a target bit of the secret exponent with an error probability bounded by $\frac{3}{7}$. We show the attack is effective even in the presence of message blinding.

Keywords: fault-based cryptanalysis, smartcards, public-key cryptosystems.

1 Introduction

In the past decade, a variety of potential attacks against supposedly tamper-proof devices have been put forward. Many of these attacks exploit side-channel information, such as that provided by timing analysis [11], differential power analysis [12], or computation faults [3,2,9,8,13].

In this paper, we focus on attacks against implementations of public-key cryptosystems based on modular exponentiation, such as RSA, El-Gamal and Diffie-Hellman. The Bellcore attack [9] revealed that induction of random faults in a device implementing RSA decryption with the Chinese Remainder Theorem (CRT) optimization could lead to disclosure of the key material.

Subsequent works have extended fault-analysis beyond CRT-based exponentiation. While revealing many potential weaknesses, these extensions have often been regarded as too idealized [1]. The original Bellcore attack just made use of one random computation fault. Subsequent models typically assumed the ability of the attacker to selectively alter the content of data registers, like flipping a few individual bits of the exponent [5], or modifying a segment of a register during the execution of a modular multiplication (e.g. the *safe errors* of [17]).

In the present paper we consider a model where truly random, hence "practical", computation faults are combined with a simple form of timing control. As pointed out

* Author's address: Dipartimento di Sistemi e Informatica, Viale Morgagni 65, I–50134 Firenze, Italy. Email: boreale@dsi.unifi.it. Work partially supported by the EU within the FET-GC2 initiative, project SENSORIA, and by University of Firenze, projects "ex-60%".

L. Breveglieri et al. (Eds.): FDTC 2006, LNCS 4236, pp. 24–35, 2006.

by several works [4,6,15], it is relatively simple to induce random computational errors in smartcards using *glitch*-based techniques. A glitch is an external perturbation, like a rapid variation in the clock frequency or power supply voltage, which causes a malfunction of the device. The effect of a glitch could be having a few instructions skipped or misinterpreted by the processor. The induced error is transient in the sense that the device will generally resume its correct functioning some μ-seconds after the glitch, with possibly the only observable effect of data corruption in some register. To quote Bar-El *et al.* [6], who have experimented using this technique: for a certain set of experiments, *"the outcome was that the value of the data could be corrupted, while the interpretation of instruction was left unchanged."*. According to [6], this method is widely researched and practiced behind closed doors by the smartcard industry. An alternative to this technique is optical fault induction, presented by Skorobogatov and Anderson in [15].

Given these premises, we can formulate our basic assumption as follows: a glitch applied during the execution of a modular multiplication $A \leftarrow B \cdot C \bmod n$ will result in a random value to be written into register A. This assumption seems reasonable, as execution of a modular multiplication provides a time window wide enough to allow a processor to resume its correct functioning after the glitch and before the next operation. Another relevant assumption we make is that the attacker has a control on the timing of the device that is fine enough to allow the choice of an appropriate instant in time for applying the glitch. This assumption (already present in other works on fault analysis) is justified by the circumstance that the clock signal is supplied to the device by an external card reader, which is presumably controlled by the attacker. In any case, we will show that precision in timing control can be traded off with success probability of the attack.

The basic idea is of the attack is easily explained. We focus on the right-to-left binary exponentiation algorithm (see e.g. [10]). For the purpose of illustration, suppose that the device implements the RSA signature scheme with secret exponent d and modulus n, and suppose for simplicity that the message to be signed is a quadratic residue mod n. Assuming the attacker has already determined the $i-1$ least significant bits of d, he can determine the initial instant in time of the i^{th} iteration (the one dealing with the i^{th} bit of d), and apply a glitch during the squaring operation that immediately precedes this iteration. As a result, a random r will be written in a certain register in place of the squaring correct result. Then, if bit d_i is set, the attacker will observe a faulty signature of the form $r \cdot C^2$, otherwise the observed faulty signature will be of the form C^2, for some C. With high probability, the attacker can tell these two cases apart by computing the Jacobi symbol of the faulty signature, thus determining the i^{th} bit of the exponent.

The rest of the paper is organized as follows. In Section 2 some preliminary notions are recalled. Section 3 introduce the basic model, where the attacker has a complete control on timing (the multiplication time is constant and known to the attacker, time due to control flow instructions is ignored). Section 4 presents the attack based on this model as a probabilistic algorithm. The attack is presented in detail for the case of a RSA modulus; the obvious modifications for a prime modulus are outlined. The results of some software simulations are also discussed. Section 5 extends the model and the attack to the case where time is randomized, possibly meaning partial control of the

attacker on timing. A few software countermeasures are discussed in Section 6; the technique of message blinding is shown to be not effective against the attack. Some concluding remarks and lines for further research are discussed in Section 7. Details of proofs have been confined to Appendix A.

2 Preliminary Notions

Recall (see [16]) that for a given prime p, x is a *quadratic residue* mod p if $\gcd(x,p)=1$ and $x=y^2 \bmod p$ for some y. If $\gcd(x,p)=1$ and x is not a quadratic residue mod p, then x is called *quadratic non-residue* mod p.

The *Jacobi symbol* $\left(\frac{m}{n}\right)$, for m and n integers, $n \geq 3$ odd, is defined as follows. If $n=p$ is prime (in this case one also speaks of *Legendre* symbol), then

$$\left(\frac{m}{p}\right) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } m \text{ is a quadratic residue mod } p \\ -1 & \text{if } m \text{ is a quadratic non-residue mod } p \\ 0 & \text{otherwise.} \end{cases}$$

If $k = p_1 \cdots p_l$, with p_j's primes not necessarily distinct, then $\left(\frac{m}{n}\right)$ is the product $\left(\frac{m}{p_1}\right) \cdots \left(\frac{m}{p_l}\right)$. It can be shown that

$$\text{If } m = m_1 \cdots m_h \mod n \text{ then } \left(\frac{m}{n}\right) = \left(\frac{m_1}{n}\right) \cdots \left(\frac{m_h}{n}\right). \tag{1}$$

It is well-known that $\left(\frac{m}{n}\right)$ can be efficiently computed without knowing the factorizations of m or n.

Suppose that $n = p \cdot q$, with p,q distinct primes. Since in $\mathbb{Z}_p$ there are exactly $(p-1)/2$ quadratic residues mod p and an equal number of quadratic non-residues mod p (similarly for q), using the Chinese Remainder Theorem and (1) above, it is immediate to check that

$$|\{r \in \mathbb{Z}_n \mid \left(\frac{r}{n}\right) = -1\}| = (p-1)\cdot(q-1)/2 = \phi(n)/2\ .$$

where $\phi(\cdot)$ is Euler's totient function.

3 The Model

Throughout the rest of the paper, unless otherwise stated, we assume a fixed modulus $n = p \cdot q$ (with p,q distinct secret primes) and a fixed document $M \in \mathbb{Z}_n$. The secret exponent $d \leq n$ has been chosen according to some possibly unknown probability distribution; in particular, we need not assume that d is an RSA exponent. The *signature* of M is $S = M^d \bmod n$. Both n and d are representable in l bits, in particular $d = (d_{l-1} \cdots d_1 d_0)_2$, where l need not to be known to the attacker.

In our scenario the attacker has got to know the device's PIN, or the device is not PIN operated. We also assume that the attacker controls the clock of the device, and can apply a glitch (e.g. through a rapid variation of clock frequency) during the computation

at an instant of his choice, and read the resulting value S'. The device can be queried in this way repeatedly. The rest of the section is devoted to a detailed description of algorithmic, timing and failure assumptions, and of faulty computations that can be induced by exploiting these assumptions.

Assumptions. We assume the device implements the *right-to-left* exponentiation algorithm (Figure 1). The algorithm uses two variables w and z, viewed as (physical or logical) s-bit registers with $s \geq l$. The value returned by the algorithm is the final content of register w, that is the (correct) signature, $S = M^d \bmod n$ (see below).

Input: M
Output: $S = M^d \bmod n$

1 $w \leftarrow 1$
2 $z \leftarrow M$
3 **for** $j = 0 \ldots l-1$ **do**
4 **if** $d_j = 1$ **then** $w \leftarrow w \cdot z \bmod n$
5 $z \leftarrow z \cdot z \bmod n$
6 **end**
7 **return** w

Fig. 1. The right-to-left exponentiation algorithm

Concerning timing and failures, we make the following assumptions:

1. each modular multiplication/squaring operation takes a constant time, say δ clock cycles, and δ is a constant known to the attacker;
2. time taken by control-flow instructions is ignored, in other words, we view the algorithm as a sequence of modular multiplications, grouped for ease of reference into the l iterations or *phases* depicted in Figure 2. Each phase i takes either δ or 2δ cycles, depending on the value of d_i, $0 \leq i \leq l-1$;

$$
\begin{array}{rl}
\text{phase } 0 & \left[\begin{array}{l} \textbf{if } d_0 = 1 \textbf{ then } w \leftarrow w \cdot z \bmod n \\ z \leftarrow z \cdot z \bmod n \end{array}\right. \\
\text{phase } 1 & \left[\begin{array}{l} \textbf{if } d_1 = 1 \textbf{ then } w \leftarrow w \cdot z \bmod n \\ z \leftarrow z \cdot z \bmod n \end{array}\right. \\
 & \vdots \\
\text{phase } l-1 & \left[\begin{array}{l} \textbf{if } d_{l-1} = 1 \textbf{ then } w \leftarrow w \cdot z \bmod n \\ z \leftarrow z \cdot z \bmod n \end{array}\right.
\end{array}
$$

Fig. 2. The right-to-left exponentiation algorithm as a sequence of l phases

3. a glitch applied onto the device during the execution of a modular multiplication will result in a random value $r \in \mathbb{Z}_{2^s}$ to be written in the involved register (w or z), in place of the multiplication's correct result.

(We will discard conditions 1 and 2 in Section 5.) If we denote by T_i the first cycle of phase i in a correct computation (counting from $T_0 = 1$), then

$$T_i = \delta\big(i - 1 + \sum_{j=0}^{i-1} d_j\big) + 1 \quad 0 \le i \le l-1\ . \tag{2}$$

Faulty computations. Let us first analyze the use of variables w and z in a correct execution of the algorithm. Variable z is used to store successive squaring of M; more precisely, when entering phase i, z contains c_i, where:

$$c_i \stackrel{\text{def}}{=} M^{2^i} \bmod n \qquad 0 \le i \le l-1\ .$$

Variable w is used to store intermediate products of the c_i's; more precisely, when leaving phase i, w contains S_i, where:

$$S_i = (c_0)^{d_0} \cdot (c_1)^{d_1} \cdot \ldots \cdot (c_{i-1})^{d_{i-1}} \cdot (c_i)^{d_i} \bmod n \tag{3}$$

in particular at, the end of phase $l-1$, S will be obtained as the product:

$$S = (c_0)^{d_0} \cdot (c_1)^{d_1} \cdot \ldots \cdot (c_{l-2})^{d_{l-2}} \cdot (c_{l-1})^{d_{l-1}} \ = M^d \bmod n.$$

Suppose the bits of the exponent from d_0 to d_{i-1} have been determined, and that bit d_i must be determined, for some $0 < i \le l-1$; note that d_0 can easily be guessed/determined by other means (and in case d is a RSA exponent, one already knows that $d_0 = 1$). The attacker computes the first instant T_i of phase i using (2), and applies a glitch at time T, for some $T_i > T > T_i - \delta$. This glitch will affect a single operation, i.e. the squaring $z \leftarrow z \cdot z$ of phase $i-1$. As a consequence, a random value $r \in \mathbb{Z}_{2^s}$ will be written in register z at the end of phase $i-1$. Let us see how this fault affects the final result of the computation, the *faulty signature* S'. It is easy to see, relying on (3) or on Figure 2, that S' will be computed as:

$$S' = (c_0)^{d_0}(c_1)^{d_1} \cdots (c_{i-1})^{d_{i-1}} \cdot r^{d_i} \cdot (r^2)^{d_{i+1}} \cdots (r^{2^{l-i-1}})^{d_{l-1}} \bmod n\,. \tag{4}$$

It is convenient to sum up the above considerations in the definition below. We code up the faulty behavior of a device where the i^{th} bit is targeted as a random variable, assuming $d_0, \ldots, d_{i-1}$ have been determined and are fixed binary constants.

Definition 1. *Let $d_i, \ldots, d_{l-1}$ be binary random variables and r be a random variable uniformly distributed in $\mathbb{Z}_{2^s}$ and independent from $d_i, \ldots, d_{l-1}$. We denote by $S'(r, d_i, \ldots, d_{l-1})$ the random variable whose value is given by the* RHS *of (4).*

4 The Basic Attack

For the rest of the section, we fix i with $0 < i \le l-1$. The target of the attack will be bit d_i, assuming that bits from d_0 to d_{i-1} have been determined. We assume without loss of generality that:

$$\left(\frac{M}{n}\right) = 1\ .$$

We shall indicate below how to modify the attack if $\left(\frac{M}{n}\right) \neq 1$ (see Remark 1 below). Equation (4) allows an attacker to extract information about the d_i by computing the Jacobi symbol of S', i.e., by taking the result of the random variable J:

$$J \stackrel{\text{def}}{=} \left(\frac{S'}{n}\right).$$

All factors of S' different from r^{d_i} have Jacobi symbol $\neq -1$. Hence, if one gets $J = -1$, one can immediately conclude that $d_i = 1$ (by (1)). On the other hand, if one gets $J \neq -1$ then one concludes that d_i is *probably* 0 (in case $\left(\frac{M}{n}\right) = -1$, we should just reverse the role of '1' and '-1'). For ease of reference, we code the test just outlined as a random variable.

Definition 2 (a test for d_i). *The random variable* $\mathbf{A}$ *is defined as:*

$$\mathbf{A} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } J = -1 \\ 0 & \text{if } J \neq -1. \end{cases}$$

Remark 1. Suppose that $\left(\frac{M}{n}\right) = -1$. A moment's thought shows that the test $\mathbf{A}$ still works if $d_0 = 0$. If $d_0 = 1$, then we can make $\mathbf{A}$ work by modifying it as follows:

$$\mathbf{A} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } J = 1 \\ 0 & \text{if } J \neq 1. \end{cases}$$

That is, if $d_0 = 1$ and one gets $J \neq -1$ then from (4) one can immediately conclude that $d_i = 1$.

Of course, if $\left(\frac{M}{n}\right) = 0$ one can immediately factor n by computing $\gcd(M,n)$. From now onward, we shall assume without loss of generality that $\left(\frac{M}{n}\right) = 1$.

The analysis of the test $\mathbf{A}$ is straightforward. In the sequel, let $\alpha \stackrel{\text{def}}{=} \Pr[d_i = 1]$, and let the success probability of $\mathbf{A}$ be $\rho \stackrel{\text{def}}{=} \Pr[A = 1 \,|\, d_i = 1]$, where we stipulate that $\rho \stackrel{\text{def}}{=} 1$ if $\alpha = 0$. Finally let the error probability of $\mathbf{A}$ be $\varepsilon \stackrel{\text{def}}{=} \Pr[d_i = 1 | \mathbf{A} = 0]$.

Lemma 1. *It holds that:*

(a) $\Pr[d_i = 1 | J = -1] = 1$;
(b) $\rho = \Pr[J = -1 | d_i = 1] \geq \phi(n)/2^{s+1}$.

PROOF: See Appendix. □

The following theorem says that $\mathbf{A}$ may be viewed as a Monte-Carlo type probabilistic algorithm.

Theorem 1. *The random variable* $\mathbf{A}$ *is a* 1*-biased probabilistic test for* d_i*, more precisely:*

(a) $\Pr[d_i = 1 | \mathbf{A} = 1] = 1$;

(b) $\varepsilon = \frac{(1-\rho)\alpha}{(1-\rho)\alpha+1-\alpha} \leq \frac{(1-\frac{\phi(n)}{2^{s+1}})\alpha}{(1-\frac{\phi(n)}{2^{s+1}})\alpha+(1-\alpha)}$.

PROOF: Part (a) follows from Lemma 1(a). For part (b), we may assume $\alpha \neq 0, 1$, otherwise the wanted equality and inequality hold trivially. First observe that $\Pr[\mathbf{A} = 0 | d_i = 0] = 1 - \Pr[\mathbf{A} = 1 | d_i = 0] = 1$, by part (a). Then, observe that, by definition of $\mathbf{A}$, $\Pr[\mathbf{A} = 1 | d_i = 1] = \Pr[J = -1 | d_i = 1] = \rho$. Apply Bayes theorem to get:

$$\varepsilon = \Pr[d_i = 1 | \mathbf{A} = 0] = \frac{\Pr[\mathbf{A}=0|d_i=1]\cdot\Pr[d_i=1]}{\Pr[\mathbf{A}=0|d_i=1]\cdot\Pr[d_i=1]+\Pr[\mathbf{A}=0|d_i=0]\cdot\Pr[d_i=0]}$$

$$= \frac{(1-\rho)\alpha}{(1-\rho)\alpha+(1-\alpha)}.$$

The last expression is decreasing with respect to ρ in $[0, 1]$. By Lemma 1(b) we know that $\rho \geq \phi(n)/2^{s+1}$, whence the thesis. □

As usual, one can make the error probability arbitrarily small by repeating the test m times independently in succession, for a suitable m, for fixed values of $d_0, ..., d_{l-1}$. In this case, the error probability is bounded above by:

$$\frac{(1-\rho)^m\alpha}{(1-\rho)^m\alpha+1-\alpha}.$$

A more precise estimation of ε is obtained by making some further assumptions. In particular, it seems reasonable to assume $\alpha = 1/2$ (this is not exact if d is a RSA exponent, but seems a good approximation in practice). Let us say that $n = p \cdot q$ is *balanced* if p and q have the same size (an integer m has size t if $2^{t-1} \leq m < 2^t$). Finally, let us assume that size of n fits the size s of the registers.

Corollary 1. *If n is balanced and has size s and $\alpha = \frac{1}{2}$ then $\varepsilon \leq \frac{3}{7}$.*

PROOF: Since p and q have the same size, it must be $p, q > 2^{\frac{s-1}{2}}$. Easy calculations then yields $\phi(n)/2^{s+1} \geq 1/4$. When we substitute this value for $\phi(n)/2^{s+1}$ and $1/2$ for α in the upper bound for ε given in the previous theorem, we get the value $3/7$. □

Here is a small example to illustrate.

i	7	6	5	4	3	2	1	0
$\mathbf{S}'$, $\mathbf{J}$	44, 1	58, -1	11, 1	86, 1	120, 0	43, -1	34, 1	-
	44, 1		106, 1	113, 1	77, 1		100, 1	-
	44, 1		35, 1	79, 1	5, 1		29, 1	-
	44, 1		43, -1	59, -1	92, 1		53, -1	-
$\mathbf{d_i}$	0	1	1	1	0	1	1	1

Fig. 3. An attack on the exponent $d = 119 = (01110111)_2$ with $n = 141$

Example 1. Suppose that $n = 141 = 3 \cdot 47$ and $M = 23$. The bits of the exponent d are determined in $l = 8$ successive stages, as in illustrated in Figure 3 (the value of l is not known in advance), starting from the least significant bit d_0 which is guessed to be 1. For each stage, the test is repeated at most $m = 4$ times independently. At each stage, the glitch time is given by $T = T_i - \varepsilon$ for some $0 < \varepsilon < \delta$. In conclusion, $d = (01110111)_2 = 119$. Of course, on $J = 0$ we could have factored the modulus right away. Also note that, in the last column, 44 is the correct value of $M^d \bmod n$: the squaring in the last but one iteration has no effect on the final result, as $d_{l-1} = 0$.

Remark 2 (software simulations). In the hypotheses of the corollary above, to obtain an error probability less say, than 2^{-10}, one may have to run the test up to $m = 25$ times independently. In practice, software simulations have shown that a bit less than 5000 queries (=faulty signatures) are sufficient to recover a RSA-768 key in about 70% of the cases. Considering a realistic time of 300 ms per query, and ignoring the time taken by a common PC to perform the test, this means that about 25 minutes are enough to recover such a key with a success probability of 0.7.

Remark 3 (Discrete log cryptosystems). The attack presented in this section can be repeated essentially unchanged when the modulus is a prime p. In this case, the success probability $\rho = \Pr[J = -1 \,|\, d_i = 1]$ can be lower-bounded by $|\phi(p)|/2^{s+1} = (p-1)/2^{s+1}$. If the size of p is $l = s$, then again $\rho \geq 1/4$ and $\varepsilon \leq 3/7$. Thus, in principle, in both El-Gamal decryption and Diffie-Hellman key-exchange an attacker might target and recover the secret exponent.

5 Randomized Time

We discard the assumption that all modular multiplications in the algorithm take the same known constant time δ. We represent multiplication times as random variables, possibly absorbing the time taken by control flow instructions. Times might change from an execution to the next, depending e.g. on instructions schedule, random delays or blinding of the argument. Or simply the randomness might represent the attacker's incomplete knowledge about the timing of the device (i.e. initial instant of each phase).

The first instant of phase i is given by the random variable

$$T_i = \sum_{j=0}^{i-1} (d_j \cdot \mu_j + \nu_j) + 1 \quad 0 \leq i \leq l-1$$

where for $0 \leq j \leq i-1$: d_j's are known values and μ_j's and ν_j's are continuous random variables, which, following [11], we assume to be normally distributed, with known variance and mean. We also assume that all these random variables (μ_j, ν_j's) are pairwise independent, and independent from d_i as a random variable. The model of the device (Definition 1) is modified as expected: S' yields the RHS of (4) whenever the glitch time T is such that $T_i > T > T_i - \nu_{i-1}$, for $0 < i \leq i-1$. Now, the midpoint in time of the squaring operation at phase $i-1$ is given by $\tau \stackrel{\text{def}}{=} T_{i-1} + d_{i-1}\mu_{i-1} + \nu_{i-1}/2$. We take the glitch time T to be the expectation:

$$T \stackrel{\text{def}}{=} \mathrm{E}[\tau] \,.$$

The definition of J and $\mathbf{A}$ remain unchanged. As we show below, with these definitions $\mathbf{A}$ yields a 2-sided probabilistic algorithm. Let $\gamma > 0$ be half the minimal duration of the squaring at phase $i-1$, i.e. take the supremum of all γ s.t. $\Pr[\nu_{i-1} < 2\gamma] = 0$, and let $\Gamma \stackrel{\text{def}}{=} \Pr[|T-\tau| < \gamma]$: this value can be computed exactly as τ is normally distributed with mean T and standard deviation $\sigma \stackrel{\text{def}}{=} \sum_{j=0}^{i-2}(d_j \mathrm{var}(\mu_j) + \mathrm{var}(\nu_j)) + d_{i-1}\mathrm{var}(\mu_{i-1}) + \frac{1}{4}\mathrm{var}(\nu_{i-1})$. Recall that $\alpha = \Pr[d_i = 1]$. The following result is proven by noting that if τ falls within γ of the glitch time T, then the glitch will be 'correct', i.e. it will affect the squaring in phase $i-1$.

Theorem 2. *The random variable* $\mathbf{A}$ *is a 2-sided probabilistic algorithm for bit* d_i*. In particular:*

a) the success probability for 1 is: $\rho \stackrel{\text{def}}{=} \Pr[\mathbf{A} = 1 | d_i = 1] \geq \frac{\phi(n)}{2^{s+1}} \cdot \Gamma$*, with* $\rho \stackrel{\text{def}}{=} 1$ *if* $\alpha = 0$*;*

b) the error probability for 1 is: $\varepsilon_1 \stackrel{\text{def}}{=} \Pr[d = 1 | \mathbf{A} = 0] \leq \frac{(1-\Gamma)(1-\alpha)}{(1-\Gamma)(1-\alpha)+\rho\alpha}$*;*

c) the error probability for 0 is: $\varepsilon_0 \stackrel{\text{def}}{=} \Pr[d = 0 | \mathbf{A} = 1] \leq \frac{(1-\rho)\alpha}{(1-\rho)\alpha+\Gamma(1-\alpha)}$.

The expressions for $\varepsilon_0, \varepsilon_1$ *are monotonically decreasing w.r.t.* $\rho \in [0,1]$.

PROOF: See Appendix □

Given that $\mathbf{A}$ is a two-sided probabilistic test, one has to run the test m times independently with fixed values of the exponent bits and take the majority of the outcomes to have a reliable result. Note that for m independent iterations of $\mathbf{A}$, with fixed values of the exponent bits, the error probabilities for 1 and 0 can be lower-bounded respectively as:

$$\frac{(1-\Gamma)^m(1-\alpha)}{(1-\Gamma)^m(1-\alpha)+\rho^m\alpha} \quad \text{and} \quad \frac{(1-\rho)^m\alpha}{(1-\rho)^m\alpha+\Gamma^m(1-\alpha)} .$$

For the test to be useful, one has to make sure that the above values vanish as m grows. This is the case precisely when $\rho + \Gamma > 1$; by virtue of (a) above, this holds if $\Gamma > \frac{1}{1+\phi(n)/2^{s+1}}$.

As a general remark, the attack performs well in situations with a moderate variance of multiplication times, that is, when timing attacks are more difficult to mount. The following example provides some numerical evidence that for typical values of Γ the randomized version of the attack is feasible.

Example 2. For ease of reference, we use numerical data drawn from Kocher's original paper [11]. The following figures refer to time measurements (in μ-seconds) of actual modular multiplications executed during modular exponentiations. The random variables μ_j and ν_j's are all normally distributed with standard deviation $\sigma_m = 12.01$ and mean $t = 1167.8$. The minimal duration of a modular multiplication can be taken 1130, hence we set $\gamma = 565$. Suppose we target the 512^{th} bit of a secret exponent of size $l = 1024$ bit. Assuming, on average, that half of the bits from d_0 to d_{511} are set, we can compute the mean of τ as $T = t(511 + \frac{1}{2} + 256) = 896286.5$ and its variance as $\sigma^2 = \sigma_m^2(511 + \frac{1}{4} + 256) = 110668.2167$. These values gives (here

$\Phi(\cdot)$ denotes the cumulative distribution function of standard normal distribution): $\Gamma = \Phi(\gamma/\sigma) - \Phi(-\gamma/\sigma) \approx 0.9105$. Under the hypotheses of Corollary 1, we get

$$\rho \geq \frac{1}{4} \cdot \Gamma \approx 0.2275 \qquad \varepsilon_0 \leq 0.4590 \text{ and } \varepsilon_1 \leq 0.2825 .$$

If we want both error probabilities to decrease under, say, 2^{-10}, we may have to run the test up to 43 times independently.

6 Countermeasures

We discuss a few software countermeasures.

Blinding Exponentiation with *blinding* (Figure 4) is a common and effective technique to thwart attacks based on timing [11].

1 choose at random $v \in \mathbb{Z}_n^*$
2 $X \leftarrow Mv^e \bmod n$
3 $Y \leftarrow X^d \bmod n$
4 $S \leftarrow Yv^{-1} \bmod n$

Fig. 4. RSA with message blinding

It is easy to see that message blinding has no effect on our attack. Suppose the attacker's target is bit d_i. Given that the values v^e, $v^{-1} \bmod n$ are usually precomputed, the attacker can easily target the i^{th} bit during the exponentiation at step 3 and induce a faulty computation yielding Y' as a result (i.e. a faulty signature of Y, without blinding), hence getting from the device a faulty signature

$$S'' = Y'v^{-1} \bmod n .$$

Let S' be the faulty signature one would obtain by targeting the i^{th} bit in the case with no blinding – but with the same choice of the random $r \in \mathbb{Z}_{2^s}$. Let $\overline{c}_i \stackrel{\text{def}}{=} (v^e)^{2^i}$, for $i = 0, \ldots, l-1$. It is easy to see, relying on equation (4), that:

$$S'' = S' \cdot C \cdot v^{-1} \bmod n$$

where $C = (\overline{c}_0)^{d_0} \cdots (\overline{c}_{i-1})^{d_{i-1}}$. Noting that that e is odd we have:

$$\left(\frac{v^e}{n}\right) = \left(\frac{v}{n}\right) = \left(\frac{v^{-1}}{n}\right)$$

and since $d_0 = 1$, hence $\overline{c}_0 = v^e$, we get

$$\left(\frac{S''}{n}\right) = \left(\frac{S'}{n}\right) .$$

Effective countermeasures *Checking before output*, i.e. checking that $S^e = M \bmod n$, with e a RSA public exponent (see [9]), before transmitting the signature has been proposed to contrast fault attacks. This is feasible in case the public exponent e is small. In the case of a prime modulus p, a strategy suggested by Shamir [14] involves doing exponentiation twice, once mod p and once mod $p \cdot r$, for r a 32-bit prime, and then comparing the results. *Random delays* (see [11]) have been proposed as a countermeasure against timing analysis. An alternative form of blinding, also proposed in [11], is blinding of the exponent, which consists in summing a quantity $k\phi(n)$, with k random, to the exponent d before performing modular exponentiation. Adoption of one of above listed methods appears to thwart our attacks.

7 Conclusions

We have demonstrated that fault analysis can be combined with timing control to potentially get effective cryptanalysis of cryptographic schemes implemented using the (right-to-left) modular exponentiation algorithm. Our model is based on random, transient computation faults, that appear to be easier to induce than faults based on modifying individual bits of data registers.

At the moment it is not clear how to extend the attack presented here to the left-to-right version of the exponentiation algorithm. Indeed, one can easily show that, in the case of a prime modulus p, a straightforward extension of this attacks based on detecting 2^i-th power mod p permits to recover the k least significant bits of the exponent, where k is the exponent of 2 in the factorization of $p-1$: however, these bits are already known to be "easy" to recover.

Also, one wonders whether an analog of the present attack might work against ECC schemes that rely on "double and add" algorithms, perhaps along the lines of the attacks presented in [7]. These extensions will be the subject of further study.

References

1. R.J. Anderson, M. Bond, J. Clulow, S. Skorobogatov. Cryptographic processors – a survey, Technical Report UCAM-CL-TR-641, University of Cambridge, Computer Laboratory, August 2005.
2. R.J. Anderson, M.J.Kuhn, Tamper resistance – a cautionary note. *The second USENIX Workshop on Electronic Commerce proceedings*, Nov. 1996.
3. R.J. Anderson, M.J. Kuhn, Low cost attacks on tamper-resistant devices, *Security protocols, 5th International Workshop*, Paris, 1997.
4. C. Aumüller, P. Bier, P. Hofreiter, W. Fischer and J.- P. Seifert. Fault attacks on RSA with CRT: Concrete Results and Practical Countermeasures, *Cryptology ePrint Archive: Report 2002/073*.
5. F.Bao, R.H.Deng, Y.Han, A.Jeng, A.D.Nirasimhalu, T.Ngair. Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. In *Proc. of the 5th Workshop on Secure Protocols*, LNCS 1361, Springer, 1997.
6. H.Bar-El, H.Choukri, D.Naccache, M.Tunstall, C.Whelan. The Sorcerer's Apprentice Guide to Fault Attacks, In *Workshop on Fault Detection and Tolerance in Cryptography*, Florence, 2004. Also in *Cryptology ePrint Archive: Report 2004/100*, 2004.
7. I. Biehl, B. Meyer, V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems, In *Advances in Cryptology - Crypto 2000*, LNCS 1880, Ed. Mihir Bellare, Springer, 2000.
8. E.Biham, A.Shamir. Differential fault analysis of secret key cryptosystem, In *Advances in Cryptology, CRYPTO '97*, LNCS 1294, Springer, 1997.
9. D.Boneh, R.A.DeMillo, R.J.Lipton. On the importance of checking cryptographic protocols for faults, *Journal of Cryptology*, 14(2), Springer, 2001.
10. D.E.Knuth. *The art of computer programming vol.2, Seminumerical algorithms*. Addison Wesley, third edition, 1997.
11. P.Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, *Advances in Cryptology-CRYPTO'96*, LNCS 1109, Springer, 1996.
12. P.Kocher, J.Jaffe, B.Jun. Differential Power Analysis, In *Advances in Cryptology, CRYPTO'99*, LNCS 1294, Springer, 1999.

13. J.J.Quisquater, G.Piret. A Differential Fault Attack Technique Against SPN Structures, with Application to the AES and KHAZAD, In *Fifth International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003)*, LNCS 2779, Springer, 2003.
14. A. Shamir. How to check modular exponentiation. Presented at *EUROCRYPT'97* rump session, Konstanz, May 1997.
15. S. Skorobogatov, R. Aderson. Optical Fault Induction Attacks. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, LNCS 2523, Springer, 2002.
16. D.R.Stinson. *Cryptography: Theory and Practice*. CRC Press, second edition, 2002.
17. S-M. Yen, M.Joye. Checking before output may not be enough against fault-based cryptanalysis. In *IEEE Transactions on Computers*, 49(9), 2000.

A Proofs

PROOF OF LEMMA 1: Part (a) follows from the discussion immediately preceding the statement of the lemma. For part (b), suppose that $d_i = 1$. Then, by definition of S' and J and by the property of the Jacobi symbol (1), $J = \left(\frac{r}{n}\right)$ with r chosen at random in $\mathbb{Z}_{2^s}$. Thus

$$\rho = |\{r \in \mathbb{Z}_{2^s} : \left(\frac{r}{n}\right) = -1\}|/2^s \;\geq\; |\{r \in \mathbb{Z}_{2^n} : \left(\frac{r}{n}\right) = -1\}|/2^s$$

since $n \leq 2^s$. But, as noted in Section 2, the set that appears at the numerator in the last expression has cardinality $\phi(n)/2$. □

PROOF OF THEOREM 2: Concerning (a), one can lower bound the success probability $\rho = \Pr[J = -1 | d_i = 1]$ by noting that if τ falls within γ of the glitch time T, then the glitch will be 'correct', i.e. it will affect the squaring in phase $i-1$. Therefore

$$\Pr[J = -1 | d_i = 1, |T - \tau| < \gamma] \geq \frac{\phi(n)}{2^{s+1}} .$$

By the independence of d_i and τ, we have:

$$\begin{aligned}\rho &= \Pr[J = -1 \,|\, d_i = 1, |T-\tau| < \gamma] \cdot \Gamma + \Pr[J = -1 | d_i = 1, |T-\tau| \geq \gamma] \cdot (1-\Gamma) \\ &\geq \Pr[J = -1 \,|\, d_i = 1, |T-\tau| < \gamma] \cdot \Gamma \\ &\geq \tfrac{\phi(n)}{2^{s+1}} \cdot \Gamma \ .\end{aligned}$$

The upper bounds for ε_0 and ε_1 follow using Bayes theorem. In particular, for ε_1 we use the lower-bound:

$$\begin{aligned}\Pr[\mathbf{A} = 0 \,|\, d_i = 0] &= \Pr[J \neq -1 \,|\, d_i = 0, |T-\tau| < \gamma] \cdot \Gamma \ + \\ &\quad \Pr[J \neq -1 | d_i = 0, |T-\tau| \geq \gamma] \cdot (1-\Gamma) \\ &= 1 \cdot \Gamma + (\cdots) \\ &\geq \Gamma \ .\end{aligned}$$

It is immediate to check that the given bounds are monotonic decreasing in ρ. □