

A Study on Public Key Cryptography

Research Paper

CS 843 – Distributed Computing System

Laurentius Purba

May 9, 2003

Contents

1. Introduction	2
2. Technical detail	5
3. Symmetric Cryptosystems: DES	7
4. Public-Key Cryptosystems: RSA	9
5. An Example	10
6. Implementation	11
7. Conclusion	12
8. Source Code	14

Abstract

Security becomes an important issue in all kind of transaction happened in the Internet. Network can be considered as insecure channel. How can a secure connection be established over an insecure channel? This problem can be solved by using cryptography method.

This paper is an introduction to Symmetric Cryptography and Public Key Cryptography as solutions for handling such problem. The discussion of the concept of Symmetric, asymmetric cryptosystems will be presented. The simple example will also be presented as well as the code, which is written in java.

1. Introduction

Internet becomes the most important communication media nowadays. It's fast, reliable, secure but also can be unsecured. It is believed that Internet will not grow fast if security does not play important part, in order to make customers have confidence on them.

Cryptography is an algorithmic process of converting a plain text (or clear text) message to a cipher text (or cipher) message based on an algorithm that both the sender and receiver know, so that the cipher text message cannot be read by anyone but the intended receiver. The act of converting a plain text message to its cipher text form is called *enciphering*. The opposite action, converting cipher text form to plain text message, is called *deciphering*. The terminology enciphering and deciphering are more commonly referred to as *encryption* and *decryption*, respectively.

All of modern algorithms use a key for performing encryption and decryption. The message can only be decrypted using the key that matches with the one it was encrypted with. Different technique can be used, by using different key for encryption as well as for decryption. There are three types of cryptography are widely used; they are *symmetric* cryptosystem, *asymmetric* cryptosystem, and cryptography *hashing* function system.

Symmetric Cryptosystem

Symmetric cryptosystems use the same key, denoted by K , for both encryption and decryption. They are dependent on the key. The range of possible values of the key is called the *keyspace*.

$$\begin{aligned} E_K(M) &= C \\ D_K(C) &= M \\ D_K(E_K(M)) &= M \end{aligned}$$

These functions can be depicted as follows:

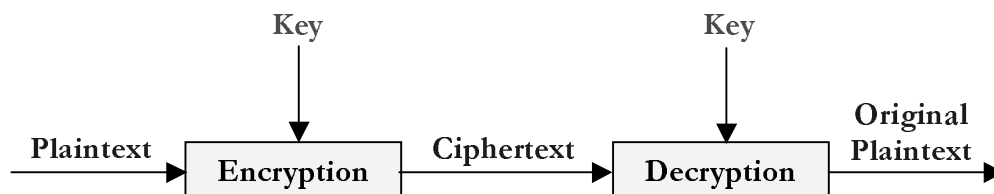


Figure 1: Encryption and Decryption with a key [3]

There are two types of algorithms in symmetric cryptosystem; they are *stream* algorithms or *stream* ciphers and *block* algorithm or *block*

cipher. Stream algorithm (stream cipher) operates on the plaintext a single bit or byte at a time. While, block algorithm (block cipher) operates on the plaintext in groups of bits. These groups of bits are called *blocks*. For modern computer algorithms, a typical block size 64 bits large enough to prevent analysis and small enough to be workable.

In order to get higher security purpose, different symmetric algorithms use different length keys. One of the examples of the symmetric cryptosystems is CAESAR, which is based on *substitutions*: each letter is substituted by another letter. The latter is obtained from the former by advancing k steps in the alphabet. At the end of the alphabet, one goes cyclically to the beginning.

Let suppose, $k = 5$, hence the substitutions are as follows.

Old:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
New:	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E

In this case, the plaintext HELLO WORLD is encrypted as MJQQT BTWQI.

Although symmetric cryptosystem works, it has an essential weakness about how to keep the key secure. If an intruder knows the key, then the message can easily decrypted. But then, the key must be available to the sender and the receiver, which might be separated. It seems the symmetric cryptosystem transform the problem of transmitting message securely into that of transmitting keys securely. This problem is referred to as key management problem.

Asymmetric Cryptosystem

Asymmetric cryptosystem handles the key management problem by using different key for encryption as well as for decryption; denoted by K_1 as encryption key, and K_2 as decryption key.

$$\begin{aligned}
 E_{K_1}(M) &= C \\
 D_{K_2}(C) &= M \\
 D_{K_2}(E_{K_1}(M)) &= M
 \end{aligned}$$

These functions can be depicted as follows:

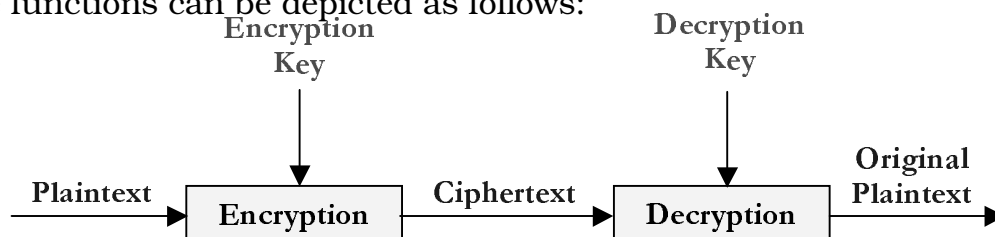


Figure 2: Encryption and Decryption with two different keys [3]

If the intruder knows only one particular key, say the encryption key, he cannot determine the other key, which is decryption key. Therefore, the encryption key can be made public, while only the party wishing to receive encrypted message holds the decryption key. In fact, anyone can use the public key to encrypt the message, but only the recipient can decrypt it. Sometimes, This terminology is called public/private key cryptography.

The keys are generated by a mathematical algorithm that generally involves very large prime numbers (will be discussed later).

Hash Function

Hash functions are functions, which map a string of arbitrary size to a short string fixed length (typically 128...160 bits). Hash functions, which satisfy security properties, are widely used in cryptographic applications such as digital signatures, password protections schemes, and conventional message authentication.

Hash function H will produce output a bit string h with fixed length, if a message M with arbitrary length is given as input.

$$h = H (M)$$

In communication system, error detection can be done by adding extra bits to a message; similar to cyclic-redundancy check (CRC). This idea is comparable to hash h .

There are several properties of hash functions, such as one-way functions, weak collision resistance, and strong collision resistance.

One-way functions are relatively easy to compute. With fixed M , it is easy to compute h , from the above function. But then, with fixed h , it is difficult to calculate M . With fixed M , it is difficult to calculate M' with $H, (M) = H (M)$. One of a good example of one-way function is breaking a plate. It is easy to smash a plate into a thousand tiny pieces. However, it is not easy to put all of those tiny pieces back together into a plate.

Weak collision resistance is also known as second pre-image resistance. It is infeasible given an input M , to find $M' \neq M$ such that $H(M) = H (M')$. By having this function, intruder has a more difficult time finding collisions because it must collide on a particular input rather than any input.

Strong collision resistance (also known as collision resistance) is a function with these properties: it is infeasible to find any two distinct inputs M and M' such that $H (M) = H (M')$. While M, M' pairs must exist by the pigeonhole principle, it may be computationally difficult to find these pairs.

2. Technical Detail

The mathematical relationship between the public/private key pair permits a general rule: any message that is encrypted using one key of the pair can be successfully decrypted only using other counterpart key.

The message that is encrypted with public key can only be decrypted with the private key. The converse is also true, the message that is encrypted with private key can only be decrypted with public key. There is no mandatory solution, which key is to keep private and which is to make public. RSA (will be discussed later), the public key uses exponents that are relatively small, in comparison to the private key, making the process of encryption and digital signature verification faster.

Figure 3 depicts the use of public/private key cryptography properly for sending messages; using Alice and Bob example.

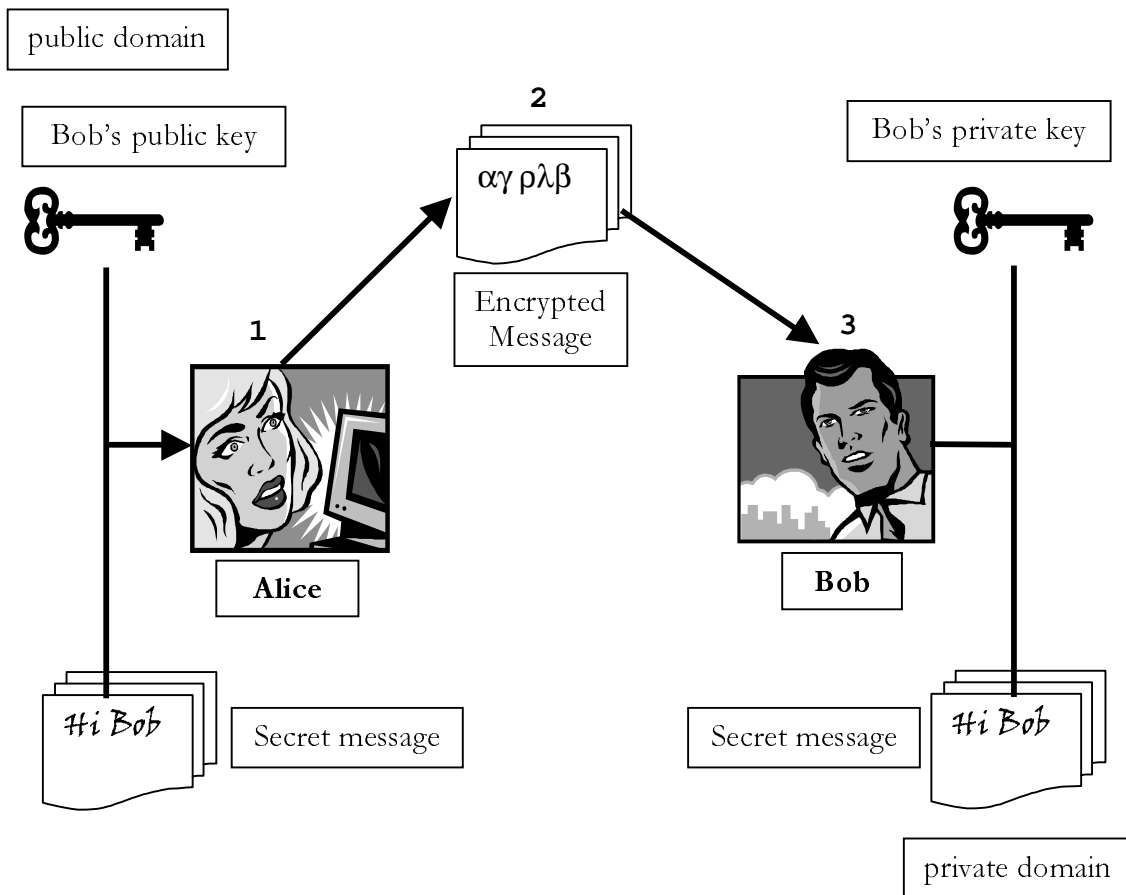


Figure 3: Proper Use of Public Key Cryptography

From figure 3, Bob has a public/private key pair. Bob's public key is placed in the public domain, such as personal website. Bob's private key is kept in a private domain, such as, on digital key card or in a password-protected file.

Let suppose Alice wants to send a message to Bob. The sequences of the process will be as follows:

1. In order to generate encrypted message, Alice passes Bob's public key and her message to the appropriate encryption algorithm.
2. After she gets the encrypted message, she sends the message to Bob, say via e-mail.
3. Bob receives the encrypted message. In order to decrypt the message, Bob has to use his private key, the message, and the appropriate decryption algorithm.

Even though the intruder can intercept Alice's encrypted message, he/she cannot decrypt the message, since only Bob has the private key to decrypt it.

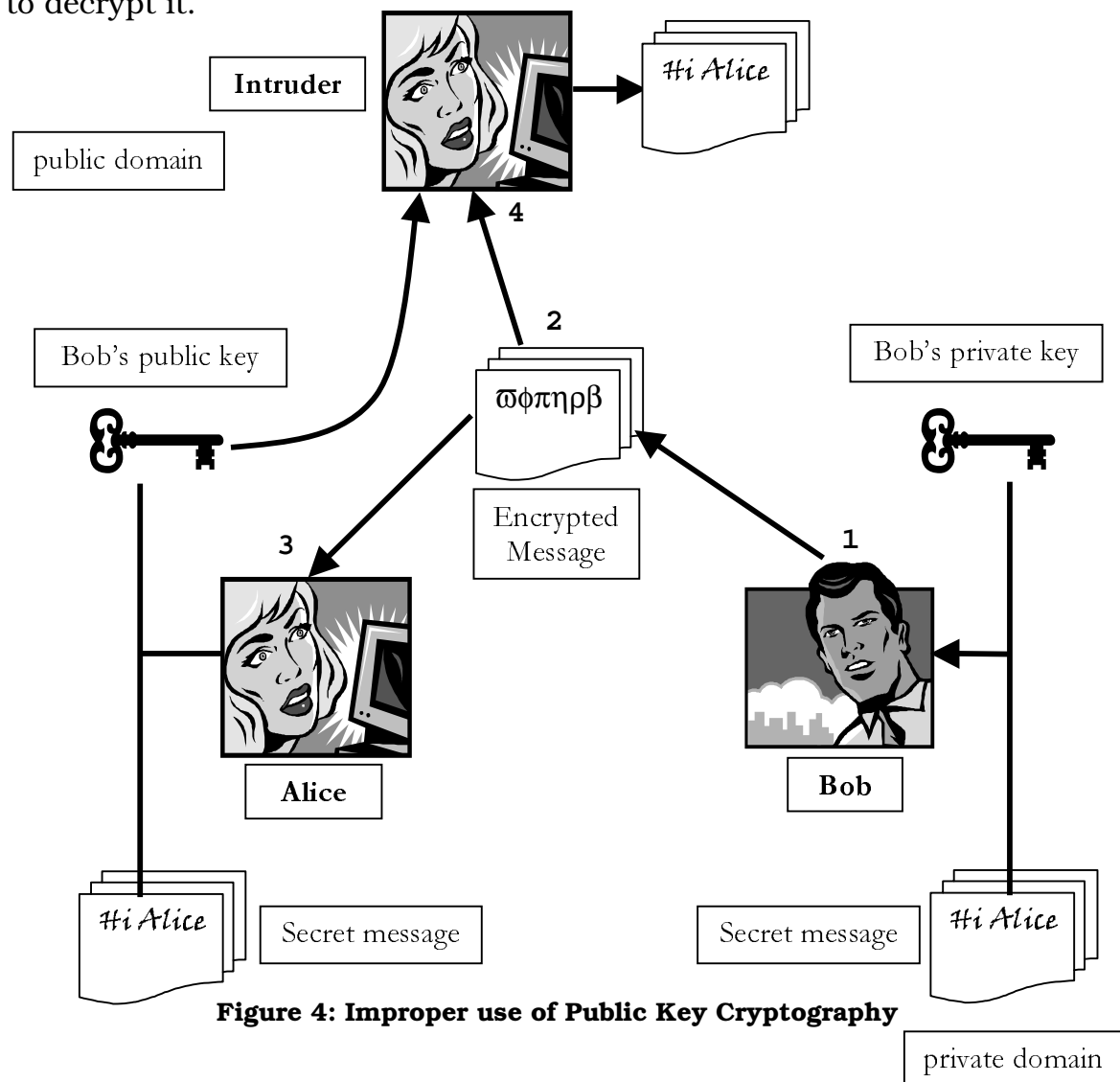


Figure 4: Improper use of Public Key Cryptography

Now, Bob tries to reply Alice's message by encrypting the original message using his private key, which is depicted in Figure 4, as follows:

1. In order to generate encrypted reply, Bob passes his private key and the secret reply to the appropriate encryption algorithm.
2. After he gets the encrypted reply, he sends the message to Alice, same way as Alice did, via e-mail.
3. Alice receives the encrypted reply. In order to decrypt the reply, Alice has to use Bob's public key, the message, and the appropriate decryption algorithm.
4. But then, suppose the intruder knows Bob's public key, since his public key is shared, the intruder can decrypt the reply and see the text of the message.

However, if Alice has her own public/private key pair, then Bob and Alice could communicate securely. Bob can send messages encrypted with Alice's public key, and Alice can send messages to Bob encrypted with Bob's public key. Since, only a particular person that has the private key can decrypt the message.

3. Symmetric Cryptosystems: DES

Data Encryption Standard (DES) is one of many examples of symmetric cryptosystems. It was originally developed by IBM and endorsed by US government in 1977.

There are 72,000,000,000,000,000 (72 quadrillion) or more possible encryption keys that can be used. For a message that will be encrypted, the key is chosen at random from among this huge number of keys. DES is similar to other private key cryptography methods, where both sender and receiver must acknowledge and use the same private key. [5]

DES processes blocks of 64 bit, which are issued as 64 bit, too. The algorithm functions symmetrically, which means as well for the encoding as for the decoding the same algorithm and the same key is used. Normally, the key is expressed as 64 bit number, but every 8-bit is a parity testing and is ignored; in this case the length of the key is 56 bit. Security is measured by the use of this key.

The uncoded text is divided into 64 bit blocks, which subsequently in the *initial permutation* (IP) are divided into two 32-bit blocks (L0, R0). Then there are 16 rounds of identical operations, function f , in which the data are combined with the key. Finally, L16 (left part) and R16 (right part) are fit together, and in a final permutation transposed once again.

In function f , on each round the key should be shifted. The selection is on 48 bits out of available 56 bits, and the right half of an *expansion permutation*. Those two parts are subsequently XOR connected and transmitted through eight S-boxes (S-Box Substitution). The boxes

generate 32 new bits, which are permuted again (P-Box Permutation). The result of function f is combined with left half by means of XOR, and this result becomes the new right half $R(i)$. The former right half is the new left half.

(^ = XOR; K_i Key i)

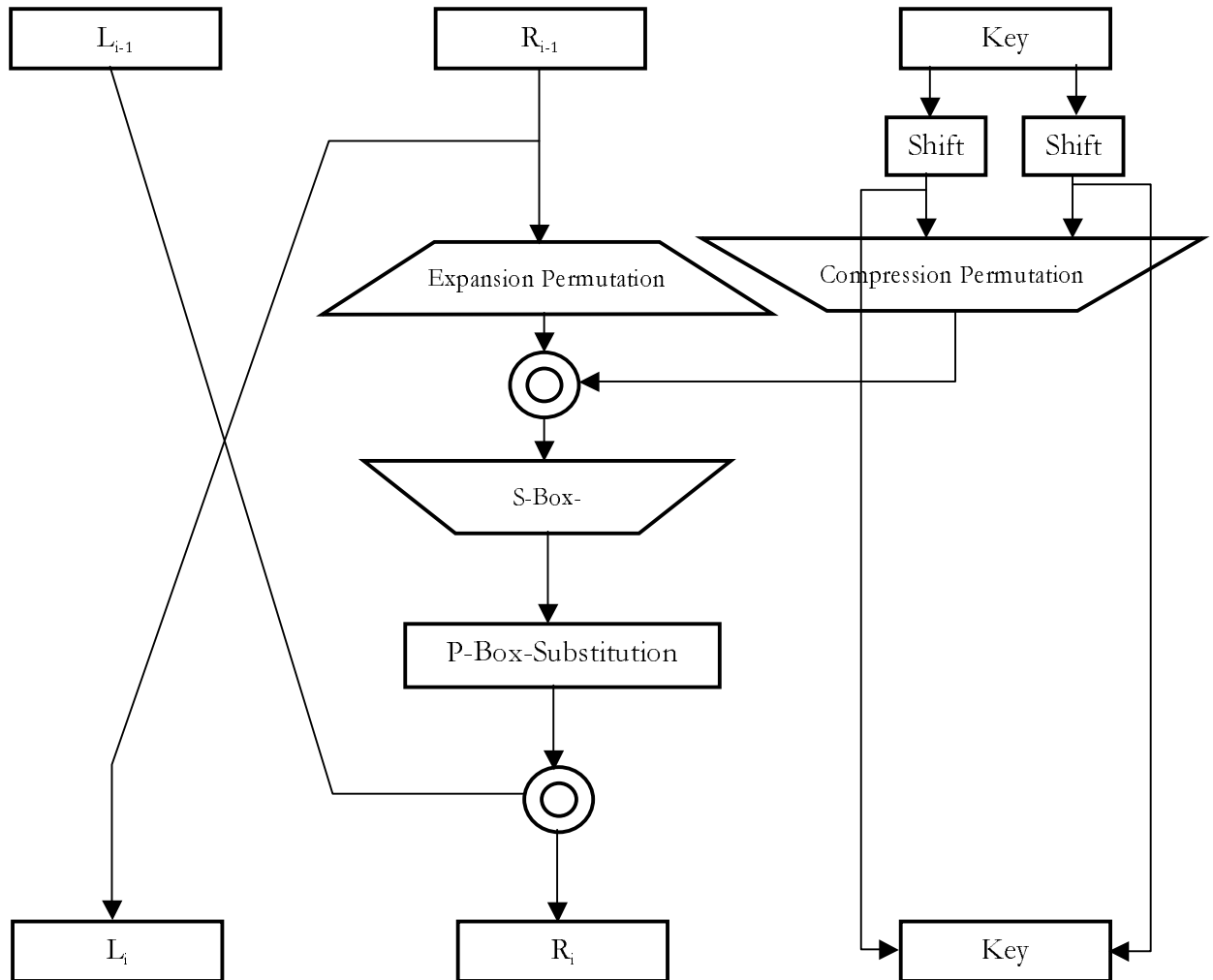


Figure 5: One round DES

Since DES is symmetric algorithm, the same function could be used to encipher and decipher. The one thing that should be taken care of is the use of the key; it should be applied in reversed order.

4. Public Key Cryptosystem: RSA

RSA is a public-key cryptosystem for encryption as well as decryption. The name RSA comes from the inventors Ron Rivest, Adi Shamir, and Leonard Adleman. RSA was invented in 1977 by Rivest, Shamir, and Adleman however; it was recently revealed that this method was originally invented in 1973 within GCHQ by Clifford Cocks. One thing that makes RSA algorithm secured is no methods are known to find the prime factors of large numbers efficiently.

The mathematical facts of RSA methods is as follows:

I will explain a little bit about “Euler Totient Function”, which is important in the RSA algorithm.

The totient function $\phi(n)$ is defined as the number of positive integers $\leq n$, which are relatively prime to (i.e., do not contain any factors in common with) n , where 1 is counted as being relatively prime to all numbers. Since a number less than or equal to and relatively prime to a given number is called *totative*, the totient function $\phi(n)$ can be simply defined as the number of totative of n . For example, there are eight totatives of 24 which are (1, 5, 7, 11, 13, 17, 19, and 23), so $\phi(24) = 8$. [6]

Back to RSA methods: If a number is raised to a power (d), modulo a prime, the original number can be obtained by raising the result to another power (e). And it can be easily determined, knowing the prime modulus, and the one power, what the power is. If the modulus (M) is not prime, there is still another power, which reverses exponentiation by any power. But finding it depends on knowing the factorization of the modulus, or, which is equivalent, knowing something called the “Euler Totient Function” of the modulus. For a prime number p , the *ETF* of it is $(p - 1)$, hence the product of two prime numbers, p and q , which is the sort of number used as a modulus in RSA, the *ETF* is $(p - 1)$ times $(q - 1)$. Then, choose for e number that is relatively prime to $(p - 1)$ and $(q - 1)$ have any other common factors, they too can be divided out so that they will only appear only once.

Finding the public and private keys pair for the impatient: [4]

1. Choose 2 very large prime numbers, p and q .
2. Compute $n = p \times q$, and $z = (p - 1) \times (q - 1)$.
3. Choose a number d that is relatively prime to z .
4. Compute the number e such that $e \times d = 1 \pmod{z}$.

The message M needs to be broken into a series of blocks, and represent each block as an integer, between 0 and $n - 1$. To encrypt the message by raising it to the e^{th} power modulo n , the result (ciphertext C) is the remainder when M^e is divided by n . On the other hand, to decrypt the ciphertext, raise it to another power d , again modulo n .

The encryption and decryption algorithms E and D are as follows:

$$C \equiv E(M) \equiv M^e \pmod{n}, \text{ for a message } M.$$

$$D(C) \equiv C^d \pmod{n}, \text{ for a ciphertext } C.$$

Notice that the encryption does not increase the size of a message, since both the message and integers in the range 0 to $n - 1$. The encryption key is the pair of positive integers (e, n) , similarly the decryption key is the pair of positive integers as well, (d, n) . Each user can make his encryption key public (i.e. by putting it on his website), but the decryption key should be kept private.

In order to compute e , the variation of Euclid's algorithm for computing the greatest common divisor of z and d can be used. Calculation of $\gcd(z, n)$ by computing a series x_0, x_1, x_2, \dots , where $x_0 \equiv z$, $x_1 \equiv d$, $x_{i+1} \equiv x_{i-1} \pmod{x_i}$, until an x_k equal to 0 is found. Then \gcd (

5. An Example [1]

Let supposed the properties are as follows:

$$p = 47, q = 59.$$

$$n = p \times q = 47 \times 49 = 2773.$$

$$\text{Let } d = 157, \text{ then } z = (p - 1) \times (q - 1) = 46 \times 58 = 2668.$$

And e can be computed as follows:

Using equation: $x_i = a_i \cdot x_0 + b_i \cdot x_1$, where x_0, x_1, x_2, \dots is mentioned in previous chapter.

Calculate of a_i and b_i such that a_i multiplied by x_0 plus b_i multiplied by x_1 equal to x_i , ($x_i = a_i \cdot x_0 + b_i \cdot x_1$).

For $i = 0$, $x_0 = a_0 \cdot x_0 + b_0 \cdot x_1$

$$\Leftrightarrow 2668 = a_0 \cdot 2668 + b_0 \cdot 157$$

$$\Leftrightarrow a_0 = 1, b_0 = 0.$$

For $i = 1$, $x_1 = a_1 \cdot x_0 + b_1 \cdot x_1$

$$\Leftrightarrow 157 = a_1 \cdot 2668 + b_1 \cdot 157$$

$$\Leftrightarrow a_1 = 0, b_1 = 1.$$

For $i = 2$, $x_2 = a_2 \cdot x_0 + b_2 \cdot x_1$

$$x_{i+1} = x_{i-1} \pmod{x_i}$$

$$x_2 = x_0 \pmod{x_1}$$

$$x_2 = 2668 \pmod{157} = 156.$$

$$\Leftrightarrow 156 = a_2 \cdot 2668 + b_2 \cdot 157$$

$$\Leftrightarrow a_2 = 1, b_2 = -16. \text{ Since } 2668 = 157 \cdot 16 + 156.$$

For $i = 3$, $x_3 = a_3 \cdot x_0 + b_3 \cdot x_1$

$$x_{i+1} = x_{i-1} \pmod{x_i}$$

$$x_3 = x_1 \pmod{x_2}$$

$$x_3 = 157 \pmod{156} = 1.$$

$$\Leftrightarrow 1 = a_3 \cdot 2668 + b_3 \cdot 157$$

$$\Leftrightarrow a_3 = -1, b_3 = 17. \text{ Since } 157 = 1 \cdot 156 + 1.$$

Therefore $e = 17$, since $e \times d \pmod{z} = 1$, ($17 \times 157 \pmod{2668} = 1$).

Since $n = 2773$, and the message should be divided into a series of blocks, with each block ranges from 0 to $n - 1$, then 2 letters per block can be used; Substituting a two-digit number for each letter: blank = 00, A = 01, B = 02, ..., Z = 26,

Blank	A	B	C	D	E	F	G	H	I	J	K	L	M
00	01	02	03	04	05	06	07	08	09	10	11	12	13
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
14	15	16	17	18	19	20	21	22	23	24	25	26	

Table 1: Substitution of a two-digit number for each letter.

the message:

HELLO WORLD

is encoded:

0805 1212 1500 2315 1812 0400

with $e = 17$, can be represented as 10001 in binary, then the first block ($M = 805$) is enciphered:

$$M^{17} = (((((1)^2 \cdot M)^2)^2)^2)^2 \cdot M = 542 \pmod{2773}$$

The whole message is enciphered as:

0542 2345 2417 2639 2056 0017

The deciphering works can be done as follows:

$$542^{157} = 805 \pmod{2773},$$

$$2345^{157} = 1212 \pmod{2773},$$

$$2417^{157} = 1500 \pmod{2773},$$

$$2639^{157} = 2315 \pmod{2773},$$

$$2056^{157} = 1812 \pmod{2773},$$

$$17^{157} = 400 \pmod{2773}.$$

6. Implementation

RSA algorithm is implemented using java. Since RSA algorithm needs big numbers to operate, *BigInteger* in java can handle such problem.

The user will be asked about the size of prime number, and it should be greater than 3. Function 'calculatePrimeNumbers' will generate two different prime numbers p , and q . Function 'calculatePublicPrivateKeys' will generate n , which is $p \times q$, and also e and d . Public keys will be the pairs of n , and e ; private keys will be the pairs of n , and d . Again, user will be asked to input the message (plaintext) that wants to be encrypted. Function 'encrypt' will divide the message into several blocks using byte data type, and encrypt the value of it using the pair keys e , and n . Function 'decrypt' will decrypt the encrypted value of the array of *BigInteger* using the pair keys d , and n , and return the original string message.

7. Conclusion

The public-key cryptography has several advantages, such as:

1. Pairs of keys are kept secret (private keys), which is increased the security.
2. Since RSA allows sender to attach a digital signature to a message to show whom the message is from and to ensure the message has not been altered or interfered during transmit. In this case, it will make sender and recipient more confident about their communication.

Besides these advantages, there are some disadvantages about RSA (public-key cryptography), such as:

1. It is much slower than the secret key models (symmetric cryptosystem).
2. Also, it is difficult to implement in hardware.

Since both types of cryptography (public/private key, and secret key) have their advantages and disadvantages, a combination of these two methods will generate appropriate result in the system. For personal use only, such as encrypt private files, secret key can be used; while other activity such as e-mail, public/private key should be used instead.

REFERENCES

- [1] Rivest, R.L., Shamir, A., Aldeman, L.: Method for Obtaining Digital Signatures and Public-Key Cryptosystems.
- [2] Data Encryption Standard (DES) (FIPS PUB 46-2).
Gaithersburg, Md. National Institute of Standards and Technology,
January 1993.
Available WWW:
<URL: <http://www.nist.gov/itl/div897/pubs/fip46-2.htm>>
- [3] Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edition by, John Wiley & Sons
ISBN: 0471128457, 1996
- [4] Tanenbaum, A.S., Steen, M: Distributed Systems: Principles and Paradigms, Prentice Hall, ISBN: 0130888931
- [5] Laynetwork.com
<http://www.laynetworks.com/users/webs/des.htm>
- [6] Wolfram Research: The way the world calculates
<http://mathworld.wolfram.com/TotientFunction.html>
- [7] Source code references:
jewel.morgan.edu/~jsmid/javafiles/encryption/RSA.java
<http://carol.wins.uva.nl/~bterwijn/Projects/Encryption/RSA/>

SOURCE CODE

```

/*****
*
* CS 843 --- DISTRIBUTED COMPUTING SYSTEM --- SPRING 2003
* Title: RSA Implementation
* Description: This program will compute public/private key, and
*              perform encryption and decryption based on RSA
*              algorithm.
* Filename: RSA.java
*
*-----
-----
*
* Instructor: Chin-Chih Chang
* Written by: Laurentius Purba
*
*****/

import java.io.* ;
import java.util.Random ;
import java.math.BigInteger ;

public class RSA
{
    int primeSize;           /* Bit length of each prime number */
    BigInteger p, q;        /* Two different large prime numbers
p and q */
    BigInteger n;           /* n = p * q */
    BigInteger r;           /* r = ( p - 1 ) * ( q - 1 ) */
    BigInteger e, d;        /* Public exponent e and Private
exponent d */

    /***
    * CONSTRUCTOR
    * PARAMETER: PRIMESIZE
    * */
    public RSA( int primeSize ) {
        this.primeSize = primeSize;

        // calculate two distinct large prime numbers p and q.
        calculatePrimeNumbers() ;

        // calculate Public and Private Keys.
        calculatePublicPrivateKeys();
    } /*** --- END OF GENERATEPRIMENUMBERS FUNCTION --- ***/

    /**
    * FUNCTION CALCULATEPUBLICPRIVATEKEYS
    * CALCULATE PUBLIC AND PRIVATE KEYS
    * */
    public void calculatePrimeNumbers() {

```

```

    p = new BigInteger( primeSize, 10, new Random() );

    do {
        q = new BigInteger( primeSize, 10, new Random() ) ;
    } while( q.compareTo( p ) == 0 );
} /**** --- END OF CALCULATEPRIMENUMBERS FUNCTION --- ***/

/**
 * FUNCTION CALCULATEPUBLICPRIVATEKEYS
 * CALCULATE PUBLIC AND PRIVATE KEYS
 * */
public void calculatePublicPrivateKeys() {
    n = p.multiply( q );          // n = p * q

    // r = ( p - 1 ) * ( q - 1 )
    r = p.subtract( BigInteger.valueOf( 1 ) ) ;
    r = r.multiply( q.subtract( BigInteger.valueOf( 1 ) ) );

    // choose e, relatively prime to r and less than r
    do {
        e = new BigInteger( 2 * primeSize, new Random() );
    } while( ( e.compareTo( r ) != -1 ) || ( e.gcd( r ).compareTo(
BigInteger.valueOf( 1 ) ) != 0 ) );

    // compute d, the inverse of e mod r
    d = e.modInverse( r );
} /**** --- END OF CALCULATEPUBLICPRIVATEKEYS FUNCTION --- ***/

/**
 * FUNCTION ENCRYPT
 * ENCRYPT THE MESSAGE USING PUBLIC KEY
 * PARAMETER: STRING MESSAGE
 * RETURN VALUE: ARRAY OF BIGINTEGER OF THE CIPHERTEXT
 * */
public BigInteger[] encrypt( String message ) {
    int i ;          // counter
    byte[] temp = new byte[1];

    byte[] numeric = message.getBytes();
    BigInteger[] bigdigits = new BigInteger[numeric.length];

    for( i = 0 ; i < bigdigits.length ; i++ ) {
        temp[0] = numeric[i];
        bigdigits[i] = new BigInteger( temp );
    }

    BigInteger[] encrypted = new BigInteger[bigdigits.length];

    for( i = 0 ; i < bigdigits.length ; i++ )
        encrypted[i] = bigdigits[i].modPow( e, n );

    return( encrypted );
} /**** --- END OF ENCRYPT FUNCTION --- ***/

```

```

/**
 * FUNCTION ENCRYPT
 * DECRYPT THE MESSAGE USING PRIVATE KEY
 * PARAMETER: ARRAY OF BIGINTEGER OF THE CIPHERTEXT
 * RETURN VALUE:  STRING MESSAGE
 * */
public String decrypt( BigInteger[] encrypted ) {
    int i;                // counter

    BigInteger[] decrypted = new BigInteger[encrypted.length] ;

    for( i = 0 ; i < decrypted.length ; i++ )
        decrypted[i] = encrypted[i].modPow( d, n );

    char[] charArray = new char[decrypted.length];

    for( i = 0 ; i < charArray.length ; i++ )
        charArray[i] = (char) ( decrypted[i].intValue() );

    return( new String( charArray ) );
} /*** --- END OF DECRYPT FUNCTION --- ***/

/*****/
/**** ACCESSORS FUNCTIONS *****/
/*****/

/**** GET PRIME NUMBER p ****/
public BigInteger getp() {
    return( p );
}

/**** GET PRIME NUMBER q ****/
public BigInteger getq() {
    return( q );
}

/**** GET VALUE OF z ****/
public BigInteger getr() {
    return( r );
}

/**** GET VALUE OF n ****/
public BigInteger getn() {
    return( n );
}

/**** GET VALUE OF e ****/
public BigInteger gete() {
    return( e );
}

/**** GET VALUE OF d ****/
public BigInteger getd() {

```

```

        return( d );
    }

    /*****
     *    MAIN PROGRAM
     *****/
    public static void main( String[] args ) throws IOException {
        String inputPrimeSize;          // PrimeSize from user's
input
        BufferedReader br = new BufferedReader( new InputStreamReader(
System.in ) );

        System.out.println (
"\n+++++ " );
        System.out.println ( "    ENCRYPTION - DECRYPTION USING RSA " );
        System.out.println (
"+++++\n" );
        System.out.println ( "-----"
);
        System.out.println ( "PrimeSize should be greater than 3.!!!"
);
        System.out.print( "Enter PrimeSize: " );
        inputPrimeSize = br.readLine();
        System.out.println ( "-----
\n" );

        // Get bit length of each prime number
        int primeSize = Integer.parseInt( inputPrimeSize );

        // Generate Public and Private Keys

        RSA rsa = new RSA( primeSize );

        System.out.println (
"*****" );
        System.out.println( "\t\tProperties of RSA: " );
        System.out.println ( "\t\tgenerated by given PrimeSize" );
        System.out.println (
"*****" );

        System.out.println( "\tKey Size: " + primeSize + "\n" );

        System.out.println( "\tLarge prime numbers p and q" );
        System.out.print( "\tp: " + rsa.getp().toString() );
        System.out.println( ", q: " + rsa.getq().toString() + "\n" );

        System.out.println( "\tThe public key pair (n, e)" );
        System.out.print( "\tn: " + rsa.getn().toString() );
        System.out.println( "\te: " + rsa.gete().toString() + "\n" );

        System.out.println( "\tThe private key pair (n, d)" );
        System.out.print( "\tn: " + rsa.getn().toString() );
        System.out.println( "\td: " + rsa.getd().toString() );
        System.out.println ( "-----
\n\n" );
    }

```

```
        // Get message from user
        String message;
        BufferedReader buf = ( new BufferedReader( new
InputStreamReader( System.in ) ) );
        System.out.print( "Enter message: " );
        message = buf.readLine();
        System.out.println( "" );

        // Encrypt Message
        BigInteger[] ciphertext = rsa.encrypt( message );

        System.out.print( "Encrypt message: " );
        for( int i = 0 ; i < ciphertext.length ; i++ ) {
            System.out.print( ciphertext[i].toString() );

            if( i != ciphertext.length - 1 )
                System.out.print( " " ) ;
        }
        System.out.println( "\n" );

        String decryptmessage = rsa.decrypt( ciphertext );

        System.out.println( "Decrypt message: " + decryptmessage + "\n"
);
    }
} /**** --- END OF MAIN FUNCTION --- ***/
```