New Modular Multiplication Algorithms for Fast Modular Exponentiation

Seong-Min Hong, Sang-Yeop Oh, Hyunsoo Yoon

Department of Computer Science and Center for AI Research
Korea Advanced Institute of Science and Technology(KAIST)
Taejeon, 305-701, KOREA
E-mail: {smhong,hyoon}@camars.kaist.ac.kr

Abstract. A modular exponentiation is one of the most important operations in public-key cryptography. However, it takes much time because the modular exponentiation deals with very large operands as 512-bit integers. The modular exponentiation is composed of repetition of modular multiplications. Therefore, we can reduce the execution time of it by reducing the execution time of each modular multiplication. In this paper, we propose two fast modular multiplication algorithms. One is for modular multiplications between different integers, and the other is for modular squarings. These proposed algorithms require single-precision multiplications fewer than those of Montgomery modular multiplication algorithms by 1/2 and 1/3 times, respectively. Implementing on PC, proposed algorithms reduce execution times by 50% and 30% compared with Montgomery algorithms, respectively.

1 Introduction

Since Diffie and Hellman had proposed public-key cryptography in 1976, many public-key cryptosystems have been developed[8, 9]. Many of them require modular exponentiations[10, 8, 11]. Therefore, a modular exponentiation becomes one of the most important operations. However, it takes much time because the modular exponentiation deals with very large operands as 512-bit integers. Therefore, many researchers have studied for the speedup of the modular exponentiation[1, 14, 18, 6, 7].

A modular exponentiation is composed of repetition of modular multiplications. Therefore, there are two possible methods to reduce the execution time of the modular exponentiation. One is to reduce the number of modular multiplications, and the other is to reduce the execution time of each modular multiplication.

Again, the latter is classified into two classes. One is the approach of considering a modular reduction apart from a multiple-precision multiplication[2, 17, 12, 13]. The other is the approach of considering together them as one operation[16, 15, 18]. Algorithms which will be proposed in this paper are included in the latter class.

It is the small-window method that can find the shortest addition-chain among systematically analyzable algorithms. Modular multiplications can be

U. Maurer (Ed.): Advances in Cryptology - EUROCRYPT '96, LNCS 1070, pp. 166-177, 1996.
© Springer-Verlag Berlin Heidelberg 1996

classified into two species when we execute a modular exponentiation with the small-window method. Ones are modular multiplications of two different integers, and the others are modular squarings. In this paper, we propose two algorithms. One is for fast modular multiplications of two different integers. It is the algorithm expanded from the Kawamura's[15]. The other is for fast modular squarings. It executes a modular squaring fast by altering the sequence of calculation. The former shows good performance for modular multiplications of two different integers, but it can not be applicable to modular squarings. Therefore, we can execute a modular exponentiation fast with both the former and the latter.

This paper is organized as follows. In Section 2, we explain briefly the process of modular exponentiation using the small-window method. In Section 3, we explain the basic method for modular multiplications and propose two modular multiplication algorithms. In Section 4, we analyze performances of proposed algorithms and compare them to that of Montgomery algorithm. In Section 5, performances of proposed algorithms implemented on a PC are presented, followed by discussions. Finally, we conclude in Section 6.

2 The Procedure of A Modular Exponentiation

The modular exponentiation used in public-key cryptosystems like RSA is defined as follows.

Definition 1. $C = M^E \bmod N \quad (b^{k-1} \leq E < N < b^k, 0 \leq M < N)$.

In the above definition, M means a message, and E means a public-key. It is desirable to obtain a sequence of modular multiplications for the fast execution of the modular exponentiation in advance, because E is known previously. An addition-chain is used to represent the sequence, which is defined as follows.

Definition 2. An *addition-chain* of length l for an integer n is a sequence of integers $a_0, a_1, ..., a_l$ satisfying

1. $a_0 = 1,\ a_l = n$.
2. $a_i = a_j + a_k,\ \ where\ \ 0 \leq j \leq k < i \leq l$.

We can enumerate the sequence of modular multiplications for a modular exponentiation according to Definition 1 and Definition 2.

$$\begin{aligned} C_0 &= M^{a_0} \bmod N &&= M, \\ C_1 &= M^{a_1} \bmod N, \\ C_2 &= M^{a_2} \bmod N, \\ &\vdots \\ C_{l-1} &= M^{a_{l-1}} \bmod N, \\ C_l &= M^{a_l} \bmod N &&= M^E \bmod N (= C). \end{aligned} \tag{1}$$

Each step of Equation (1) is connected with previous steps by the following relation.

$$C_i = (C_j \times C_k) \bmod N, \quad where \ 0 \le j \le k < i \le l. \tag{2}$$

As we can see in Equations (1) and (2), the shorter the addition-chain is the shorter the execution time of the modular exponentiation is. Many researchers have proposed addition-chain algorithms[1, 2, 3, 4, 5, 6, 7]. Among these algorithms, the Bos-Coster's heuristic algorithm[1] shows the best performance. However, the gap between the small-window method[2] and the Bos-Coster's algorithm is very small and the latter is more complex. Modular multiplication algorithms which will be proposed later in this paper are applicable to both algorithms. Therefore, we suppose that the small-window method is used to execute a modular exponentiation for the sake of convenient explanation.

A modular exponentiation is executed by the repetition of following two kinds of operations according to the above supposition, where w means the window size in the small-window method.

$$C_i = C_{i-1} \times C_\alpha \bmod N, \quad where \ 0 < i \le l, \ 0 \le \alpha < 2^{w-1}. \tag{3}$$

$$C_i = C_{i-1}^2 \bmod N, \quad where \ 0 < i \le l. \tag{4}$$

3 Algorithms

In this section, we explain the basic method for modular multiplications and propose two modular multiplication algorithms. We call Equation (3) a *window modular multiplication* and Equation (4) a *modular squaring*.

3.1 Basic Method

The basic method for a modular multiplication is as follows. A multiple-precision multiplication and a modular reduction are separate tasks. The former is executed first and the latter is executed secondly. The result of the multiplication is the input of the modular reduction. Equation (3) and Equation (4) are calculated by Equation (5) and Equation (6), respectively.

$$\begin{aligned} C_i &= C_{i-1} \times M^\alpha \bmod N \\ &= (C_{i-1} \times (M^\alpha \bmod N)) \bmod N \\ &= (C_{i-1} \times T[\alpha]) \bmod N, \\ &\quad where \ T[\alpha] = M^\alpha \bmod N, 0 \le \alpha < 2^{w-1}. \end{aligned} \tag{5}$$

$$\begin{aligned} C_i &= C_{i-1}^2 \bmod N \\ &= (C_{i-1}^2) \bmod N. \end{aligned} \tag{6}$$

3.2 Proposed Algorithms

In this section, we propose two modular multiplication algorithms. One is for the window modular multiplication and the other is for the modular squaring. We express a multiple-precision integer C_{i-1} as follows:

$$C_{i-1} = \sum_{j=0}^{k-1} c_j b^j. \tag{7}$$

Note that we write C_{i-1} as C for the sake of simplicity in this section.

Window Modular Multiplication. A feature of the small-window method is that one of two operands in the window modular multiplication is restricted in narrow limits. As we suppose that the small-window method is used to decide the sequence of modular multiplications, Equation (3) can be expanded to Equation (8).

$$\begin{aligned} C_i &= C \times M^\alpha \bmod N \\ &= (\textstyle\sum_{j=0}^{k-1} c_j b^j) \times M^\alpha \bmod N \\ &= (\textstyle\sum_{j=0}^{k-1} c_j \times (b^j \times M^\alpha \bmod N)) \bmod N \\ &= (\textstyle\sum_{j=0}^{k-1} c_j \times T[\alpha][j]) \bmod N, \\ &\quad \textit{where } T[\alpha][j] = b^j \times M^\alpha \bmod N \textit{ and } 0 \le \alpha < 2^{w-1}. \end{aligned} \tag{8}$$

On executing a modular exponentiation with the small-window method, M^α and N can be considered constant numbers. Therefore, $b^j \times M^\alpha \bmod N$ can be calculated in advance. The table T in Equation (8) can be calculated by the following equation.

$$\begin{aligned} T[\alpha][0] &= M^\alpha \bmod N, \\ T[\alpha][j] &= (T[\alpha][j-1] \times b) \bmod N, \\ &\quad \textit{where } 0 \le \alpha < 2^{w-1} \textit{ and } 0 < j \le k-1. \end{aligned} \tag{9}$$

Modular Squaring. It is required to execute more modular squarings than window modular multiplications in order to execute a modular exponentiation. However, the algorithm explained in the previous section can not be applicable to modular squarings, because C_{i-1} in Equation (6) is not known in advance. Therefore, we need a fast modular squaring algorithm.

In this section, we propose a fast modular squaring algorithm. It uses the fact that modular reductions can be executed fast for small operands. Equation (4) can be expanded as follows.

$$\begin{aligned} C_i &= C^2 \bmod N \\ &= (c_{k-1}b^{k-1} + c_{k-2}b^{k-2} + \cdots + c_1 b^1 + c_0)^2 \bmod N \\ &= ((\cdots(c_{k-1}b + c_{k-2})b + \cdots + c_1)b + c_0)^2 \bmod N \\ &= (C^{(1)}b + c_0)^2 \bmod N \\ &= ((C^{(1)})^2 b^2 + 2c_0 C^{(1)} b + c_0^2) \bmod N \\ &= (((C^{(1)})^2 \bmod N)b^2 + 2c_0 C^{(1)} b + c_0^2) \bmod N, \\ &\quad \textit{where } C^{(1)} = (\cdots(c_{k-1}b + c_{k-2})b + \cdots + c_2)b + c_1. \end{aligned} \tag{10}$$

Looking at the first and the last term among all of right terms in the above equation, $C^2 \bmod N$ and $(C^{(1)})^2 \bmod N$ are included, respectively. They have the same form as each other. Therefore, we can compute Equation (10) recursively. In addition to it, $C^{(1)}$ in Equation (10) is the value which is shifted to the right by one digit from C. If we repeat the recursive function call $\frac{k}{2}$ times, the final value is as follows:

$$C^{(\frac{k}{2})} = \sum_{j=0}^{\frac{k}{2}-1} c_{j+\frac{k}{2}} b^j. \tag{11}$$

Because $C^{(\frac{k}{2})}$ is a $\frac{k}{2}$-digit integer, the result of squaring is a k-digit integer at most, and $(C^{(\frac{k}{2})})^2 \bmod N$ can be calculated by only one subtraction or no operation except for a squaring. Therefore, we can consider Equation (11) basis of recursive calls and get the result of a modular squaring by repeating the following equation $\frac{k}{2}$ times.

$$\begin{aligned} &(C^{(j-1)})^2 \bmod N \\ &= (((C^{(j)})^2 \bmod N)b^2 + 2c_0 C^{(j)} b + c_0^2) \bmod N, \\ &\quad \textit{where } 1 \le j \le \tfrac{k}{2} \textit{ and } C^{(0)} = C. \end{aligned} \tag{12}$$

The only term to be reduced by modulus N is $((C^{(j)})^2 \bmod N)b^2$ when we calculate Equation (12). Because $(C^{(j)})^2 \bmod N$ is the value calculated in the previous step, $((C^{(j)})^2 \bmod N)b^2 \bmod N$ can be calculated by shifting $(C^{(j)})^2 \bmod N$ to the left by two digits and by executing a simple modular reduction to the result of shifting. This procedure can be processed using only subtractions, and is appeared in the following equation.

$$\begin{aligned} &(((C^{(j)})^2 \bmod N)b^2) \bmod N \\ &= (((((C^{(j)})^2 \bmod N)\log s - T[m_0])\log s - T[m_1])\cdots)\log s - T[m_{t-1}], \\ &\quad \textit{where } T[m_x] = m_x \times N,\ 0 \le m_x < s,\ 0 \le x < t,\ t = \tfrac{2\log b}{\log s}. \end{aligned} \tag{13}$$

In the above equation, t must be as small as possible because it means the number of shifts and subtractions. However, b is determined by the system on which the algorithm is implemented, and s is limited by the memory capacity. The reasonable value of b and s are 2^{16} and 2^8 on current 32-bit computer systems, respectively. Note that m_x can be determined easily using backward pointer array, which can be made during the table construction.

4 Time Complexity

In this section, we compute time complexities of algorithms proposed in this paper and compare them with those of existing algorithms. The metric of the time complexity is the number of single-precision multiplications required for the execution of the corresponding algorithm.

4.1 Basic Method

The number of single-precision multiplications required to multiply two large integers which is represented like Equation (7) is k^2. All of existing modular reduction algorithms require $k(k+c)$ single-precision multiplications, where c is the constant according to the algorithm used. Therefore, the number of single-precision multiplications required to execute a modular multiplication by the basic method is as follows:

$$2k^2 + ck.$$

The value of c is '1' if the used algorithm is the Montgomery's which is the best modular reduction algorithm[12, 18, 14].

4.2 Proposed Algorithms

Window Modular Multiplication. It is not required to execute any explicit modular reduction to calculate Equation (3) using Equation (8). Only a few additional operations are required to reduce an intermediate result into a k-digit integer.

First, k single-precision multiplications are required to multiply each digit and the residue equivalent to it. The residue is $T[\alpha][j]$ in Equation (8). Because an integer has k digits, the number of single-precision multiplications required is k^2 totally. The intermediate result is larger than N. However, we can get the final result with no single-precision multiplication using the table explained in Section 3.2, because the difference is very small. Therefore, the number of single-precision multiplications required to calculate Equation (8) is as follows:

$$k^2. \tag{14}$$

Next, we consider single-precision multiplications required to construct the table explained in Section 3.2. Seeing Equation (9), there is no additional operation required to calculate $T[\alpha][0]$s, because they are values that are calculated in the previous step. Each of the next operations needs one shift to the left and one modular reduction. The integer to be reduced by modulus N has $k+1$ digits at most. Therefore, the modular reduction can be executed by k single-precision multiplications. The number of single-precision multiplications required to construct the table is as follows:

$$2^{w-1} \times k(k-1). \tag{15}$$

In the above equation, w means the window size. If we use the table explained in Section 3.2, all table entries can be calculated with no single-precision multiplication.

Modular Squaring. We count the number of single-precision multiplications required to execute a modular squaring by the proposed algorithm. First, $\frac{k^2+2k}{8}$ single-precision multiplications are required to calculate Equation (11). Second, we count the number of single-precision multiplications required to execute Equation (12). The first term in the right side of the equation needs no single-precision multiplication, as we can see in Equation (13). The second term requires $\frac{3k^2-2k}{8}(=\sum_{j=\frac{k}{2}}^{k-1} j)$ single-precision multiplications during all recursive function calls. The third term requires $k/2$ single-precision multiplications because it requires one for each recursive call. Therefore, the number of single-precision multiplications required to execute Equation (4) by the proposed algorithm is as follows:

$$\frac{k^2+k}{2}. \tag{16}$$

The table T in Equation (13) can be constructed without any single-precision multiplication.

4.3 Comparison

We have computed time complexities of proposed algorithms and the basic method so far. However, there are many existing algorithms for modular multiplications other than the basic method[15, 13, 18, 7, 16]. Some are algorithms using pre-computation table and others are algorithms which regard a multiple-precision multiplication and a modular reduction as one operation. We briefly examine them.

An efficient algorithm was proposed when a modular exponentiation was executed using the binary method[2] as an addition-chain algorithm in [7] and [15]. However, it is not good for a modular exponentiation because the binary method is very inefficient. Findlay et. al. proposed an algorithm using partial modular reductions based on sums of residues in [13]. However, the number of single-precision multiplications required for partial modular reductions is k^2. Furthermore, a few additional operations are required because it is not the final result. Therefore, the number of single-precision multiplications required for this algorithm is much the same as that of the basic method. Morita et. al. proposed a new modular multiplication algorithm in [16]. However, it reduces only the required available memory for the computation but not the number of single-precision multiplications.

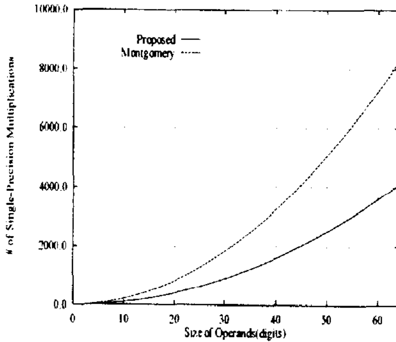
As we look at in the above paragraph, existing algorithms using a pre-computation table or combining a multiple-precision multiplication and a modular reduction into a single operation are not much different from the basic method in regard to the number of single-precision multiplications. The best modular reduction algorithm known to us is the Montgomery algorithm[14]. Therefore, we compare our algorithms with the basic method which uses the Montgomery reduction algorithm.

Time complexities of proposed algorithms and those of Montgomery algorithms are compared in Table 1 and Figure 1. The time required for table construction in proposed algorithms and the time required for pre-/post-calculation

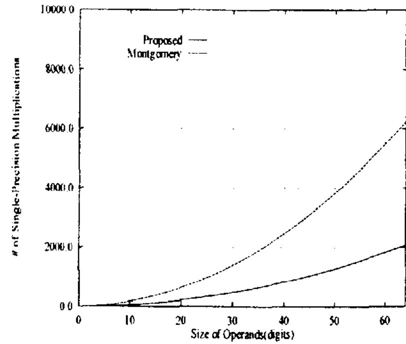
in Montgomery algorithms are excluded, because many modular multiplications are required to execute a modular exponentiation. The number of single-precision multiplications required to execute an ordinary multiple-precision multiplication is recorded together, for reference. The ordinary multiple-precision multiplication means one which needs not modular reduction. s in Table 1 is the same as that in Equation (13).

Table 1. *The time complexity and the memory requirement of each algorithm. NOSPM means the number of single-precision multiplications. The metric of memory requirement is the number of operands.*

Algorithm		NOSPM	Memory Requirement
Window Modular Multiplication	Montgomery	$2k^2 + k$	2^{w-1}
	Proposed	k^2	$k \times 2^{w-1}$
Ordinary Multiplication		k^2	2^{w-1}
Modular Squaring	Montgomery	$\frac{3}{2}(k^2 + k)$	0
	Proposed	$\frac{1}{2}(k^2 + k)$	$k \times s$
Ordinary Squaring		$\frac{1}{2}(k^2 + k)$	0



(a) *Window Modular Multiplication*



(b) *Modular Squaring*

Fig. 1. *The time complexity of each algorithm*

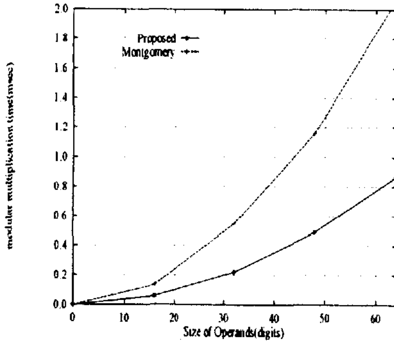
As we can see in Table 1 and Figure 1, numbers of single-precision multiplications required for proposed algorithms are the same as those required for ordinary multiple-precision multiplications. That is, explicit modular multiplications are not required for proposed algorithms. The proposed window modular multiplication algorithm removes them by pre-calculations and the proposed modular squaring algorithm removes them by pre-calculations and subtractions.

5 Implementation and Discussion

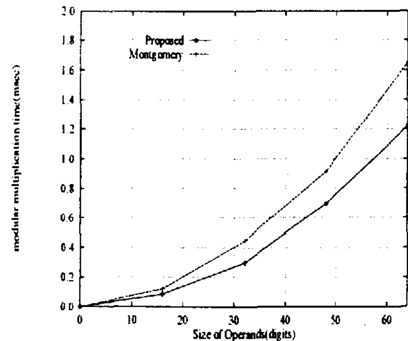
We implemented proposed algorithms and the Montgomery algorithm. The system on which we implemented is a PC with Pentium-90 microprocessor. We implemented them in C language, and compiled with Watcom C(version 10.0) compiler. A digit is 16-bit($b = 2^{16}$). Results are appeared in Table 2.

Table 2. *The execution time of each algorithm*

Algorithm		Time of Execution(msec.)			
		256-bit	512-bit	768-bit	1024-bit
Window Modular Multiplication	Montgomery	0.137	0.544	1.16	2.07
	Proposed	0.0604	0.220	0.489	0.868
Modular Squaring	Montgomery	0.121	0.445	0.917	1.65
	Proposed	0.0851	0.297	0.698	1.22



(a) *Window Modular Multiplication*



(b) *Modular Squaring*

Fig. 2. *The execution time of each algorithm*

According to Table 1, proposed algorithms are faster than Montgomery algorithms by two and three times, respectively. As we can see in Table 2 and Figure 2(a), the real execution time agrees with Table 1 and Figure 1(a) in the case of window modular multiplications. However, it is not in the case of modular squarings. The reason is that we count only the number of single-precision multiplications in Table 1. While it is a fact that a multiplication takes much time than an addition in general purpose processors, as the proposed modular squaring algorithm uses many additions we must consider the number of single-precision additions, also. Time complexities are appeared in consideration of the number of single-precision additions in Table 3. r is the constant required to

represent the time complexity as the number of single-precision multiplications. It is defined as follows and determined by the system on which the corresponding algorithm is implemented.

Definition 3. $r = \dfrac{\text{the time required for a single-precision addition}}{\text{the time required for a single-precision multiplication}}$

Table 3. *The time complexity in consideration of single-precision additions. NOSPM means the number of single-precision multiplications.*

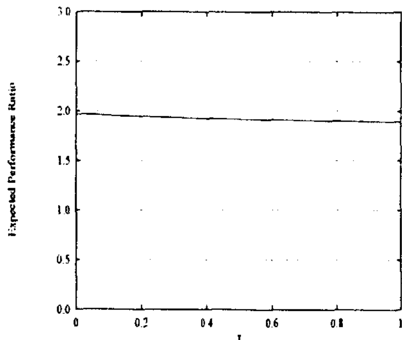
Algorithm		NOSPM
Window Modular Multiplication	Montgomery	$2k^2 + k + r \times (2k^2)$
	Proposed	$k^2 + r \times (k^2 + 3(k+1))$
Modular Squaring	Montgomery	$\frac{3}{2}(k^2 + k) + r \times (\frac{3}{2}k^2 + k)$
	Proposed	$\frac{1}{2}(k^2 + k) + r \times \frac{5}{2}(k^2 + k)$

Figure 3 is the graph to represent changes of expected performance ratios of proposed algorithms to Montgomery algorithms according to r when the value of k is 32. The expected performance ratio is defined as follows.

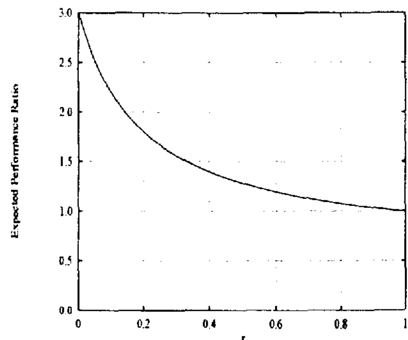
Definition 4. The *expected performance ratio* of the proposed algorithm to Montgomery algorithm

$$= \frac{\text{NOSPM required to execute Montgomery algorithm}}{\text{NOSPM required to execute the proposed algorithm}}$$

NOSPM has the same meaning as that used in Table 3.



(a) *Window Modular Multiplication*



(b) *Modular Squaring*

Fig. 3. *Change of the expected performance ratio according to r*

As we can see in Figure 3(a), the performance ratio of the proposed window modular multiplication algorithm to the Montgomery algorithm is little changed although the r is largely changed. This means that the proposed window modular multiplication algorithm is faster than the Montgomery algorithm by two times in almost systems. However, the performance ratio in Figure 3(b) is largely changed according to r. That is, the performance of the proposed modular squaring algorithm is largely dependent on the system on which the algorithm is implemented. For example, as the value of r is about 0.37 in the case of Pentium PC, the proposed modular squaring algorithm is faster than the Montgomery algorithm by 30% on it.

6 Conclusion

We proposed two algorithms to execute modular multiplications fast. One is the window modular multiplication algorithm and it uses the feature of the addition-chain found with the small-window method. The other is the modular squaring algorithm and it uses the fact that a modular reduction can be executed easily for an integer which is not much larger than a modulus. Proposed algorithms require single-precision multiplications 1/2 and 1/3 times of that required for Montgomery algorithms, respectively. Implementing on PC, proposed algorithms reduce execution times by 50% and 30% compared with Montgomery algorithms, respectively.

References

1. J.Bos, M.Coster: Addition chain heuristics. Crypto'89, 400 407 (1989)
2. D.E.Knuth: The art of computer programming Vol.2. Addison–Wesley,Inc. (1981)
3. M.J.Coster: Some algorithms on addition chains and their complexity. CWI Report CS–R9024 (1990)
4. Y.Yacobi: Exponentiating faster with addition chains. Eurocrypt'90, 222–229 (1991)
5. P.Downey, B.Leong, R.Sethi: Computing sequences with addition chains. SIAM J. Comput., vol.10, NO.3, August, 638–646 (1981)
6. J.Jedwab, C.J.Mitchell: Minimum weight modified signed-digit representations and fast exponentiation. Electronics Letters, vol.25, 1171–1172 (1989)
7. A.Selby, C.Mitcheil: Algorithms for software implementations of RSA. IEE Proceedings(E), vol.136, NO.3, May", 166 170 (1989)
8. W.Diffie, M.E.Hellman: New directions in cryptography. IEEE Trans. Computers, vol.IT-22, NO.6, June, 644–654 (1976)
9. W.Diffie: The first ten years of public-key cryptography. Proceeding of the IEEE, vol.76, NO.5, May, 560–576 (1988)
10. R.L.Rivest, A.Shamir, L.Adleman: A method for obtaining digital signatures and public key cryptosystems. CACM, vol.21, 120–126 (1978)
11. T.ElGmal: A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, vol.IT-31, NO.4, 469–472 (1985)

12. P.L.Montgomery: Modular multiplication without trial division. Mathematics of Computation, vol.44, 519–521 (1985)
13. P.Findlay, B.Johnson: Modular exponentiation using recursive sums of residues. Crypto'89, 371–386 (1990)
14. A.Bosselaers, R.Govaerts, J.Vandewalle: Comparison of three modular reduction functions. Crypto'93, 175–186 (1994)
15. S.Kawamura, K.Takabayashi, A.Shimbo: A fast modular exponentiation algorithm. IEICE Transactions., vol.E-74, NO.8, August, 2136–2142 (1991)
16. H.Morita, C.Yang: A modular-multiplication algorithm using lookahead determination. IEICE Trans. Fundamentals, vol.E76-A, NO.1, January, 70–77 (1993)
17. P.Barrett: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. Crypto'86, 311–323 (1987)
18. S.R.Dusse, B.S.Kaliski: A cryptographic library for the motorola DSP56000. Eurocrypt'90, 230–244 (1991)