Improvements of RSA algorithm for hardware encryption implementation based on FPGA

Li Youguo
Department of Computer Science
Xinyang Agricultural College
Xinyang, Henan, China
liyouguoxs@163.com

***Abstract*—In the field of information security, encryption and decryption operations in RSA cryptographic algorithm are both modular exponentiation. The direct computing method to implement this algorithm through software programming is to multiply M e times with iteration statements and later make modulo operations. The operational speed of this method is feasible when M, e and n are relatively small. However, in order to enhance the security of RSA algorithm, we need to take at least 512-bit values for M, e and n. The operation speed in the way of software implementation is slow, and intermediate results will also take up lots of temporary storage space. Thus, software implementation has great difficulties. In contrast, hardware encryption has high security, fast speed and strong real-time property. At present, FPGA-based RSA hardware encryption and decryption is a new research direction, and an improved study on the existing low-radix Montgomery algorithm has been made in this article.**

Keywords-Montgomery modular multiplication algorithm, Exclusive or arithmetic unit, RSA algorithm, shift arithmetic unit

I. INTRODUCTION TO HARDWARE COMPUTING IN RSA ENCRYPTION ALGORITHM

RSA security is based on the presumed difficulty of factoring large integers, and the reason is that till now the assumption that factoring integers is an np issue has not yet been proved. Perhaps, there is still a polynomial time decomposition algorithm which has not yet been found[1]. The factorization technique adopted in 1995 showed that, for a 512-bit secret key, we had to spend more than 1 million dollars on factorization in 10 months. In an attacking experiment in 1999, for a specific 512-bit secret key, it took 7 months to complete factorization. The risks of secret key attacks mainly rely on the constant growth of computing power and the improvement of factorization algorithm. It is estimated that in the future for a longer period of time, RSA is secured which has a secret key length between 1024 bits and 2048 bits. Also, RSA's creators required similar bit-length for p and q, and the modulus n itself could not be a prime number. In modular arithmetic, we often calculate what is formed like $m^e \bmod n$, and work out $m^2, m^3, \cdots, m^e \bmod n$ in turn when e is relatively small. But when e is large, this calculation is unrealistic, for the amount of calculation and the intermediate storage data size will be tremendous, so as to lose the meaning of fast encryption and decryption. In the practical calculation, we often use repeated modular squaring algorithm: the number of modular multiplication operations that $m^e \bmod n$ requires is reduced to at most 2k, of which k=log 2e.Suppose |n|=l, then the complexity of $m^e \bmod n$ is O(kl2).

II. MATHEMATICAL FOUNDATIONS OF RSA ALGORITHM

A. Repeated modular squaring algorithm

In modular arithmetic, we often calculate what is formed like $m^e \bmod n$, and work out $m^2, m^3, \cdots, m^e \bmod n$ in turn when e is relatively small. But when e is large, this calculation is unrealistic, for the amount of calculation and the intermediate storage data size will be tremendous, so as to lose the meaning of fast encryption and decryption. In the practical calculation, we often use modular repeated squaring algorithm: the number of modular multiplication operations $m^e \bmod n$ requires is reduced to at most 2k, of which k=log 2e. Suppose |n|=l, then the complexity of $m^e \bmod n$ is O(kl2). Suppose e's binary representation is $(e_{k-1} \cdots e_1 e_0)_2$, namely $e = \sum_{i=0}^{k-1} e_i 2^i$.

1) Compute in turn:

$$y_0 = m, y_i = y_{i-1}^2 \bmod n, (i = 1,2,3,\cdots,k-1)$$

2) Compute in turn:

$$x_0 = \begin{cases} y_0 & e_0=1 \\ 1 & e_0=0, \end{cases} x_i = \begin{cases} x_{i-1} y_i \bmod n & e_i=1 \\ x_{i-1} & e_i=0, \end{cases}, (i = 1,2,\cdots,k-1)$$

3) The x_{k-1} which has been finally obtained is just the required $m^e \bmod n$.

B. Euler function

Suppose n is the number of m which are positive integers less than n and that are relatively prime to n defined by the Euler functionφ(n), namelyφ(n) = {0 ≤ m < n | gcd(m ,n) = 1}.

If p is a prime number, then φ(p) = p −1, or φ(p) + 1 = p

If p and q are prime numbers, and n = pq, then

$\varphi(n) = \varphi(p \cdot q) = \varphi(p)\,\varphi(q) = (p-1)(q-1)$

C. Euler Theorem

For any positive integers a and n, if gcd(a ,n) = 1, then $a^{\varphi(n)} \equiv 1 \bmod n$.

In Euler Theorem, for any positive integers a and n, if gcd(a、n) = 1, then a(n) ≡ 1mod n.

RSA is a block cipher, generated by a secret key. From the encryption and decryption process, we may find that the secret key, plaintext and ciphertext all correspond to a certain number, which are then encrypted and decrypted by mathematical operations. An exponential expression has been used in RSA scheme. The plaintext is encrypted by block, of which each block is a binary value less than a number n. That is, the bits of each block should be less than or equal to $\log_2 n$.

D. Seek the greatest common divisor

Euclidean algorithm is an algorithm to seek the greatest common divisor, also known as Method of Successive Division. Theorem: for any nonnegative integers a and b, and $r_0 = a, r_1 = b$, with the division method with a remainder, there exists the following equation: gcd(a,b)=gcd(b,amodb).

Divide successively:

$$r_0 = r_1 q_1 + r_{2,}\ 0 \le r_2 < r_1$$

$$r_1 = r_2 q_2 + r_{3,}\ 0 \le r_3 < r_2$$

………

$$r_{n-2} = r_{n-1} q_{n-1} + r_{n,}\ 0 \le r_n < r_{n-1}$$

$$r_{n-1} = r_n q_n + r_{n+1,}\ r_{n+1} = 0$$

$\gcd(a,b)$ can be calculated by the remainder of each operation dividing the divisor in this operation, which can gradually reduce the values of operands in the operation. In the limited steps n, there must be the fact that $r_{n+1} = 0$. The last nonzero remainder is the greatest common divisor. When the encrypted secret key e is generated, e is any number co-prime toφ(n). To select a number e, check if it is co-prime toφ(n). If not, make e add 1, and check once again until it is co-prime toφ(n). The method to check whether two numbers are co-prime is as follows: check if the gcd of the two numbers is 1; if so, they are co-prime. According to Euclidean algorithm, if a=bn+c and gcd(a,b)=gcd(b,amodb), then gcd(a,b)=gcd(b,c) . So the method of successive division can be used to seek the greatest common divisor.

III. Fast Modular Exponentiation

For RSA algorithm, the secret key 0 or 1 corresponds to the two operations of modular multiplication and modular squaring, so it is easy to obtain the information about the secret key by power analysis. The most commonly used algorithm to seek modular exponentiation of large numbers is BR (Binary Representation) algorithm which can decompose modular exponentiation into a series of modular multiplication operations[3]. First, e's binary conversion is represented as $(e_{k-1} \cdots e_1 e_0)_2$, namely $e = \sum_{i=0}^{k-1} e_i 2^i$. Then, it is inserted into the modular exponentiation formula $m^e \bmod n$. Due to the theorem that ABmodC=(AmodC)BmodC, to seek modular exponentiation of large numbers can be decomposed into a series of issues on seeking modular multiplication of large numbers, so that $m^e \bmod n$ =((me $^{k-1}$ mod n)mod n)m$^{e_{k-2}}$ mod n)m^{e_0} mod n. In this way, the algorithm has been simplified. But after e's binary conversion with BR algorithm, its binary bit becomes very long, and also multiple scannings (iterations) are needed in computing. This method has too low efficiency, and takes up more resources. So there is a need to speed up modular exponentiation algorithm. In hardware implementation (binary scanning algorithm), repeated modular squaring and modular exponentiation algorithm can be simplified to repeated modular squaring and modular multiplication. Based on different scanning directions of the exponent e, the binary scanning algorithm can be divided into the H algorithm from high-order digit to low-order digit and the L algorithm from low-order digit to high-order digit.

With the H algorithm, a register is needed to save C and a modular multiplier, but m can be unsaved, which can be computed by being directly input from the computing port. Seen from the number of registers, there is no much difference between the two algorithms but just one register.

Besides, the third step in H algorithm needs the value in the second step, so sequential execution is required for the two modular multiplication modules; in L algorithm, the two steps are independent of each other, which thus can be executed simultaneously. Therefore, the speed of the H schema is faster than that of the L schema. In addition, seen from the above two algorithms, e's binary bit is n, and the two algorithms both need n-1 modular squaring operations and about （n-1）/2 modular multiplication operations. Modular squaring and modular multiplication are serial operations, which will reduce computing efficiency, and can have some appropriate improvements. The two operations can be computed in advance and saved in registers, which are then executed in parallel and called at any time. But correspondingly, one more register needs to be added. For the overall consideration of the area of a chip and its computing speed, the H algorithm will be used to achieve modular exponentiation, which can save hardware implementation area. With this method, each digit of e can be scanned from high-order digit to low-order digit, so as to check if e is 1. In the subsequent modular exponentiation, we complete the two processes of modular squaring and modular multiplication in this algorithm in parallel, so that computing efficiency can be improved.

IV. THE IMPROVED MONTGOMERY MODULAR MULTIPLICATION ALGORITHM

The key to the speed of modular multiplication is the speed of modular arithmetic that is actually an operation of seeking the remainder of a division. But for the division of large numbers, whether with software or hardware, it takes much more time to realize the division operation than the plus, minus or multiplication operation. So if division is not used, less used, or replaced with other methods, the computing speed of modular multiplication will be greatly improved, so as to correspondingly improve RSA's computing speed to process plaintexts and ciphertexts. Currently, the common methods to seek modular multiplication of large numbers are[4]: division after multiplication, Blakley algorithm, Brickell algorithm, Barrett algorithm and Montgomery algorithm.

A. The original Montgomery algorithm

Montgomery algorithm has relatively fast modular multiplication and short encryption and decryption time. In 1985, PeterMontgomery put forward an algorithm to achieve modular multiplication just through division and right shift of numbers, which was later known as Montgomery modular multiplication algorithm. Its theorem[5]: this algorithm is mainly to convert the multiplication of two numbers to the concentrated multiplication of residue classes of a modulus n, and then to convert seeking the remainder of n to seeking the remainder of any modulus r (r>n); to seek the power of the modulus r that is 2. In this way, the division of the modulus n has been converted to the division of the modulus 2, and the binary counting of the computer and the electronic circuit has been changed into a simple shift operation which will be much faster than division and accessing large primes. This thought is the core of Montgomery algorithm, and all the later improvements in this algorithm are almost based on this basic thought. For an individual modular multiplication operation, this method is not so efficient, but if it is used in repeated modular multiplication operations, its efficiency will reflect its superiority.

The original solution of Montgomery algorithm that is $ABS^{-1} \bmod N$ can be described as follows:

To select the radix S and the modulus N (N>1,S>N), in case that gcd(S,N)=1, for any integer that meets 0≤T=AB<SN, the fast Montgomery algorithm expressed as Mont(T):

```
Suppose S<SN,S=2k,N=$\sum_{i=0}^{k-1} k_i 2^i$, gcd(N,S)=1;
REDC(T)
REDC(T,N)=$TS^{-1} \bmod N$
First, p=(TmodS)N'modS;
Then, R=(T+pN)/S;
At last, IF R≥N
 R=R—N;
Return R;
```

To some extent, the original Montgomery algorithm has reduced the difficulty in modular multiplication of large numbers. k*k bit multiplication and 2k bit addition are needed, and the intermediate results can reach up to 2k bits. Besides, in each modular multiplication operation, there must be multiplications of large numbers such as T=AB,TN^{-1}、pN. When A,B and N have more than 1024 bits, there will be more storage and processing data, leading to the demand of more hardware resources for encryption and decryption directly with Montgomery algorithm. Therefore, further improvements in Montgomery algorithm are needed.

B. Radix 2-Montgomery algorithm and its simplified algorithm

For the current hardware encryption, its superiority is taken into account from area and speed, so it is necessary to carry out appropriate optimization. The optimization of the original Montgomery algorithm can be roughly made from the following aspects: 1) calculate just part of values each time, instead of computing the value of R at a time in the original algorithm; 2) select the appropriate radix S (S=2k), and the corresponding division can be just realized by the shift operation.

Based on the size of S, the optimized algorithm can be divided into two categories: one is the high-radix optimized algorithm, namely S=2k, which is a method based on speed optimization; the other is the binary Montgomery modular multiplication algorithm, namely S=2, which is a method based on area optimization. According to different key demands of the actual situation of the project for the performance indexes, the two different implementation algorithms are chosen to meet different design requirements. The design of a secured wireless serial can process a small size of data. Taken the costs of the user into account, the radix 2 algorithm based on the optimization of the circuit area will be given a priority selection. For the simplified radix 2 Montgomery algorithm, it should be modified based on the following 3 points:

(1) Improvement of the initial value in the algorithm. Radix 2 Montgomery algorithm is introduced to seek p_i so as to ensure that R is the result of being divided by 2 evenly, which has reduced operation efficiency. Based on the improved p_i solution proposed by Eldridge and Walter[6], in the modified radix 2 Montgomery algorithm, the multiplier B is shifted left by a single bit, and $b_0 = 0$, so that the value of p_i is only introduced from R_0. Since R_0=0, $p_0 = 0$.

(2) Replacement of the subtraction operation. In the original Montgomery algorithm, when R≥N, in order that the iterative operation of the result from the current modular multiplication can be directly taken as the input of the next modular multiplication, one more subtraction operation is needed, and the condition that 0<R<N must also be met. The

subtraction operation here has obviously increased operational steps and hardware resource consumption, which has also affected operation efficiency. Therefore, it is necessary to replace the subtraction operation. For this, the multiplicand A and the multiplier B must satisfy that: A and B are both less than 2N; $a_n = 0,$ and $b_{n+1} = 0$; $m_{n+1} = m_n = 0$. Thus, the result is R<2N. Here, to increase a_n means to increase one more cycle operation, and to increase b_{n+1}、m_{n+1} and m_n has ensured the storage of R's intermediate result information during the operation.

(3) Compression of the number of addition in cycle operation. The key operation in Montgomery algorithm is the solution of R. The currently cycled values of a_i and p_i are used to compute $a_i \times b_j + p_i \times N$ and then the result of R. Addition in the formula is processed one time when a_i、b_j are known, with the operation of only a bit of radix 2 each time. In the cycle, the solution of R each time includes a selection and an addition operation, which can reduce the number of addition in modular multiplication.

While the simplified radix 2 Montgomery algorithm has guaranteed the correctness of results, it has also improved operation efficiency. The pseudocode can be expressed as follows: Simpled-Monts_2(A,B,N)

```
Simpled-Monts_2(A,B,N) $= AB \times 2^{-n-1} \bmod N$
  $A = (a_0 a_1 \cdots a_{n-2} a_{n-1} a_n)_2,\ a_n = 0$ ,
  $B = (b_0 b_1 \cdots b_{n-2} b_{n-1} b_n b_{n+1})_2, b_0 = b_{n+1} = 0$ ,
  $N = (n_0 n_1 \cdots n_{n-2} n_{n-1} n_n n_{n+1})_2, n_0 = 1, n_n = n_{n+1} = 0$

  $T_i = b_i + n_i$ ;
  $R_0 = 0;$
  for(i=0;i<=n;i++)
  { $q_i = R_0 + a_i b_0 (\bmod 2);$
    for(j=0;j<=n+1;j++)
    {
      switch $a_i, p_i$
        1and1: $T_j \leftarrow$ mux[j];
        1and0: $b_j \leftarrow$ mux[j];
        0and1: $n_j \leftarrow \mathrm{mux[j]}$ ;
        0and0:0 $\leftarrow$ mux[j];
      }
      $R_{j-1} = R_j + mux[j]$
  }
  input: $AB2^{-n-1} \bmod N$
```

Here, different values of a_i、p_i can influence variables of the selector, and the selection operation of multiple input and single output can be realized with the selector; to shift B left by a bit has avoided addition and multiplication at the third step; to increase two cycles has got rid of comparison and subtraction operations.

V. Conclusion

The security of RSA encryption and decryption is based on the difficult decomposability of large numbers. At present, the bottleneck in encryption and decryption algorithm is the slow modular exponentiation of large numbers. So how to improve the speed of modular exponentiation is the key to this study. In this article, the improved radix 2 Montgomery modular multiplication algorithm has been proposed based on the existing radix 2 Montgomery modular multiplication algorithm, so that modular exponentiation can be converted into several modular multiplication processes, and the free modular multiplication storage can be used to save hardware resources.

References

[1] Hu Lei. Handbook of Applied Cryptography[D]. Beijing. Publishing House of Electronics Industry, 2005,P1-17

[2] Zhang Huanguo, Wang Zhangyi. A General Introduction to Cryptography[D].Wuhan. Wuhan Press, 2009,P20-35

[3] Yang Bo. Modern Cryptography[D].Beijing. Tsinghua University Press, 2005,P11-14

[4] Zhou Yujie. Public Key Cryptographic Algorithms and Its Fast Implementation[D]. Beijing. National Defence Industry Press, 2002.,P98-102

[5] Chen Fenglin. The Improvement of Montgomery Algorithm and Its Application in RSA. Computer Applications and Software, 2006.6

[6] Montgomery PL.Modular Multiplication without Trial Division Mathematics of Computation,1985