Vol. 45 No. 8 IPSJ Journal Aug. 2004

Regular Paper

Timing Attacks against a Parallelized RSA Implementation

Yasuyuki Sakai[†] and Kouichi Sakurai[††]

We discuss timing attacks against RSA using the parallel modular exponentiation. We describe a parallel algorithm for the modular exponentiation $y \equiv x^d \bmod n$. Then timing attacks against the parallel implementation are demonstrated. When we have two processors, which perform the modular exponentiation, an exponent d is scattered into two partial exponents $d^{(0)}$ and $d^{(1)}$, where $d^{(0)}$ and $d^{(1)}$ are derived by bitwise AND operation from d such that $d^{(0)} = d \wedge (0101 \cdots 01)_2$ and $d^{(1)} = d \wedge (1010 \cdots 10)_2$. Two partial modular exponentiations $y_0 \equiv x^{d^{(0)}} \bmod n$ and $y_1 \equiv x^{d^{(1)}} \bmod n$ are performed in parallel using two processors. Then we can obtain y by computing $y \equiv y_0 y_1 \bmod n$. In general, the hamming weights of $d^{(0)}$ and $d^{(1)}$ are smaller than that of d. Thus a fast computation of the modular exponentiation $y \equiv x^d \bmod n$ can be achieved. We describe a timing attack against RSA with and without the Chinese Remainder Theorem (CRT) using the parallel modular exponentiation. Both the secret exponents $d_p = d \bmod p-1$ and $d_q = d \bmod q-1$, where $n = pq$, are scattered into two partial exponents, respectively. We show that timing attacks are still applicable to that case.

1. Introduction

Embedded systems are one of the major application fields of cryptographic algorithms, and may contain sensitive data, such as RSA private key. Some implementations of cryptographic algorithms often leak "*side channel information.*" Side channel information includes power consumption, electromagnetic radiation and timing to process. Side channel attacks, which use side channel information leaked from real implementation of cryptographic algorithms, were first introduced by Kocher [6),7)]. Side channel attacks can be often much more powerful than mathematical cryptanalysis. Thus, many literatures on side channel cryptanalysis have been published [5),9)~12)].

In this paper, we focus on timing attacks against RSA without Chinese Remainder Theorem (CRT) using the parallel algorithm for the modular exponentiation which will be described in later sections. We also discuss timing attacks against RSA with CRT. The running time of a cryptographic device can constitute an information channel, providing the attacker with valuable information on the secret parameters involved. The timing attack is the attack to determine a secret parameter from differences between running times needed for various input values. The timing attack was first introduced by Kocher at Crypto 96 [7)]. He showed that a careful statistical analysis could lead to the total recovery of secret parameters.

Dhem, et al. showed a successful timing attack against the modular exponentiation without CRT [2)]. Schindler, et al. proposed further improved timing attacks [10)]. These attacks assume that implementations do not use CRT for the modular exponentiation. Schindler presented a powerful timing attack against an implementation using CRT [9)]. The exponent of the modular exponentiation can be a secret parameter in RSA-based cryptosystems. Thus we must implement the modular exponentiation very carefully.

As we mentioned before, our target of implementation is the parallel modular exponentiation algorithm. Garcia, et al. gave parallel algorithms for elliptic scalar multiplication [4)]. When we have plural processors, which can perform elliptic scalar multiplications, we scatter a given scalar into several sub-scalars. Partial scalar multiplication are carried out with subscalar in parallel on plural processors. Then scalar multiplication can be computed with the outputs from the processors. Non-zero bits in a scalar representation can be distributed into plural processors. Thus this parallel algorithm provides an efficient implementation. This algorithm for the elliptic scalar multiplication can be easily extended to the modular exponentia-

† Information Technology R&D Center, Mitsubishi Electric Corporation
†† Kyushu University

☆ This work was partially presented at Applied Cryptography and Network Security (ACNS 2003).

tion.

In this paper, we consider vulnerability of RSA implementation with and without CRT using the parallel exponentiation. We describe timing attacks against those implementations. Our attack is an extension of Dhem's method [2)] or Schindler's method [9)]. When RSA-based cryptosystems are implemented with CRT using the parallel exponentiation, the both secret exponents $d_p = d \bmod p-1$ and $d_q = d \bmod q-1$, where $n = pq$, are scattered into two partial exponents, respectively. We show that the timing attack is still applicable to that case.

If plural processors for the modular exponentiation can be mounted, the parallel exponentiation discussed in this paper can achieve significant speed-up of RSA cryptosystems. Typical environment for the parallel exponentiation can be embedded systems such as network devices and cryptographic boards.

2. Parallel Exponentiation

In this section we describe the parallel algorithm for the modular exponentiation $y \equiv x^d \bmod n$. Garcia, et al. [4)] gave a parallel algorithm for elliptic scalar multiplication, which is the basis of the parallel exponentiation.

2.1 The Algorithm of Parallel Exponentiation

Assume that a t-bit non negative integer d has a binary representation $d = (d_{t-1} \cdots d_1 d_0), d_i \in \{0, 1\}$. We divide the binary representation of d in $\lceil t/b \rceil$ blocks of b bits each one and then we scatter d into $d^{(0)}, d^{(1)}, \cdots, d^{(b-1)}$. The binary representation of the block $d^{(i)}$ is formed by $\lceil t/b \rceil$ blocks of b bits set to 0, except for the i-th bit, which has the same value that the i-th bit of the corresponding block of the binary representation of d. Thus we can define $d^{(i)}$, for $i = 0, \cdots, b-1$, as follows.

$$\begin{aligned} d^{(0)} &= d_0 + d_b 2^b + d_{2b} 2^{2b} \\ &\quad + \cdots + d_{(\lceil \frac{t}{b} \rceil - 1)b} 2^{(\lceil \frac{t}{b} \rceil - 1)b} \\ d^{(1)} &= d_1 2 + d_{b+1} 2^{b+1} + d_{2b+1} 2^{2b+1} \\ &\quad + \cdots + d_{(\lceil \frac{t}{b} \rceil - 1)b+1} 2^{(\lceil \frac{t}{b} \rceil - 1)b+1} \\ &\vdots \\ d^{(b-1)} &= d_{b-1} 2^{b-1} + d_{2b-1} 2^{2b-1} \\ &\quad + d_{3b-1} 2^{3b-1} + \cdots + d_{b\lceil \frac{t}{b} \rceil - 1} 2^{b\lceil \frac{t}{b} \rceil - 1} \end{aligned}$$

That is

$$d^{(i)} = \sum_{j=0}^{\lceil \frac{t}{b} \rceil - 1} d_{jb+i} 2^{jb+i}$$

for $i = 0, 1, \cdots, b-1$, where we consider some padding bits $b_l = 0$ for $t-1 < l < b\lceil t/b \rceil - 1$.

Clearly we have

$$d = d^{(0)} + d^{(1)} + \cdots + d^{(b-1)}.$$

Thus we can compute the modular exponentiation $y \equiv x^d \bmod n$ by the following equation.

$$x^d \bmod n = \Big((x^{d^{(0)}} \bmod n)(x^{d^{(1)}} \bmod n) \cdots (x^{d^{(b-1)}} \bmod n) \Big) \bmod n$$

The exponent set $\{d^{(0)}, d^{(1)}, \cdots, d^{(b-1)}\}$ can be easily obtained from d by bitwise AND operation with the appropriate mask.

We show the algorithm for the parallel modular exponentiation in Algorithm 1.

Algorithm 1 Parallel modular exponentiation

```
Input  $x, n, d$
Output  $y \equiv x^d \bmod n$
1. Bits scattering:
     Compute the set $\left\{d^{(0)}, d^{(1)}, \cdots, d^{(b-1)}\right\}$
2. Parallel computation:
     Using $b$ processors, compute in parallel
   the exponentiations
     $y \equiv \Big( t(x^{d^{(0)}} \bmod n)(x^{d^{(1)}} \bmod n) \cdots (x^{d^{(b-1)}} \bmod n) \Big) \bmod n$
3. Return $y$
```

Notice that in the case that block-length b equals to one, Algorithm 1 is equivalent to the simple modular exponentiation.

2.2 Computational Complexity

Let us discuss the computational complexity of the parallel modular exponentiation algorithm 1. We use the words "computational complexity" in the sense of the running time of the total (not partial) modular exponentiation. The running time can be evaluated by the number of modular multiplication $\mathcal{M}$ and modular squaring $\mathcal{S}$ required. Assume d has the bitlength t and has the hamming weight $H(d)$. Then we have the following lemma [4)].

Lemma 1 *Assume each individual exponentiation is performed by the binary method. The parallel exponentiation Algorithm 1 has the* *computational complexity, on the best case,*

$$(t-1)\mathcal{S} + (\lceil H(d)/b \rceil + b - 1)\mathcal{M}$$

and, on the worst case,

$$(t-1)\mathcal{S} + (H(d) - 1)\mathcal{M}$$

where $\mathcal{M}$ and $\mathcal{S}$ denote modular multiplication and modular squaring, respectively.

The computational complexity of the parallel exponentiation can be evaluated by the individual exponentiation which has the most expensive complexity. Since the most significant bit d_{t-1} is 1 by assumption, one of the individual exponentiation has to perform $t-1$ modular squarings.

On the best case, all the bits set to one in the binary representation of d are equally scattered among the exponent set $\{d^{(i)}\}$, so the computational cost is perfectly balanced on all the individual exponentiations. The worst case implies that there exists some index i, for $0 \le i \le b-1$, such that all of bit "1" are mapped to $d^{(i)}$ and then $x^{d^{(i)}} \bmod n = x^d \bmod n$. In such the case, the computational cost is the same as the traditional binary exponentiation.

The computational cost for computing the set $\{d^{(0)}, d^{(1)}, \cdots, d^{(b-1)}\}$ from d can be ignored, because we can obtain the set by simple bitwise AND operations.

Remark 1 To achieve the computationally best case, we can implement a scattering function, rather than the simple AND operation, so that all the bits set to one are equally scattered among the exponent set.

3. Timing Attacks against RSA

In this section we give brief description of timing attacks against RSA, which were presented by Dhem, et al.[2] and by Schindler[9]. Those methods are attacks against RSA without CRT and with CRT, respectively.

3.1 Modular Exponentiation

Before we move on to timing attacks, let's have a brief description of the modular exponentiation. In this paper we will consider the binary representation for a given exponent and the left-to-right binary (square-and-multiply) method for the modular exponentiation. The left-to-right binary method can be described as Algorithm 2.

Algorithm 2 Left-to-right binary method of the modular exponentiation

```
Input  $x, n, d$, where $d = (d_{t-1}d_{t-2}\cdots d_0)$,
       $d_i \in \{0,1\}$ for $0 \le i \le t-2$ and $d_{t-1} = 1$
Output  $y \equiv x^d \bmod n$
1. $y \leftarrow x$
2. for $i$ from $t-2$ downto 0
        $y \leftarrow y^2 \bmod n$
        if $d_i = 1$ then $y \leftarrow y \cdot x \bmod n$
   endfor
3. return $y$
```

Timing attacks will be demonstrated on an implementation with the following algorithms.

- The modular exponentiation is performed by Algorithm 2.
- Modular multiplication and squaring are performed with Montgomery's method[8].

In the Montgomery's method, when the intermediate value during the computation becomes larger than the modulus n, we have to perform "*extra reduction.*"

The goal for an attacker is to recover the exponent d, which can be a secret parameter of the decryption and the signature generation in RSA-based cryptosystems. The attacker determines a secret parameter d from differences between running times needed for various input values x.

3.2 Model of the Attacker

We assume that the attacker can be modeled as the following.

- The attacker has access to a device which performs the modular exponentiation with a secret exponent.
- The attacker can measure the running time of the modular exponentiation with various input x.
- The attacker knows the modulus n.
- The attacker has the knowledge of the implementation.

The attack algorithm will be developed with the assumption that the running times are reproducible, i.e., for fixed d and n the running time of the modular exponentiation $x^d \bmod n$ only depends on the base x but not on other influences.

3.3 Timing Attack against RSA without CRT

Dhem, et al. proposed a practical timing attack against an implementation of the modular exponentiation without CRT[2]. In this subsection we briefly explain their strategy. See Ref. 2) for details.

Assume again that a given exponent d has binary representation $d = (d_{t-1}\cdots d_1 d_0)$, where $d \in \{0,1\}, d_{t-1} = 1$. Their attack recovers the exponent bit by bit from d_{t-2} to the least significant bit d_0. Notice that the MSB d_{t-1} is always 1. We start by attacking d_{t-2}. When $d_{t-2} = 1$, at `Step 2` of Algorithm 2, modular multiplication with Montgomery's method has to be performed. For some input x, intermediate value can be larger than the modulus n, and then the extra reduction has to be performed. For the other input x, the extra reduction is not required. Let X be the set of inputs. We can define two subsets of inputs $X_1, X_2 \subset X$ as follows.

$$X_1 = \{x \in X | x \cdot x^2 \text{ has to be performed with extra reduction}\}$$
$$X_2 = \{x \in X | x \cdot x^2 \text{ can be performed without extra reduction}\}$$

If the value of d_{t-2} is 1, then we can expect that the running times for the inputs $x \in X_1$ to be slightly higher than the corresponding times for $x \in X_2$. On the other hand, if the actual value of d_{t-2} is 0, then the modular multiplication in `Step 2` will not be performed. In this case, for any input x, there is no reason that the extra reduction is induced. Therefore, the separation in two subsets should look random, and we should not observe any significant differences in the running time.

When the attacker wants to guess the bit d_{t-2}, he should take the strategy below.

Algorithm 3 Guessing d_{t-2}

```
1. Generating two subsets $X_1, X_2$:
   For various inputs $x$, the attacker does the following simulation with the knowledge of the implementation. At modular multiplication phase in Step 2 of Algorithm 2 with $i = t - 2$, if the extra reduction has to be performed, $x$ should be classified into $X_1$. Else if the extra reduction is not required, $x$ should be classified into $X_2$.
2. Measuring the running times:
   Using the device, on which a modular exponentiation is implemented, the attacker measures the running time of the modular exponentiation for $x \in X_1$ and $x \in X_2$.
3. Guessing $d_{t-2}$:
   The attacker does a statistical analysis on the difference of the running times between $x \in X_1$ and $x \in X_2$. Then he guesses $d_{t-2} = 0$ or 1.
```

Based on the time measurement, the attacker has to decide that the two subsets X_1 and X_2 are significantly different or not. Some statistical analysis can be of help. Possible use for statistics could be the mean value and χ^2 test. The attacker first guesses the bit d_{t-2} based on Algorithm 3. The same strategy can be applied continuously bit by bit from MSB to LSB. The attacker may recover the total secret exponent d.

There is a more subtle way to take advantage of our knowledge of Montgomery's method: instead of the multiplication phase, we could turn ourselves to the square phase at `Step 2` of Algorithm 2 [2)]. The same strategy as multiplication phase described before can be applicable.

3.4 Timing Attack against RSA with CRT

Schindler proposed an efficient timing attack against RSA with the Chinese Remainder Theorem (CRT) [9)]. Moreover Schindler, et al. proposed an improved error detection/correction strategies [10)]. In this subsection we give a brief description of the attack described in Ref. 9). The reader is referred to Ref. 9) for details.

The same model of the attack as defined in Section 3.2 can be applicable to the CRT case. Of course the attacker knows the modulus n, but does not know the prime factors p and q.

Let $n = pq$ be an RSA modulus, where p and q are distinct two primes and have the same bitlength. A modular exponentiation with CRT is performed as follows.

(1) $y_p \leftarrow x^{d_p} \bmod p$
(2) $y_q \leftarrow x^{d_q} \bmod q$
(3) *Re-combination*

where, $d_p = d \bmod p-1$ and $d_q = d \bmod q-1$.

In this paper, we assume that RSA computation is performed using CRT with Montgomery multiplication [8)]. The formulation of the attack make the following assumptions concerning the implementation.

- The modular exponentiation is carried out with simple binary method.
- Both Montgomery multiplications $\bmod p$ and $\bmod q$ use the same Montgomery constant R.
- Montgomery multiplications require time c if no extra reduction is needed and $c + c_{ER}$ otherwise.

Remark 2 *Schindler described a timing attack against RSA with CRT using advanced exponentiation algorithms such as the ary method* [9)].

We define the mapping $\mathbb{Z} \rightarrow \mathbb{Z}_p$ by $\mathtt{MONT}(u) := uR^{-1} \bmod p$. Schindler stated the following lemma [9)].

Lemma 2 (Schindler) *Let B denote a random variable being uniformly distributed on $\mathbb{Z}_p$.*

$$\begin{aligned} pr :&= Prob\left(\textit{extra reduction in } \texttt{MONT}(B^2)\right) \\ &= \frac{p}{3R} \end{aligned}$$

and, unless the ratio $R/gcd(R,(u \ mod \ p))$ is extremely small, also

$$\begin{aligned} pr(u) :&= Prob\,(\textit{extra reduction in} \\ &\quad \texttt{MONT}(uB)) \\ &= \frac{u(mod\, p)}{2R} \end{aligned}$$

for $u \in \mathbb{Z}_n$.

Remark 3 *Lemma 2 is not exact in a mathematical sense [9)]. The proof of Lemma 2 uses some heuristic arguments. The word "extremely" stated in Lemma 2 is an ambiguous expression. However, results from practical experiments by Schindler (thousands of pseudo-random factors and various moduli) match perfectly with Lemma 2. The details can be found in Ref. 9).*

For base uR^{-1} mod n hundreds of Montgomery multiplication have to be performed with factors $u(\text{mod } p)$ and $u(\text{mod } q)$. Theorem 2 says that the probability for an extra reduction within any of these multiplication is linear in the respective factor. Differences between running times required for two modular exponentiations result from different numbers of extra reductions within the respective modular multiplications and squarings. Moreover we can see from Lemma 2 that the expected number of extra reductions is discontinuous at each integer multiple of p or q.

Let $T(u)$ denote the running time of the modular exponentiation $(uR^{-1})^d$ mod n. For $u_1 < u_2$ with $u_2 - u_1 \ll p, q$ the time difference $T(u_2) - T(u_1)$ should reveal whether the interval $\{u_1+1, \cdots, u_2\}$ contains an integer multiple of at least one prime factor p, q or not. In the first case $T(u_1)$ should be significantly larger than $T(u_2)$ while in the second case both running times should approximately be equal.

Let $0 < u_1 < u_2 < n$ with $u_2 - u_1 < p, q$. The following three cases are possible. Case A: $\{u_1+1, \cdots, u_2\}$ does not contain a multiple of p or q. Case B: $\{u_1+1, \cdots, u_2\}$ contains a multiple of p or q but not of both. Case C: $\{u_1+1, \cdots, u_2\}$ contains a multiple of both p or q.

The running time for the input uR^{-1} mod n, denoted by $T(u)$, is interpreted as a "*realization*" of a random variable X_u [9)]. The expected difference $X_{u_2} - X_{u_1}$ depends essentially on the fact whether Case A, Case B or Case C is true:

$$E(X_{u_2} - X_{u_1}) \approx \begin{cases} 0 & \text{in Case A} \\ -\frac{c_{ER}}{8}\frac{\sqrt{n}}{R} & \text{in Case B} \\ -\frac{c_{ER}}{8}\frac{2\sqrt{n}}{R} & \text{in Case C} \end{cases}$$

See Ref. 9) for the detailed derivation of the above.

Let $\beta := \sqrt{n/R^2}$. We assume $p/R \approx \beta$ and $q/R \approx \beta$. The algorithm 4 below can be applicable to the RSA implementation with CRT using Montgomery multiplication [9)].

Algorithm 4 Timing attack against RSA with CRT using Montgomery multiplication (basic scheme)

```
Step 1. Choose an integer $u$ with $\beta R \leq$
  $u < n$ and set (e.g.) $\Delta \leftarrow 2^{-6}R$.
  $u_2 \leftarrow u$
  $u_1 \leftarrow u_2 - \Delta$
  while$\left(T(u_2)-T(u_1) > -c_{ER}\frac{\log_2(n)\beta}{16}\right)$ do:
      $u_2 \leftarrow u_1$
      $u_1 \leftarrow u_1 - \Delta$
  end of while
Step 2. while$(u_2 - u_1 > 1000)$ do:
      $u_3 \leftarrow \lfloor (u_1 + u_2)/2 \rfloor$
      if$\left(T(u_2) - T(u_3) > -c_{ER}\frac{\log_2(n)\beta}{16}\right)$
      then $u_2 \leftarrow u_3$
      else $u_1 \leftarrow u_3$
  end of while
Step 3. Compute $\gcd(u, n)$ for each $u \in$
  $\{u_1 + 1, \cdots, u_2\}$.
```

The Algorithm 4 has three steps. In `Step 1` an interval set $\{u_1+1, \cdots, u_2\}$ has to be found which contains an integer multiple of p or q. In `Step 2` a sequence of decreased interval subsets have to be determined, each of which contain an integer multiple of p or q. More precisely, in each step of `Step 2` it is checked whether the upper subset $\{u_3+1, \cdots, u_2\}$ with $u_3 \leftarrow \lfloor (u_1+u_2)/2 \rfloor$ contains such a multiple or not. In `Step 3`, when the subset $\{u_1+1, \cdots, u_2\}$ is sufficiently small, the attacker computes $\gcd(u, n)$ for all u contained in this subset. If all decisions within `Step 1` and `2` were correct, then the gcd computations will deliver the factorization of RSA modulus n.

Efficient error detection/correction strategies, which should be adopted to the attack algorithm 4, are also discussed in Refs. 9), 10).

4. Timing Attacks against RSA Using Parallel Exponentiation

In this section we propose timing attacks against RSA with and without CRT using the parallel Algorithm 1 which has been described in Section 2.

4.1 Without CRT

We first consider a timing attack against the parallel algorithm without CRT. Two parallelized exponentiations will be discussed in this subsection. In the case of two parallelized implementation without CRT, the running time should be faster compared to traditional implementations.

4.1.1 The Difficulty

The total running time of the parallel algorithm for the modular exponentiation depends on the most low-speed partial exponentiation among $x^{d^{(0)}} \bmod n, x^{d^{(1)}} \bmod n, \cdots, x^{d^{(b-1)}} \bmod n$. This property causes difficulty such that the running time of a cryptographic device could not constitute an information channel on all bits of d.

Let's consider the case of two parallelism. In that case two partial exponents $d^{(0)}$ and $d^{(1)}$ should be derived from the given exponent d as below.

$$d^{(0)} = d \wedge (0101\cdots01)_2$$
$$d^{(1)} = d \wedge (1010\cdots10)_2$$

where $\wedge$ denotes bitwise AND operation. The computational complexity of the partial modular exponentiations using left-to-right method can be evaluated as

$$(t_i - 1)\mathcal{S} + (H(d^{(i)}) - 1)\mathcal{M} \qquad (1)$$

for $i = 0, 1$, where t_i denotes the bitlength of $d^{(i)}$. Note that $d^{(1)}$ always has bitlength t. The hamming weight $H(d^{(i)})$ of the partial exponents $d^{(i)}$ has significant effect on the running time of the total modular exponentiation.

In the next subsection we will discuss this effect.

4.1.2 The Case that Hamming Weight of $d^{(0)}$ and $d^{(1)}$ are Almost the Same

We can state the following theorem.

Theorem 1 *Assume that the running time of the modular exponentiation can be evaluated by the number of modular multiplication and squaring required, that is, other influences can be ignored. Let $d^{(0)}$ and $d^{(1)}$ are derived from d by masking as before. If the following equation holds, the running time of the two partial modular equations $x^{d^{(0)}} \bmod n$ and $x^{d^{(1)}} \bmod n$ can be the same, except the influence of the extra reduction.*

$$\begin{aligned} &H(d^{(0)}) - \text{“the run-length of the} \\ &\text{leading bit 0 in } d^{(0)}\text{”} - 1 \\ &= H(d^{(1)}) - 1 \qquad (2) \end{aligned}$$

Proof. Notice that the MSB of $d^{(1)}$ is always 1. The evaluation (2) can be easily derived from (1). □

Following is a small example.

$$\begin{array}{cccccccccc} d & = & 1 & 1 & 1 & 1 & \cdots & 1 & 1 & 0 & 1 \\ d^{(0)} & = & 0 & 1 & 0 & 1 & \cdots & 0 & 1 & 0 & 1 \\ d^{(1)} & = & 1 & 0 & 1 & 0 & \cdots & 1 & 0 & 0 & 0 \end{array}$$

In this case, the running time of $x^{d^{(0)}} \bmod n$ and $x^{d^{(1)}} \bmod n$ can be the same, except the influence of the extra reduction. For randomly chosen input x, the running time of the total exponentiation $x^d \bmod n$ could be an information on $d^{(0)}$ and $d^{(1)}$.

We experimentally confirmed that how often Eq. (2) holds for randomly chosen exponent d. We randomly generated d by SHA-1 based pseudo-random number generator specified in FIPS 186-2 [3]. The number of d generated was 1,000,000. As the results, when d has the bitlength of 128, 256, 512, 768 and 1024, the probability that Eq. (2) holds are 6.4%, 4.1%, 2.0%, 1.0%, 0.5%, respectively. The probability decrease for larger d, but does have non-negligible percentages. With these probability above, the secret exponent d can be perfectly recovered by our attack. Experiments of the attack will be shown in the later section.

We should note that computational efficiency (in the sense of the running time of the exponentiation) can be the best case if all the bits set to one in the binary representation of d are equally scattered, i.e., Eq. (2) holds, among the exponent set $d^{(0)}$ and $d^{(1)}$. Therefore, the masking function to generate $d^{(0)}$ and $d^{(1)}$ may be chosen so that Eq. (2) holds, rather than the simple AND operation. In such the case our attack may perfectly recover the secret exponent d.

4.1.3 The Case that the Difference between Hamming Weight of $d^{(0)}$ and $d^{(1)}$ is Large

When $H(d^{(0)})$ is significantly larger than

$H(d^{(1)})$, the running time of the total modular exponentiation $x^d \bmod n$ can be regarded as that of $x^{d^{(0)}} \bmod n$. Therefore, in this case any information on $d^{(1)}$ can not be leaked in the running time of the total modular exponentiation. Then the attacker has little chance to recover $d^{(0)}$ from the running time.

We have "*partial key exposure attacks*" which is a class of attacks to answer the question: how many bits of d does an adversary require in order to reconstruct all of d?. Boneh, et al. presented such the attacks [1]. For instance, when the public exponent $e < 2^{(n/4)-3}$, the quarter of the least significant bits of d suffice to reconstruct the entire d. However, unfortunately, Boneh et al's attacks can not be applicable to our case, because recovered bits by the timing attack are not consecutive. Constructing a mathematical attack such as the partial key exposure attacks against our case should be a future work.

4.1.4 The Attack against the Parallel Implementation without CRT

In this subsection we state an algorithm for timing attack against our parallel algorithm of the modular exponentiation Algorithm 1. We consider the two parallelized implementation.

Similar to the attack against the traditional modular exponentiation, we define the model of the attacker as follows.

- The attacker has access to a device which performs the modular exponentiation with a secret exponent.
- The attacker can measure the running time of the total (not partial) modular exponentiation with various input x.
- The attacker knows the modulus n.
- The attacker has the knowledge of the implementation such that:
  - The modular exponentiation is performed by the two parallelized algorithm (i.e., $b = 2$).
  - The each partial modular exponentiation is performed by left-to-right binary method.
  - The partial exponentiations are performed without CRT.
  - Modular multiplication and modular squaring are performed with Montgomery's method.

We assume again that the MSB d_{t-1} of the secret exponent d is always 1. We also assume that the MSB of $d^{(1)}$ is always 1 by appropriate masking. The strategy for guessing d_i is quite similar to Algorithm 3. Algorithm 5 shows the strategy to guess the second significant bit d_{t-2} of the given exponent d.

Algorithm 5 Guessing d_{t-2} in the two parallelized implementation

```
1. Generating two subsets $X_1, X_2$:
   For various inputs $x$, the attacker does the following simulation with the knowledge of the implementation. At modular multiplication phase in Algorithm 1 with $i = t-2$, if the extra reduction has to be performed, $x$ should be classified into $X_1$. Else if the extra reduction is not required, $x$ should be classified into $X_2$.
2. Measuring the running times:
   Using the device, on which the modular exponentiation is implemented, the attacker measures the running time of the modular exponentiation for $x \in X_1$ and $x \in X_2$.
3. Guessing $d_{t-2}$:
   The attacker does a statistical analysis on the difference of the running times between $x \in X_1$ and $x \in X_2$. Then he guesses $d_{t-2} = 0$ or 1.
```

The attacker first guesses the bit d_{t-2} based on Algorithm 5. The same strategy can be applied continuously bit by bit from MSB to LSB. The attacker may recover the total secret exponent d.

4.1.5 Experiments

In this subsection we show an experiment on the attack to recover the secret exponent d. As in the previous section, we will consider the two parallelized implementation of our Algorithm 1 (i.e., $b = 2$). We made a software simulation on Pentium IV-based PC, running at 1.8 GHz. Programs were written in C-language with VC++ 6.0 compiler.

The experiments we have carried out is a simulation. Even if the PC does not have an architecture which can execute the exponentiation in parallel, we can simulate our attacks as the following. We first measure the running times of two partial exponentiations separately. We guess that the attacker can observe the running time of the slower one, which is the running time observed from real parallelized implementations. Then we apply the timing attack described before.

In this environment, when the modulus n and the exponent d has the size of 128 bits, the mean value of clock cycles needed to perform extra reduction was several hundreds. We used the mean value for statistical analysis as follows.

- For guessing d_i, if the difference of the mean value of the running time between input $x \in X_1$ and input $x \in X_2$ is larger than several hundreds clock cycles, then the attacker should guess $d_i = 1$.
- If the difference is smaller than several hundreds clock cycles, then the attacker should guess $d_i = 0$.

In the case that both n and d have 128 bits size, when we took 100,000 time measurements, the following results were obtained by our experiment.

- When the relation between two partial exponents $d^{(0)}$ and $d^{(1)}$ meets Eq. (2), we successfully recovered the entire exponent d.
- When the relation between two partial exponents $d^{(0)}$ and $d^{(1)}$ does not meet Eq. (2), one of the partial exponents $d^{(0)}$ or $d^{(1)}$ can be recovered.

4.1.6 More Parallelism

When we have more processors which perform the modular exponentiation, more speed-up can be achieved by the parallel implementation. Assume we have four processors, given exponent d should be scattered into four partial exponents $d^{(0)}$, $d^{(1)}$, $d^{(2)}$ and $d^{(3)}$. In such the case, our timing attack may be still applicable, but the probability analogous Eq. (2) holds will be much lower. Therefore higher parallelism can achieve not only speed-up but also security against our timing attack.

4.2 With CRT

Next we will move on to the case that RSA is implemented with the parallel exponentiation with CRT.

Assume again that we have two processors which perform the modular exponentiation in parallel. In such the case we can compute the modular exponentiation with CRT in parallel. Both $x^{(d \bmod p-1)} \bmod p$ and $x^{(d \bmod q-1)} \bmod q$ can be computed in parallel using two processors, respectively[☆]. We will make same assumptions, stated in the previous section, on the attacker. A different situation from the traditional non-parallel implementation is that four modular exponentiations are performed with two processors in parallel.

[☆] With two processors, we can also perform $x^{(d \bmod p-1)} \bmod p$ and $x^{(d \bmod q-1)} \bmod q$ in parallel. This computation may be faster than the computation described in this subsection. However, whether an analogy to Shindler's chosen input attack can be constructed is an open problem and remains as a future work. The discussion in this subsection could lead to attacks against four parallelism, which is an efficient setting for parallelized CRT implementation.

Let $d_p = d \bmod p-1$ and $d_q = d \bmod q-1$. In our parallel algorithm d_p and d_q are scattered into:

$$\begin{aligned} d_p^{(0)} &= d_p \wedge (0101\cdots01)_2 \\ d_p^{(1)} &= d_p \wedge (1010\cdots10)_2 \\ d_q^{(0)} &= d_q \wedge (0101\cdots01)_2 \\ d_q^{(1)} &= d_q \wedge (1010\cdots10)_2 \end{aligned} \tag{3}$$

We have four cases of the running time of the modular exponentiation:

- $T(u) =$ $Timing(u^{d_p^{(0)}} \bmod p) + Timing(u^{d_q^{(0)}} \bmod q)$ if $H(d_p^{(0)}) > H(d_p^{(1)})$ and $H(d_q^{(0)}) > H(d_q^{(1)})$
- $T(u) =$ $Timing(u^{d_p^{(1)}} \bmod p) + Timing(u^{d_q^{(0)}} \bmod q)$ if $H(d_p^{(1)}) > H(d_p^{(0)})$ and $H(d_q^{(0)}) > H(d_q^{(1)})$
- $T(u) =$ $Timing(u^{d_p^{(0)}} \bmod p) + Timing(u^{d_q^{(1)}} \bmod q)$ if $H(d_p^{(0)}) > H(d_p^{(1)})$ and $H(d_q^{(1)}) > H(d_q^{(0)})$
- $T(u) =$ $Timing(u^{d_p^{(1)}} \bmod p) + Timing(u^{d_q^{(1)}} \bmod q)$ if $H(d_p^{(1)}) > H(d_p^{(0)})$ and $H(d_q^{(1)}) > H(d_q^{(0)})$

where $H(u)$ denotes the hamming weight of u.

The attacker can observe the running timing $T(u)$. However he does not know which case has occurred. In the computationally best case, all the bits set to one in the binary representation of d_p and d_q are equally scattered among the exponent set $d_p^{(i)}$ and $d_q^{(i)}$ for $i = 0, 1$, so the computational cost is perfectly balanced on two individual exponentiations. In such the case, for the timing attack, we should make slight modification to Schindler's attack (Algorithm 4).

The expected hamming weight of the exponents $d_p^{(0)}$ (or $d_p^{(1)}$) and $d_q^{(0)}$ (or $d_q^{(1)}$) should be $0.25 \log_2 p$ and $0.25 \log_2 q$, respectively on average. Then we can modify the Algorithm 4 for attacking against the parallel implementation.

We have estimated the probability such that Eq. (2) holds in the case of without CRT. An analogous theorem to theorem 1 can be stated, but probability such that the evaluation (an analog of Eq. (2) in Theorem 1) holds may be lower. However, as we said previously, a balanced scattering function could be implemented on real devices for efficient implementation.

When the computational cost is not balanced, i.e., $H(d_p^{(0)})$ and $H(d_p^{(1)})$ are significantly different or $H(d_q^{(0)})$ and $H(d_q^{(1)})$ are significantly different, any information on $d_p^{(1)}$ and $d_q^{(1)}$ can not be leaked (assume that $H(d_p^{(0)}) \gg H(d_p^{(1)})$ and $H(d_q^{(0)}) \gg H(d_q^{(1)})$) in the running time of the total modular exponentiation. Then the attacker has little chance to recover the entire d from the running time as we said previously. Some mathematical attacks such as partial key exposure attacks could be a further work. Note that the revealed exponent by our attack can be $d \bmod p-1$ or $d \bmod q-1$. Therefore, there will be a different situation than the setting of without CRT.

4.2.1 The Scenario for Timing Attack

At the `Step 1` and `2` of Algorithm 4, the attacker has to decide that an integer multiple of p or q is contained in the interval set $\{u_1+1, \cdots, u_2\}$ or not. The criterion of the decision was that whether the timing difference between $T(u_2)$ and $T(u_1)$ is greater than $-c_{ER}\dfrac{\log_2(n)\beta}{16}$ or not.

Remark 4 *We should notice that the value* $-c_{ER}\dfrac{\log_2(n)\beta}{16}$ *is approximately equal to*

$$0.5\left[E\left(X_{u_2}-X_{u_1} \mid \text{Case A is true}\right) + E\left(X_{u_2}-X_{u_1} \mid \text{Case B is true}\right)\right]$$

The above criterion can be obtained from Lemma 2 and the assumption that the hamming weight of the secret exponents d_p and d_q have $0.5\log_2 p$ and $0.5\log_2 q$, respectively.

In our case, the parallel exponentiation, the expected hamming weight of the four secret exponents $d_p^{(0)}$, $d_p^{(1)}$, $d_q^{(0)}$ and $d_q^{(1)}$ strongly depend on the masking function. If we apply the simple bitwise AND operation as Eq. (3), the four hamming weights can be expected to $0.25\log_2 p$ for $d_p^{(i)}$ and $0.25\log_2 q$ for $d_q^{(i)}$, respectively. Thus a timing attack against that implementation is described by a very slight modification of Algorithm 4.

Algorithm 6 Timing attack against RSA with CRT using Montgomery multiplication and the parallel exponentiation

```
Step 1. Choose an integer $u$ with $\beta R \le u < n$ and set (e.g.) $\Delta \leftarrow 2^{-6}R$.
  $u_2 \leftarrow u$
  $u_1 \leftarrow u_2 - \Delta$
  while$\left(T(u_2)-T(u_1) > -c_{ER}\frac{\log_2(n)\beta}{32}\right)$ do:
     $u_2 \leftarrow u_1$
     $u_1 \leftarrow u_1 - \Delta$
  end of while
Step 2. while$(u_2 - u_1 > 1000)$ do:
     $u_3 \leftarrow \lfloor (u_1+u_2)/2 \rfloor$
     if$\left(T(u_2) - T(u_3) > -c_{ER}\frac{\log_2(n)\beta}{32}\right)$
     then $u_2 \leftarrow u_3$
     else $u_1 \leftarrow u_3$
  end of while
Step 3. Compute $\gcd(u,n)$ for each $u \in \{u_1+1, \cdots, u_2\}$.
```

We performed experiments of the timing attack based on the Algorithm 6 above. In the case that modulus n has 1,024-bit, our experiments showed that the attacker can reveal the secret prime factors p and q by about 1,000 timing measurements on average.

4.2.2 More Parallelism

When we have more processors which perform the modular exponentiation, we can have the same arguments stated in Subsection 4.1.6. In this case, d is scattered into $d_p^{(0)}$, $d_p^{(1)}$, $d_q^{(0)}$ and $d_q^{(1)}$. Then four partial exponentiations $x^{d_p^{(0)}} \bmod p$, $x^{d_p^{(1)}} \bmod p$, $x^{d_q^{(0)}} \bmod q$ and $x^{d_q^{(1)}} \bmod q$ can be computed in parallel. In such the case, our timing attack may be still applicable, but the probability analogous Eq. (2) holds will be much lower. Therefore higher parallelism can achieve not only speed-up but also security against our timing attack.

4.2.3 Error Correction and Detection Strategies

In Refs. 9), 10) error detection and correction strategies are described. At any instant within `Step 2` of Algorithm 4 the attacker can verify whether his decisions were correct so far, i.e., whether the actual interval $\{u_1+1, \cdots, u_2\}$ really contains an integer multiple of p or q. He just applies the decision rule to a time difference for neighboring values of u_1 and u_2. The attacker should examine the difference $T(u_2-1) - T(u_1+1)$. If this examination leads to the same decision, it is confirmed with high probability that the interval $\{u_1+1, \cdots, u_2\}$ truly contains a multiple of p or q.

The same strategy will be reasonable to the attack against implementations with the parallel exponentiation. However, as mentioned in Ref. 10), this strategy will lead the additional time measurements to be carried out.

5. Countermeasures

The timing attack described in this paper is against implementations with Montgomery multiplication. Thus the obvious countermeasures, such as a dummy operation within each Montgomery multiplication, will be applicable. A blinding the secret exponent d could also be a countermeasure.

References

1) Boneh, B., Durfee, G. and Frankel, Y.: An attack on RSA given a small fraction of the private key bits, *Advances in Cryptology — ASIACRYPT'98*, LNCS 1514, pp.25–34, Springer-Verlag (1998).
2) Dhem, J.F., Koeune, F., Leroux, P.A., Mestré, P. and Quisquater, J.J.: A practical implementation of the timing attack, *CARDIS 1998*, LNCS 1820, pp.175–190, Springer-Verlag (1998).
3) FIPS PUB 186-2: Digital Signature Standard (DSS), available at `http://csrc.nist.gov/CryptoToolkit/tkrng.html`
4) Garcia, J.M.G. and Garcia, R.M.: Parallel algorithm for multiplication on elliptic curves, *Cryptology ePrint Archive*, Report **2002/179**, `http://eprint.iacr.org` (2002).
5) Hachez, G. and Quisquater, J.J.: Montgomery exponentiation with no final subtractions: Improved Results, *Cryptographic Hardware and Embedded Systems — CHES 2000*, LNCS 1965, pp.293–301, Springer-Verlag (2000).
6) Kocher, P.C., Jaffe, J. and Job, B.: Differential power analysis, *Advances in Cryptology — CRYPTO'99*, LNCS 1666, pp.388–397, Springer-Verlag (1999).
7) Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, *Advances in Cryptology — CRYPTO'96*, LNCS 1109, pp.104–113, Springer-Verlag (1996).
8) Montgomery, P.L.: Modular multiplication without trial division, *Math. Comp.*, Vol.44, No.170, pp.519–521 (1885).
9) Schindler, W.: A timing attack against RSA with the Chinese Remainder Theorem, *Cryptographic Hardware and Embedded Systems — CHES 2000*, LNCS 1965, pp.109–124, Springer-Verlag (2000).
10) Schindler, W., Koeune, F. and Quisquater, J.J.: Improving divide and conquer attacks against cryptosystems by better error detection correction strategies, *Proc. 8th IMA International Conference on Cryptography and Coding*, pp.245–267 (2001).
11) Walter, C.D.: Montgomery exponentiation needs no final subtractions, *Electric Letters*, Vol.35, No.21, pp.1831–1832 (1999).
12) Walter, C.D. and Thompson, S.: Distinguishing exponent digits by observing modular subtractions, *RSA Conference 2001*, LNCS 2020, pp.192–207, Springer-Verlag (2001).

(Received November 28, 2003)
(Accepted June 8, 2004)

Yasuyuki Sakai received the B.S. and M.S. degree from Waseda University in 1990 and 1992, respectively. He is a researcher of Information Technology R&D Center at Mitsubishi Electric Corporation. His current research interests are in cryptography and information security. He is a member of the Institute of Electronics, Information and Communication Engineers.

Kouichi Sakurai received the B.S. degree in mathematics from Faculty of Science, Kyushu University and the M.S. degree in applied science from Faculty of Engineering, Kyushu University in 1986 and 1988, respectively. He had been engaged in the research and development on cryptography and information security at Computer & Information Systems Laboratory at Mitsubishi Electric Corporation from 1988 to 1994. He received the Dr. degree in engineering from Faculty of Engineering, Kyushu University in 1993. Since 1994 he has been working for Department of Computer Science of Kyushu University as an associate professor, and now he is a full professor. His current research interests are in cryptography and information security. Dr. Sakurai is a member of the Institute of Electronics, Information and Communication Engineers, the Mathematical Society of Japan, ACM and the International Association for Cryptologic Research.