Journal of Electronic Research and Application
OPEN

Implementation and Improvement Optimization of RSA Encryption Algorithm

Yu Siqing
Beijing Anstitute of Electronic Science and Technology, Beijing 100070, China

Abstract: First, we encrypted and decrypted data by RSA encryption algorithm and achieved a secure transmission of data. Aiming at the drawbacks of time-consuming and complicated program in RSA encryption algorithm, an exponentiation algorithm for solving large number of modules is proposed. Through the realization of the exponentiation algorithm, the deficiency of RSA in modulus of large number is improved, and the processing speed is improved.

Key words: RSA; Encryption; Counting; Pascal
Published online: 30th Nov 2017
Corresponding author: Yu Siqing, E-mail: siqing-yu@outlook.com

1 Introduction

In the information society, a tremendous amount of encrypted information is having been transmitted, exchanged, stored and processed. Consequently, how to ensure the safe transmission of information has become a hot issue in the field of information transmission.

First, the principles and virtues of RSA asymmetric encryption algorithm are studied in this paper, and then implement its encryption and decryption. The exponentiation used in this algorithm is time-consuming and restricts its wide application. Consequently, to increase the speed of encryption and decryption, in this paper, a fast power algorithm is proposed to solve the fast algorithm of large number exponentiation by optimizing with binary arithmetic[1-3].

2 The Principle of RSA Encryption Algorithm

2.1 The Background of RSA Encryption Algorithm

RSA is the first relatively perfect public key algorithm，which has withstood years of intensive cracks. Although cryptanalysts can neither justify nor deny the security of RSA, it just shows that this algorithm has some credibility, and now it has become the most popular public key algorithm.

The security of RSA is based on the difficulty of factorization of great numbers. Its public key and private key are functions of a pair of great prime numbers (100 to 200 digits, decimal numbers, or greater). The difficulty of regaining plaintext from a public key and ciphertext is equivalent to decomposing the product of two great prime numbers.

2.2 The Principle of RSA Encryption Algorithm

The theoretical basis of the RSA encryption algorithm is a special invertible modulus exponent operation. The algorithm principles are described as follows:

(1) Select two distinct prime numbers, P and Q (P and Q must be kept secret and large enough)；

(2) Calculate n = p*q, f(n) = (p-1) * (q-1);

(3) Choose a number “e” which is smaller than n,e and f (n) are coprime.

(4) Find the number “d” that allows e*d-1 to be divided with no remainder by f (n), this formula is: d≡e-1 mod f(n)

(5) RSA is a block cipher system, therefore public key is KU=(e, n), private key is KP=(d,n);

Among them, “n” is the modulus must be known by both parties, “e” is the index of encryption operations must be

known by the sender. “d” is the decryption operation index must be known by decrypting party.

The above algorithms are further abstracted as follows:

Public key: KU=(e,n); n=p*q, e and （p-1）*（q-1）are coprime.

Private key: KP=（d,n）, $d = (e-1)\bmod\{(p-1)*(q-1)\}$

Encrypt: C=$C^e \bmod n$ ^() D_Dd__

Decrypt: M=$C^d \bmod n$

3 Optimization and Implementation of RSA Encryption Algorithm

3.1 Selection of Prime Number

The first step of the RSA encryption algorithm is to select great prime numbers, the prime number is obtained by sieve method. By look-up table method, preserve a certain range of prime numbers in the array prime numbers[4].

3.2 Random Selection of Encrypted Prime Numbers

Because of the presence of pseudorandom numbers, when choosing prime numbers as encryption factors, the prime numbers are the same at each choice. In Pascal, after the randomization of randomize, the random function is called, and the results are not in the presence of pseudorandom numbers[5].

3.3 Selection of Public Key

By randomly selecting less than f (n) integer, the public key “e”, by extended euclidean algorithm, calculate the greatest common divisor. If the greatest common divisor of 1, e and f (n) coprime.

```
function    gcd (a, b: qword):qword; // The greatest common factor by extended Euclid
begin
if b=0 then gcd:=a
else gcd:=gcd (b,a mod b);
end;
```

the calculation of the private key “d” can be done by calculating $(e * d)\bmod z = 1$. So calculation of the formula deformation of $d = (z * i + 1)$ mode can be gained by enumerating “i”[6]

3.4 Raw Data Processing

For letters and numbers, encryption will be processed during inputting data. Gain Ascll codes by means of the ord function, preserve single letters and numbers in array data.

```
begin
writeln ('please input data:');
sum:=0;
while not eoln do
begin
 read(ch);
inc(sum);
data[sum]: =ord(ch);// Convert characters to Ascll
end;
end;
```

3.5 Modular Arithmetic

The hard core of RSA encryption algorithm is (a^b)mod c, when a and b are very great, directly solving the problem is almost impossible. Consequently, there are the following algorithms:

Algorithm 1: Direct Solution

```
int ans = 1;
for (int i = 1; i<=b; i++)
ans = ans * a;
ans = ans % c;
```

Faults: there are obvious problems with the algorithm, if a and b are excessively large, the data will overflow a buffer.

Let's take a look at the first improvement plan. Before this plan, let's look at a formula like this: $a^b \bmod c = (a \bmod c) | b \bmod c$, and after modding some factor, multiply, and then mod, keep the remainder unchanged. So the new *ans* can also be taken out. We get the new improvement plan based on the two points[7].

Algorithm 2: Further improved algorithm

```
int ans = 1;
a = a % c; //add this one
for (int i = 1; i<=b; i++)
ans = (ans * a) % c; // mod once again
ans = ans % c;
```

The algorithm is not improved in the time complexity, is still *O(b)*. But it's been much better. However, under the condition that C is too large, it is still possible to timeout, so the following algorithm of exponentiation is adopted.

Algorithm 3: Exponentiation algorithm

Based on dichotomy theorem, the complexity of the algorithm is reduced to O(logn)

B can be expanded by binary like:

b = c (n) * 2n + c (n-1) * 2n-1 + • • • + c (1) * 2 + c (0)

c (i) ($0 \le i \le n$) is 0 or 1;

Thus, $a^b = a^{c(n)*2^n + c(n-1)*2^{n-1} + \ldots c(1)*2 + c(0)}$

$$= a^{c(n)*2^n} * a^{c(n-1)*2^{n-1}} * \ldots * a^{c(1)*2} * a^{c(0)}$$

We don't have to deal with the situation of $c(i) = 0$, result displays 1; We must consider the case of $c(i) = 1$.

Simplification: $a^{2^i} = a^{2^{i-1}*2} = (a^{c(i)*2^{i-1}})^2$

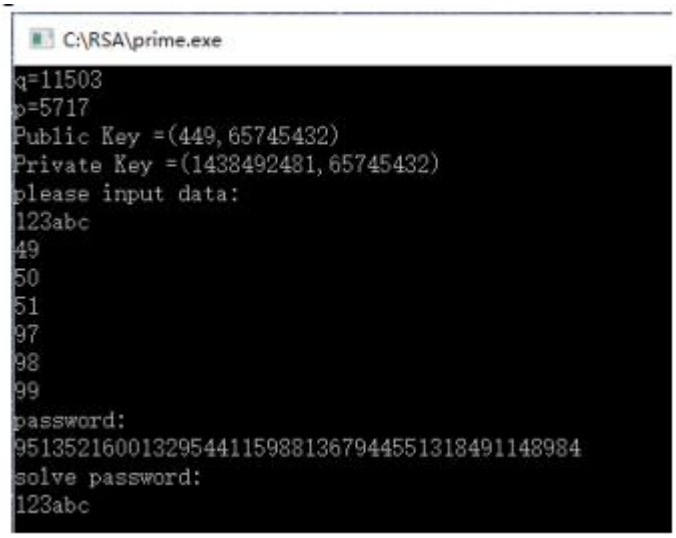
All a^{2^i} can be worked out recursively

Thus, modulo operation should be:

$$a^{2^i} \% d = (d^{c(i)*2^{i-1}}) \% d * (a^{c(i)*2^{i-1}}) \% d$$

Therefore, all $c(i) = 1$ a^{2^i}%d, according to the algorithm 1 multiply up, and then %c is the result. The binary scan has been scanned from top to the lowest point.

```
function f (a, b, n: int64): int64;
var t, y: int64;
begin
t:=1; y:=a;
while b<>0 do
begin
```




```
if (band1)=1 then t:=t*y mod n;
y:=y*y mod n; {here y*y is the result of a^(2^(i-1)) }
b:=b shr 1; { Move right to remove the already dealt one }
end;
exit(t);
end;
```

3.6 Program Operation Implementation

By selecting encryption factor "e" and decryption factor "d" randomly, convert each character input into ascll code. Then get the encrypted password in figure. Then restore password to original characters. The running results are shown in figure[8]:

4 Conclusion

Aiming at the drawbacks of time-consuming and complicated program in RSA encryption algorithm, the exponentiation algorithm is used to solve the problem of large number power modulo and optimized by binary operation.

The improved algorithm can reduce the time consuming of RSA modular exponentiation and increase the speed of RSA encryption and decryption. Through the practical comparison, the implementation of different arithmetic of finger patterns proves that the fast exponentiation algorithm is excellent[9-10].

References

[1] Yang Kunwei. *The application of Chinese Remainder Theorem in cryptography, Computer Technology And Development* [J] 2014.

[2] Lan Haibing. Chen Shengli, *Research on the improvement of RSA algorithm and its implementation technology, Transportation and Computer.*[J] 2006(10).

[3] Tian Wenyan. Li Zeming, *Research on the improvement of RSA algorithm and its implementation technology, Shanxi Electronic Technology* [J] 2013(6).

[4] A Chitra, K Sangeethalakshmi, P Sivalakshmi, et al. A FPGA Implementation of High Security Hybrid Reconfigurable Cryptographic Processor with RSA and SEA.

[5] K Shehata, H Hussien, S Yehia. FPGA Implementation of RSA Encryption Algorithm for E-Passport Application.

[6] O Nibouche, M Nibouche, A Bouridane, et al. Fast architectures for FPGA-based implementation of RSA encryption algorithm. IEEE International Conference on Field-programmable Technology.

[7] S Kumar Sahu, M Pradhan. FPGA Implementation of RSA Encryption System. International Journal of Computer Applications[J].

[8] N Qi, J Pan, Q Ding. The Implementation of FPGA-based RSA Public-key Algorithm and its Application in Mobile-phone SMS Encryption System. First International Conference on Instrumentation.

[9] T W. Ockinger. High-Speed RSA Implementation for FPGA Platforms.

[10] SS Ghoreishi, MA Pourmina, H Bozorgi, et al. High Speed RSA Implementation Based on Modified Booth's Technique and Montgomery's Multiplication for FPGA Platform. International Conference on Advances in Circuits.