A High-Speed Processing LSI for RSA Cryptograms using High-Radix Signed-Digit Numbers and a new algorithm of modulo operation

Yoshinori Fujisawa
Nagano National College of Technology
Nagano-shi Tokuma 716
Nagano-ken 381-8553 JAPAN

Yasushi Fuwa
Shinshu University
Nagano-shi Wakasato 4-17-1
Nagano-ken 380-8553 JAPAN

Abstract

In this work, we developed a high speed LSI for encoding and decoding the RSA cryptogram and describe the processing method in this paper. This cryptogram is used not only for encrypting data, but also for such purposes as authentication. However, the RSA encoding and decoding processes take a long time because they require a great deal of calculations. As a result, this cryptogram is not suited for practical use. In order to make a high-speed processing method, we introduce the following ideas: 1. To reduce the number of summation operations, we increase the number of coding bits used to represent a digit. 2. We propose a high-speed addition operation for handling the case in which each digit has a large number of bits. 3. We guarantee the value of modulo operations by determining the possible range and create parallel subtraction circuits as a result. By applying these concepts, we are able to reduce processing times comparing to the previous methods. We also developed the LSI to realize our proposed algorithm.

1 Introduction

Recently, the development of computer technology is progressing very quickly and the polarization of the Internet is spreading in general. Various cash transactions using the Internet have begun. On the other hand, computer crimes of illegal access, spoofs, and so on have been increasing for a few years. The popularization of cash transactions is disrupted by these crimes. The cryptosystem is one of the means to cope with these crimes.

There are public-key and secret-key cryptograms in a cryptosystem. The public-key cryptograms have drawn attention compared with secret-key cryptograms because in addition to the encoding and decoding functions, they can also be used in authentication applications.

However, in a public-key cryptogram, the encryption and decryption take a long time because they require a great deal of calculations making it unsuitable for practical use. On the other hand, the number of coding bits used to represent an encryption and decryption key has a tendency to increase when improving crypto-strength. As a result, the problem of increasing encryption time is taken seriously.

In this work, we developed the high speed LSI processing for public-key cryptogram. When both encryption and decryption are executed in a public-key cryptogram it is necessary to calculate powers and remainders with large numbers. These calculations can be realized by repeated additions and subtractions. So, reducing the number of repetitions is significant in improving the speed of processing time. Until now, many scholars researched speed improvement methods for public-key cryptograms[2] and a number of researchers have developed some hardware for encryption and decryption processes using these results. However, a feasible processing speed has not yet been attained. It is necessary to create a faster implementation for public-key cryptograms to be used more popularly.

In this paper, we propose a new method of high-speed encryption and decryption for the public-key cryptogram. This method is considered on the basis of three ideas as follows:

1. To reduce the number of summation operations, we increase the number of coding bits used to represent a digit.

2. We propose a high-speed addition processing method in the case that the number of coding bits used to represent each digit is large.

3. We guarantee the value of remainders to be within a constant range. Using this result, we can realize parallel subtraction processes.

We can achieve higher speeds of encryption and decryption processing than in previous works with these ideas.

In Section 2, we explain our target public-key cryptogram. Next, we propose new high-speed encryption and decryption processing methods. In Section 5, we introduce our high speed LSI processing for the RSA cryptogram. Finally, we evaluate our proposal against previous methods.

2 Outline of RSA Cryptosystem

Among the various public-key cryptograms, the RSA cryptosystem[3], [7], [8], [9] invented by *Rivest*, *Shamir* and *Adleman* in 1977 is considered the most powerful. In this cryptosystem, the encryption key is a pair (e, F) and the decryption key is a pair (d, F) where component e is called a public key and d is called a private key.

Let the plain text be M and the cipher text be C. Then the RSA encryption and decryption algorithms are given by

$$Encryption : C = M^e \bmod F. \tag{2.1}$$

$$Decryption : M = C^d \bmod F. \tag{2.2}$$

where the range of M and C is between 0 and F-1. In (2.1) and (2.2), the keys e, d and F are determined as follows:

1. Choose two large prime numbers p and q.
2. Calculate $F = p \times q$.
3. Calculate $L = LCM((p-1),(q-1))$.
4. Choose e that satisfies the following two conditions, $GCD(L,e) = 1$ and $(1 < e < L)$.
5. Calculate d which satisfies the following condition, $e \times d \bmod L = 1$.

The security of an RSA cryptosystem depends on the complexity of calculation in factorizing F into p and q. Consequently, the value of F is increased for improving the safety. The inventors of the RSA cryptosystem recommend selecting p and q as having more than 100 digits in decimal representation. Therefore, the factorized F is more than 200 digits in decimal representation. On the other hand, the complexity of calculation for encryption and decryption is also increased for a larger F, so the processing time becomes longer.

In an RSA cryptosystem, the algorithm of encryption and decryption uses the same algorithm as (2.1) and (2.2). In this paper, we propose a high-speed processing algorithm for encryption. However, the algorithm can also be used for decryption.

3 Proposed Processing Algorithm

In the most powerful processing method[2] among previously developed methods, a radix-4 signed-digit (SD) system is used. Propagations of carry and borrow do not occur in this system.

A radix-4 SD number system[1] is a representation method of numbers proposed by A. Avizienis. In a radix-4 SD number representation, the signs of each digit are in symmetry. For instance, an integer X is represented by a radix-4 SD number of n digits as follows:

$$\begin{aligned} X &= x_{n-1} \times 4^{n-1} + x_{n-2} \times 4^{n-2} + \cdots \\ &\quad \cdots + x_1 \times 4^1 + x_0 \times 4^0 \\ &= \sum_{i=0}^{n-1} x_i \times 4^i. \quad (x_i \in \{\bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}, \bar{x} = -x) \end{aligned}$$

Here, X is between $-4^n + 1$ and $4^n - 1$.

However, the value of e, F requires more than 1,000 bits in an RSA cryptosystem. So, the problem is processing time. Because it is assumed that (2.1) is calculated using a radix-4 SD system, the number of additions is more than 1.5 million times. In this section, we introduce a new processing method to further reduce the number of addition operation. This method is based on the following ideas.

1. To reduce the number of repetitions, let us consider using a higher radix SD number than a radix-4 or radix-8 SD number. We adopt a general radix-2^k SD number for the new algorithm.

2. We design an algorithm that decreases the number of additions using a radix-2^k SD number.

3. Using 2, the number of repetitions becomes larger in the remainder algorithm because the range for the value of R is extended. Therefore, we have to reconsider its algorithm.

Also, we introduce a new function (G(d,k)) that is considered to be a function of the number of digits of F (d) and k. As a result, we can obtain the number of additions from this function. We discuss this function in Section 4.

3.1 General addition of radix-2^k SD numbers

Until now, there were many proposals for algorithms to be used in addition (or subtraction) circuits using not only radix-4 SD numbers but also radix-8 SD numbers. Addition using an SD number system has the property that the carry propagation of each digit is always constant in both radix-4 and radix-8 SD number cases. Therefore, there have been no reports of speed improvement for addition circuits using a high radix SD number system.

However, increasing the number of coding bits used to represent a digit has a vital role in our work in reducing the number of addition repetitions. We designate the number of coding bits used to represent a digit by k. Also, the correctness of addition using a radix-2^k SD number system needs to be clarified. This is discussed in our paper[4].

3.2 Proof of proposed algorithms

There is a discussion about the correctness of our proposed algorithms using radix-2^k SD numbers in our paper [5],[6]. Here, we explain our paper briefly.

Assume that the encryption key e is represented by a radix-2^k SD number $(e_{m-1} \cdots e_0)_{SD}$ such that $e = \sum_{i=0}^{m-1} e_i \times (2^k)^i$. Then (2.1) can be calculated by the following definition and theorem.

Definition 3.1. *For all T_i such that $T_i = M^{e_i} \bmod F$*

$$\begin{aligned} Pow(M,e,F,k)_0 &= T_{m-1} \\ Pow(M,e,F,k)_{i+1} &= (((Pow(M,e,F,k)_i)^{2^k} \bmod F) \times T_{m-i-1} \bmod F \end{aligned}$$

The following theorem can be deduced.

Theorem 3.1. $Pow(M,e,F,k)_{m-1} = M^e \bmod F$

Eq.(2.1) is calculated by following algorithm and its correctness is guranteed by definition 3.1 and theorem 3.1.

Algorithm 3.1.

Step 1	:		C	$\leftarrow$	*1*
Step 2	:		i	$\leftarrow$	$m-1$
Step 3	:		j	$\leftarrow$	k
Step 4	:		Q	$\leftarrow$	C
			P	$\leftarrow$	$Q \times C \bmod F$
			C	$\leftarrow$	P
			j	$\leftarrow$	$j-1$
Step 5	:	*If*	j	$\neq$	*0, then go to Step 4.*
Step 6	:	*If*	e^i	$\neq$	*0, then*
			Q	$\leftarrow$	$M^{e_i} \bmod F$
			P	$\leftarrow$	$Q \times C \bmod F$
			C	$\leftarrow$	P
Step 7	:	*If*	i	$\neq$	*0, then* $i \leftarrow i-1$
					go to Step 3.
Step 8	:	*If*	C	$<$	*0, then* $C \leftarrow C+F$.

In Algorithm 3.1, $M^{e_i} \bmod F$ is calculated in Step 6. But, it is assumed that $M^2 \bmod F \sim M^{2^k-1} \bmod F$ is calculated in advance. In this algorithm, the calculation of the following expression needs a long processing time:

$$P = Q \times C \bmod F. \tag{3.3}$$

Equation(3.3) can be calculated by the following algorithm. Assume that C is represented by a radix-2^k SD number $(c_{m-1} \cdots e_0)_{SD}$ such that $C = \sum_{j=0}^{m-1} c_j \times (2^k)^i$.

Definition 3.2. *For all U_j such that $U_j = (Q \times c_j) \bmod F$*

$$\begin{aligned} Mul(Q,c,F,k)_0 &= U_{m-1} \\ Mul(Q,c,F,k)_{j+1} &= ((2^k \times Mul(Q,c,F,k)_j) + U_{m-j-1} \bmod F \end{aligned}$$

The following theorem can be deduced.

Theorem 3.2. $Mul(Q,c,F,k)_{m-1} = (Q \times C) \bmod F$

Eq.(3.3) is calculated by following algorithm and its correctness is guranteed by definition 3.2 and theorem 3.2.

Algorithm 3.2.

Step 1	:		P	$\leftarrow$	*0*
			J	$\leftarrow$	n
			G	$\leftarrow$	*1*
Step 2	:		J	$\leftarrow$	$J-1$
Step 3	:		G	$\leftarrow$	$Q \times c_J \bmod F$
Step 4	:		R	$\leftarrow$	$2^k \times P + G$
Step 5	:		P	$\leftarrow$	$R \bmod F$
Step 6	:	*If*	J	$=$	*0, then stop.*
					Otherwise, go to Step 2.

In Algorithm 3.2, it is assumed that $Q \times c_J \bmod F$ ($c_J = -2^k + 1 \sim 2^k - 1$) is calculated in advance.

The number of repetitions can be reduced in Algorithms 3.1, 3.2 if the value of k is increased because the number of repetitions m depends on radix number 2^k. If the value of k increases, then the value of m decreases.

3.3 Calculation for Remainder

In this section, we introduce two methods for calculating remainder results.

3.3.1 Calculation using parallel processes

In the first algorithm proposed for calculating remainders, the range of values for R becomes wide in Step 4 of Algorithm 3.2. This is a problem since the number of repetitions increases for obtaining the value of $R \bmod F$. To solve this problem, we introduce a new calculation method of $R \bmod F$.

The value of R is within the range of 0 and $(2^k + 1) \times F$ in Algorithm 3.2. $2^i \times F$ $(i = 1, 2 \cdots, k)$ can be calculated in advance because the value of F is fixed. So, $R \bmod F$ can be calculated by the circuit in Fig.1.

The addition processing time does not depend on the number of digits and the number of coding bits in a radix-2^k SD number representation. Then, it can be assumed that the total processing time is the same as that for obtaining the value of one $R \bmod F$ calculation.

3.3.2 Sequential calculation

The previous algorithm needs many addition circuits and it is difficult to realize it in a LSI because of its limited gates. So, we propose another algorithm for remainders using only one addition circuit. This algorithm uses a fact that the value of R is within the range of 0 and $(2^k - 1) \times F$. When the operation of subtract F from R using addition circuit is repeated until the result is smaller than F, the maximum number of subtractions is $2^k - 1$. To reduce the number of subtractions, we use a binary search technique and obtain the following algorithm.

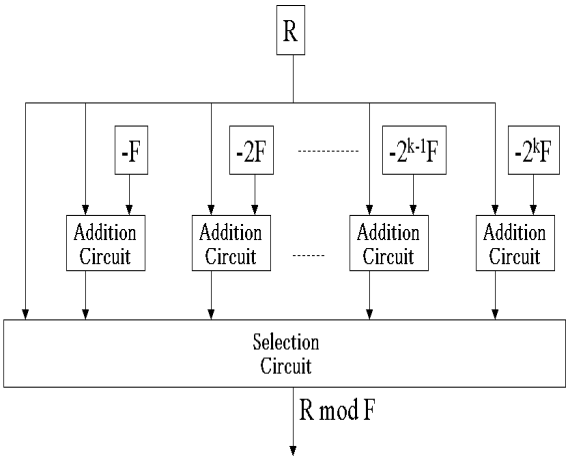

Figure 1: Block diagram for the calculation circuit of $R \bmod F$.

Algorithm 3.3.

$$
\begin{array}{llllll}
\textit{Step 1} & : & & n & \leftarrow & 2^k \\
\textit{Step 2} & : & & t & \leftarrow & R - n \times F \\
\textit{Step 3} & : & \textit{If} & t & > & \textit{0, then } R \leftarrow t. \\
\textit{Step 4} & : & \textit{If} & n & = & \textit{1, then stop.} \\
 & & & & & \textit{Otherwise, } k \leftarrow k-1 \\
 & & & & & \textit{go to Step 2.}
\end{array}
$$

In Algorithm 3.3, $n \times F (n = 2^k, 2^{k-1}, \cdots, 2^1, 2^0)$ is calculated. However, this calculation needs almost no processing time because this can be realized by bit shifts. Then, the result of remainder calculations can be obtained by k addition.

4 Evaluation

In this section, we evaluate our proposed processing method. As a method of evaluating the processing time, we compare the number of additions (subtraction is considered as the same process) in a previous algorithm with the proposed algorithm. We compare the cases of coding bit lengths of keys being 512 bits. In the case of the previous algorithm, the coding bit length of each digit is fixed. The number of additions is not constant because the calculation of remainders depends on the bit pattern used. So, we used the average number of additions in the previous algorithm for comparison. In the case of our proposed algorithm, the number of additions depends on the value of k. So, we discuss the optimum bit lengths for k (function G(d,k)). Also, we use the worst value of the key because the number of additions depends on the coding bit pattern of the key used.

Assume that e is L bits. Then, the number of repetitions of (3.3) is $[L/k]$ ([M] is the smallest integer greater than M) in Algorithm 3.1. Here, assume that $[L/k]$ is the number of digits d. In one repetition, (3.3) is calculated k times in Step 4 and 1 time in Step 6. It

is necessary to calculate the values of M^2 mod F, M^3 mod F, ... , M^{2^k-1} mod F in advance before the execution of Algorithm 3.1. These expressions can be calculated $2^k - 1$ times in the same way as (3.3). As a result, (3.3) is calculated a total $d \times (k+1) + 2^k - 1$ times.

Eq.(3.3) can be calculated using Algorithm 3.2. In this algorithm, $Q \times c_J$ mod F ($c_J = -2^k + 1 \sim 2^k - 1$, expect for the cases of $c_J = -1, 0, 1$) is calculated before execution. In the cases of $c_J = 2 \sim 2^k - 1$, the number of addition repetitions is $2^k - 2$ times. Assume that Q, C and F are L bits constantly in Algorithm 3.2. Then, the number of digits n is represented by d digits. So, $2^k \times P + G$ in Step 4 and R mod F in Step 5 are calculated d times for each calculation.

In the case of first remainder method in Section 3.3.1, R mod F can be realized by a parallel processing circuit. In this circuit, processing time is equivalent to one addition processing time. To calculate (3.3), addition is executed $2^k - 2 + 2 \times d$ times. Also, it is necessary to calculate $-2 \times F \sim -2^k \times F$ in advance before using the parallel processing circuit. These results can be obtained by addition processing of $2^k - 2$ times before (2.1) and (2.2) are calculated. So, we can obtain the number of additions by the following function (G(d,k)). Here, the number of coding bits used to represent a digit is k bits.

$$\begin{aligned} G(d,k) &= (d \times (k+1) + 2^k - 1) \\ &\times (2^k - 2 + 2 \times d) + (2^k - 2). \end{aligned} \tag{4.4}$$

In the case of the second remainder calculation method in Section 3.3.2, R mod F can be realized by sequential processing. The number of additions depends on k. So, we can obtain the number of additions by the following function (G(d,k)).

$$\begin{aligned} G(d,k) &= (d \times (k+1) + 2^k - 1) \\ &\times (2^k - 2 + (k+1) \times d) + (2^k - 2). \end{aligned} \tag{4.5}$$

Fig.2 based on (4.4), (4.5) and the previous method of processing shows the change in the number of additions for various key lengths. Here, the number of additions using the first remainder method (proposed algorithm1) is indicated with a solid line, the second remainder method (proposed algorithm2) is indicated with a bold line, and the previous algorithm result is indicated with a broken line. The dot on the broken line is the average number of additions.

As seen from this diagram, $k = 5 \sim 6$ is the most suitable value in the proposed algorithms. As a result, we clarified that the processing time become higher than the previous method using our proposed algorithm.

5 LSI for RSA Cryptograms

In this work, we developed LSI to realize the proposed algorithm shown in Fig.3. In this LSI, we used proposed algorithm2 (Algorithm 3.3) because of the smaller number of gates

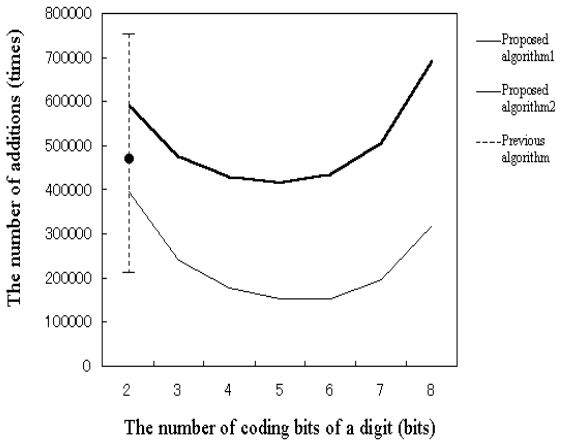

Figure 2: Change in the number of additions with a 512 bit key length.

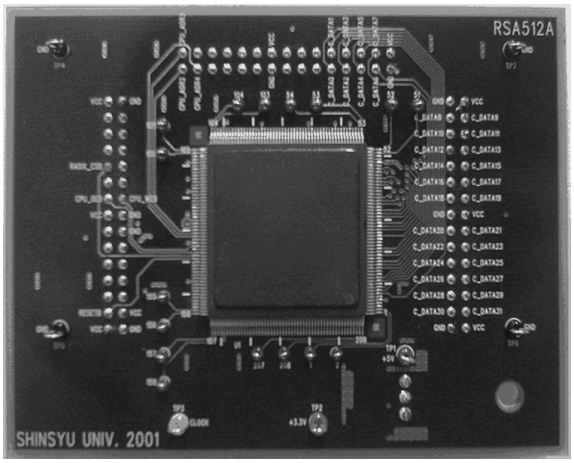

Figure 3: High speed LSI for RSA cryptogram.

needed.

The operation specifications of this LSI are as follows.

Coding bits of a digit	5 bits
Number of key bits	512 bits
Active frequency	38 MHz
Number of Gates	440,000 gates
Processing performance	20,271 bits/sec
(case of 512 bits of key)	(see Fig.4)
Type of a wafer	880KGate Standard Cell

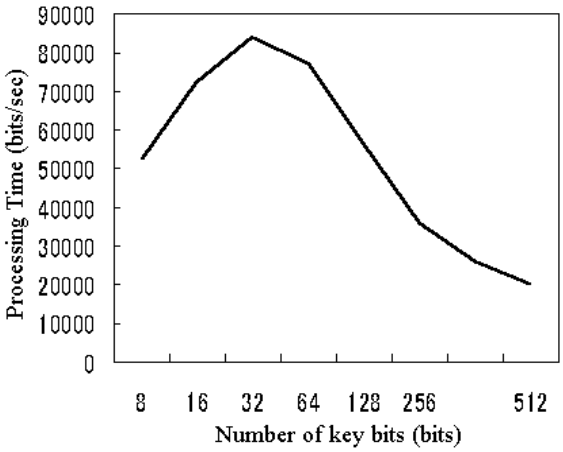

Figure 4: Performance of LSI.

6 Conclusion

In this paper, we proposed two new processing algorithms for an RSA public key cryptogram system. Then, we developed LSI to realize our proposed algorithm using a sequential processing algorithm for calculating remainders. In our future work, we will develop a faster algorithm for remainders and the corresponding LSI.

References

[1] A.Avizienis, "Signed-digit number representations for fast parallel arithmetic", IRE Trans.Elect.Comput., EC-10, pp.389-400, 1961.

[2] M. Kameyame, S. Wei, T. Higuchi, "Design of an RSA encryption processor based on Signed-Digit multivalued arithmetic circuits", Trans.(D), I.E.I.C.E., Japan,Vol.J71-D, No.12, pp.2659-2668, 1988.

[3] Yoshinori Fujisawa, Yasushi Fuwa, Hidetaka Shimizu, "Public-Key cryptography and pepin's test for the primality of Fermat numbers", Formalized Mathematics, Vol.7(2), pp.317-321, 1998.

[4] Yoshinori Fujisawa, Yasushi Fuwa, "Definitions of Radix-2^k Signed-Digit number and its adder algorithm", Mechanized Mathematics and Its Applications, Vol.1(1), pp.11-20, 2000.

[5] Yasushi Fuwa, Yoshinori Fujisawa, "High-speed algorithms for RSA cryptograms", Formalized Mathematics, in press.

[6] Yoshinori Fujisawa, Yasushi Fuwa, "Proposal of a High-Speed Processing Method for RSA Cryptograms using High-Radix Signed-Digit Numbers", Personal Computer Users' Application Technology Association, Vol.10(1), pp.61-70, 2000.

[7] Kineo Matsui, "Cryptographic algorithm", Morikita Publishing Co., Ltd., Japan, 1987.

[8] Shinichi Ikeno, Kenji Koyama, "Modern cryptosystem [Gendai angouriron]", I.E.I.C.E., Japan, 1995.

[9] Kouichi Sakurai, "CRYPTOGRAPHY : Theorey and Practice", Kyouritsu Publishing Co., Ltd., Japan 1996.