I.J.Modern Education and Computer Science, **2011, 3, 8-14**
Published Online June 2011 in MECS (http://www.mecs-press.org/)

Design and Implementation of a Low Power RSA Processor for Smartcard

Zhen Huang
Institute of Microelectronics Tsinghua University, Beijing, China
Email: huangz04@mails.tsinghua.edu.cn

Shuguo Li
Institute of Microelectronics Tsinghua University, Beijing, China
Email: lisg@tsinghua.edu.cn

***Abstract*—Power consumption limits the application of public key cryptosystem in portable devices. This paper proposes a low power design of 1,024-bit RSA. In algorithm, the Chinese Remainder Theorem (CRT) and an improved Montgomery algorithm are selected to decrease the computation of RSA. In architecture and circuit, the operand isolation technique is applied to avoid unnecessary flip-flops of the combinational logic, and the clock gating technique is used to reduce the power dissipation of the registers. The proposed design is functionally verified on Altera FPGA EP2C8Q208C8N device. With SMIC 0.18μm CMOS process, the Synopsys synthesizing result shows that the area and the critical path are 7.1k gates and 5.3ns respectively, while the power is 2.56mW and the throughput can reach 49 kbps. Thus the proposed design requires lower power than previous designs.**

***Index Terms*—Montgomery algorithm; Chinese Remainder Theory(CRT); operand isolation; clock gating**

I. Introduction

As the information technology develops, the security of information becomes more and more important. Public key cryptography is becoming the preferred solution for information security because of its advantage in distribution and management of the keys. Nowadays, the RSA cryptography [1] is the most widely used public key cryptography, the security of which is based on the Integer Factorization Problem (IFP). The primary operation of the RSA signature is modular exponentiation, which can be realized with modular multiplication. Montgomery [2] algorithm realizes the modular multiplication with addition and shift, making the VLSI implementation of modular multiplication possible.

One of the most critical indices for portable devices is power consumption. The length of the operands of RSA cryptosystem may be up to 1,024-bit or even 2,048-bit to achieve required security class. However, the large area and high power consumption usually make the application of RSA cryptosystem impossible for battery-powered and passive devices. Therefore, low power design becomes the challenge for application of public key cryptosystem RSA in portable devices.

This paper is organized as below: Section II, the description of the design of algorithm level; Section III, the design of architecture and circuit level; Section IV, comparison between the result of the proposed design and previous work; and Section V, the summary and the conclusion.

II. Algorithm Selected

A. RSA Modular Exponentiation

The major operation is modular exponentiation when the RSA cryptosystem performs encryption and decryption. That means the implementation of RSA cryptography is actually the implementation of modular exponentiation. There are several methods to realize modular exponentiation, for example, binary, m-ray and sliding window. For low power design, the binary method is preferred because of its simplicity, low memory requirement and low power consumption.

There are two ways to execute the binary modular exponentiation, namely the left-to-right method and the right-to-left method. They are shown in Algorithm 1, where E is the key; N = P×Q is the module (P and Q are the two primes); M is the message to be encrypted and C is the cryptography:

Algorithm 1: Binary Modular Exponentiation
The Left-to-Right Method:
Input: N, M, E
Output: $C = M^E \pmod N$
1. If $E_{n-1} = 1$ then C = M else C = 1
2. For i = k -2 downto 0
   a) $C = C \times C \pmod N$
   b) If $E_i = 1$ then $C = C \times M \pmod N$

The Right-to-Left Method:
Input: N, M, E
Output: $C = M^E \pmod N$
1. If $E_0 = 1$ then C = M else C = 1
2. For i = 1 upto k-1
   a) $M = M \times M \pmod N$
   b) If $E_i = 1$ then $C = C \times M \pmod N$

A reasonable assumption can be made that half of E's digital bits are logic "1", resulting in that the n-bit

modular exponentiation requires 1.5n modular multiplications on average.

B. Chinese Remainder Theorem (CRT)

Chinese Remainder Theorem (CRT) [3] can realize a 2n-bit RSA modular exponentiation with two n-bit modular exponentiations, which can effectively reduce both time complexity and power consumption of RSA cryptosystem.

As mentioned above, the major operations of an n-bit modular exponentiation are about 1.5n modular multiplications on average. The most effective modular multiplication algorithm so far needs three n-bit multiplications to realize an n-bit modular multiplication (This will be discussed later). Thus 4.5n n-bit multiplications are required to realize an n-bit modular exponentiation and that for 2n-bit modular exponentiation is 9n.

Comparing the complexity of one 2n-bit modular exponentiation with that of two n-bit modular exponentiations, it is found that the numbers of multiplications are the same, but the complexity of an n-bit multiplication is approximately a quarter of that of a 2n-bit multiplication. That means the complexity of a RSA modular exponentiation without CRT is about 4 times higher than a RSA modular exponentiation with CRT.

Low power consumption is the most critical for portable device design. Chinese Remainder Theorem can reduce the amount of operations for RSA modular exponentiation to about a quarter of that of original RSA modular exponentiation, and this reduction will significantly decrease the power consumption. So CRT is chosen for our design.

Algorithm 2 describes the Chinese Remainder Theorem, where M is the message to signature; P and Q are the two primes and P<Q, N = P×Q; E is the private key, 0 < A, B < Q-1 and A×P ≡ 1 (mod Q), B×Q ≡ 1 (mod P).

Algorithm 2: Chinese Remainder Theorem (SRC)
Input: M, N, P, Q, E, A, B
Output: $C = M^E \pmod N$

1. $Mp = M \pmod P$, $Mq = M \pmod Q$
2. $Ep = E \pmod{P-1}$, $Eq = E \pmod{Q-1}$
3. $Cp = C \pmod P = Mp^{Ep} \pmod P$
4. $Cq = C \pmod Q = Mq^{Eq} \pmod Q$
5. $C = CpBQ + CqAP \pmod N$

In practice, Ep and Eq must be pre-computed because Montgomery algorithm can only work with odd module, so it is impossible to get Ep and Eq with Montgomery algorithm.

The method to execute Chinese Remainder Theorem in Algorithm 2 is Single-Radix Conversion (SRC). To realize RSA modular with SRC, four modular multiplications and one modular addition with module N should be performed. This is not good news for hardware implementation, because one more module means not only one more input parameter, but also a few more pre-computation. On the other hand, two modular inverses should be performed in the pre-computation and then both modular inverse results should be added to the list of input parameters.

The Mixed-Radix Conversion (MRC) was first proposed by H.L.Garner in 1958, and then improved by D. E. Kunth. To execute CRT with MRC, the modular operations with module N is unnecessary and only one modular inverse is required. Both of these two improvements are good for hardware implementation.

Algorithm 3 describes the Chinese Remainder Theorem with Mixed-Radix Conversion, where M is the message to signature; P and Q are the two primes and P<Q, N = P×Q; E is the private key, 0 < A < Q-1 and A×P ≡ 1 (mod Q).

Algorithm 3: Chinese Remainder Theorem (MRC)
Input: M, P, Q, E, A
Output: $C = M^E \pmod N$

1. $Mp = M \pmod P$, $Mq = M \pmod Q$
2. $Ep = E \pmod{P-1}$, $Eq = E \pmod{Q-1}$
3. $Cp = C \pmod P = Mp^{Ep} \pmod P$
4. $Cq = C \pmod Q = Mq^{Eq} \pmod Q$
5. $C = [((Cq + Q - Cp)A) \pmod Q]P + Cp$

The purpose of the extra addition of Q is to make sure that the Intermediate result is positive. To realize CRT with MRC, only one modular multiplication with module P and one multiplication are required. The more important thing is that with MRC, we don't parameters N and B in algorithm 2. This improvement can reduce the size of required memory and eliminate the hardware for the modular operations with module N. Because N = P×Q, so the length of N is normally twice as the length of P and Q. So the elimination of the modular operations with module N means all the modular operations have modules with same length, so they can be realized with the same hardware device.

On the other hand, the performance of RSA modular exponentiation can be improved by Chinese Remainder Theorem as well. In high speed applications, step 3 and step 4 can be executed in parallel, reducing the time complexity of modular exponentiation to only a quarter of the original value.

C. Improved Montgomery Algorithm

The main operation of RSA encryption and decryption is modular exponentiation, and modular exponentiation consists of modular multiplications, so the actual implement of RSA cryptography is the implement of modular multiplication.

The general process of modular multiplication consists of two steps: computing T = A×B and reducing T to yield M = A×B (mod N). The traditional reduction, which is implemented with division operation, is not easy for VLSI implement as there is not any good solution for VLSI implement of division proposed yet. Several modular multiplication algorithms without division operation have been proposed to replace the traditional reduction. Among them the two most popular algorithms are Montgomery algorithm and Barrett algorithm.

Montgomery algorithm is the most widely used as well as the most efficient modular multiplication algorithm being applied. It realizes the reduction with addition and shift operations, and they are much easier for VLSI

implement than division operation. Algorithm 4 shows the original Montgomery algorithm where N is the n-bit module, should be odd and N>1; R is relative prime to N and normally $R = 2^n$, R^{-1} and N satisfy $0 < R^{-1} < N$, $0 < N' < R$, $RR^{-1} - NN' = 1$. That is $RR^{-1} \pmod N = 1$ or $NN' \pmod R = -1$; A and B are the two multipliers:

Algorithm 4: Montgomery Modular Multiplication

Input: A, B, N, N′

Output: $t = A \times B \times R^{-1} \pmod N$

1. $T = A \times B$
2. $m = (T \bmod R)\, N' \bmod R$
3. $t = (T + m \times N) / R$

if $t \geq N$ then $t = t - N$

Note that the result of Algorithm 4 is $ABR^{-1} \pmod N$ instead of AB (mod N). When using it in practice the constant R^{-1} should be eliminated. That means Algorithm 4 should be executed one more time.

The primary operation for the hardware implementation of Montgomery algorithm is to divided T by 2 (because $TR^{-1} = T2^{-n}$), and the operation of division by 2 can be realize by shift right. It should be remembered that the shift operations are with modular N, so if T is odd, T = T + N should be performed before the shift operation. After divisions by 2 for n times, the result t is obtained. The result t will satisfy $0 \leq t < 2N-1$, but it is not the final result of the modular multiplication. It is $T/2^n$ mod N instead of T mod N, so we can not get the final result until another transfer, which is a multiplication by 2^n is performed.

The hardware implementation of the original Montgomery algorithm is shown in Algorithm 5. A and B are the two n-bit binary multipliers and can be presented as $A = (a_{n-1}, a_{n-2}, \ldots\ldots, a_1, a_0)$, $B = (b_{n-1}, b_{n-2}, \ldots\ldots, b_1, b_0)$, N is the n-bit modular and $R = 2^n$:

Algorithm 5: Hardware Implementation of Original Montgomery Algorithm

Input: A, B, N

Output: $T = ABR^{-1} \bmod N$

1. $S_0 = 0$
2. For i = 0 upto n-1
   if ($S_i + a_iB$ is even) then $S_{i+1} = (S_i + a_iB)/2$
   else $S_{i+1} = (S_i + a_iB + N)/2$
3. $T = S_n$

In fact, Algorithm 4 is not used very much, in practice, another method called "High-base" algorithm is more widely used, and it can be shown as Algorithm 6. A, B and N are presented as $A = (a_{w-1}, a_{w-2}, \ldots\ldots, a_1, a_0)_r$, $B = (b_{w-1}, b_{w-2}, \ldots\ldots, b_1, b_0)_r$, $N = (n_{w-1}, n_{w-2}, \ldots\ldots, n_1, n_0)_r$, where $R = 2^n = r^w$, and $0 \leq a_i < r$, $0 \leq b_i < r$, $0 \leq n_i < r$, and n_0 and n_0' satisfy $n_0 n_0' \bmod r = -1$.

Algorithm 6: Modified Hardware Implementation of Montgomery Algorithm

Input: A, B, N, n_0'

Output: $T = ABR^{-1} \bmod N$

1. $S_0 = 0$
2. For i = 0 upto w-1
   $m_i = (S_i + a_iB)\, n_0' \bmod r$
   $S_{i+1} = (S_i + a_iB + m_iN)/r$
3. $T = S_n$

When r = 2, then $n_0 = 1$ and $n_0' = 1$, Algorithm 6 will degenerate to Algorithm 5.

The original Montgomery modular multiplication is difficult to realize with VLSI because the operands in the cryptosystem applications are usually very large. When the length of the integers is 1,024-bit or 2,048-bit, even the implementation of the simplest operations like addition and shift are impossible. As a result, the big operands are usually divided into a serious of small operands, so that operations on big operands can be executed.

Apparently, the number of arithmetic units can be reduced when the big operands are divided into small operands, and many registers can also be saved because the storage of the big intermediate results will become unnecessary.

Various methods have been proposed to improve the Montgomery algorithm under this idea [4]. With these modified algorithms, it is much easier to implement Montgomery modular multiplication, and higher performance can be achieved with lower power.

The FIPS [4] algorithm proposed by KoC is one of those modified algorithms suitable for implementation with VLSI, especially for the implementations with digital signal processors. The most important feature of this algorithm is that in this algorithm, there is no need to obtain the result of T = A×B. The less significant words of the temporary and the more significant words of the intermediate result can be stored in the same memory units, and the temporary value and the result can be stored in the same memory address, too. As a result, it reduces the size of the memory as well as the number of operations. Both the cost of hardware and the power consumption are therefore lowered. The FIPS algorithm is described in Algorithm 7:

Suppose A, B and N are decomposed to $A = (a_{s-1}\, a_{s-2} \ldots a_1\, a_0)_r$, $B = (b_{s-1}\, b_{s-2} \ldots b_1\, b_0)_r$, and $N = (n_{s-1}\, n_{s-2} \ldots n_1\, n_0)_r$, where $0 \leq a_i < r$, $0 \leq b_i < r$, and $0 \leq n_i < r$. Normally r is chosen to be the power of 2. Let $R = r^s$, so $N<R$ ($A<N$, $B<N$), and $n_0 n_0' \pmod r = -1$, where n_0' is pre-computed.

Algorithm 7: FIPS Modular Multiplication

Input: A, B, N, n_0'

Output: $M = ABR^{-1} \bmod N$

1. $S = 0$

//Step A: calculate the temporary value m[i]

2. for i = 0 to s-1
   a) for j = 0 to i-1
      i. $S = S + a[j]b[i-j] + m[j]n[i-j]$
   b) $S = S + a[i]b[0]$
   c) $m[i] = S\, n_0' \pmod r$
   d) $S = S + m[i]n[0]$
   e) $S = S/r$

//Step B: calculate the result and store the result in m[i]

3. for i = s to 2s -1
   a) for j = i – s + 1 to s – 1
      i. $S = S + a[j]b[i-j] + m[j]n[i-j]$
   b) $m[i-s] = S \pmod r$
   c) $S = S/r$

//Step C: adjust the result to [0, N)

4. temp = S (mod r)

5. carry = 1
6. for i = 0 to s-1
   a) (carry, b[i]) = m[i] + not(n[i]) + carry
7. temp = temp + not(0) + carry
8. if temp =0
   then for i = 0 to s-1 m[i] = b[i]

III. Archetecture and Circuit Design

A. The Architecture Design

Algorithm 7 shows that most operations of the FIPS Montgomery modular multiplication are multiplications and accumulation. Considering the tradeoff between the power consumption and the necessary speed, the width of the multiplier is chosen as 16-bit. Thus, in Algorithm 4, r = 2^{16}, and s = 512 /16 = 32.

The length of the accumulator can then be decided. For our design, the accumulator will store the largest temporary result when performing Step A of Algorithm 4. When i = s-1, the largest temporary result is the sum of (1), (2) and the result of S/r in i = s-2:

$$S1 = a[0]b[s-1] + a[1]b[s-2] + \ldots + a[s-1]b[0]. \quad (1)$$

$$S2 = m[0]b[s-1] + m[1]n[s-2] + \ldots + m[s-2]n[1]. \quad (2)$$

Equations (1) and (2) show that there will be 2s-1 32-bit numbers to be accumulated, which means that the sum will be less than $(2s-1)\times 2^{32} = 2^{38} - 2^{32}$. After performing the shifting S/r, the length of the initial value in the accumulation register is 32-bit, and hence the final result will be less than 2^{38}. Then the length of the adder can be chosen as 38-bit. Because the width of the SRAM used in this design is 16-bit, taking the length of register to be times of 16 will make the shifting of operands much easier. As a result, the length of the accumulating register is 48-bit. The data path of the modular multiplier is shown in Fig. 1.

B. Operand Isolation

The power consumption can be divided into two parts: static and dynamic power consumption. The static power is determined by the fabrication process, where possible input from the designers is quite limited. The calculations of the combinational logic and the toggles of the registers take up most of the dynamic power. As a result, the general idea of low power design is to eliminate the unnecessary calculations of the combinational logic and the redundant toggles of the registers.

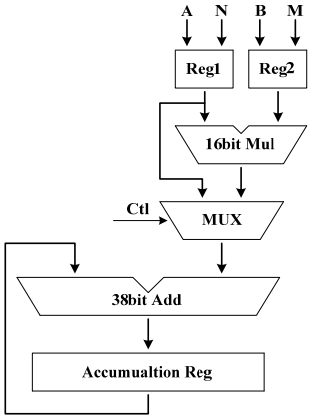

Figure 1. The architecture of the modular multiplier

Combinational logic circuits are used to realize different logic and arithmetic functions. In general, no circuit is expected to keep working at any time. Ideally, we want them to work only when we want to use them. But this is usually not that case, the combinational logic circuit will begin to calculate once its input signals are changed. Fig. 2 shows the structure of an ALU for a RISC CPU, where the arithmetic circuit, the logical circuit, the shift circuit and the comparison circuit are connected parallel. When the input signals are changed, all of the four parts will work together, but obviously only one of the four results will be used at the most. That means the circuits consume power without generating any useful data. So in low power design, we hope that the circuits will not work until we want them to work. The only way to stop the combinational circuits from working is to keep the input of the circuits still. And an effective method to keep the input still is the so-called Operand Isolation.

The idea of operands isolation is to keep the inputs of the combinational logics constant when they are not being used, so that the combinational logics will keep quiet and power required during the idle period is reduced. Fig. 3 illustrates the operands isolation of the modular multiplier. The shadows are the isolation modules. The inputs of both the multiplier and the adder stay constant when shift is being performed. Only the former stays constant when performing additions and subtractions.

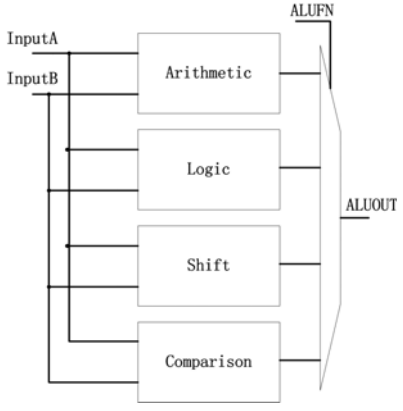

Figure 2. The structure of ALU

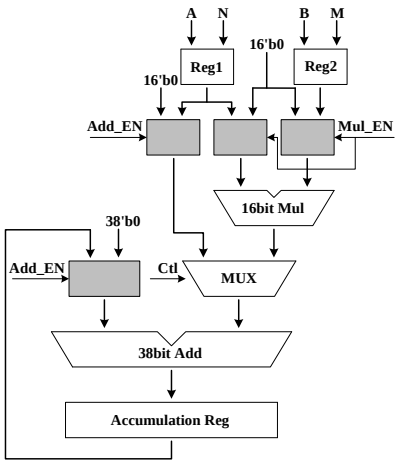

Figure 3. The operands isolation of the modular multiplier

C. Clock Gating

In digital logic circuits, the clock signal has the maximum fan out and the highest activity, so the power consumption from the clock signal is an important part of the total power consumption of the whole circuit.

Fig. 4 is the functional schematic of a D-type flip-flop. It shows that the clock signal needs to drive four transfer gates in every D-type flip-flop. When the transfer gates are switch, no matter whether the value of the flip-flop is changed, the charging and discharging of the capacity of the transfer gates will consume dynamic power. There can be thousands of flip-flops in a large design, so the power consumption on all these flip-flops will be a problem for low power design.

Fig. 5 shows the principle of clock gating. The upper part of Fig. 5 shows the equivalent schematic of a D-type flip-flop with synchronous active-high enabling signal. It can be seen that the enabling signal acts as the control signal of a 2-to-1 multiplexer, but it has no influence on the clock signal, which is connected to the control signals of the transfer gates. This means that the clock in this flip-flop will consume power, no matter whether the enabling signal is high.

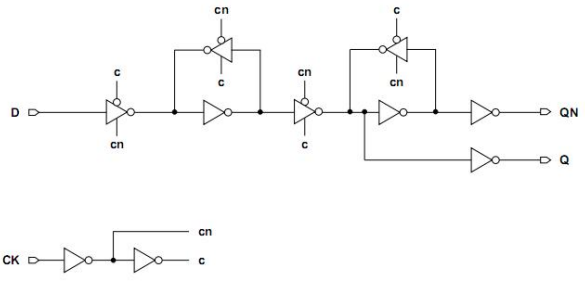

Figure 4. The functional schematic of D-type flip-flop

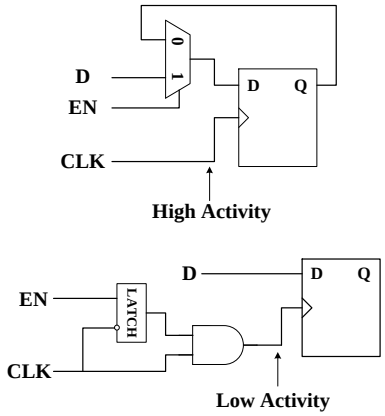

Figure 5. The principle of clock gating

As a result, reducing the clock activity effectively reduces the unnecessary power dissipation. One solution is to insert clock gating. When clock gating is inserted, the clock net of the flip-flop will be kept at logic "0", as shown in the lower part of Fig. 5, thus the power dissipated on the switch is saved. The most power in an ASIC design is consumed by the clock tree, so the reduction of clock activity will notably reduce the total power.

IV. The Result and Comparison

The proposed design has been implemented by Verilog HDL, simulated with ModelSim 6.2b and synthesized with Synopsys Design Compiler with SMIC 0.18μm process. The result shows that the critical path of the design is 5.3ns, so the highest clock frequency of the design can be up to 188MHz and its area is 7.1k gates. It spends about 3.9M clock cycles to finish a 1,024bit RSA signature, so the throughput of this design is about 49kbps at 188MHz. When working on the frequency of 188MHz, it requires a power of 2.56mW. Implemented with Altera FPGA EP2C8Q208C8N device, the proposed design costs 2,439 logic elements and can work at the frequency of 47.32MHz.

The performance and power consumption of proposed design are compared with previous works in TABLE I and TABLE II.

TABLE I. Comparison of Performance

Design	Year	Technique	Frequency	Throughput (1024-bit)
[6]	2006	UMC0.18μm	460MHz	586kbps
[7]	2008	TSMC0.18 μm	200MHz	107.5kbps
[8]	2009	SMIC0.13 μm	196MHz	5.4kbps
proposed	2010	SMIC0.18 μm	188MHz	49kbps

TABAE II. COMPARISON OF POWER

Design	Area	Power	Power/Throughput
[6]	5.76mm^2	830mW	1.42mW/kbps
[7]	61k Gates	32.5mW	0.30mW/kbps
[8]	2.6k Gates	2.27mW	0.42mW/kbps
proposed	7.1k Gates	2.56mW	0.052mW/kbps

From these tables, we can see that the proposed design has lower ratio (power/throughput), which means it requires less power than the previous designs when working at the same speed.

The proposed RSA processor is implemented by Cadence SoC Encounter 8.1 with SMIC 0.18μm CMOS process and is integrated in a smartcard with Anti-counterfeiting capability. The two SRAM blocks are generated by SMIC SRAM Generator and the EEPROM IP is S018EE16KBS_LPI from SMIC. The design can execute 1,024-biy RSA digital signature as well as User data read/write. The layout of the smartcard with proposed RSA processor is shown in Fig.6.

V. CONCLUSION

The proposed 1,024-bit RSA design achieves ultra low power by using the Chinese Remainder Theorem, improved Montgomery algorithm and several low power techniques. The synthesizing result shows that it has a performance of 49kbps at 188MHz while consuming only 2.56mW and the area is only 7.1k gates. The low power and low area make it suitable for smartcards and portable device.

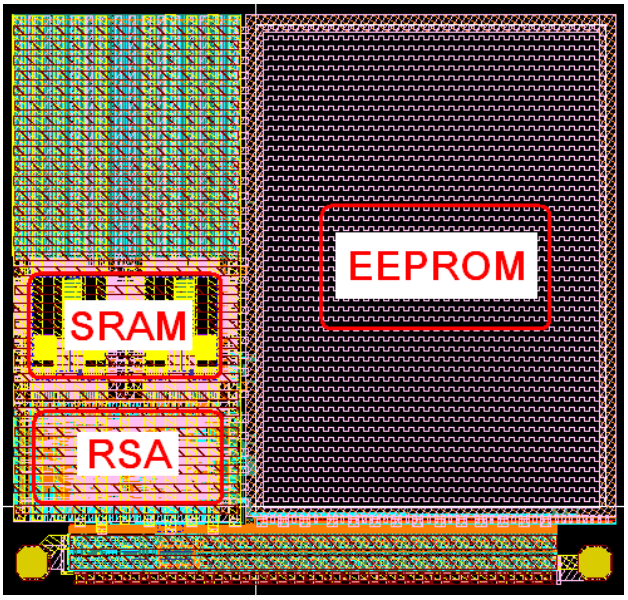

Figure 6. Layout of the smartcard with RSA processor

APPENDIX A THE PROOF OF CHINESE REMAINDER THEOREM

Two theorems will be used when performing RSA modular exponentiation. They are shown as Theorem 1 and Theorem 2.

Theorem 1: Chinese Remainder Theorem [9]

Suppose $m_1, m_2, \ldots, m_k$ are relative primes to each other, then congruence equation:

$$\begin{cases} x \equiv b_1 \pmod{m_1} \\ x \equiv b_2 \pmod{m_2} \\ \quad\cdots\cdots \\ x \equiv b_k \pmod{m_k} \end{cases}$$

has unique solution.

Theorem 2: Fermat's Little Theorem [10]

Suppose a is an integer and p is a prime number, then

$$a^p \equiv a \pmod{p}$$

And it can also be written as

$$a^{p-1} \equiv 1 \pmod{p}$$

There are two different algorithms to obtain the unique solution in Theorem 1. The first algorithm is the Single-Radix Conversion, which is shown as Algorithm 8, and the improved Mixed-Radix Conversion algorithm is shown as Algorithm 9.

Algorithm 9: The Single-Radix Conversion [11]

1. $M = m_1 m_2 \ldots m_k$
2. $M_i = M/m_i \ (1 \le i \le k)$
3. $c_i = M_i^{-1} \pmod{m_i}$
4. $x = b_1 c_1 M_1 + b_2 c_2 M_2 + \ldots + b_k c_k M_k \pmod{M}$

Algorithm 10: The Mixed-Radix Conversion [10]

1. $A_{ij} = m_i^{-1} \pmod{m_j} \ (1 \le i < j \le k)$
2. $C_1 = b_1 \pmod{m_1}$
   $C_2 = (b_2 - C_1)A_{12} \pmod{m_2}$
   ……
   $C_k = (b_k - (C_1 + m_1(C_2 + m_2(C_3 + \ldots + m_{k-2}C_{k-1}))))A_{1k}A_{2k}\ldots A_{(k-1)k} \pmod{m_k}$
3. $x = C_k m_{k-1} \ldots m_2 m_1 + \ldots + C_3 m_2 m_1 + C_2 m_1 + C_1$

In the case of RSA modular exponentiation, there are only two prime numbers P and Q, and $N = P \times Q$, so k=2. Then the solutions for the two algorithms can be shown as follow:

With SRC: $x = b_1 A_{pq} P + b_2 A_{qp} Q \pmod{N}$

With MRC: $x = ((b_2 - b_1)\, A_{pq} \pmod{Q})P + b_1$

where $A_{pq} = P^{-1} \pmod{Q}$ and $A_{qp} = Q^{-1} \pmod{P}$

The last symbols need to be replaced are b_1 and b_2. According to RSA cryptography and Chinese Remainder Theorem, the original expressions of b_1 and b_2 are:

$$b_1 = M^E \pmod{P},\ b_2 = M^E \pmod{Q}$$

M and E can be written as $M = UP + Mp$ and $E = V(P-1) + Ep$, so $Mp = M \pmod{P}$ and $Ep = E \pmod{P-1}$, Then $b_1 = M^E \pmod{P} = (UP + Mp)^{V(P-1)+Ep} \pmod{P} = Mp^{V(P-1)} Mp^{Ep} \pmod{P} = Mp^{Ep} \pmod{P}$.

With the same process, we can deduce that $b_2 = Mq^{Eq} \pmod{Q}$. Let $Cp = Mp^{Ep} \pmod{P}$ and $Cq = Mq^{Eq} \pmod{Q}$, the solutions can be written as:

With SRC: $x = CpA_{pq}P + CqA_{qp}Q \pmod{N}$

With MRC: $x = ((Cq - Cp)\, A_{pq} \pmod{Q})P + Cp$

APPENDIX B THE PROOF OF MONTGOMERY ALGORITHM

Proof of Algorithm 3:

Known: N is odd and N>1; R is relative prime to N, and $0 < R^{-1} < N$, $0 < N' < R$, $RR^{-1} - NN' = 1$. That is $RR^{-1} \pmod N = 1$ or $NN' \pmod R = -1$, $m = (T \bmod R)\, N' \bmod R$.

Solution:

Because $mN = ((T \bmod R)N' \bmod R)N$
Then $mN \bmod R = TNN' \bmod R$
And because $NN' \bmod R = -1$
So $mN \bmod R = -T \bmod R$
Then $(T + mN) \bmod R = (T-T) \bmod R = 0$
So $t = (T + mN)/R$ is an integer and $tR = T + mN$
Then $tR \bmod N = T \bmod N$
Then $tRR^{-1} \bmod N = TR^{-1} \bmod N$
Because $RR^{-1} \bmod N = 1$
Then $t \bmod N = TR^{-1} \bmod N$
So $t = TR^{-1} \bmod N$
Because $m = (T \bmod R)\, N' \bmod R$
So $0 \le m < R$
And $0 \le T < RN$
So $0 \le T + mN < RN + RN$
And because $t = (T + mN)/R$
So $0 \le t < 2N$

Proof of Algorithm 5:

Known: A, B and N are n-bit integers and are presented as $A = (a_{w-1}, a_{w-2}, \ldots\ldots, a_1, a_0)_r$, $B = (b_{w-1}, b_{w-2}, \ldots\ldots, b_1, b_0)_r$, $N = (n_{w-1}, n_{w-2}, \ldots\ldots, n_1, n_0)_r$, where $R = 2^n = r^w$, and $0 \le a_i < r$, $0 \le b_i < r$, $0 \le n_i < r$, and n_0 and n_0' satisfy $n_0 n_0' \bmod r = -1$.

Solution:

Because $n_0 n_0' \bmod r = -1$
So $((S_i + a_i B)\, n_0' \bmod r) n_0 \bmod r = -(S_i + a_i B) \bmod r$
And $S_i + a_i B + m_i N$
$= (S_i + a_i B) + ((S_i + a_i B)\, n_0' \bmod r)N$
So $(S_i + a_i B + m_i N) \bmod r = 0$
So $S_i + a_i B + m_i N$ is divisible by r
So $rS_{i+1} = S_i + a_i B + m_i N$
Then we get the follow equations:

$rS_1 = a_0 B + m_0 N$ ……(1)
$rS_2 = S_1 + a_1 B + m_1 N$ ……(2)
……
$rS_w = S_{w-1} + a_{w-1} B + m_{w-1} N$ ……(w)

(1) + (2) × r + (3) × r^2 + … + (w) × r^{w-1}
Then $r^w S_w = AB + MN$
That is $RS_w = AB + MN < 2RN$
Where $M = m_0 + m_1 r + m_2 r^2 + \ldots + m_{w-1} r^{w-1}$
So $S_w = AB\, R^{-1} \bmod N$ and $S_w < 2N$

ACKNOWLEDGMENT

This work was supported by the National Natural Science foundation of China (No.61073173), National High-Tech Research and Development Program of China (No.2006AA01Z418). The authors would like to thank the editor and the reviewers for their comments.

REFERENCES

[1] R. L. Rivest, A. Shamir and L. A. Adleman, "Method for obtain digital signatures and public-key cryptosystems," Communications of the ACM, 1978, 21(2): 120-126.
[2] P. L. Montgomery, "Modular mulbtiplication without trial division," Mathematics of Computation, 1985, 44(170): 519-521.
[3] J. -J. Quisquater, C. Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," Electronics Letters, 1982, 18(21): 905-907.
[4] C. K. KoC, T. Acar, "Analyzing and comparing Montgomery multiplication algorithms," [J]. IEEE Micro, 1996, 16(3): 26-33.
[5] Xiqing Yu, ASIC Design Practical Course (in Chinese), Zhejiang University Press, Hangzhou, China: Jan. 2007, pp. 229-280.
[6] Chingwei Yeh, En-Feng Hsu, Kai-Wen Cheng, Jinn-Shyan Wang, Nai-Jen Chang, "An 830mW, 586kbps, 1024-bit RSA chip design," Processings of the Conference on Design, Automation and Test in Europe, Mar. 2006.
[7] Xinjian Zheng, Zexiang Liu, Bo Peng, "Design and implementation of an ultra low power RSA coprocessor," WiCOM' 08. 4th International Conference on Wireless Communications, Networking and Mobile Computing, Oct. 2008. pp. 1-5.
[8] Wei Huang, Kaidi You, Suiyu Zhang, Jun Han, Xiaoyang Zeng, "Unified low cost crypto architecture accelerating RSA/SHA-1 for security processor," ASICON'09. IEEE 8th International Conference on ASIC, Oct. 2009. pp. 151-154.
[9] Guanzhang Hu, Dianjun Wang, Applied Modern Algebra 3rd edition (in Chinese), Tsinghua University Press, Beijing, China: Jul. 2006, pp. 29-34.
[10] K. H. Rose, Elementary Number Theory and Its Application [M], Addison-Wesley, 1984.
[11] D. E. Kunth, The Art of Computer Programming: Semi-numerical Algorithms[M], Volume 2. Addison-Wesley, 3rd edition, 1998

Zhen Huang was born in the province of Guangxi, China, in March, 1985. He received the Bachelor degree of Engineering in Department of Micro and Nano Electronics, Tsinghua University, Beijing, China in 2008.

He is now pursuing the Master degree of Engineering in Institute of Microelectronics, Tsinghua University. His research interest is VLSI Implementation of information security processor.

Shuguo Li received the Bachelor, the Master and the PhD degree in Computer Department from Xidian University, Shandong University and Northwestern Polytechnical University in China in 1986, 1993 and 1999 respectively. In 2001, he finished his postdoctoral position research at Tsinghua University.

Now he is an associate professor at the Institute of Microelectronics at Tsinghua University. His current research interests the algorithm for cryptography and design for encryption processor and microprocessor processor.

