

Research on Fast Implementation of RSA with Java

Guicheng Shen^{1,2}, Bingwu Liu², Xuefeng Zheng¹

¹ School of Information Engineering, Beijing University of Science and Technology, Beijing, China
Email: shenguicheng@bvu.edu.cn

² School of Information, Beijing Wuzi University, China
Email: liubingwu@bvu.edu.cn

Abstract— this paper introduces the principles of RSA, then based on these principles the implementation of RSA is discussed. After that several classes are analyzed and designed with Object-Oriented methods, and these classes are implemented with Java language. In the third fourth part, we give the fast implementation of RSA cryptosystem, and it is tried to encrypt, decrypt, signature and verify the signature. In the end, we put forward the conclusion that the RSA cryptosystem implemented with java can be applied in electronic commerce.

Index Terms—Information Security; Electronic Commerce Security; Cryptography; RSA

I. INTRODUCTION

Digital Signature is becoming a practical authentication technology in recent years, and it will replace traditional authentication technologies of seal and handwriting in the field of digital media in the future. It is proposed by Diffie and Hellman in 1976, and in 1978, RSA cryptography and its scheme of digital signature was put forward in 1978. Digital signature has made a breakthrough at conditional encryption boundary, and it has solved the problem of key distribution and showed the existence of the feasibility of digital signature scheme. Now, many business, research institutes and even governments are using RSA. The wide-spread application represents its practicality, stability, and broad prosperity.

RSA is the first public cryptography, and its safety is based on big integer factorization. It can be used in encryption, decryption, and digital signature at the same time. Since it has been put forward in 1978, there is no serious drawback in it after it has experienced much cryptography analysis.

In RSA, the most operation are power operation and module operation, and we often compute the result of $a^e \bmod N$ where N is the product of two big primes and e is the public key of encryption or the private key of decryption. To forestall the attack of public module number and low exponent, N and e must be large enough, and in some hardware implementation, N has 664 bits, and now its bit size will be 1024. Because of the limit of bit length, power module operation can not be computed directly, several integers can be combined as one big integer. Therefore this reduces the speed of RSA, and RSA may be slower than DES 100 times. Thus, how to

find practical methods to develop RSA software is becoming an import research direction.

II. PRINCIPLES OF RSA

The safety of RSA algorithm is base on the problem of big integer factorization, and the algorithm is Eule theory in number theory.

A. System Parameters

- P and Q are two big primes whose bit sizes are than 300, and they are kept secret. For the safety case, the difference of two sizes must be big enough.
- $N = P * Q$, $\Phi = (P-1) * (Q-1)$, and Φ is kept secret.
- E can any plus integer but the great common division of e and Φ must be 1. D is an integer and $d * e \equiv 1 \pmod{\Phi}$.
- $Kp = (E, N)$ is the public key and $Ks = (D, N)$ is the private key.
- Let M is a plaintext, $(M^E)^D \equiv M \pmod{N}$.

B. The Procedures

- Encryption: $C = \text{Enc}(E, M) = M^E \pmod{N}$
- Decryption: $M = \text{Dec}(D, C) = C^D \pmod{N}$
- Signature: $S = \text{Sig}(D, M) = M^D \pmod{N}$
- Signature Verification: $SV(S) = \text{VeriSign}(E, S) = S^E \pmod{N} = M'$. If $M' = M$, the signature is true, and otherwise the signature is false. As anybody who has the public key of Alice can verify whether the signature is given by Alice and one Alice knows d . when a disputation takes place, an impartial third-party can give an impartial arbitration.

III. ANALYSIS AND DESIGN OF RSA CRYPTOSYSTEM

From the principle of RSA, RSA cryptosystem must include the following several parts:

- ### A. Select two big primes, and compute N and Φ ;

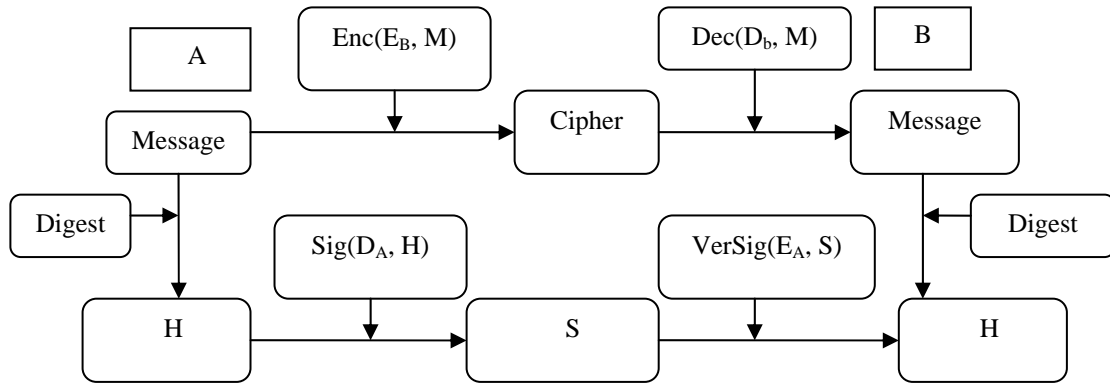


Figure 2. Encryption, Decryption, Signature, and Verification

B. Compute a suitable E , where the greatest common division of E and Φ is one. To speed up digital signature verification and encryption, E must be very small. Based on E and N , D can be computed;

C. Compute $Mkey \bmod N$, where key can be E as well as D .

When E is being computed, E can be tried from 3 for $\Phi = (P-1)*(Q-1)$ is even, therefore E can be increased with 2 once. The algorithm of computing E can be displayed in a language similar to C language.

```

LongInteger ComputerE(LongInteger  $\Phi$ ) {
    LongInteger e=3;
    LongInteger division = 0;
    While(GCD( $\Phi$ , e) != 1) {
        E = e + 2;
    }
    Return e;
}

```

Here GCD is the function to compute the greatest common division of two integers, and it can be implemented with Euclid Algorithm.

As the bit length of D and E can reach more than 1000, We can not compute $L=M^{key}$, then we compute $L \bmod N$ because this may cause the bit length of L reaches 1000000, thus it will reduce computing speed greatly. Therefore in general, the method of computing power module is to square and module.

Suppose we have computed the bit representation of A , such as $A = b_0b_1b_2\dots b_k$, and we want to compute $Ma \bmod N$. We demonstrate the algorithm with a language similar to C language.

```

LongInteger PowerModule(LongInteger M, int b[],
    LongInteger N, int k) {
    LongInteger Result = 1;
    For(int i=k; i >= 0; i--) {
        Result = Result * Result;
        If(b[i] == 1) {
            Result = Result * M;
        }
    }
    If(Result >= N) {
        Result = Result % N;
    }
}

```

```

}
}
Return Result;
}
}

```

As the bit number of a key may reach to hundreds of bits, even to one thousand bit, thus one integer in programming languages can not meet our request, therefore we can combine several integers to represent a long integer. As we need to hide the implementation information and to reduce the maintenance cost, we can use the object-oriented method to implement this cryptography system.

Based on the above discussion, we can conclude that RSA class may have two properties and four methods. These two properties are N and Key, and in order to speed up computation, we can use a bit array to represent the key; the four methods are encryption, decryption, signature, and verifySignature; besides these, we must have methods to compute greatest common division of two integers, and to generate a big prime. Thus, this can be displayed in the following way.

```

class RSACryptography {
    private LongInteger N;
    private int a[];
    public RSACryptography()...
    private static int[] computeBitArray(LongInteger
        Key)...
    private static int getBitLength(LongInteger Key)...
    private static LongInteger GCD(A, B)...
    private LongInteger PowerModule(A, M)
    public LongInteger Encryption(LongInteger M)...
    public LongInteger Decryption(LongInteger C)...
    public LongInteger Signature (LongInteger M)...
    public Boolean VerifySignature(LongInteger S,
        M')...
}

```

IV. FAST IMPLEMENTATION OF RSA

There are many methods of implementing RSA cryptosystem, and in the environment that the speed is required to be very high, it can be developed with C language, and the shortcomings are that we must implement all algorithms including the implementation of large integer, the generation of big primes, and the

method of computing greatest common division of two integers, therefore, these are very tedious.

In the environment of electronic commerce, electronic commerce systems must run on different operating system platforms such as windows, Unix, and Linux. If the system is developed with C language, it will bring up a problem that the system can not run on all platforms although the system developed with C language is easy to transplant. As the same data type has different length on different platforms, this makes it difficult to transplant.

To make it easy to maintain the system, develop quickly and to solve the problem of cross platform, we try to develop this system with java language. Java is an object-oriented language, and the system developed with java language can run on all platforms. Although the execution efficiency of language is slower than that of C language and C++ language and it is said that java speed is only one tenth of C, its speed is about one third of C and C++ really as java language provides many class libraries which are implemented with native languages.

According to the discussion on the third part, we need to define a class of long integer, and java provides a class BigInteger which meets our requirements. BigInteger has basic operation methods including addition, subtraction, multiplication, and division, and besides these, it also provides the methods to generate a big prime whose bit length is predefined, the method of compute the greatest common division, the method of compute the module inverse, and so on.

Based on this analysis, we can define three classes. The first one is RSACore class which provides the implementation of core algorithms; the second class is publicRSA which is the subclass of RSACore, and it has two methods of encryption and signature verification; the third class is privateRSA which is the subclass of RSACore, and it has only two methods of decryption and signature.

RSACore class has two properties which are N and Key. Key can be public key as well as private Key and it is an integer array of its bit presentation. It has five methods and these are the method of computing the bit representation of Key, the method of generating public key and private key, the method of computing the power module. This class is displayed in the following way.

```
import java.math.*;
import java.util.*;
public class RSACore {
    private BigInteger N=null;
    private int[] key=null;
    private static int[] getBitArray(BigInteger M) {
        byte[] bytes = k.toByteArray();
        int[] bits = new int[8*bytes.length];
        BigInteger one = BigInteger.ONE;
        BigInteger two = one.add(one);
        BigInteger temp = k;
        int i=0;
        while(temp.compareTo(BigInteger.ZERO) != 0)
        {
            if(temp.mod(two).compareTo(one) == 0)
                bits[i] = 1;
```

```
        else bits[i] = 0;
        temp = temp.divide(two);
        i++;
    }
    int index = bits.length-1;
    while(index >= 0 && bits[index] == 0) {
        index--;
    }
    if(index == -1) return null;
    int[] newbits = new int[index+1];
    for(int j=0; j <= index; j++) {
        newbits[j] = bits[j];
    }
    return newbits;
}
public static BigInteger generatePrime(int
    bitlength){
    return BigInteger.probablePrime(bitlength,
        new Random());
}
public static BigInteger generatePublicKey(
    BigInteger phai) {
    BigInteger one = BigInteger.ONE;
    BigInteger two = one.add(one);
    BigInteger key = one.add(two);
    while(phai.gcd(key).compareTo(one) != 0) {
        key = key.add(two);
    }
    return key;
}
public static BigInteger generatePrivateKey (
    BigInteger phai, BigInteger E) {
    return E.modInverse(phai);
}
public BigInteger powerModule()...
```

We can find that the programming code is very short, and we use the methods provided by BigInteger class.

We use this class to generate two primes P and Q, where the bit length of P is 500 while the bit length of Q is 520. Based on these two primes, we compute N, Φ , E, and D. These numbers are displayed as follows.

P=27778143967984925872857130196948488892856
0003370391977364091129875556857245076095626889
4630463933293256615224079082487916388108388118
281627770269380647.

Q=21083375654623834531347917973607370063253
5802460506133197476478413611437159949254172575
4665327520195845481759111230389371752509333786
547047226909900294110681.

E=7,D=8366529203789275814439622700307581351
7898668811825642875543780561729719484428316250
2736821873916405956342993141478660083955402708
2285071461837889090006072522317232848660847732
7419168349059826755515709863447994525911881503
6869306517865290679008311003381563761333381976
611581902733141575781780943817853414183.

As for the implementation of publicRSA and privateRSA, it is very easy, thus we only demonstrate the implementation of publicRSA class. Class publicRSA

has no properties, but it has two methods. These methods are encryption and verifySignature. Class publicRSA is demonstrated as follows.

```
import java.math.*;
public class publicRSA extends RSACore {
    public publicRSA()...
    public BigInteger Encryption (BigInteger M) {
        return powerModule(M);
    }
    public boolean VerifySignature(BigInteger S,
        BigInteger H) {
        BigInteger newh = powerModule(S);
        if(newh.compareTo(H) == 0) return true;
        return false;
    }
}
```

Although the speed of encryption is very fast, the speed of decryption is slower. The total time we cost in generating two primes, computing N, D, and E, encrypting and decrypting is 7 seconds. As the bit length of private key is 1200, we think that the speed can be suitable in electronic commerce. As the bit length of E is only 3 while the bits length of D is 1200 about, the speed of encryption is about 0.02 second. As the bit length of N is 1200, we think that our test is successful. We use one BigInteger to test our system. The test result is demonstrated as follows.

M=10715086071862673209484250490600018105614
 0481170553360744375038837035105112493612249319
 8378815695858127594672917553146825187145285692
 3140435984577574698574803934567774824230985421
 0746050623711418779541821530464749835819412673
 9876755916554394607706291457119647768654216766
 0429831652624386837205668069376.

C=13441042556607850502301339153152295196553
 9850525665748330729065896934540966492174580942
 0052799854092930064422881235861225922598913196
 0821123161977516671882184522916350469279583111
 4983576369805163905916848111898922124849170812
 7448045001148106244552149092687532698885981104
 549889088747160935615481636578613717.

M'=1071508607186267320948425049060001810561
 4048117055336074437503883703510511249361224931

9837881569585812759467291755314682518714528569
 2314043598457757469857480393456777482423098542
 1074605062371141877954182153046474983581941267
 3987675591655439460770629145711964776865421676
 60429831652624386837205668069376.

V. CONCLUSION

As java is object-oriented, cross-platform language and it also provides lots of class library which are implemented with native programming language, its execution efficiency is very high. As this RSA system is implemented with java language, the system can run on all platforms, therefore it provides the sound base for its application in electronic commerce.

ACKNOWLEDGEMENT

This paper is supported by three projects including Funding Project for Academic Human Resources Development in Institutions of Higher Learning Under the Jurisdiction of Beijing Municipality (PHRIHLB), Funding Project for Base Construction of Scientific Research of Beijing Municipal Commission of Education, and Funding Project for Science and Technology Program of Beijing Municipal Commission of Education Under the grant number KM200810037001) .

REFERENCES

- [1] Diffie W, Hellman M. New Directions in Cryptography[J]. IEEE Transactions on Information Theory, 1976, 22(6): 644-654.
- [2] Rivest R, Shamir A, Adleman L. A Method for Obtaining Digital Signatures and Public Key Cryptosystems[J]. Communications of the ACM, 1978, 21(2): 120-126.
- [3] Yonglaong Luo, Liusheng Huang. A Faster Algorithm of Modular Exponentiation in RSA. Mini-Micro Systems[J]. Jan. 2004
- [4] Ziwei Zheng, Cuihua Li. Realization of Class-Based RSA System. Journal of Huaqiao University (Natural Science) [J]. Jul. 2003
- [5] Qi Xie. A New Fast RSA Algorithm. Computer Engineering[J]. February. 2003