数理解析研究所講究録
第1186巻 2001年 164-176



# Correctness of adder algorithm using high-radix signed-digit number system and its adaptation to RSA cryptogram

信州大学工学部 藤澤義範 (Yoshinori Fujisawa)
信州大学工学部 不破泰 (Yasushi Fuwa)
信州大学工学部 中村八束 (Yatsuka Nakamura)

## ABSTRACT

In this paper, we propose a new high-speed processing method for encoding and decoding the *RSA* cryptogram which is a kind of public-key cryptogram. This cryptogram is not only used for encrypting data, but also for such purposes as authentication. However, the 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 to approximately $1/3 \sim 1/4$ of the time of previous methods.

## 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, spoof 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 become a matter of great concern compared with secret-key cryptograms because of an authentication can be performed, kinds of private keys are not required, and the transfer of the pri-

vate key is unnecessary. However, in a public-key cryptogram, the encryption and decryption take a long time because they require a great deal of calculations and this cryptogram is not suited 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 a public-key cryptogram, when both encryption and decryption are executed it is necessary to calculate powers and remainders with large numbers. Their calculation can be realized by addition and subtraction repetitions. 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]. We can obtain some good results. Reserchers 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 algorithm for 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 this paper, we explain our target public-key cryptogram in Section 2 and previous processing methods in Section 3. Next, in Section 4, we propose the new high-speed encryption and decryption processing methods. Finally, we evaluate our proposal against previous methods.

## 2 Outline of $RSA$ Cryptosystem

There are various public-key cryptograms. Among these, especially, the $RSA$ cryptosystem[3], [6], [7] invented by *Rivest*, *Shamir* and *Adleman* in 1977 is considered as the most powerful. In this

cryptosystem, the encryption key is a pair $(e, F)$ and the decryption key is a pair $(d, F)$. The 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

$$Encryption : C = M^e \bmod F \quad (1)$$

$$Decryption : M = C^d \bmod F \quad (2)$$

where the range of $M$ and $C$ is between 0 and $F$-1. In Eqs. (1) and (2), 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 $q$. Consequenty, 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 speed becomes slower.

In an $RSA$ cryptosystem, the algorithm of encryption and decryption uses the same algorithm as Eqs.(1) and (2). In this paper, we propose the high-speed processing algorithm for encryption. However, the algorithm can also be used for decryption.

# 3 Previous Processing Method

In this section, we introduce the most powerful processing method[2] among previously developed methods. Assume that the encryption key $e$ is represented by a radix-4 number of m digits $(e_{m-1} \cdots e_0)_4$ $(e_i \in \{0, 1, 2, 3\})$ such that $e = \sum_{i=0}^{m-1} e_i 4^i$. Then Eq.(1) can be calculated by the following expression.

$$\begin{aligned} M^e \bmod F &= M^{\left[\sum_{i=0}^{m-1} e_i 4^i\right]} \bmod F \\ &= (M^{e_0} \times M^{4e_1} \times \\ &\quad \cdots \times M^{4^{m-1}e_{m-1}}) \bmod F \\ &= ((M^{e_0} \bmod F) \times \\ &\quad \cdots \times (M^{4^{m-1}e_{m-1}} \bmod F)) \bmod F \end{aligned}$$

The value of Eq.(1) can be obtained by the following algorithm.

**Algorithm 3.1**

*Step 1* : $C \leftarrow 1$

*Step 2* : $i \leftarrow m-1$

*Step 3* : $Q \leftarrow C$

$P \leftarrow Q \times C \bmod F$

$Q \leftarrow P$

$P \leftarrow Q \times C \bmod F$

$C \leftarrow P$

*Step 4* : *If* $e_i \neq 0$, *then*

$Q \leftarrow M^{e_i} \bmod F$

$P \leftarrow Q \times C \bmod F$

$C \leftarrow P$

*Step 5* : *If* $i \neq 0$, *then* $i \leftarrow i-1$, *go to Step3.*

*Step 6* : *If* $C < 0$, *then* $C \leftarrow C+F$.

In this algorithm, the calculation of the following expression needs a long processing time:

$$P = Q \times C \bmod F \tag{3}$$

Reducing the number of calculations of Eq.(3) has a meaning for improving the speed of Algorithm 3.1. In Step 4 of this algorithm, $M^{e_i} \bmod F$ is calculated. But, assume that $M^{e_i} \bmod F (\, e_i \in \{0,1,2,3\})$ is calculated in advance. Here, it is not necessary to calculate $M^0 \bmod F$ and $M^1 \bmod F$ because $M$ is lower than $F$-1.

Eq.(3) contains multiplication and remainder calculations with a large number. These calculations are realized by addition and subtraction. However, increased processing time is a problem because of carry and borrow propagation. Therefore, a radix-4 signed-digit (SD) number system, in which propagation of carry and borrow does not occur, is used in the previous method.

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

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

Here, $X$ is between $-4^n+1$ and $4^n-1$. Then, the calculation algorithm of Eq.(3) is shown in Algorithm 3.2 in a radix-4 SD number representation.

Here, $Q$ and $C$ are integers between $-F$ and $F$. It is assumed that these are represented by a radix-4 SD number of $n$ digits. It is also assumed that $F$ is represented by a radix-4 number of $n$ digits.

**Algorithm 3.2**

*Step 1* : $P \leftarrow 0$

$J \leftarrow n$

*Step 2* : $J \leftarrow J-1$

*Step 3* : $G \leftarrow Q \times c_J \bmod F$

*Step 4* : $R \leftarrow 4P+G$

*Step 5* : $P \leftarrow R \bmod F$

*Step 6* : *If* $J = 0$, *then stop*

*otherwise, go to Step 2.*

In this algorithm, $Q \times c_J \bmod F$ is calculated in Step 3. But, we can calculate $Q \times c_J \bmod F$( $c_J \in \{-3, -2, -1, 0, 1, 2, 3\}$) in advance. However, it is not necessary to calculate $Q \times \pm 1 \bmod F$ and $Q \times 0 \bmod F$. $R \bmod F$ in Step 5 can be calculated with the following algorithm. Here, it is assumed that $2F$ is calculated in advance.

**Algorithm 3.3**

*Step 1* : *If* $|R| < F$, *then stop*

$S \leftarrow SIGN(R)$

$R \leftarrow R - 2F \times S$

*Step 2* : *If* $|R| < F$, *then stop*

$S \leftarrow SIGN(R)$

$R \leftarrow R - F \times S$

*Return to Step 2.*

In this algorithm, the function $SIGN(x)$ differentiates positive and negative signs of integer $x$ and is defined as follows:

**Definition 3.1** *For all integers* $x$

$$SIGN(x) = \begin{cases} -1 & ,x < 0 \\ 1 & ,x \geq 0 \end{cases}$$

Using the fact that the result is guaranteed to be between $-F + 1$ and $F - 1$, the value of $R \bmod F$ is obtained to subtract $2F$ from $R$ (if $R$ is negative, then add $2F$ to $R$) in Step 1. In Step 2, the result is obtained to subtract $F$ from $R$ (if $R$ is negative, then add $F$ to $R$). This algorithm is repeated until the value is between $-F+1$ and $F-1$. Also, an addition (or a subtraction) is done between 0 and 3 times. This is because the result (obtained by Step 4 in Algorithm 3.2) is between -5$F$ and 5$F$.

# 4 New processing method

Eq.(1) contains calculations of powers and remainders with large numbers. These calculations can be realized by algorithms explained in Section 3. 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 Eq.(1) is calculated using Algorithm 3.3, the number of additions is more than 1.5 million times. In this section, let us consider a new processing method to further reduce the number of addition operation. This method is considered based on the following ideas.

1. To reduce the number of repetitions in Algorithm 3.1, 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 Algorithm 3.3 becase the range for value of $R$ is extended. Therefore, we have to reconsider Algorithm 3.3.

## 4.1 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 carry propagation of each digit is always constant in both radix-4 and radix-8 SD number cases. Therefore, there have been no reports on speed improvement addition circuits using a high radix SD number system.

However, increasing the number of coding bits used to represent a digit has a meaning for 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 clarify. There is a discussion about the correctness of addition using radix-$2^k$ SD numbers in our paper[4]. Here, we explain our paper briefly.

To represent a radix-$2^k$ SD number, we considered the following set:

**Definition 4.1** *For all integers k*

$$set\ k\text{-}SD = \{integer\ e\text{:-}2^k + 1 \leq e \leq 2^k\text{-}1\ \}$$

A number represented as a radix-$2^k$ SD number of $n$ digits is a finite sequence in which each digit of the sequence is an element of $k$-SD. Assume that $x$ is represented by a radix-$2^k$ SD number of $n$ digits. Then we show the number of the $i^{th}$ digit of $x$ as $x_i$ ($x_i \in k$-SD).

We defined a function called SDDec($x$) which translates the number represented by a radix-$2^k$ number of $n$ digits to an integer as follows:

**Definition 4.2** *For all integers x such that radix-$2^k$ SD number of n digits*

$$SDDec(x) = \sum_{i=1}^{n} (2^k)^{i-1} \times x_i$$

Conversely, we defined a function called DecSD ($y$,$k$) which translates integers $y$ into a radix-$2^k$ number of $n$ digits as follows:

**Definition 4.3** *For all integers y*

$$DecSD(y,k)_i = (y \bmod (2^k)^i) \div (2^k)^{i-1}$$

Assume that $x$ and $y$ are represented by radix-$2^k$ SD numbers of $n$ digits. Let us consider addition of $x$ and $y$. Data and carry occur from $x_i$+$y_i$. We defined functions to obtain the data and carry, respectively, called $SD\_Add\_Carry(x_i + y_i)$ and $SD\_Add\_Data(x_i + y_i, k)$ as follows:

**Definition 4.4** *For all integers z such that $z = x_i + y_i$*

$$SD\_Add\_Carry(z) = \begin{cases} 1 & ,z > 2 \\ -1 & ,z < -2 \\ 0 & ,otherwise \end{cases}$$

**Definition 4.5** *For all integers $z$ such that $z = x_i + y_i$*

$$SD\_Add\_Data(z,k) = z - SD\_Add\_Carry(z) \times 2^k$$

The following theorem can be easily deduced.

**Theorem 4.1** *If $k \geq 2$, then $-2^k + 2 \leq SD\_Add\_Data\ (x_i + y_i, k) \leq 2^k - 2$*

We defined the function called Add$(x,y,k)$ using Defintion4.4, 4.5 as follows:

**Definition 4.6** *For all natural numbers $k$ and $x,y$ which are represented by a radix-$2^k$ SD number of $n$ digits*

$$\begin{aligned} Add(x,y,k)_i &= SD\_Add\_Data(x_i + y_i, k) \\ &\quad + SD\_Add\_Carry(x_{i-1} + y_{i-1}) \end{aligned}$$

The value of Add$(x,y,k)_i$ is an element of $k$-SD. Then, there is no a carry propagation. This is proved by Theorem 4.1. In view of this result, the additon time is constant as shown in Fig. 1:

In this figure, $d_i$ is SD_Add_Data($x_i + y_i$,k), $c_i$ is SD_Add_Carry($x_i + y_i$) and $s_i$ is Add$(x,y,k)_i$. We proved the following theorem to guarantee the correctness of this addition.

**Theorem 4.2** *For all $x,y$ such that $x,y$ are represented by a radix-$2^k$ SD number of $n$ digits*

$$\begin{aligned} x + y &= (SDDec(DecSD(x,k) \\ &\quad + DecSD(y,k))) + (2^k)^n \\ &\quad \times SD\_Add\_Carry(DecSD(x,k)_n) \\ &\quad + DecSD(y,k)_n) \end{aligned}$$

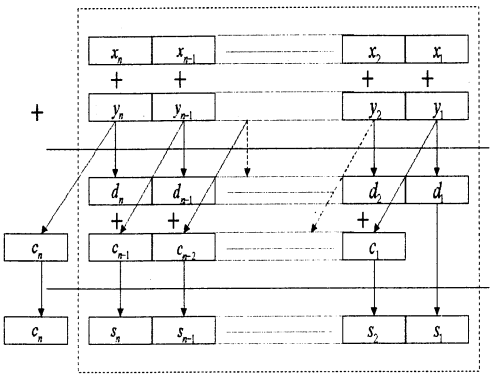


Figure 1: $n$ digit addition based on a radix-$2^k$ SD number.

## 4.2 Reducing the number of repetitions for Eqs.(1) and (3)

There is a discussion about the correctness of algorithms 3.1,3.2 using radix-$2^k$ SD numbers in out paper[5]. Here, we explain out 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 $e = \sum_{i=0}^{m-1} e_i (2^k)^i$. Then Eq.(1) can be calculated by the following definition and theorem.

**Definition 4.7** *For all $T_i$ such that $T_i = M^{e_i} \bmod F$*

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

The following theorem can be deduced.

**Theorem 4.3** $Pow(M, e, F, k)_{m-1} = M^e \bmod F$

Eq.(1) is calculated by following algorithm and its correctness is guranteed by definition 4.7 and theorem 4.3.

**Algorithm 4.1**

*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 4.1, $M^{e_i} \bmod F$ 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.

Then Eq.(3) can be calculated by the following definition and theorem. Assume that the $C$ is represented by a radix-$2^k$ SD number $(c_{m-1} \cdots e_0)_{SD}$ such that $C = \sum_{j=0}^{m-1} c_j (2^k)^i$.

**Definition 4.8** *For all* $U_j$ *such that* $U_j = (Q \times c_j) \bmod F$

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

The following theorem can be deduced.

**Theorem 4.4** $Mul(Q, c, F, k)_{m-1} = (Q \times C) \bmod F$

Eq.(3) is calculated by following algorithm and its correctness is guranteed by definition 4.8 and theorem 4.4.

**Algorithm 4.2**

*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* : $R \leftarrow 2^k P + G$

*Step 5* : $P \leftarrow R \bmod F$

*Step 6* : *If* $J = 0$, *then stop.*

*Otherwise, go to Step 2.*

In Algorithm 4.2, 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 Algorithm 4.1, 4.2 if the value of $k$ is increased because the number of repetitions $m$ depends on radix number $2^k$. If the value of $k$ is increases, then the value of $m$ is decreases.

### 4.3 Introduction of parallel processes to calculate $R$ mod $F$

The range of values for $R$ becomes wide in Step 4 of Algorithm 4.2. Therefore, it is a problem that the number of repetitions increases for obtaining the value of $R$ mod $F$ in Algorithm 3.3. To solve this problem, we introduce a new calculation method of $R$ mod $F$.

The value of $R$ is within the range of $(-2^k+1)F$ and $(2^k+1)F$ in Algorithm 4.2. $2^iF$ $(i = 1, 2 \cdots, k)$ can be calculated in advance because the value of $F$ is fixed. So, $R$ mod $F$ can be calculated by following circuit.

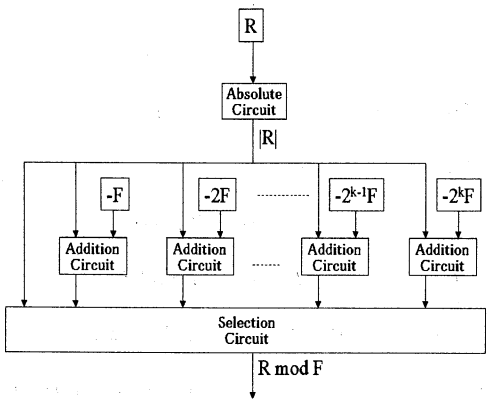


Figure 2: Block diagram for the calculation circuit of $R$ mod $F$.

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 one processing time for obtaining the value of $R$ mod $F$.

## 5 Evaluation

In this section, we evaluate our proposed processing method. As a method of evaluating the processing time, we compare the number of additions (and subtraction 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, 1024 and 2048 bits. Also, the number of additions depends on the value of $k$ in the proposed algorithms. So, we discuss the optimum bit lengths for $k$. We consider that the number of additions is one time in the case of parallel addition processing. Also, we use the worst value of the key because the number of additions depends on the coding bit pattern of the key used.

### 5.1 The number of additions in the previous algorithm

We obtain the number of calculations of Eq.(3) in Algorithm 3.1. It is assumed that bit lengths of $e$ are $L$ bits. Then the number of repetitions $m$ is represented by $[L/2]$ ($[M]$ is the number raised to unit fractions lower than zero of $M$). In one

repetition, Eq.(3) is calculated 2 times in Step 3 and 1 time in Step 4 of Algorithm 3.1. As a result, Eq.(3) is calculated 3 times every time. Also, it is necessary to calculate Eqs.(4), (5) in advance before executing this algorithm. These are calculated with the following expression:

$$M^2 \bmod F = M \times M \bmod F \quad (4)$$

$$M^3 \bmod F = M \times (M^2 \bmod F) \bmod F \quad (5)$$

Eqs.(4), (5) can be calculated in the same way as Eq.(3). As a result, the number of Eq.(3) is a total $L/2 \times 3 + 2$ times.

Eq.(3) is realized by Algorithm 3.2. In this algorithm, the following calculation is done in advance.

$$Q \times (\pm 2) \bmod F = ((\pm Q) + (\pm Q)) \bmod F \quad (6)$$

$$Q \times (\pm 3) \bmod F = ((Q \times (\pm 2) \bmod F) + (\pm Q)) \bmod F \quad (7)$$

In Eq.(6), $Q + Q$ is calculated and if $Q + Q$ is smaller than $F$, then it is the result. If $Q + Q$ is bigger than $F$, then $Q + Q - F$ is calculated and obtained as the result of Eq.(6). In the cases of negative numbers, we can obtain the result in the same way. The result of Eq.(7) can be obtained using the result of Eq.(6) in the same way. As a result, the number of addition repetitions is 8 times.

In Algorithm 3.2, it is assumed that $Q$, $C$ and $F$ are $L$ bits constantly. Then the number of digits is $n$ for each $[L/2]$. So, $4P + G$ in Step 4 and $R \bmod F$ in Step 5 are calculated $[L/2]$ times for each calculation. $R \bmod F$ can be calculated by Algorithm 3.3. In this algorithm, the number of additions is between 0 and 3. After all, to calculate Eq.(3), addition must be executed $8 + [L/2] \sim 8 + [L/2] \times 4$ times. $2F$ is used in Algorithm 3.3. But, this is not a serious problem because $2F$ can be calculated by only one addition.

To conclude, in the cases of calculating Eq.(1), we can obtain the number of additions by the following expression in the previous Algorithms 3.1, 3.2 and 3.3. In these algorithms, the number of coding bits used to represent a digit is 2 bits.

$$([L/2] \times 3 + 2) \times (8 + [L/2]) + 1 \sim ([L/2] \times 3 + 2) \times (8 + [L/2] + \times 4) + 1 \quad (8)$$

## 5.2 The number of additions in the new algorithms

Assume that $e$ is $L$ bits. Then the number of repetitions of Eq.(3) is $[L/k]$ in Algorithm 4.1. In one repetition, Eq.(3) is calculated $k$ times in Step 4 and 1 time in Step 6. It is necessary to calculate the values of $M^2 \bmod F$, $M^3 \bmod F$, ... , $M^{2^k-1} \bmod F$ in advance before execution of Algorithm 4.1. These expressions can be calculated $2^k - 1$ times in the same way as Eq.(3). As a re-

sult, Eq.(3) is calculated a total $[L/k] \times (k+1) + 2^k - 1$ times.

Eq.(3) can be calculated using Algorithm 4.2. In this algorithm, $Q \times c_J \bmod F$ ($c_J = -2^k + 1 \sim 2^k - 1$, where, $c_J = -1, 0, 1$ is removed) 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 4.2. Then the number of digits $n$ is represented by $[L/k]$. So, $2^k P + G$ in Step 4 and $R \bmod F$ in Step 5 are calculated $[L/k]$ times for each calculation.

$R \bmod F$ can be realized by a parallel processing circuit that we propose in Section 4.3. In this circuit, processing time is equivalent to one addition processing time. After all, to calculate Eq.(3) addition is executed $2^k - 2 + 2[L/k]$ times. Also, it is necessary to calculate $-2F \sim -2^k F$ in advance before using the parallel processing circuit in Section 4.3. These results can be obtained by addition processing of $2^k - 2$ times before Eqs.(1) and (2) are calculated. In the case of Eqs.(1) and (2), we can obtain the number of additions by the following expression. Here, the number of coding bits used to represent a digit is $k$ bits.

$$([L/k] \times (k+1) + 2^k - 1) \times (2^k - 2 + 2[L/k]) + (2^k - 2) \quad (9)$$

## 5.3 Comparison of the number of additions of the proposed algorithm with the previous algorithm

Fig. 3, 4 and 5 based on Eqs.(8) and (9) show the change in the number of additions for various key lengths. Here, the number of additions using the new algorithm is indicated with a solid line and the number using the previous algorithm is indicated with a broken line. In Algorithm 3.3, the number of additions is not constant. Then it is assumed that the average number of additions is 1.5 times in Algorithm 3.3. A dot on the broken line is its number. In the new algorithm, $k = 4 \sim 6$ is the most suitable value as seen from these diagrams. As a result, the number of additions becomes $1/3 \sim 1/4$ the number of previous algorithms. If Eqs.(1) and (2) are calculated using the most suitable value of the key, then the calculation speed is higher than the pre-

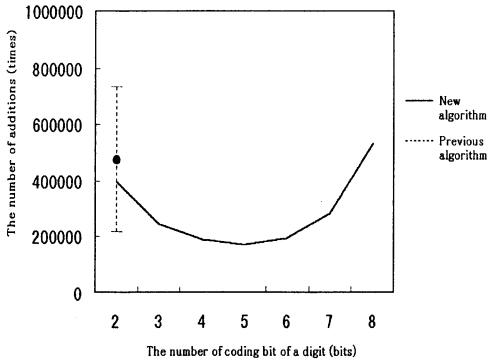


Figure 3: Change in the number of additions with a 512 bit key length.

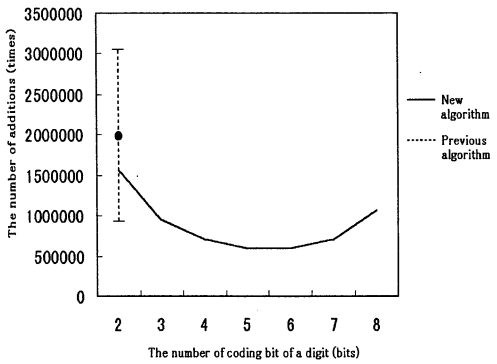


Figure 4: Change in the number of additions with a 1024 bit key length.

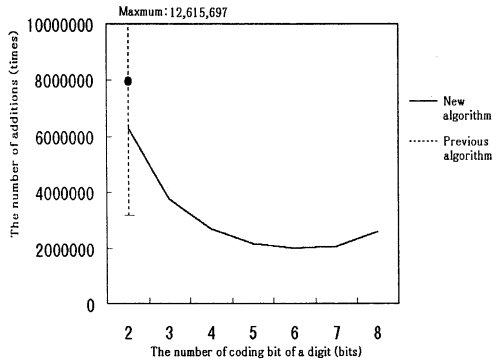


Figure 5: Change in the number of additions with a 2028 bit key length.

vious method.

## 6 Conclusion

In this paper, we described the correctness of addition and its propeties based on a radix-$2^k$ SD number system. Also, we guarantee the value of remainders by determining the possible range and propose parallel subtraction circuits to handle them. We propose a new processing method for an $RSA$ public key cryptogram system which includes our ideas and proved that the new processing method outperforms the previous method. In our future work, we will make the $LSI$ for our proposed algorithm.

## References

[1] A.Avizienis, "Signed-digit number refresentations 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, Vol.2, 2000.

[6] K.Matsui, "Cryptographic Algorithm", Morikita Publishing Co., Ltd., Japan, 1987.

[7] S.Ikeno, K.Koyama, "Modern Cryptosystem (Gendai angouriron)", I.E.I.C.E., Japan, 1995.