Timing Attack against Implementation of a Parallel Algorithm for Modular Exponentiation

Yasuyuki Sakai[1] and Kouichi Sakurai[2]

[1] Mitsubishi Electric Corporation,
5-1-1 Ofuna, Kamakura, Kanagawa 247-8501, Japan
ysakai@iss.isl.melco.co.jp
[2] Kyushu University,
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan
sakurai@csce.kyushu-u.ac.jp

Abstract. We describe a parallel algorithm for modular exponentiation $y \equiv x^k \bmod n$. Then we discuss timing attacks against an implementation of the proposed parallel algorithm for modular exponentiation. When we have two processors, which perform modular exponentiation, an exponent k is scattered into two partial exponents $k^{(0)}$ and $k^{(1)}$, where $k^{(0)}$ and $k^{(1)}$ are derived by bitwise AND operation from k such that $k^{(0)} = k \wedge (0101 \cdots 01)_2$ and $k^{(1)} = k \wedge (1010 \cdots 10)_2$. Two partial modular exponentiations $y_0 \equiv x^{k^{(0)}} \bmod n$ and $y_1 \equiv x^{k^{(1)}} \bmod n$ 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 weight of $k^{(0)}$ and $k^{(1)}$ are smaller than that of k. Thus fast computation of modular exponentiation $y \equiv x^k \bmod n$ can be achieved. Moreover we show a timing attack against an implementation of this algorithm. We perform a software simulation of the attack and analyze security of the parallel implementation.

Keywords: Parallel modular exponentiation, Montgomery multiplication, Side channel attack, Timing attack, RSA cryptosystems

1 Introduction

1.1 Timing Attack

Smart cards 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 fields and timing to process. Side channel attacks, which use side channel information leaked from real implementation of cryptographic algorithms, were first introduced by Kocher [Ko96,KJJ99]. Side channel attacks can be often much more powerful than mathematical cryptanalysis. Thus, many literatures on side channel cryptanalysis have been published [IT02,IYTT02,OS00,Sc00,Wa99,WT01].

J. Zhou, M. Yung, Y. Han (Eds.): ACNS 2003, LNCS 2846, pp. 319–330, 2003.

In this paper, we focus on a timing attack against an implementation of a parallel algorithm for modular exponentiation which will be described in later section. 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 [Ko96]. He showed that a careful statistical analysis could lead to the total recovery of secret parameters.

In [DKLMQW98] a successful timing attack against a modular exponentiation without Chinese Remainder Theorem (CRT) was implemented. In [SQK01] further improved timing attacks were proposed. These attacks assume that implementations do not use CRT for modular exponentiation. In [Sc00] Schindler presented a powerful timing attack against an implementation using CRT. The exponent of modular exponentiation can be a secret parameter in RSA-based cryptosystems. Thus we must implement modular exponentiation very carefully.

In RSA-based cryptosystems, modular exponentiation is the most expensive part in implementation. Therefore, it is very attractive to provide algorithms that allow efficient implementation of modular exponentiation. Montgomery's method [Mo85] perform modular multiplication without a division instruction which is an expensive operation in almost processors. Thus this method can achieve computational speed-up and is often used in implementations of modular exponentiation. Timing attacks described in the above literature were against modular exponentiation with Montgomery's method. The running time of Montgomery's method for modular multiplication depends on the input value. If the intermediary result of the multiplication is greater than the modulus, an "*extra subtraction*" has to be performed. The extra subtraction lead differences between running times needed for various input values. Previous works on timing attacks against the modular exponentiation [DKLMQW98,SQK01,Sc00] made good use of the timing differences. Dhem et. al. presented a practical attack such that a secret exponent of 512 bits can be totally recovered with 300,000 time measurements [DKLMQW98]. In their attacks, an attacker has to know information on the implementation such that modular exponentiation is implemented with the binary method and Montgomery's multiplication. Walter et. al. gave a way to recover a secret exponent without knowledge of the modulus or input values [WT01].

1.2 Parallel Algorithm

Recently Garcia et. al. gave parallel algorithms for elliptic scalar multiplication [GG02]. When we have plural devises, which can perform elliptic scalar multiplications, we scatter a given scalar into several sub-scalars. Partial scalar multiplication are carried out with sub-scalar in parallel on plural devises. Then scalar multiplication can be computed with the outputs from the devises. Nonzero bits in scalar representation can be distributed into plural devises. Thus this

parallel algorithm provides an efficient implementation. This algorithm for elliptic scalar multiplication can be easily extended to modular exponentiation. In this paper we show a parallel algorithm for modular exponentiation. Our parallel algorithm can achieve speed-up of computation. However, as we noted before, computational advantage can not be enough for implementations, especially on smart cards. That is, side channel attacks have to be cared when we implement our parallel algorithm.

1.3 Our Contribution

In this paper we first describe a parallel algorithm for modular exponentiation. Then the computational efficiency will be discussed. Our parallel algorithm can gain speed in modular exponentiation if we have plural devises for modular exponentiation. When we have b processors, the computational complexity can be reduced to $(t-1)\mathcal{S} + (\lceil H(k)/b \rceil + b - 1)\mathcal{M}$ on the best case, where t and $H(k)$ denote the bitlength of k and hamming weight of k respectively, $\mathcal{S}$ and $\mathcal{M}$ denote modular squaring and multiplication respectively.

Moreover we analyze vulnerability of implementations of our parallel algorithm against the timing attack. We will state that there is difficulty in the attack against our parallel implementation. We adopt Dhem's strategy [DKLMQW98] based timing attack. We make an experiment on the timing attack with software simulation. Our results show that Dhem's method based attack can still work against the parallel implementation.

2 Parallel Algorithm for Modular Exponentiation

In this section we describe a parallel algorithm for modular exponentiation $y \equiv x^k \bmod n$. Garcia et. al. [GG02] gave a parallel algorithm for elliptic scalar multiplication, which is the basis of our algorithm.

2.1 The Algorithm

Assume that a t-bit non negative integer k has a binary representation $k = (k_{t-1} \cdots k_1 k_0), k_i \in \{0,1\}$. We divide the binary representation of k in $\lceil t/b \rceil$ blocks of b bits each one and then we scatter k into $k^{(0)}, k^{(1)}, \cdots, k^{(b-1)}$. The binary representation of the block $k^{(i)}$ 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 k. Thus we can define $k^{(i)}$, for $i = 0, \cdots, b-1$, as follows.

$$
\begin{aligned}
k^{(0)} &= k_0 + k_b 2^b + k_{2b} 2^{2b} + \cdots + k_{(\lceil \frac{t}{b} \rceil - 1)b} 2^{(\lceil \frac{t}{b} \rceil - 1)b} \\
k^{(1)} &= k_1 2 + k_{b+1} 2^{b+1} + k_{2b+1} 2^{2b+1} + \cdots + k_{(\lceil \frac{t}{b} \rceil - 1)b+1} 2^{(\lceil \frac{t}{b} \rceil - 1)b+1} \\
&\vdots \\
k^{(b-1)} &= k_{b-1} 2^{b-1} + k_{2b-1} 2^{2b-1} + k_{3b-1} 2^{3b-1} + \cdots + k_{b\lceil \frac{t}{b} \rceil - 1} 2^{b\lceil \frac{t}{b} \rceil - 1}
\end{aligned}
$$

That is

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

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

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

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

$$x^k \bmod n = \left(\left(x^{k^{(0)}} \bmod n \right) \left(x^{k^{(1)}} \bmod n \right) \cdots \left(x^{k^{(b-1)}} \bmod n \right) \right) \bmod n$$

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

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

Algorithm 1 Parallel modular exponentiation

`Input` x, n, k
`Output` $y \equiv x^k \bmod n$

1. `Bits scattering:`
   Compute the set $\left\{ k^{(0)}, k^{(1)}, \cdots, k^{(b-1)} \right\}$
2. `Parallel computation:`
   Using b processors, compute in parallel the exponentiations

$$y \equiv \left(\left(x^{k^{(0)}} \bmod n \right) \left(x^{k^{(1)}} \bmod n \right) \cdots \left(x^{k^{(b-1)}} \bmod n \right) \right) \bmod n$$

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 we discuss the computational complexity of the parallel modular exponentiation Algorithm 1. Assume k has the bitlength t and has the hamming weight $H(k)$. Then we have the following theorem [GG02].

Theorem 1. *Assume each individual exponentiation is performed by the binary method (Algorithm 2). The parallel exponentiation Algorithm 1 has the computational complexity, on the best case,*

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

and, on the worst case,

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

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 k_{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 k are equally scattered among the exponent set $\{k^{(i)}\}$, so the computational cost is perfectly balanced on all the individual exponentiation.

The worst case implies that there exists some index i, for $0 \leq i \leq b-1$, such that all of bit "1" are mapped to $k^{(i)}$ and then $x^{k^{(i)}} \bmod n = x^k \bmod n$. In such the case, the computational cost is the same as the traditional binary exponentiation.

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

3 Timing Attack against Modular Exponentiation

In this section we explain a framework of the timing attack against implementation of the basic modular exponentiation without CRT.

3.1 Modular Exponentiation

In this paper we consider the binary representation for a given exponent and the left-to-right binary (square-and-multiply) method for modular exponentiation. The left-to-right binary method for modular exponentiation $y \equiv x^k \bmod n$ can be described as Algorithm 2.

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

```
Input  x, n, k, where k = (k_{t-1} k_{t-2} ··· k_0), k_i ∈ {0,1} for 0 ≤ i ≤ t-2 and
       k_{t-1} = 1
Output y ≡ x^k mod n
1.  y ← x
2.  for i from t-2 downto 0
        y ← y^2 mod n
        if k_i = 1 then
            y ← y · x mod n
    endfor
3.  return y
```

The timing attack will be demonstrated on an implementation with the following algorithms.

- The modular exponentiation is performed by Algorithm 2.
- The modular exponentiation is performed without CRT.
- Modular multiplication and modular squaring are performed with Montgomery's method.

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

The goal for an attacker is to recover the exponent k, which can be a secret parameter of the decryption and the signature generation in RSA-based cryptosystems. The attacker determine a secret parameter k 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 perform 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. In this paper, we assume that the modular exponentiation is performed by the method described in the previous section.

The attack algorithm will be developed with the assumption that the running time of modular exponentiation only depend on the base x but not on other influences.

3.3 Dhem's Method of Timing Attack

Dhem et. al. proposed a practical timing attack against an implementation of modular exponentiation without CRT [DKLMQW98]. In this subsection we briefly explain their strategy. See [DKLMQW98] for details. Our timing attack, which will be described in the next section, will be developed based on their attack.

Assume again that a given exponent k has binary representation $k = (k_{t-1} \cdots k_1 k_0)$, where $k \in \{0, 1\}, k_{t-1} = 1$. Their attack recovers the exponent bit by bit from k_{t-2} to the least significant bit k_0. Notice that the MSB k_{t-1} is always 1. We start by attacking k_{t-2}. When $k_{t-2} = 1$, 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 subtraction has to be performed. For the other input x, the extra subtraction

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.

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

If the value of k_{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 k_{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 subtraction 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 k_{t-2}, he should take the strategy below.

Algorithm 3 Guessing k_{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 subtraction has to be performed, x should be classified into X_1. Else if the extra subtraction is not required, x should be classified into X_2.
2. `Measuring the running times:`
   Using the device, on which modular exponentiation is implemented, the attacker measures the running time of modular exponentiation for $x \in X_1$ and $x \in X_2$.
3. `Guessing` k_{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 $k_{t-2} = 0$ 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 k_{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 k.

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 [DKLMQW98]. The same strategy as multiplication phase described before can be applicable.

4 A Timing Attack against the Parallel Modular Exponentiation

In this section we consider a timing attack against the parallel algorithm for modular exponentiation Algorithm 1. Two parallelized exponentiation will be discussed.

4.1 The Difficulty

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

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

$$k^{(0)} = k \wedge (0101 \cdots 01)_2$$
$$k^{(1)} = k \wedge (1010 \cdots 10)_2$$

where $\wedge$ denotes bitwise AND operation.

The computational complexity of the partial modular exponentiations using left-to-right method of modular exponentiation Algorithm 2 can be evaluated as

$$(t_i - 1)\mathcal{S} + \left(H(k^{(i)}) - 1\right)\mathcal{M} \tag{1}$$

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

In the next subsection we will discuss this effect.

4.2 The Case That Hamming Weight of $k^{(0)}$ and $k^{(1)}$ Are almost the Same

We can state the following theorem.

Theorem 2. *Assume that the running time of modular exponentiation can be evaluated by the number of modular multiplication and squaring required, that is, other influences can be ignored. Let $k^{(0)}$ and $k^{(1)}$ are derived from k by masking as before. If the following equation holds, the running time of the two partial modular equations $x^{k^{(0)}} \bmod n$ and $x^{k^{(1)}} \bmod n$ have almost the same running time.*

$$H(k^{(0)}) - \text{“}\textit{the run-length of the leading bit 0 in } k^{(0)}\text{”} - 1 = H(k^{(1)}) - 1 \tag{2}$$

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

Following is a small example.

$$\begin{aligned} k &= 1\,1\,1\,1\cdots 1\,1\,0\,1 \\ k^{(0)} &= 0\,1\,0\,1\cdots 0\,1\,0\,1 \\ k^{(1)} &= 1\,0\,1\,0\cdots 1\,0\,0\,0 \end{aligned}$$

In this case, the running time of $x^{k^{(0)}} \bmod n$ and $x^{k^{(1)}} \bmod n$ can be almost the same.

For randomly chosen input x, The running time of the total exponentiation $x^k \bmod n$ could be an information on $k^{(0)}$ and $k^{(1)}$.

4.3 The Case That the Difference between Hamming Weight of $k^{(0)}$ and $k^{(1)}$ Is Large

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

When the relation between $H(k^{(0)})$ and $H(k^{(1)})$ is far from the evaluation (2) in theorem 2, how better chance for successful attack the attacker can get may depend on the following.

- The ratio of the running time of the extra subtraction to the running time of $x^{k^{(0)}} \bmod n$.
- The influences on running time other than Montgomery's method of modular multiplication and squaring.

4.4 An Algorithm for Attack against the Parallel Implementation

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

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

- The attacker has access to a device which perform 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 k_{t-1} of the secret exponent k is always 1. We also assume that the MSB of $k^{(1)}$ is always 1 by appropriate masking. The strategy for guessing k_i is quite similar to Algorithm 3. Algorithm 4 shows the strategy to guess the second significant bit k_{t-2} of the given exponent k.

Algorithm 4 Guessing k_{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 subtraction has to be performed, x should be classified into X_1. Else if the extra subtraction is not required, x should be classified into X_2.
2. `Measuring the running times:`
   Using the device, on which modular exponentiation is implemented, the attacker measures the running time of modular exponentiation for $x \in X_1$ and $x \in X_2$.
3. `Guessing` k_{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 $k_{t-2} = 0$ or 1.

The attacker first guesses the bit k_{t-2} based on Algorithm 4. The same strategy can be applied continuously bit by bit from MSB to LSB. The attacker may recover the total secret exponent k.

5 An Experiment

In this section we show an experiment on the attack to recover the secret exponent k. 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.

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

– For guessing k_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 $k_i = 1$.

– If the difference is smaller than several hundreds clock cycles, then the attacker should guess $k_i = 0$.

In the case that both n and k has 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 $k^{(0)}$ and $k^{(1)}$ almost holds equation (2), we successfully recovered the entire exponent k.
– When the relation between two partial exponents $k^{(0)}$ and $k^{(1)}$ is far from equation (2), one of the partial exponents $k^{(0)}$ or $k^{(1)}$ can be recovered.

Countermeasure. For an implementation of our parallel algorithm for modular exponentiation, possible countermeasures could be similar way as the case of traditional implementation of modular exponentiation with Montgomery's method. Always performing ("*dummy*") extra subtraction strategy could be a counter measure [HQ00,Wa99], but lead extra computational cost.

6 Conclusion and Further Work

In this paper we described an algorithm for parallel modular exponentiation, and then the computational efficiency has been discussed. Moreover, we showed a timing attack against an implementation of our parallel algorithm.

If one of the partial exponent $k^{(0)}$ or $k^{(1)}$ is recovered, that is half bits of given exponent k is recovered, some number theoretic approaches (e.g. [BDF98]) could be applicable for total break.

Moreover, we demonstrated that the similar strategy as Dhem's attack can be applicable to our parallel modular exponentiation. But we showed the difficulty of the timing attack, which is caused from the parallelism.

To extend our attack to larger n and k, possible further work could be: Attacking on the modular square phase may be useful for the timing attack [DKLMQW98]. Error detection strategies [DKLMQW98,Sc00,SQK01] for the parallel exponentiation should be considered.

References

[BDF98] D. Boneh, G. Durfee, and Y. Frankel, "An attack on RSA given a small fraction of the private key bits," *Advances in Cryptology – ASIACRYPT'98*, LNCS, **1514** (1998), Springer-Verlag, 25–34.

[DKLMQW98] J.F. Dhem, F. Koeune, P.A. Leroux, P. Mestré, and J.J. Quisquater, "A practical implementation of the timing attack," *CARDIS 1998*, LNCS, **1820** (1998), Springer-Verlag, 175–190.

[GG02] J.M.G. Garcia and R.M. Garcia, "Parallel algorithm for multiplication on elliptic curves," *Cryptology ePrint Archive*, Report **2002/179**, (2002), `http:eprint.iacr.org`.

[HQ00] G. Hachez and J.J. Quisquater, "Montgomery exponentiation with no final subtractions: Improved Results," *Cryptographic Hardware and Embedded Systems – CHES 2000*, LNCS, **1965** (2000), Springer-Verlag, 293–301.

[IT02] T. Izu and T. Takagi, "Fast parallel elliptic curve multiplications resistant to side channel attacks," *Public Key Cryptography – PKC 2002*, LNCS, **2274** (2002), Springer-Verlag, 335–345.

[IYTT02] K. Itoh, J. Yajima, M. Takenaka, and N. Torii, "DPA countermeasures by improving the window method," *Cryptographic Hardware and Embedded Systems – CHES 2002*, (2002).

[KJJ99] P.C. Kocher, J. Jaffe, and B. Job, "Differential power analysis," *Advances in Cryptology – CRYPTO'99*, LNCS, **1666** (1999), Springer-Verlag, 388–397.

[Ko96] P.C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology – CRYPTO'96*, LNCS, **1109** (1996), Springer-Verlag, 104–113.

[Mo85] P.L. Montgomery, "Modular multiplication without trial division," *Math. Comp.*, **44** (no. 170) (1885), 519–521.

[OS00] K. Okeya and K. Sakurai, "Power analysis breaks elliptic curve cryptosystems even secure against the timing attack," *INDOCRYPT 2000*, LNCS, **1977** (2000), Springer-Verlag.

[Sc00] W. Schindler, "A timing attack against RSA with the Chinese Remainder Theorem," *Cryptographic Hardware and Embedded Systems – CHES 2000*, LNCS, **1965** (2000), Springer-Verlag, 109–124.

[SQK01] W. Schindler, J.-J. Quisquater, and F. Koeune, "Improving divide and conquer attacks against cryptosystems by better error detection correction strategies," *Proc. of 8th IMA International Conference on Cryptography and Coding*, (2001), 245–267.

[Wa99] C.D. Walter, "Montgomery exponentiation needs no final subtractions," *Electric Letters*, LNCS, **35** (no. 21) (1999), 1831–1832.

[WT01] C.D. Walter and S. Thompson, "Distinguishing exponent digits by observing modular subtractions," *RSA Conference 2001*, LNCS, **2020** (2001), Springer-Verlag, 192–207.