AN EFFICIENT VLSI ARCHITECTURE FOR RSA PUBLIC-KEY CRYPTOSYSTEM

Jen-Shiun Chiang and Jian-Kao Chen

Department of Electrical Engineering
Tamkang University
Tamsui, Taipei
Taiwan
E-mail: chiang@ee.tku.edu.tw

ABSTRACT

In this paper, a new efficient VLSI architecture to compute RSA public-key cryptosystem is proposed. The modified H-algorithm is applied to find the modular exponentiation. By this modified H-algorithm, the modular multiplication steps reduced about 5n/18. For the modular multiplication the L-algorithm (LSB first) is used. In the architecture of the modular multiplication the iteration times are only half of Montgomery's algorithm and the H-algorithm. By these arrangements, this architecture of RSA has very good area-time product.

1. INTRODUCTION

The development of the Internet is amazed and amused by people. Nowadays, we can buy many items from the network, and life seems easier than several years ago. However, due to the opening characteristics of the internet the safety problems are always concerned by the people. Everybody hates his/her credit card number to be robbed by the network hiker. Therefore, the safety problem of the network is one of the most important problems is needed to be solved nowadays. The most efficient solution is to use cryptographic to encode and decode the data. RSA cryptosystem is the best cryptographic we have ever known [1]. In the RSA cryptosystem, we need two keys, public key and private key, and associate these two keys we can get safety data. The safety depends on the length of the key, usually the longer the key the more safety the data. Generally we need at least a 512-bit key. The processing of the key is composed of many modulo multiplication, modulo additions, and modulo exponent operations. Due to the large number, by the software approach the operation speed can not be fast, and can not be real time. Because of the development and advancing of the VLSI process the RSA cryptosystem can be implemented in the VLSI chip. In the reported RSA cryptosystem architecture or chip the hardware cost is still very high.

The RSA cryptosystem is briefly described as follows:
Let p and q be two distinct large random primes. Denote

$$N = pq$$

Let us choose a large random number d>1 such that

$$\gcd[(p-1)(q-1),d] = 1$$

and compute the number e, $1<e<(p-1)(q-1)$,

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

The numbers N, e, and d are called modulus, encryption, and decryption exponent respectively. The numbers N and e constitute the public encryption key, and p, q, (p-1)(q-1), and d form the secret trapdoor. To encrypt and decrypt, the input text is first encoded to a number and is divided into blocks of suitable size. The blocks are then processed separately as follows:

- $C = M^e \bmod N$ (1)
- $M = C^d \bmod N$ (2)

C and M are referred to as plaintext and ciphertext blocks respectively.

Equation (1) and (2) are in modular exponentiation operation and are the most critical operation in the RSA. Therefore, how to increase the speed of the modular exponentiation is the main task for the RSA public-key cryptosystem. Basically the modular exponentiation needs modular multiplication. The modular multiplication is accomplished by addition and shift operations. To avoid unnecessary carry propagation, the addition can be finished by the redundant binary adder [2] or carry save adders [3-6]. However, the adder cell of the redundant binary adder is very complicated, and people prefer to use carry save adder. In the shift operation, there are two approaches, left shift (multiply by 2) [2,5,6] and right shift (divide by 2) [3,4]. In this paper, we use the left shift approach. The modular operation can be finished by comparators or by checking the overflow of the adder [5,6]. The former approach needs more hardware and the speed is slower. The latter approach needs less hardware and the speed is faster. Therefore, we use the latter approach to implement modulo operations.

This paper is organized as follows. In section 2, the modified modular exponentiation algorithm is described. The modified modular multiplication algorithm is described in section 3. The hardware design of the RSA cryptosystem and the simulation results are explained in section 4. Finally we give the conclusion in section 5.

2. MODIFIED MODULAR EXPONENTIATION ALGORITHM

The basic algorithm for computing modular exponentiations is to repeat squaring and multiplying. To compute $C = M^e \pmod N$, the algorithm operates as follows [10]:

The H-algorithm (MSB first)

$P_0 = 1$
for (i=n-1; i>=0; i--){
$M_{n-i}=M^2_{n-i-1} \pmod N$
if (e_i==1)
$P_{n-i} = M_{n-i} \times M \pmod N$;
else $P_{n-i} = M_{n-i}$ }

0-7803-5471-0/99/$10.00©1999 IEEE

Where $e=[e_{n-1},e_{n-2},\ldots,e_1,e_0]_2$ is the encryption key, and P_i is the partial product. In the modular operation, '1' needs two iteration steps in e[]. In the worst case, the iteration steps of computing the exponentiation are 2n. In order to reduce the iteration times, we partition the encryption key e[] into several segments, and each segment consists of four bits. In the 4-bit segment, we find some rule to reduce the iteration times. For example, e[i]=0000 needs squaring four times, and e[i]=0001 needs squaring three times and the 1 may be combined with next segment. Generally there need at most five iteration times in each segment. Whereas e[i]=0111, the operation needs seven iteration times with the traditional H-algorithm. By this 4-bit segment, we can build a table as show in Table I.

Table I. The Encryption Key Table

0000	010	010	010	010	
0001	010	010	010		
0010	010	010			
0011	010	010	010	010	011
0100	010	010	001	010	010
0101	010	010	010	010	101
0110	010	010	010	011	010
0111	010	010	010	010	111
1000	010	001	010	010	010
1001	010	001	010	010	
1010	010	010	010	101	010
1011	010	010	010	101	
1100	010	010	011	010	010
1101	010	010	011	010	
1110	010	010	010	111	010
1111	010	010	010	111	

We can pre-calculate $M3=M^3$, $M5=M^5$, and $M7=M^7$, and store these 3 n-bit numbers to RAMs. Thus we can combine the encryption key table and the pre-calculated M3, M5, and M7, and the modulo exponentiation can be computed as follows:

- [001] means multiply by M;
- [010] means square;
- [011] means multiply by M3;
- [101] means multiply by M5;
- [111] means multiply by M7;

By the above arrangement, [111] can reduce 2 iteration times, and [011] and [101] can reduce 1 iteration time. In the worst case, the iteration times of the modular multiplication of this approach are 4n/3, and the multiplication average times are 11n/9. Compared to H-algorithm (worst case =2n, average=3n/2), our approach reduce the multiplication times significantly.

3. MODIFIED MODULAR MULTIPLICATION ALGORITHM

In the modular multiplication, the Montgomery's algorithm (divide by 2) [3,4] or the H-algorithm (MSB first) [2,5] is used widely. However, they have their drawbacks in the proposed architectures [11, 12, 13]. Here we would like to use a modified L-algorithm (LSB first). The L-algorithm [10] and our modified L-algorithm are shown as follows:

The L-algorithm (LSB first)

```
A×B;
P0=0, M0=A;
  for (i=0; i<=n-1; i++){
    if (bi==1)
      Pi=Pi-1+Mi-1   (mod N);
    else
      Pi=Pi-1;
    Mi=Mi-1<<1   (mod N);   }
```

Where P_i is the partial product, and $B=[b_{n-1},b_{n-2},\ldots,b_1,b_0]_2$.

The modified L-algorithm

```
A×B;
P0=0, M0=A, s=0, c=0;
  for (i=0; i<=(n>>1)+1; i++){
    s=b2i+b2i+1+c, c=s>>2, s=s&3;
switch(s){
  case 3:
    c+=1;
    Pi=(Pi-1+(-Mi-1)) (mod N);
    Mi=Mi-1<<2(mod N);
  case 2:
    Mi=(Mi-1<<1) (mod N);
    Pi=(Pi-1+Mi) (mod N);
    Mi=(Mi<<1) (mod N);
  case 1:
    Pi=(Pi-1+Mi-1) (mod N);
    Mi=(Mi-1<<2) (mod N);
  case 0:
    Pi=Pi-1;
    Mi=(Mi-1<<2) (mod N);   } }
```

In the modified L-algorithm, each modular multiplication iteration takes two bits, therefore the total iteration times are only half of Montgomery's algorithm and H-algorithm.

In this paper, modulo reductions are finished by checking overflows of the operation. There are four kinds of overflow. If overflow occurs, we have to add k1, k2, k3, or k4 to the number [5,6]. Where

- $k1\equiv 2^n \pmod N$,
- $k2\equiv 2^{n+1} \pmod N$,
- $k3\equiv 3\times 2^n \pmod N$,
- $k4\equiv 2^{n+2} \pmod N$.

The relationship between overflow and k values is shown in Table II.

Table II. Overflow Revised k Value

$C(S)_{n+1}$	C_n	S_n	k
0	0	0	0
0	0	1	$k1\equiv 2^n \pmod N$
0	1	0	$k1\equiv 2^n \pmod N$
0	1	1	$k2\equiv 2^{n+1} \pmod N$
1	0	0	$k2\equiv 2^{n+1} \pmod N$
1	0	1	$k3\equiv 3\times 2^n \pmod N$
1	1	0	$k3\equiv 3\times 2^n \pmod N$
1	1	1	$k4\equiv 2^{n+2} \pmod N$

By the overflow checking method, it is very easy to implement modulo operations, and the speed can be increased.

In the modular exponentiation and multiplication, $(-M_{i-1}) \pmod N$ is needed in our algorithm. The value of $(-M_{i-1}) \pmod N$ can be calculated by adding a number of multiples of N to $-M_{i-1}$ to make it positive. Since

$$0 < M_{i-1} = C_{Mi-1} + S_{Mi-1} < 2 \times 2^n + 2^n = 3 \times 2^n$$

When there is an overflow, i.e. c=1, we have to add a number, k6, $3 \times 2^n < k6 < 4 \times 2^n$, which is multiples of N, and the range of $(k6 - M_{i-1})$ is

$$\begin{aligned} & 2^n \leqq M_{i-1} < 3 \times 2^n \\ & -3 \times 2^n < -M_{i-1} \leqq -2^n \\ & 0 < k6 - M_{i-1} < 3 \times 2^n \end{aligned} \qquad (3)$$

When there is no overflow, i.e. c=0, we can add k5, $2 \times 2^n < k5 < 3 \times 2^n$, which is multiples of N, and the range of the $(k5 - M_{i-1})$ can be found as follows:

$$\begin{aligned} & 0 < M_{i-1} < 2 \times 2^n \\ & 2^{n+1} < -M_{i-1} < 0 \\ & 0 < k5 - M_{i-1} < 3 \times 2^n \end{aligned} \qquad (4)$$

Since equations (3) and (4) are within the range of the carry save adder. Therefore, these two numbers, $(k6 - M_{i-1})$ and $(k5 - M_{i-1})$, can be used in the next step, and no further conversion is needed.

Table III lists the clock cycles that are needed to perform the modular multiplication with different algorithms.

Table III. Clock Cycle Needed for Modular Multiplication

Algorithm	Each addition	Each multiplication
Montgomery	2	3n *
H-algorithm	3	4n *
L-algorithm	3	3n
Ours	4	2n

*include the addition for next multiplication

4. HARDWARE DESIGN

The main operation of the modular multiplier is addition. The carry save adder is commonly used to avoid unnecessary carry propagation [3-6]. In our modular multiplier, there are two units. One is the partial product adder to find the partial products, and the other is "the summand generator" to generate the summand for the partial product. In order to reduce the hardware cost, the message is partitioned into four segments, and we need four steps to finish the modular multiplication. In the modified L-algorithm as mentioned above, two bits are scanned each time, and this 2-bit number can decide 0, A, 2A, or –A to be added to the partial product. These four cases are summarized in Table IV.

Table IV. Summand Factor

s	SUMMAND
00	0
01	A
10	2A
11	-A

The block diagram of the summand generator (SG) is shown in Figure 1. From Figure 1, the inputs are k1~k6 and s, and the outputs are Ci and Si. Ci and Si are then used to find the partial product. There are three steps to find the partial product, and they are

- (1)$P_i = P_{i-1} + C_i$;
- (2)$P_i = (P_{i-1} + C_i) + S_i$;
- (3)$P_i = ((P_{i-1} + C_i) + S_i) \pmod N$.

The block diagram of the architecture of the partial product adder (PPA) is show in Figure 2. Both Figure 1 and Figure 2 are pipelined. The shift register is used to store B that is showed in Figure 3. Figure 4 shows the architecture of a 512-bit RSA processor. The Table block of Figure 4 stores M1, M3, M5, M7, and k1~k6.

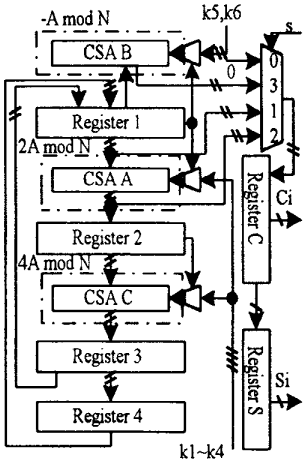

Figure 1. The Summand Generator

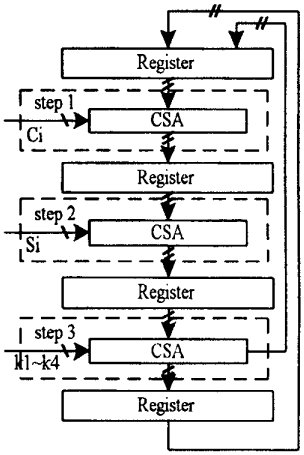

Figure 2. The Partial Product Adder

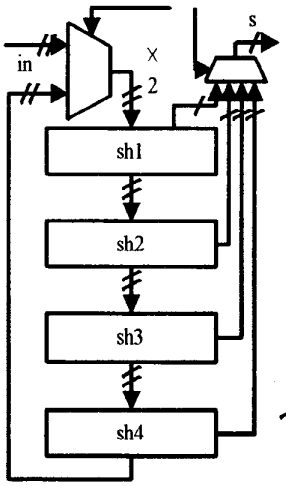

Figure 3. The Shift Register

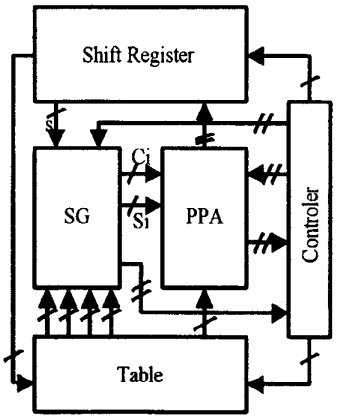

Figure 4. RSA Processor

We use Compass standard cell library (TSMC 0.6um process) to implement our design, and simulate the chip with Compass ISM (input slope model) delay model to estimate the critical path delay that includes the gate delay and wire-loading delay. The simulation results show the critical path delay is only 6ns. The chip can operate up to 166-MHz clock. The processor delivers a baud rate of 122 kbit/s in the worst case.

5. CONCLUSION

We propose two methods to speed up the operation of modular exponentiations and modular multiplication. The modified modular exponentiation algorithm limits the number of modular multiplication to 4n/3. The modified L algorithm for the modular multiplication reduces the operations times to half. In order to reduce the hardware requirement, only n/4 bits are executed in each clock cycle. In modulo reduction, we use the idea of replacing the overflow with the equivalent values that are pre-computed, and thus no comparison with the modulus (N) is needed. Based on the algorithm, this RSA cryptosystem can achieve high performance.

The simulation results show the critical path delay is only 6ns. In the worst case, the architecture takes 0.7 M clock cycles to finish the modular exponentiation (512 bits modulus, 512 bits exponent). The processor delivers a baud rate of 122 kbit/s with 166-MHz clock in the worst case.

The comparisons of hardware requirement and time complexity of the mentioned algorithms are listed in Table V and Table VI respectively. From the comparisons, the hardware of our architecture is small, and the speed is reasonable. The area-time product is very good.

Table V. Hardware Requirement

Authors		Hardware Requirement			
		FAs	REGs	MUXs	RAM
[3]	Wang I	2n	13n	10n	0
[3]	Wang II	4n	26n	20n	0
[6]	Sheu I	3.18n	10.24n	9n	0
[6]	Sheu II	3.38n	13.88n	19.1n	0
[7]	Eldridge	3n+A1*	16n	9n	0
[8]	Wu	2n+A2*	12n	6n	512*512
[9]	Juang	2n	14n	10n	10*512
	Ours	1.5n	6.75n	n	10*512

*A1=n and A2=$\log_2(n+1)$ represent the number of used half adders.

Table VI. Time Complexity

Authors		Time Complexity		
		Addition	Comparison	Cycle time
[3]	Wang I	$1.5n^2$	No	FA
[3]	Wang II	n^2	No	FA
[6]	Sheu I	$2.4n^2$	Simple	2FA
[6]	Sheu II	$2.5n^2$	Simple	FA
[7]	Eldridge	$4n^2$	Simple	2FA
[8]	Wu	$2n^2$	No	FA
[9]	Juang	$6n^2$	Simple	FA
	Ours	$2.67n^2$	Simple	FA

6. REFFRENCES

[1]R. L Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Com. of ACM*, vol. 21, no. 2, pp. 120-126, Feb. 1978.

[2]Naofumi Takagi and Shuzo Yajima, "Modular multiplication hardware algorithms with a redundant representation and their application to RSA cryptosystem", *IEEE Trans. on Computers*, vol. 41, no. 7, pp. 887-891, July 1992.

[3]P. Adrian Wang, Wei-Chang Tsai, and C. Bernard Shung, "New VLSI architecture of RSA public-key cryptosystem," *IEEE Int. Symp. on Circuits and Systems*, vol. 3, pp. 2040-2043, 1997.

[4]Po-Song Chen, Shih-Arn Hwang, and Cheng-Wen Wu, "A systolic RSA public key cryptosystem," *IEEE Int. Symp. on Circuits and Systems*, vol. 4, pp. 408-411, 1996.

[5]Yong-Jin Jeong, and Wayne P. Burleson, "VLSI array algorithm and architectures for RSA modular multiplication," *IEEE Trans. on Very Large Scale Integration (VLSI) System*, vol. 5, no. 2, pp. 211-217, June 1997.

[6]Jia-Lin Sheu, Ming-Der Shieh, Chien-Hsing Wu, and Ming-Hwa Sheu, "A pipelined architecture of fast modular multiplication for RSA cryptography," *IEEE Int. Symp. on Circuits and Systems*, vol. 2, pp. 121-124, 1998.

[7]S. E. Eldridge, "A faster modular multiplication algorithm," *Int. J. Computer Math*, vol. 40, pp. 63-68, 1991.

[8]C. Y. Su and C. W. Wu, "A practical VLSI architecture for RSA public-key cryptosystem," *Proc. Sixth VLSI Design/CAD Symp.*, pp. 273-276, August 1995.

[9]Y. J. Juang, E. H. Lee, and C. H. Chen, "A new architecture for fast modular multiplication," *Int. Symp. on VLSI Technology, System, and Application*, pp. 357-360, 1989.

[10]D. E. Knuth, *The Art of Computing*, vol. 2, Seminumerical Algorithms, Addison-Wesley, New York, 1981.

[11]C. Yang, "IC design of a high speed RSA processor," *Master Thesis, Institute of Electronic, National Chiao Tung University*, Hsin-Chu, Taiwan, June 1996.

[12]N. Takagi, "A radix-4 modular multiplication hardware algorithm efficient for iterative modular multiplcations," *10th IEEE Symp. on Computer Arithmetic*, pp. 35-42, 1991.

[13]M. Shand and J. Vuillemin, "Fast implementation of RAS cryptograph," *11th IEEE Symp. on Computer Arithmetic*, pp. 252-259, 1993.