High-Speed Implementation Methods for RSA Scheme

Keiichi Iwamura†, Tsutomu Matsumoto††, and Hideki Imai††

† Canon Research Center, 21 Laboratory, 5-1 Wakamiya, Morinosato, Atsugi-shi, Kanagawa 243-01, Japan

†† Yokohama National University, Division of Electrical & Computer Engineering, 156 Tokiwadai, Hodogaya, Yokohama, 240 Japan

Abstract. This paper proposes two novel implementation methods for the RSA cryptographic scheme. (1) The most efficient RSA implementation known to the present authors. This implementation achieves 50 Kbps at about 25 Kgates for a 512-bit exponent e and a 512-bit modulus N. Thus the efficiency is 2.0 bps/gate. (2) A systolic architecture useful for high-speed and efficient and flexible chip implementation of the RSA scheme.

1 Introduction

Modular exponentiations (or powerings) are elementary operations for cryptographic transformations in public-key cryptosystems like the RSA scheme [RSA78]: $C = M^e \bmod N$, the ElGamal schemes, and so on. These cryptosystems' security is based on the difficulty of factoring integers or that of computing discrete logarithms. To achieve enough security level, the word lengths in the modular exponentiations should be significantly greater than those used in conventional general-purpose computer hardwares. The required typical word length is around 512 bits or more. A modular exponentiation can be accomplished by iterating modular multiplications. Thus, for obtaining high-speed implementations of the RSA scheme and the like, it is quite natural to pursue techniques for speeding up modular multiplications for integers of 512 bits or longer.

A lot of efforts have been done in this field [Bric82]-[WQ90]. Reference [Bric89] surveys various hardware implementations of the RSA scheme. We add Table 1 for introducing other implementations, not mentioned in [Bric89],

Table.1: Some of the implementations for the RSA scheme so far presented

Reference	Baudrate	bits	Gates	Reference	Baudrate	bits	Gates
[Miya83]	50K	512	280K	[TAA87]	100K	256	448K
[KMT87]	70K	512	160K	[THAA88]	500K	512	640K
[Mori90]	80K	512	50K	[IWDS91]	64K	512	50K

R.A. Rueppel (Ed.): Advances in Cryptology - EUROCRYPT '92, LNCS 658, pp. 221-238, 1993.

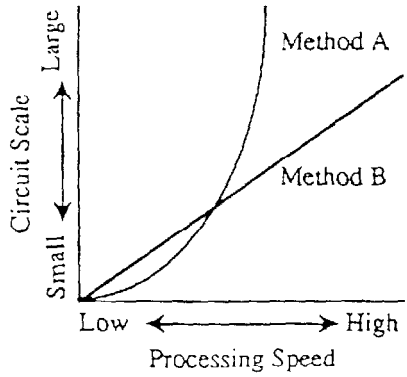

Fig.1: Efficiency and Flexibility

with known circuit scale (Gates) and known processing speed (Baudrate). It should be noted that in Table 1 [Mori90] shows an implementation only for modular multiplication and [TAA87] presents an RSA implementation for a 256-bit exponent and a 256-bit modulus. If we define the efficiency of implementation as

$$\text{(efficiency of implementation)} = \text{(processing speed)/(circuit scale)}$$

then many of hardware implementations so far reported are not so efficient. This measure may be appropriate from the practical view point of achieving high-speed and smaller circuits. The better efficiency is realized by the implementation with the smaller circuit scale and the higher processing speed.

We define another measure called the flexibility, which represents the degree of linearity between the processing speed and the circuit scale. To illustrate this notion, we examine two virtual implementations named Method A and Method B, each of which relationship between the circuit scale and the processing speed is given in Fig.1. We consider that Method A is not flexible because it is not efficient in the wide range of processing speed. In contrast, Method B is flexible since it realizes implementations with the same efficiency in the wide range of processing speed. In this paper we propose two implementation methods for modular multiplication from these points of view.

Section 2 shows a modular multiplication algorithm and circuit with the best efficiency rating of all algorithms and circuits introduced to date and known to the present authors. Based on this Section 2 described an RSA implementation achieving 50 Kbps at about 25 Kgates for a 512-bit exponent e and a 512-bit modulus N. Thus the efficiency is 2.0 bps/gate.

Section 3 presents a flexible modular multiplication algorithm based on systolic array, a way of parallel processing. This high speed processing method utilizes a pipeline processing by means of several types of processing elements (henceforth abbreviated as PEs) each of which is controlled locally and regularly. Also the algorithm is suitable for chip implementations because a chip can be constructed simply and regularly with favorite circuit scale and because

the RSA processing speed increases in proportion to the number of chips. This implementation can allow an increase in speed based on the Chinese remainder theorem in the same chip implementation because it permits variable bit number for the encryption keys.

Lastly in Section 4, the results of this paper are compared with previously obtained results and are shown to be better from the practical view point. See Fig.12. Parallel processing applied to the RSA scheme gives an architecture that allows for the increasing of speed for processing and the reduction in size of the circuit scale systematically.

2 An Efficient Implementation for RSA Scheme

2.1 An Efficient Modular Multiplication Circuit

Consider the modular multiplication $R = A \cdot B \bmod N$ (with N, A, B, R being $k = m \cdot n$ bit integers). We express A in binary and B and R in radix $X = 2^m$ as follows,

$$\begin{aligned} A &= A_{k-1} \cdot 2^{k-1} + A_{k-2} \cdot 2^{k-2} + \cdots + A_1 \cdot 2 + A_0 && (1)\\ B &= B_{n-1} \cdot X^{n-1} + B_{n-2} \cdot X^{n-2} + \cdots + B_1 \cdot X + B_0 && (2)\\ R &= R_{n-1} \cdot X^{n-1} + R_{n-2} \cdot X^{n-2} + \cdots + R_1 \cdot X + R_0 && (3) \end{aligned}$$

where $A_{k-j} \in \{0,1\}$ $(j = 1, \cdots, k)$ and B_i, $R_i \in \{0,1\}^m$ $(i = 0, \cdots, n-1)$. Then the modular multiplication can be calculated by the consecutive execution of the following two algorithms:

ALGORITHM 1A
(Input: $A_0, \cdots, A_{k-1}$; $B_0, \cdots, B_{n-1}$; N)
(Output: $R_{k,0}, \cdots, R_{k,n-1}$)

$D_{0,i} = 0$; $C_{0,i} = 0$
FOR $j = 1$ **TO** k
  $E_{j-1} = E_{j-1,n-1} \cdot X^{n-1} + \cdots + E_{j-1,0} = (C_{j-1,n-1} \cdot X^n) \bmod N$ (4)
  FOR $i = 0$ **TO** $n-1$
    $R_{j,i} = D_{j-1,i} \cdot 2 + C_{j-1,i-1} + A_{k-j} \cdot B_i + E_{j-1,i}$
    $D_{j,i} = dw_{m-1}(R_{j,i})$
    $C_{j,i} = up_{m-1}(R_{j,i})$
  NEXT
NEXT

where $dw_a(Z) = Z \bmod 2^a$ and $up_a(Z) = (Z - dw_a(Z))/2^a$.

ALGORITHM 1B
(Input: $R_{k,0}, \cdots, R_{k,n-1}$; N)
(Output: $R_0, \cdots, R_{n-1}$)

$$\begin{aligned} R &= R_{n-1} \cdot X^{n-1} + R_{n-2} \cdot X^{n-2} + \cdots + R_1 \cdot X + R_0 \\ &= (R_{k,n-1} \cdot X^{n-1} + R_{k,n-2} \cdot X^{n-2} + \cdots + R_{k,1} \cdot X + R_{k,0}) \bmod N \qquad (5) \end{aligned}$$

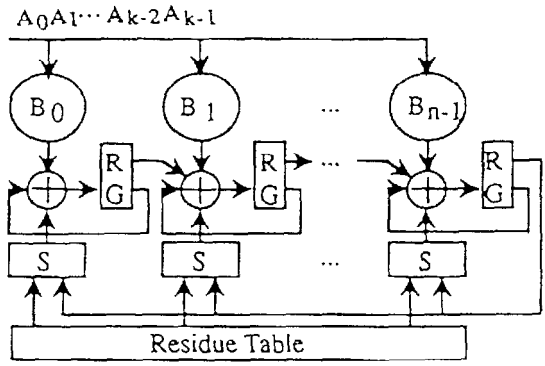

Fig.2: An efficient modular multiplication circuit

Fig.2 illustrates the principal part of our modular multiplication circuit, namely the circuit implementing ALGORITHM 1A. The index i goes from left ($i = 0$) to right ($i = n - 1$) and corresponds to the position of the register, multiplier, adder and the modular-reduction circuit, whereas the value j corresponds to the clock. Each circuit except registers is realized as below.

1) Multiplication: the multiplier is easily realized by AND gates which output B_i only when A_{k-j} is 1. Therefore the circuit scale of the multipliers is small.

2) Addition: the adder is realized by 4-input addition functions of which inputs are $A_{k-j} \cdot B_i$ from the multiplier and $E_{j-1,i}$ from the modular-reduction circuit and $D_{j-1,i} \cdot 2$ and $C_{j-1,i-1}$. Since $A_{k-j} \cdot B_i$ and $E_{j-1,i}$ are m-bit and $D_{j-1,i}$ is $(m-1)$-bit, the fact that $C_{j-1,i-1}$ is less than an m-bit value leads that the value enters the register from the adder is $(m+2)$-bit and thus, $C_{j,i}$ is 3-bit. Therefore in the case where $m \geq 3$ the circuit scale for the carry-bit registers is small compared to the whole circuit scale for registers, because there are 2 carry bits for every $m-$bit added result.

3) Modular Reduction: it is easy to see that $C_{j,i}$, the 3-bit carry, doesn't take the value [111] (See Appendix). The residue $E_{j-1,i}$ $(i = 0, \cdots, n-1)$ is derived as Expression (4) from $C_{j-1,n-1}$. Since $C_{j-1,n-1}$ takes 3-bit values [000], [001], $\cdots$, [110], except for [111], each $E_{j-1,i}$ takes only 7 patterns including $E_{j-1,0} = 0$. These residue patterns can be selected by the selectors S in Fig.2. Therefore the delay time of modular reduction is short. With this circuit, the modular reduction processing is simplified because of the no need to compare divisors with dividends and operate negative numbers. Also, since $E_{j-1,n-i}$ is a true value, these values can be input into the addition circuit as they are.

The outputs $R_{k,0}, \cdots, R_{k,n-1} \in \{0,1\}^{m+2}$ from the circuit should be modified by an auxiliary small circuit for ALGORITHM 1B, which compensates the carry bits in $R_{k,i}$ so that they become all zero after the final calculation and corrects

the final value of R to be less than N. Such a small circuit will have little effect on the overall processing speed. This is the reason why it is omitted from Fig.2.

2.2 Circuit Scale and Processing Speed

Assuming that the proposed circuit is fabricated by C-MOS technology, let us evaluate the circuit scale according to those of the Fujitsu Standard Cells [FujG88], [FujM88].

When k equals 512 and m equals 4, the modular multiplication circuit of roughly 25 Kgates is possible. (The residues $E_0, E_1, \cdots, E_{n-1}$ are calculated and set by another circuit from the value of N in relation of Expression (4).) Since the processing time required for 1 clock is realized at the order of $20 \sim 30$ ns using C-MOS gates, the circuit in Fig.2 needs approximately $13\mu s$ to execute one modular multiplication $R = A \cdot B \bmod N$. Subsequently, when the keys e and N are each at the 512 bit, RSA encryption using this modular multiplication circuit can be achieved at a processing speed of 50 Kbps. Also, if the 6 non-zero values for $E_{j,i}$ derived from N are used as the public-key instead of the modulus N, then the preprocessing required for $E_{j,i}$ can be eliminated.

Table 1 and the reference [Bric89] show various implementations for the RSA scheme up to date. Even though an unconditional comparison cannot be made, when efficiency is defined as the processing speed per one gate (*processing speed /circuit scale*), we attempt an evaluation of several examples of the RSA scheme. Among the RSA scheme implementations developed until now, that of [THAA88] achieves the highest processing speed of 500 Kbps, however it needs a scale greater than or equal to 640 Kgates. It yields an efficiency of 0.78 bps/gate. A modular multiplication circuit for the RSA scheme [Mori90] can achieve a processing speed of 80 Kbps using a 50 Kgates circuit scale. The modular multiplication circuit yields an efficiency of 1.6 bps/gate, and can be considered to be the most efficient before this paper.

While, the circuit proposed in 2.1 yields an efficiency of 2.0 bps/gate, and can thus be described as more efficient than that of [Mori90]. Further, since the modular multiplication circuit in [Mori90] requires the circuit for Booth algorithm, both the control and calculation algorithm of the circuit proposed herein can be considered simpler than that of [Mori90]. Consequently, the modular multiplication circuit in Fig.2 allows the most efficient implementation of the RSA scheme to date.

3 Systolic Arrays for RSA Scheme

3.1 An Approach to Design Systolic Arrays for Modular Multiplication

Modular multiplication: $R = A \cdot B \bmod N$ can be calculated by the recursive execution of

$$R_j = 2 \cdot R_{j-1} + A_{k-j} \cdot B + E_{j-1} \tag{6}$$

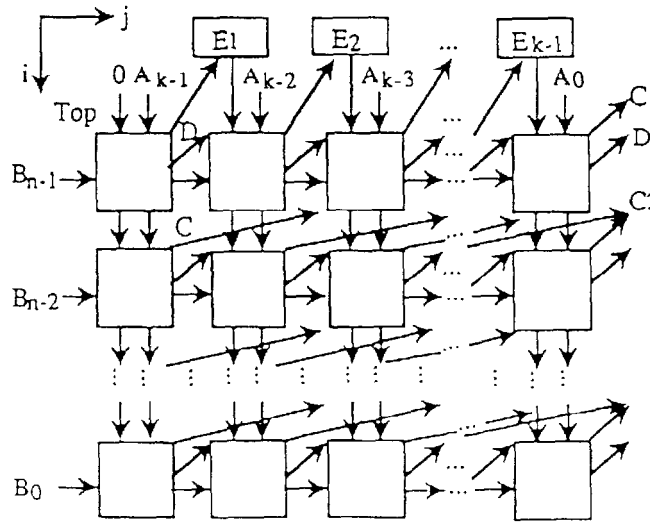

Fig.3: Data dependence graph for modular multiplication

from $j = 1$ to $j = k$, where R_0 is 0, and where E_{j-1} is the residue shown in Expression (4).

A data dependence graph can represent a version of the recursion obtained by using m-bitwise slice. We propose the graph given in Fig.3 where each column of n cells performs Expression (6) and the row of k such columns illustrate the recursive execution. In the rest of this subsection, we give an intuitive explanation of the key idea for our systolic arrays.

The inputs B flow from left to right along the horizontal arrows, and the inputs A flow downward along the vertical arrows. The result of the computation done in each cell is divided into the least significant $m-1$ bits D and the rest of bits C (the carry bits). The 1-bit shifted D (double of D) is input to the cell in the same row and in the right next column. The 1-bit shifted C (double of C) is input to the cell in the upper next row and in the column two positions to the right. For each cell in the top row the carry bits C is translated into a residue E and E affects the right next column of cells. Such a trick is to avoid the increasing of the size of data treated in each cell and to ensure a regular and efficient pipeline structure.

Observe that between each pair of consecutive columns there are three types D, $C1$, and $C2$ of data flows as well as the flow of B. For each column, $C1$ denotes the type of C made in the column and $C2$ denotes the type of C passing through the column. Each column can be realized by a single processing element (PE) defined as Fig.5 and the row of k such columns can be implemented as the connection of k PEs. See Fig.4. The j th PE holds the value of A_{k-j} at one of its internal registers. The carry bits C produced at the top cell in a column in Fig.3 is hold at a register as S_{out} in the corresponding PE in Fig.5. The ROM

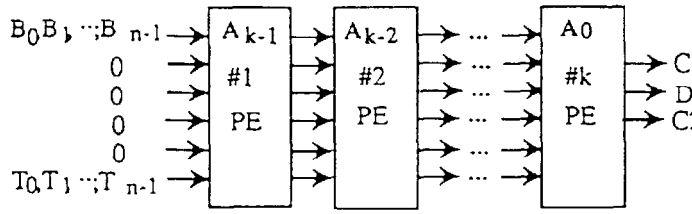

Fig.4: Systolic array for modular multiplication

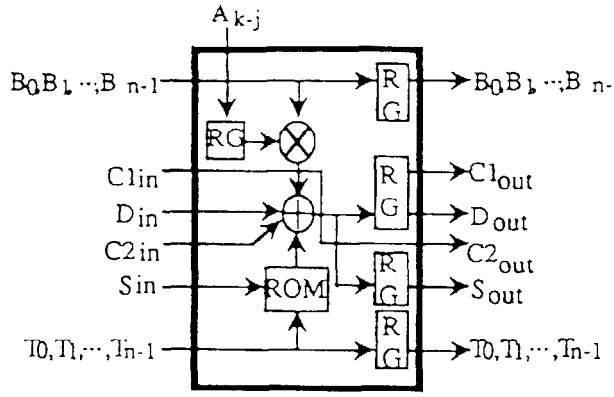

Fig.5: Processing element in Fig.4

in each PE accepts the S_{out} coming from the left next PE as

S_{in} and outputs a residue E specified by S_{in} in m-bit wise according to an input sequence of timing signals $T_{n-1}, T_{n-2}, \cdots, T_0$.

Since the form of the data D, $C1$, and $C2$ appears in the array differs from that of the input data, the modular product of two integers, output from the array cannot be directly used as another input to the same array. Thus, using only the array we cannot execute modular exponentiation. We describe in the next subsection a method to eliminate such a drawback and propose an effective systolic array.

3.2 A Systolic Array for Modular Multiplication and Compensating Carry Bits

We consider the modular multiplication $R_c = A_c \cdot B_c \mod N$, where $A_c = A + a \cdot X$, $B_c = B + b \cdot X$ and $R_c = R + r \cdot X$. A, B and R are shown in Expressions (1) - (3), respectively and a, b and r are expressed as follows,

$$
\begin{aligned}
a &= a_{k-m} \cdot X^{n-2} + \cdots + a_{2 \cdot m} \cdot X + a_m \\
b &= b_{n-1} \cdot X^{n-2} + \cdots + b_2 \cdot X + b_1 \\
r &= r_{n-1} \cdot X^{n-2} + \cdots + r_2 \cdot X + r_1
\end{aligned}
$$

where, a_{k-j}, b_{n-i}, $r_{n-i} \in \{0,1\}$ $(j = m, 2 \cdot m, \cdots, (n-1) \cdot m)$ $(i = 1, \cdots, n-1)$, and a_{k-j}, b_{n-i}, r_{n-i} are 1-bit carry bits by every m-bit group. The modular multiplication $R_c = A_c \cdot B_c \mod N$ is calculated as shown in ALGORITHM 2.

Fig.9 is the systolic array for ALGORITHM 2 and PEA, PEB, and PEC's processing is executed by the PE shown in Fig.6, Fig.7, and Fig.8, respectively. The neighboring PEs is connected between X_{out} and X_{in} ($X = b, B, S, D, C$ and T) In ALGORITHM 2, the value i refers to the clock, and j refers to the number for the PEs, so that from right to left $j = 1$ to $j = k + n$ refers to each position of the PEs. PEA calls 1 PEB for every m turns, and for the final calculation calls PEC instead of PEB in Fig.9.

In the first PE($j = 1$) of Fig.9, B_{n-i} and $b_{n-i}(i = 1, \cdots, n)$ are input from B_{in} and b_{in} simultaneously in order from the upper row. Also, the timing signal $T_{n-i} = n - i$ $(i = 1, \cdots, n)$ referring to the modular-reduction circuit is input from T_{in} synchronized with B_{n-i}. B_{n-i}, b_{n-i} and T_{n-i} are held back by one clock in internal registers of the PEs, then are output to the next PE from B_{out}, b_{out} and T_{out}. In the first PE the values input for D_{in}, S_{in} and C_{in} are set to 0. Each PE's composition elements and actions can be explained in the following:

PEA : The value A_{k-j+s} is preset in the internal 1-bit register of j-th PEA. The multiplier is constructed with $m+1$ AND gates and outputs $A_{k-j+s} \cdot B_{n-i}$ and $A_{k-j+s} \cdot b_{n-i}$. The adder is based on the 5-input from the multiplier outputs $A_{k-j+s} \cdot B_{n-i}$ and $A_{k-j+s} \cdot b_{n-i}$, the modular-reduction circuit's output $E_{j-1,n-i}$, and the previous PE's output $2 \cdot dw_{m-1}(R_{j-1,n-i})$ and the output $2 \cdot C_{j-2,n-i-1}$ of the two PEs before. Since $A_{k-j+s} \cdot B_{n-i}$ and $E_{j-1,n-i}$ are m-bit and $dw_{m-1}(R_{j-1,n-i})$ is $m-1$-bit and $A_{k-j+s} \cdot b_{n-i}$ is 1-bit, the fact that $2 \cdot C_{j-2,n-i-1}$ is less than an m-bit value leads that the value enters the register from the adder is $m + 2$-bit. The residue $E_{j-1,n-i}$ is derived from Expression (7) in relation of $up_{m-1}(R_{j-1,n-1}) = C_{j-1,n-1}$. Since B_{n-i} and $E_{j-1,n-i}$ are in the same digit, $E_{j-1,n-i}$ $(i = 1, \cdots, n)$ can be gradually output according to the timing signal T_{n-i} which is synchronized with B_{n-i}. Since $C_{j-1,n-1}$ is 3-bit and T_{n-i} is $\log n$-bit, the output circuit for $E_{j-1,n-i}$ can be realized by ROM (read-only-memory) holding input of $3 + \log n$ bits and output of m bits and the selector and the register for the 3 bits to save the value of $C_{j-1,n-1}$.

PEB : In the internal register of this PE the value $a_{k-j+s+1}$ is preset. Since $a_{k-j+s+1}$ and A_{k-j+s} in the previous PE correspond to the same bit in the binary expression, $dw_m(R_{j-1,n-i})$ and $C_{j-2,n-i-1}$ are not doubled and $C_{j-1,n-i}$ is $up_m(R_{j-1,n-i})$.

PEC : This PE aims at to make the output r_{n-i} $(i = 1, \cdots, n)$ one bit. This PE adds the outputs D_{out}, S_{out}, and C_{out} from the previous PE and creates the single value $R_{k+n,n-i}$. Next the PE delays the value $dw_m(R_{k+n,n-i})$ in the register and adds it to the value $up_m(R_{k+n,n-i})$, and produces $R_{k+n+1,n-i} \in \{0,1\}^{m+1}$ $(i = 1, \cdots, n)$. The most significant bits $up_m(R_{k+n,n-1})$ is saved in a separate register. Since the residue E_{k+n+1}

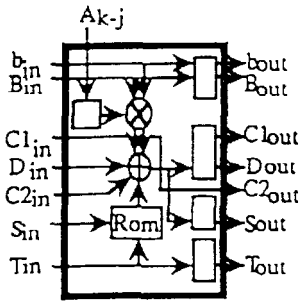

Fig.6: PEA

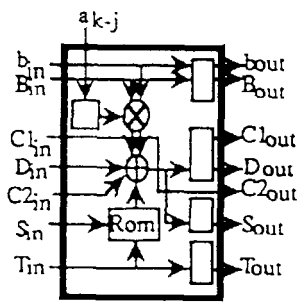

Fig.7: PEB

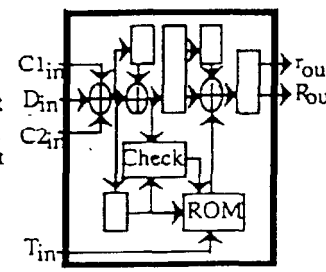

Fig.8: PEC

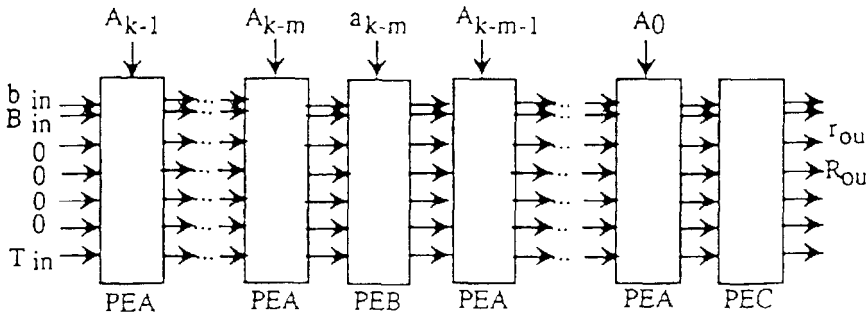

Fig.9: Systolic-array for modular multiplication using PEA, PEB and PEC

is calculated by $(CC \cdot X^n) \bmod N$, $R_{k+n+2,n-i} = dw_m(R_{k+n+1,n-i}) + up_m(R_{k+n+1,n-i-1}) + E_{k+n+1,n-i}$ and $R_{k+n+2,n-i} \in \{0,1\}^{m+1}$ $(i = 1, \cdots, n)$. However, pre-modular multiplication is performed in the check circuit shown as C in Fig.8 to set the most significant bit $r_{n-1} = 0$. Since the most significant bits of residue E_{ad} from the value CC is calculated in advance, the pre-modular multiplication $dw_m(R_{k+n+1,n-1}) + up_m(R_{k+n+1,n-2}) + E_{ad}$ can be calculate. If the most significant bit of the pre-modular multiplication is 1, the fact that E_{k+n+1} takes $((CC+1) \cdot X^n) \bmod N$ leads $r_{n-1} = 0$. This checking circuit is executed with 3 bits of ROM and an adder.

Consequently, the repeating modular multiplication such as the RSA enciphering and deciphering can be achieved with a systolic array as shown in Fig.9.

ALGORITHM 2
(Input: $A_0, \cdots, A_{k-1}$; $a_m, \cdots, a_{k-m}$; $B_0, \cdots, B_{n-1}$; $b_1, \cdots, b_{n-1}$; N)
(Output: $R_0, \cdots, R_{n-1}$; $r_0, \cdots, r_{n-1}$)

$R_{0,n-i=0}$; $C_{0,n-i=0}$
FOR $s = 0$ **TO** $k/m - 1$
FOR $c = 1$ **TO** m
$j = s \cdot (m+1) + c$
$E_{j-1} = E_{j-1,n-1} \cdot X^{n-1} + \cdots + E_{j-1,0}$
$= (up_{m-1}(R_{j-1,n-1}) \cdot X^n) \bmod N$ (7)
FOR $i = 1$ **TO** n
P $R_{j,n-i} = 2 \cdot dw_{m-1}(R_{j-1,n-i}) + 2 \cdot C_{j-2,n-i-1}$
E $+A_{k-j+s} \cdot (B_{n-i} + b_{n-i}) + E_{j-1,n-i}$
A $C_{j-1,n-i} = up_{m-1}(R_{j-1,n-i})$
NEXT
NEXT
IF $s = k/m - 1$ **THEN** GOTO PEC
$j = j + 1$
$E_{j-1} = E_{j-1,n-1} \cdot X^{n-1} + \cdots + E_{j-1,0}$
$= (up_m(R_{j-1,n-1}) \cdot X^n) \bmod N$ (8)
FOR $i = 1$ **TO** n
P $R_{j,n-i} = dw_m(R_{j-1,n-i}) + C_{j-2,n-i-1}$
E $+a_{k-j+s+1} \cdot (B_{n-i} + b_{n-i}) + E_{j-1,n-i}$
B $C_{j-1,n-i} = up_m(R_{j-1,n-i})$
NEXT
NEXT
FOR $i = 1$ **TO** n
P $R_{k+n,n-i} = R_{k+n-1,n-i} + C_{k+n-2,n-i-1}$
E $R_{k+n+1,n-i} = dw_m(R_{k+n,n-i}) + up_m(R_{k+n,n-i-1})$
C $CC = (up_m(R_{k+n,n-1}) + up_m(R_{k+n+1,n-1}))$
$E_{ad} = ((CC \cdot X^n) \bmod N)/X^{n-1}$
IF $dw_m(R_{k+n+1,n-1}) + up_m(R_{k+n+1,n-2}) + E_{ad} \geq X$
THEN $CC = CC + 1$
$E_{k+n+1} = E_{k+n+1,n-1} \cdot X^{n-1} + \cdots + E_{k+n+1,0}$
$= (CC \cdot X^n) \bmod N$
$R_{k+n+2,n-i} = dw_m(R_{k+n+1,n-i}) + up_m(R_{k+n+1,n-i-1})$
$+E_{k+n+1,n-i}$
$R_{n-i} = dw_m(R_{k+n+2,n-i})$
IF $i = 1$ **THEN** $r_{n-1} = 0$ **ELSE** $r_{n-i} = up_m(R_{k+n+2,n-i})$
NEXT

where $dw_a(Z) = Z \bmod 2^a$ and $up_a(Z) = (Z - dw_a(Z))/2^a$.

3.3 Circuit Scale and Processing Speed

At first we evaluate the circuit scales of PEA, PEB and PEC, and show them in Table.2 for $m = 8$.

Table.2: Circuit scale of PEs for $m = 8$

	PEA	PEB	PEC
multiplication	20	20	0
addition	294	294	464
modular reduction	404	306	404
delay	94	94	38
total	812	714	906

Table.3: Efficiency of PEA for m

m	16	32	64	128
multiplication	31	55	103	199
addition	451	891	1771	3531
modular reduction	410	410	349	318
delay	138	232	425	819
PEA's circuit scale	1030	1603	2648	4867
number of PEs	544	528	520	516
total circuit scale	561K	847K	1377K	2512K
processing speed	400K	800K	1600K	3200K
efficiency	0.71	0.94	1.16	1.27

In order to construct a systolic array shown in Fig.9, 512 PEAs, 63 PEBs and 1 PEC are necessary. For simplicity's sake, if the systolic array in Fig.9 consists of 576 PEAs, its circuit scale becomes 468 Kgates.

Since the processing time needed for 1 clock is the time to go from the selector to pass the ROM and the adder, if a bi-polar ROM is used a processing time of about 50 ns can be achieved. When the RSA scheme keys e and N are 512 bits, a processing speed for the RSA scheme can achieve at about 200Kbps. This means that even if data is entered at the speed of 200Kbps successively, real-time processing by means of the pipeline processing of the systolic array without buffer overflow is possible.

Table.3 shows PEA's circuit scale and processing speed for selected m's. Since the delay time for the adder increases as the value of m increases even when m is doubled it can be considered that the processing speed will not double, but, rather, somewhat decrease. However, if the processing speed as m increases is compared to the heap of the delay time, the influence of the delay time is ignorably small. Consequently, we realize that as m increases the efficiency increases.

Also, since the functions of PEA, PEB and PEC resemble each other, a common PE can be constructed only increasing a small circuit scale as compared with that of PEA. The common PE calculates a partial modular multiplication such as $R_j = 2 \cdot R_{j-1} + A_{k-j} \cdot B_c \bmod N$ and/or $R_j = R_{j-1} + a_{k-j} \cdot B_c \bmod N$

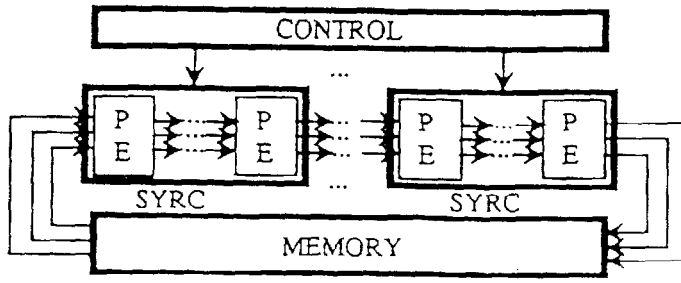

Fig.10 : Chip implementation using Systolic RSA Chip

, and the pipeline structure by q PEs realizes partial modular multiplications q times. Consequently, processing speed of a modular multiplication can be increased in proportion to the increase of the number of the PEs. Further, the pipeline structure with the PEs can keep the same efficiency for an increase in speed. For example, if we take $m = 8$, a circuit scale of $q*812$ gates (with q being an integer representing the number of PEs, $131072 \geq q \geq 1$) and a processing speed of $q \cdot 347$ bps can thus be achieved for the RSA scheme implementation. If we take $m = 128$, the RSA scheme implementation at a circuit scale of $q \cdot 4867$ gates and a processing speed of $q \cdot 6.2$ Kbps can be achieved. Consequently, it is shown that this method can realize an efficient RSA scheme implementation for the increase in speed.

3.4 Chip Implementation Using the Systolic-Array Approach

As shown in 3.2, since the repeating modular multiplication circuit can be constructed using optional q PEs ($131072 \geq q \geq 1$), we construct a modular multiplication circuit with favorite circuit scale. At that point the creation of chips (hereafter referred to as SYRCs, Systolic RSA Chips) in units with favorite number of PEs, in combination with RAM, both of which can be controlled by external programming thus affecting a way to achieve the RSA scheme implementation. The external programming can be constructed in a flexible way using the ROM. Also, if a greater processing speed is required, many SYRCs can be connected as shown in Fig.10. Fig.11 shows the difference of processing speed between using 2 SYRCs and using 3 SYRCs in the implementation of Fig.10. It is easy to understand that the implementation using 3 SYRCs is 1.5 times faster than that of using 2 SYRCs. Like these examples, this implementation can achieve a speed-up proportional to the number of SYRCs and can realize high-speed real-time processing. The same characteristic can be also easily accomplished by changing the number of times the SYRC processing occurs with a control circuit.

While, since this method is based on a systolic array, this method is suitable for the VLSI (Very Large Scale Integration) implementation. If VLSI is utilized,

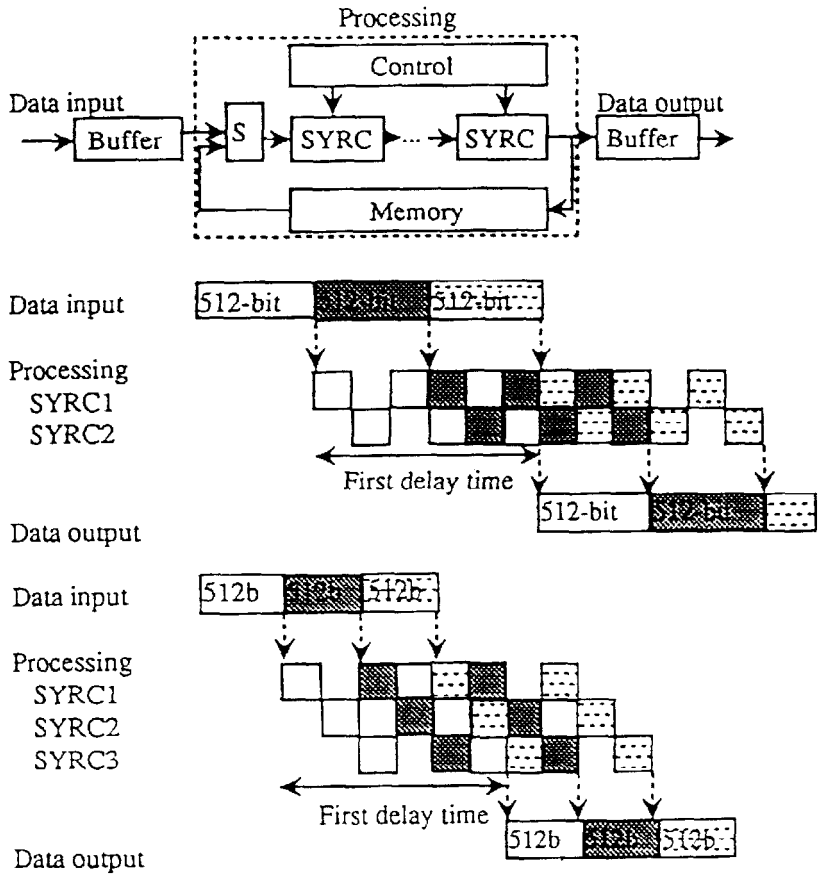

Fig.11 : Difference of processing speed between using 2 SYRCs and using 3 SYRCs

high-speed RSA scheme implementation in one chip is easily constructed because of the regularity and simplicity of the structure. Also, since the processing that occurs in one PE is simple integer calculations, even without putting the PEs onto a separate chip the modular multiplication algorithm shown in **3.1** can be realized using the normal DSP or CPU etc.. As shown above, the modular multiplication method demonstrated in this section is useful because it is extraordinarily easy to circuitize and improvements in speed are easily attainable.

3.5 Another Utility of the Systolic-Array Approach

It is known that the processing can be sped up with a same circuit scale using the CRT (Chinese Remainder Theorem) [QC82]. When an RSA implementation using the CRT is attempted, the problem of creating different bit numbers for

multipliers and divisors makes use of the same modular multiplication circuit for both enciphering and deciphering difficult to realize. However, since the modular multiplication circuit in this section enables an easy trade-off between circuit scale and processing events, enciphering and deciphering can be realized on the same circuit if we change the processing events for the difference in multipliers and divisors. This can be easily affected by simply changing the control of the number of feedback inputs to the SYRC for enciphering and deciphering with a control circuit. Using the CRT to increase processing speed can yield at most a quadrupled rate on the same circuit scale. Consequently, when the method outlined in this section uses the CRT, the processing speed demonstrated in the previous section can be increased by approximately 4 times, thus making the construction of an extremely efficient RSA scheme implementation possible. Also, since the RSA deciphering using the CRT can be basically achieved in parallel processing, when modular multiplication is achieved with multiple chips, this paper's method is suitable from this point of view.

This also means that effective support for changes in the bits of keys e and N can be achieved with the method shown in this section. In order to increase the security of the cryptosystem, even when N is made larger, if the number of feedbacks to the SYRC and the size of ROM for the modular reduction is increased then support is possible. Although this lowers the processing speed, if the number of SYRCs is increased, it is possible to maintain the same processing speed.

4 Conclusion

Each + in Fig.12 shows the RSA implementations with known processing speeds and circuit scales. Except for [THAA88], a trend of pursuing smaller circuit scales within 100Kbps and of getting higher efficiency can be observed. If the horizontal co-ordinate is the same, then the lower the vertical co-ordinate the more efficient the system is. In particular, the propose in Section 2 exhibits the best efficiency, i.e., 2.0 bps/gate. For processing speeds more than 100Kbps, however, like the implementation described in [THAA88], no efficient way has been reported for constructing appropriate scaled circuits for the RSA implementation. Any conventional method, even one in Section 2, would require a circuit scale which rapidly grows as the processing speed increases.

Section 3 resolved this difficulty by introducing the systolic array architecture and attained RSA implementations of constant efficiency as shown in lines a and b in Fig.12, where $m = 64$ is adopted. In these systolic implementations the circuit scale increases linearly in the required processing speed. And the efficiency is improved if the value of m is increased.

The systolic approach gives a systematic way of designing RSA hardwares in a wide range of circuit scales and processing speeds. Its characteristics are listed below:

1) Since the number of PEs on a chip is optional, we can make a chip to favorite scale. To increase the processing speed, one would merely have to increase the number of the chips.

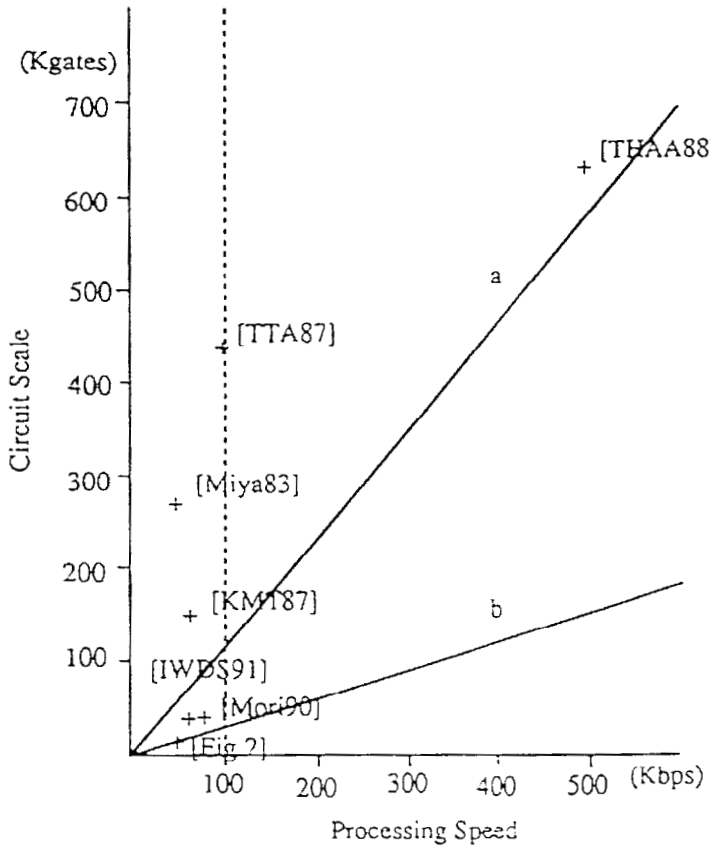

Fig.12 : Comparison of implementations proposed in this paper with those proposed so far

2) This method allows the same chip to implement the deciphering or signing (the secret transformation) with the aid of the CRT and the enciphering or signature-verification (the public transformation). Using C-MOS gates, which can be produced cheaply in large quantity due to the present-day semi-conductor technology, a one-chip RSA implementation achieving 64Kbps but using at most 20Kgates can be realized readily by adopting CRT-based secret transformations and moderate-sized public exponents e.

3) A systolic array architecture is simple and regular, and thus, suitable for VLSIs. An RSA implementation using VLSI enables a single chip to achieve Mega bps.

4) Since the calculations processed within the PEs are simple integer calculations, an efficient RSA implementation can be constructed even by the use

of CPUs or DSPs.

In conclusion, the methods outlined in this paper lead to the constructive and effective ways for implementing the RSA scheme.

On the other hand it is known that the Montgomery method [Mont85] is an effective algorithm for modular multiplication. We already have a short paper [IMI92] proposing systolic arrays for the Montgomery method which can realize more efficient and more flexible implementation in the wide range of processing speed.

References

[RSA78] R.Rivest, A.Shamir, and L.Adleman, "A method of obtaining digital signatures and public key cryptosystems," Comm. of ACM, Vol.21, No.2, pp.120-126, Feb. 1978.

[Brick89] E.F.Brickell, "A Survey of hardware implementation of RSA," Advances in Cryptology — CRYPTO'89, pp.368-370, Springer-Verlag.

[Bric82] E.F.Brickell, "A fast modular multiplication algorithm with applications to two key cryptography," Advances in Cryptology, Proc. of CRYPTO'82, pp.51-60, Plenum.

[DK90] S.R.Dusse and B.S.Kaliski Jr., "A cryptographic library for the Motorola DSP56000," Advances in cryptology — EUROCRYPT'90, pp.230-244, Springer-Verlag.

[Even90] S.Even, "Systolic modular multiplication," Advances in Cryptology — CRYPTO'90 pp.619-624, Springer-Verlag.

[HDVG87] F.Hoornaert, M.Decroos, J.Vandewalle, and R.Govaerts, "Fast RSA-hardware: Dream or reality ?" Advances in Cryptology — CRYPTO'88, pp.257-264, Springer-Verlag.

[HTAA90] T.Hasebe, N.Torii, M.Azuma, and R.Akiyama, "Implementation of high speed modular exponentiation calculation," Proc. IEICE Spring Conference, A-284, 1990.

[IWDS91] P.A.Ivey, S.N.Walker, S.Davidson, and J.M.Stern, "A VLSI architecture for RSA encryption," Secure Design and Test of Crypto-Chips, IFIP WG10.5 Workshop, Oct. 1991.

[Kawa88] S.Kawamura, "A modulo multiplication algorithm using a small size residue table," Proc. Sympo. on Cryptography and Information Security, Session J, 1988.

[KMT87] Y.Kano, N.Matsuzaki, and M.Tatebayashi, "A modulo exponentiation LSI using high-order modified Booth's algorithm," Proc. Workshop on Cryptography and Information Security, WCIS87-11, 1987.

[Koch85] M.Kochanski, "Developing an RSA chip," Advances in Cryptology — CRYPTO'85, pp.350-357, Springer-Verlag.

[Miya83] S.Miyaguchi, "A fast computing scheme for RSA public-key cryptosystem and its VLSI organization," Trans. Info. Processing Soc. Japan, Vol.24, No.6, pp.764-771, Nov. 1983.

[Mori90] H.Morita, "A fast modular multiplication algorithm based on a radix 4 and its application," Trans. IEICE, Vol.73, No.7, July 1990.

[Omur90] J.K.Omura, "A public key cell design for smart card chips," Proc. 1990 Int. Symp. Info. Theory and Its Applications, 65-1, 1990.

[ORSPT86] G.Orton, M.Roy, P.Scott, L.Peppard, and S.Tavares, "VLSI implementation of public-key encryption algorithms," Advances in Cryptology — CRYPTO'86, pp.277-301, Springer-Verlag.

[OSA90] H.Orup, E.Svendsen, and E.Andreasen, "VICTOR an efficient RSA hardware implementation," Advances in Cryptology — EUROCRYPT'90, pp.245-252, Springer-Verlag.

[OT88] E.Okamoto and K.Tanaka, "A realization of RSA cryptosystem using digital signal processor, " Proc. Sympo. on Cryptography and Information Security, Session J, 1988.

[QC82] J.J.Quisquater and C.Couvreuer, "Fast decipherment algorithm for RSA public-key cryptosystem," Electron. Letters Vol.18, No.21, pp.905-907, Oct. 1982.

[Rive84] R.L.Rivest, "RSA chips (past/present/future),"Advances in Cryptology — EUROCRYPT'84, pp.159-168, Springer-Verlag.

[Taka88] K.Takaragi, "Hardware of RSA encryption,"Proc. Sympo. on Cryptography and Information Security, Session J, 1988.

[TAA87] N.Torii, M.Azuma, and R.Akiyama, "A study on high speed RSA encryption LSI using parallel processing," Proc. IEICE National Convention, p.1388, 1987.

[THAA88] N.Torii, T.Hasebe, M.Azuma, and R.Akiyama, "The hardware technologies for RSA encryption systems," Proc. Sympo. on Cryptography and Information Security, Session J, 1988.

[TKS90] K.Takabayashi, S.Kawamura and A.Shinbo, "A modular exponentiation method using a fast constant multiplication algorithm," Proc. IEICE Spring Conference, A-285, 1990.

[Walt91] C.D.Walter, "Fast modular multiplication by operand scaling," Abstracts of CRYPTO'91, pp.8-1-8-6.

[WQ90] S.Waleffe and J.J.Quisquater, "CORSAIR: a smart card for public key cryptosystems," Advances in Cryptology — CRYPTO'90, pp.475-487, Springer-Verlag.

[FujG88] Fujitsu Limited, "ASIC technical information," GATI0172C, 1988.

[FujM88] Fujitsu Limited, "CMOS standard cell design manual," MATI10702, 1988.

[Mont85] P.L.Montgomery, "Modular multiplication without trial division," Math.of Computation, Vol.44, pp.519-521, 1985.

[IMI92] K.Iwamura, T.Matsumoto and H.Imai, "Systolic-arrays for modular exponentiation using Montgomery method," presented in Rumpsession of Eurocrypt'92, May 24-28, 1992.

Appendix

Theorem
In ALGORITHM 1A when $A_{k-j} \cdot B_i$, $E_{j-1,i}$, $D_{j-1,i}$ and $C_{j-1,n-i}$ are respectively m-bit, m-bit, $(m-1)$-bit and 3-bit values, the most significant 3 bits of

$$R_{j,i} = A_{k-j} \cdot B_i + E_{j-1,i} + 2 \cdot D_{j-1,i} + C_{j-1,i-1}$$

is less than [111] for $m \geq 3$.

Proof
Because $A_{k-j} \cdot B_i \leq 2^m - 1$, $E_{j-1,i} \leq 2^m - 1$, $2 \cdot D_{j-1,i} \leq 2^m - 2$ and $C_{j-1,i-1} \leq 7$, we have

$$R_{j,i} \leq (2^2 + 2^1) \cdot 2^{m-1} + 2^1 + 2^0.$$

Thus, if $m - 1 \geq 2$, that is, $m \geq 3$, then the most significant 3 bits of the right-hand side is equal to [110].