RSA Encryption Algorithm Design and Verification Based on Verilog HDL

Bei Cao[1](✉), Tianliang Xu[1,2], and Pengfei Wu[1]

[1] Electronic Engineering School, Heilongjiang University, Harbin 150080, China
caobei@hlju.edu.cn
[2] Microelectronic Center, Harbin Institute of Technology (Wei Hai), Harbin 150000, China

Abstract. Prime number generation and the large number operations directly affect the efficiency of RSA encryption algorithm. In order to reduce the number of the calculation process about modular operation and to reduce the difficulty of division in the calculation process, the Montgomery optimization algorithm is used to carry out the modular multiplication of RSA encryption algorithm, so that the efficiency of the algorithm is improved. Based on the application and research of hardware implementation to information encryption, the Verilog hardware description language is used to design the RSA encryption algorithm in 1024 bits. The simulation results of encryption and decryption experiment show that Montgomery modular multiplication algorithm and RSA encryption algorithm are verified to be correct and effective.

Keywords: RSA encryption algorithm · Verilog HDL
Modular multiplication · Montgomery optimization algorithm

1 Introduction

The public key cryptography, which plays an important role in the modern encryption system, has high information security. RSA encryption algorithm is the first to meet the requirements of the public key cryptography [1]. The inverse of the modular exponentiation operation is calculated so that the decomposition of large integer is difficult. In this way, we can ensure the security of information [2]. It is a perfect password system which is in theoretical research and application.

At present, the security of RSA encryption system is also threatened. In 1999, the 512-bit RSA encryption algorithm was worked out after five months. In recent years, the RSA encryption algorithm with 768-bit length of secret key has been cracked, and the crack time is thousands of times as long as 512-bit encryption's crack time. Obviously, the key length is closely related to the encryption reliability. The security of RSA algorithm with 1024-bit secret key length can be guaranteed. A lot of research has been done for the 1024-bit RSA at home and abroad. Montgomery optimization algorithm based on the pipeline technology and the hard algorithm of the hardwire multiplier are proposed to improve the computational speed [3]. The modular multiplication circuit architecture is used to 4-radix multiplication RSA algorithm [4].

CMOS technology is used to design and implement Montgomery encryption algorithm [5]. RSA encryption algorithm has received extensive attention in the field of research and application.

RSA encryption algorithm can be used software and hardware to achieve. In order to ensure information security, tedious data calculations are needed. For the hardware implementation, the encrypted data are applied to the hardware device to obtain the encrypted information code. In this paper, RSA encryption algorithm and optimization technology will be studied and design. Based on hardware information encryption technology, Verilog hardware description language (HDL) is used to implement the algorithm, and the RSA system is verified by simulation tool.

2 RSA Encryption Algorithm Basis

The public-key cryptography belongs to asymmetric cryptosystem whose encryption and decryption process use different keys. They are called public key and private key respectively. Double secret keys make it possible for both parties to communicate securely, even on insecure information channels.

2.1 Cryptosystem

In the encryption and decryption process, first, pair of encryption key "e" and decrypt key "d" are generated in the receiving terminal. Encryption key can be disclosed to other parties. The sender encrypt the information of the plaintext M which will be conveyed, and generate incomprehensible symbol or information as a ciphertext C. Then, it is sent to the receiving terminal. The encryption process:

$$C = E_e(M) \tag{1}$$

The receiving terminal obtains the ciphertext C. It can decrypt the ciphertext C by using the decryption algorithm D and the decryption key "d". The plaintext M is restored. The decryption process:

$$M = D_d(C) \tag{2}$$

RSA encryption algorithm has high strength and security. Key management is simple and clear. The two sides of the communication use different keys so that the encryption key can be spread. In this way, it is difficult to crack from the encryption key to get the decryption key. So it can reduce the burden of key management. The algorithm is difficult and slow to operate. The process of generating keys is complex for the mathematical theory, and these also are the advantages to protect the information security.

2.2 RSA Encryption Algorithm Principle and Large Number Processing

The working of the algorithm consists of three processes: key generation, encryption and decryption. The important technology is mainly large prime generation, large number of processing, solving equations and fast modular exponentiation for the hardware implementation. The encryption algorithm requires the prime number detection before the key is generated. The modular exponentiation operation will be used. And the modular exponent operation use the Euclidean algorithm, which is the process of the decrypt key "d" generating. RSA encryption algorithm as a system is interrelated, and it is not independent of each other.

Key generation is achieved through the key generation algorithm. The specific steps of the algorithm are as follows.

[1] Generate two different large prime p, q;
[2] Calculate $n = p \bullet q$ and $\Phi(n) = (p - 1)(q - 1)$, where $\Phi(n)$ is the Euler function of n;
[3] Select random integer e to make $gcd(e, \Phi(n)) = 1$, and satisfy $1 < e < \Phi(n)$, that is, e and $\Phi(n)$ are coprime;
[4] Seeking private key d to make $ed \equiv 1(\mathrm{mod}\ \Phi(n))$, and d is the modulo anti-element, also called multiplication inverse d; satisfying $1 < d < \Phi(n)$;
[5] (e, n) is the public key, (d, n) is the private key, then p, q, $\Phi(n)$ are destroyed.

Encryption and decryption process is also the key to the design of the algorithm. If Party B sends an encrypted message M to the Party A, it would need to use the public key (e, n) to encrypt the plaintext M to obtain ciphertext C. In this paper, when encrypting, the plaintext is first grouped. And the value of each group is less than the integer N. The binary width of the packet is less than the value of $\log 2^N$ [6]. Then, plaintext grouping for the integer m, and the string can be obtained ASCII value to conduct encryption. Such as function (3), note that m must be less than n.

$$C = E_e(M) = M^e(\mathrm{mod}\,N) \tag{3}$$

Party A receives the information code from Party B, and it is necessary to decrypt the ciphertext packet, such as in function (4).

$$M = D_d(C) = C^d(\mathrm{mod}\,N) \tag{4}$$

3 RSA Encryption Algorithm and System Design

3.1 Large Prime Generation and Computing Processing

An important question of RSA encryption algorithm is how to generate large prime numbers quickly. In this paper, the binomial probability detection method is used to generate random numbers, and then we can judge the number of prime numbers by the designed test algorithm. The steps are as follows.

[1] Calculate the odd number M so that $N = (2^r) * M + 1$;
[2] Select the random number $A < N$;
[3] for $i < r$, $A^{((2^i)*M)} \bmod N = N - 1$, then N through the random number A test;
[4] or, if $A^M \bmod N = 1$, then N passes the test of random number A;
[5] Through the value of A is different to give N five tests. If the results are passed, then N is prime.

As the number of tests N increases, N satisfies that the probability is not prime, that is, the probability that N is prime can reach $1 - (1/4^t)$. For example, when $t = 5$, the probability that N is prime is more than 99.99%. Most compilers support 64-bit. In order to avoid the problem of low efficiency in establishment of large number of arithmetic and decimal number and a large number of storage space. In this paper, the 1024-bit large number is expressed as 0x1_0000_0000 hexadecimal. For example, 0x0000_0000 $\sim$ 0xffff_ffff, that is, with 32-bit * 32-bit interval, 1024-bit large number conversion into 32 elements of the interval array, for the interval array is only more 32 Sub-cycle operation than before. In terms of computers, the binary is similar to the 0x1_0000_0000 system and it is easy to convert.

3.2 Modular Multiplication

The modular exponential operation complicates the data through the power exponential operation to make the operation process complicated and achieve the effect of information security. Therefore, the modular exponentiation is transformed into multiple modular multiplication operations, that is, $C = A * B \bmod N$, where A, B and N are integers that satisfy the relation of $0 \leq A, B \leq N$. Considering the security, the 1024-bit modulo operation requires a higher hardware design. And the multiplication rule will produce 2048 bits of the process. And avoiding the direct calculation of $A * B$ is the primary problem to be solved by the modular multiplication. Montgomery algorithm is used to modular multiplication. It is defined as follows.

Definition 1. Assuming that the two integers of x and y are satisfied with $0 < a, b < N$, select a radix r, and then select a number R with respect to N and satisfy $R = r^n$, then $X = x * R(\bmod N), Y = y * R(\bmod N)$. Represent the mapping of a, b in N residual.

$$Z = X * Y * R^{-1}(\bmod N) \tag{5}$$

Where R^{-1} is the inverse of the R-mode N and satisfy $R^{-1}R = 1(\bmod N)$, which is called the Montgomery multiplication.

Montgomery multiplication can avoid the use of division. On the contrary, it can use shift operation to achieve modular operation. Under the K power of 2, division only needs to perform the left shift operation. In this way, we can simplify the complexity of the operation and reduce the number of modes. Table 1 shows the Montgomery modular expression, N represents the modulus, and T is the reduced number $R = 2^n$, in the binary conditions, n is the number of bits width. $T + qN$ is the actual number of the remaining classes in which T is actually represented.

It can be seen that $Mon \bmod N = T * R^{-1} \bmod N$, the value of $Mon \bmod N$ can be used to calculate the value of $T * R^{-1} \bmod N$. If q is found to be a multiple of R, the divisor would be an integer. If

$$N[0] * N[0]'\%r = 1, q = (C'[0] + A * B[0]) * (r - N[0]')\%r \tag{6}$$

Table 1. Expressions about Montgomery modular multiplication.

Character	Expression
Montgomery reduction	Mon = $(T + qN)/R = (T + qN) * 1/R$
Operation of residue class	$\mathrm{M}(T) * \mathrm{M}(1) = \mathrm{M}(T) * \mathrm{M}(R) * \mathrm{M}(R-1)$

Then,

$$(C'[0] + A * B[0] + q * N[0]\%r = (C'[0] + A * B[0] - ((C'[0] + A * B[0]) * N[0]' * N[0])\%r)\%r = 0) \tag{7}$$

Where C′[0] represents the initial state. If r = 0x1_0000_0000 is used, division and modulo operation will become simple. For example, the result is q[63:0], %r = q[31:0], /r = q[63:32]. There is no more than 32 times the number of cycles k for the key length in 1024-bit. Only the intermediate variables of $C' = C' + A * B + q * N$ will appear multiple times so that the efficiency of the algorithm will be improved. The only thing that needs to be calculated is the N[0]′ value, which makes $N[0] * N[0]'\%r = 1$. Since N is a fixed value, it is only necessary to perform a calculation on N[0]′ without affecting the efficiency of the operation.

3.3 RSA Encryption System Design

RSA encryption hardware application requirements are analyzed. RSA encryption algorithm top-level module is described based on Verilog HDL and named RSA_1024. The system module and external pin definition are shown in Fig. 2. The parallel data input is used in this system, and the maximum can support 1024-bit data operations. In order to optimize the structure, reduce the port cost, the input data will be taken 32-bit as a group and 32 consecutive units of input in cycles (Fig. 1).

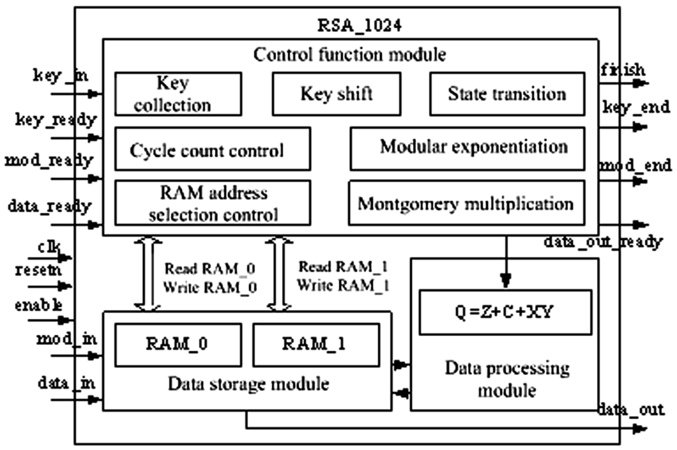


Fig. 1. 1024-bit RSA encryption system.

When data_ready signal of encryption key *e*, mod *n* and whisper data are pulled high, the data are entered. After the encryption operation is completed, the data_out_ready signal is pulled high and the outputted encrypted data is valid. Then, the system mainly includes the control function realization module, the data processing module and the data storage module.

3.4 RSA Control Module Design

The RSA control function module performs packet acquisition on the encryption key e and d input. We can select RAM_0, RAM_1 for the module of store data by judging the data by the key shift, executes the data Montgomery modular multiplication, and controling the two sub-modules through the RAM address selection control and the cyclic count control to store the data The module selects RAM_0, RAM_1, and stores the data values in the middle of the operation. In addition, the control state transfer switch plays a role of connecting each module in the process of RSA algorithm implementation. Finally, the signal processing of the plaintext ciphertext, plus the key, the modulo input are done by the relevant module data storage and encryption operation. The realization process can be represented by the state Fig. 2.

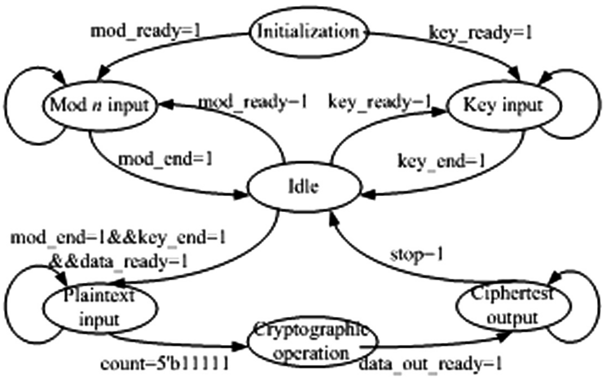


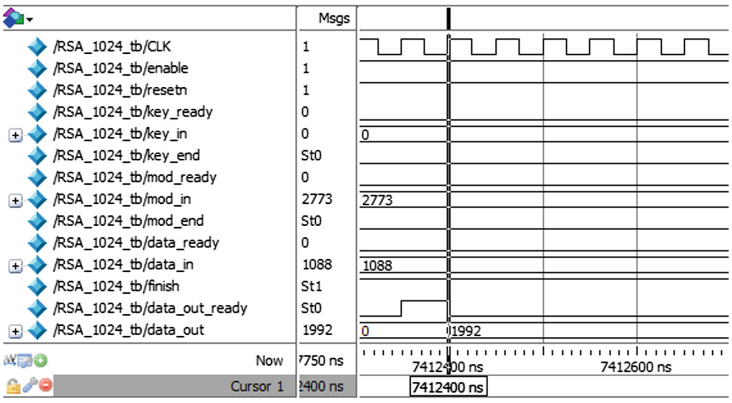
Fig. 2. State diagram about implementation module of RSA control function

The data input and output sections are analyzed according to the actual situation of the state transition and the parallel input data shown in Fig. 2. Starting from the data initialization, when the input key data signal is 1, the key e is input until 32 groups 32 Bit data input is completed, and the key data input end signal is 1 and jumps to the idle state. When the input modulo data signal is 1, the input data N is started. When 32 sets of 32-bit data input are completed. The modulo data input end signal is 1 and jumps to the idle state. In addition, when the key e and the modulo data N input are completed and the input plaintext data signal is ready to be 1, the plaintext data is input until 32 sets of 32-bit data input completion with countering from 0 to 31, before entering the encryption operation state; when the encryption operation is over, then ready to output ciphertext data signal is 1, and all the data output to stop the entire operation process, then jump back to the end of the idle state.

Considering the problem of large data storage capacity, the data storage module uses memory RAM_0 and RAM_1, which are 128 × 32 and used as input group of plaintext or ciphertext data input and module n input. Finally, it will store the middle of the calculation process, Remainder and modulo and other data values.

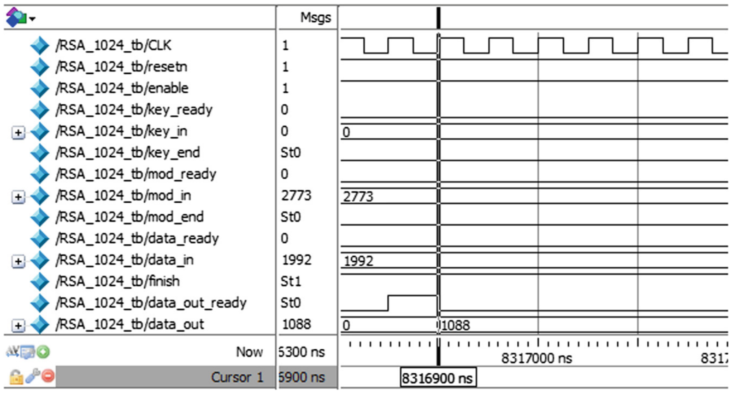
4 RSA Encryption Algorithm Simulation

In order to verify the function of Verilog HDL based RSA encryption algorithm, the simulation tool ModelSim is used to simulate the algorithm and we will analyse whether the sub-module achieves the expected function. The security key is set to p = 47, q = 59, e = 101, d = 317, n = 2773, i.e. the public key is (101, 2773) and the private key is (317, 2773). When 32 sets of plaintext data are M = 1088, 32 sets of ciphertext data are C = 1992, and this data is taken as an example to verify the function of the corresponding module.



```
                               Msgs
/RSA_1024_tb/CLK               1
/RSA_1024_tb/enable            1
/RSA_1024_tb/resetn            1
/RSA_1024_tb/key_ready         0
/RSA_1024_tb/key_in            0       0
/RSA_1024_tb/key_end           St0
/RSA_1024_tb/mod_ready         0
/RSA_1024_tb/mod_in            2773    2773
/RSA_1024_tb/mod_end           St0
/RSA_1024_tb/data_ready        0
/RSA_1024_tb/data_in           1088    1088
/RSA_1024_tb/finish            St1
/RSA_1024_tb/data_out_ready    St0
/RSA_1024_tb/data_out          1992    0  1992
Now                            7750 ns       7412400 ns        7412600 ns
Cursor 1                       2400 ns       7412400 ns
```

Fig. 3. Encryption process simulation results

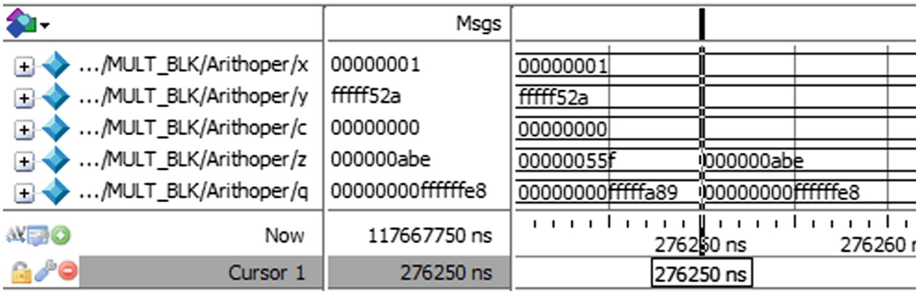


```
                               Msgs
/RSA_1024_tb/CLK               1
/RSA_1024_tb/resetn            1
/RSA_1024_tb/enable            1
/RSA_1024_tb/key_ready         0
/RSA_1024_tb/key_in            0       0
/RSA_1024_tb/key_end           St0
/RSA_1024_tb/mod_ready         0
/RSA_1024_tb/mod_in            2773    2773
/RSA_1024_tb/mod_end           St0
/RSA_1024_tb/data_ready        0
/RSA_1024_tb/data_in           1992    1992
/RSA_1024_tb/finish            St1
/RSA_1024_tb/data_out_ready    St0
/RSA_1024_tb/data_out          1088    0  1088
Now                            5300 ns              8317000 ns        831
Cursor 1                       5900 ns       8316900 ns
```

Fig. 4. Decryption process simulation results

When the 32 groups of input text are $M = 1088$, encryption key $e = 101$, 32 group mode are $N = 2773$, Fig. 3 shows the encryption operation of 32 sets of ciphertext data are $C = 1992$; when the input ciphertext $C = 1992$, decryption key $d = 317$, modulo $N = 2773$, Fig. 4 shows the decryption operation of plaintext M is 1088. After the encryption and decryption of the data can be found in each other, the function is correct, that is, algorithm can be achieved. It should be noted that the plaintext, modulo, ciphertext are 1024 bits of data, except that each of the 32 groups is the same, and the encryption key is 32 bits and only have one group. Finally, the remaining 31 groups are 0.

As shown in Fig. 5, the modular multiplication algorithm realize the function of the algorithm by outputting the 12_bit command through the output terminal. These commands control the data path and the data storage of the four registers to realize the core algorithm calculation, that is, $Q = Z + C + XY$.



```
                              Msgs
...MULT_BLK/Arithoper/x       00000001            00000001
...MULT_BLK/Arithoper/y       fffff52a            fffff52a
...MULT_BLK/Arithoper/c       00000000            00000000
...MULT_BLK/Arithoper/z       000000abe           0000055f          000000abe
...MULT_BLK/Arithoper/q       00000000fffffe8     00000000fffffa89  00000000fffffe8
Now                           117667750 ns              276250 ns         276260 ns
Cursor 1                      276250 ns                 276250 ns
```

Fig. 5. Modular multiplication simulation results

We can check when the error is obtained by $x = 32'h0000\_0001$ and $y = 32'hffff\_f52a$, $c = 32'H0000\_0000$, $z = 32'h0000\_055f$, the simulation result is the same as the calculated result. In this paper, the latter set of data is also consistent with the simulation results. So we can verify the correctness of the Montgomery modular multiplication.

5 Conclusion

There are six security parameters in the RSA encryption algorithm, which are named p, q, n, $\Phi(n)$, e, d. There is the public key (e, n) only on the encryption. And the decryption side have the private key (d, n) and the parameter p, q. In this paper, the design of Verilog HDL is achieved by 1024-bit key length algorithm. And it used Rabin Miller primality probability detection method to generate the binomial number. And the large number is processed, stored and computed by the N array and the Montgomery modular multiplication algorithm. In this way, we can solve the huge amount of data problems. In order to solve the input and output data time between the numbers of pins of mutual restriction. And a large number of input and output are used a parallel way. The simulation results show that this paper can realize the information encryption and decryption function requirements.

References

1. Adleman, L., Rivest, R.: The use of public key cryptography in communication system design. Commun. Soc. Mag. **16**, 20–23 (1978)
2. Yan, G.X.: Information security under encryption technology. Netw. Secur. Technol. Appl. **393**, 100–104 (2013). (in Chinese)
3. Hentabli, W., Merazka, F.: An extension of RSA_512 to RSA_1024 core under hardware platform based on Montgomery powering. In: International Conference for Internet Technology and Secured Transactions, pp. 448–453 (2013)
4. Tamura, S., Yamada, C., Ichikawa, S.: Implementation and evaluation of modular multiplication based on coarsely integrated operand scanning. In: Networking and Computing, pp. 334–335 (2012)
5. Tenca, A.F., Ruggiero, W.V.: CRT RSA decryption: modular exponentiation based solely on Montgomery multiplication. In: Asilomar Conference on Signals, Systems and Computers, pp. 431–436 (2015)
6. Liu, Z., Jing, J., Xia, L.: An optimized architecture to speed up the Montgomery modular. In: International Conference on Computers, Communications, Control and Automation, pp. 58–62 (2011)