ISSN No. 0976-5697

Volume 3, No. 3, May-June 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

New Variant MultiPrime Jordon Totient–RSA Cryptosystem with one Public key and two Private Keys

*E.Madhusudhana Reddy and M.Srujan Kumar Reddy
*Professor & Head and Asst. Professor
Department of Information Technology
Madanapalle Institute of Technology and Science Madanapalle,India
e_mreddy@yahoo.com*srujanit18@gmail.com

Abstract: Security protocols are a must for communication between parties. We studied and came up with new applications of Jordon Totient function and applied them to RSA public key cryptosystem with one public key and two private keys, and developed protocols for communication between two parties using java for E-Commerce.

Keywords: Protocols, Jordon Totient function, public key cryptosystem.

I. INTRODUCTION

In this article we develop a new Public Key Cryptosystems which was extension of the work of Cesar Alison Monteiro Paixao [1] some variants of the RSA Cryptosystem. We extend variant analyzed in [2] using the properties of Jordan Totient function [3]. We briefly discuss the possibility and validity of combining new variant with algorithm, java code, test result and graphical performance analysis to obtain a new efficient and general Cryptosystem.

II. JORDAN–TOTIENT FUNCTION

A. *Definition:*

A generalization of the famous Euler's Totient function is the Jordan's Totient function [1,4] defined by

$J_k(n)=n^k \prod_{p|n}\left(1-p^{-k}\right)$, Where k, n $\in Z^+$

We define the conjugate of this function as

$\overline{J_k}(n)=n^k \prod_{p|n}\left(1+p^{-k}\right)$

B. *Properties:*

a. $J_k(1)=1, J_k(2)=2^k-1\equiv 1(\text{mod } 2)$

b. $J_k(n)$ is even if and only if $n \geq 3$

c. If p is a prime number then

$J_k(p)=p^k\left(1-p^{-k}\right)=\left(p^k-1\right)$

$J_k\left(p^\alpha\right)=p^{(\alpha-1)k}\left(p^k-1\right)$

d. If n = $p_1^{\alpha_1}.p_2^{\alpha_2}.....p_r^{\alpha_r}$ Then

$J_k(n)=p_1^{(\alpha_1-1)k}.p_2^{(\alpha_2-1)k}.....p_r^{(\alpha_r-1)k}.\left(p_1^k-1\right)\left(p_2^k-1\right).....\left(p_r^k-1\right)$

e. $J_1(n)=\phi(n)$

III. M- PRIME RSA CRYPTOSYSTEM

Multi Prime RSA Cryptosystem was introduced by Collins who modified the RSA modulus so that it consists of r primes $p_1,p_2,$-------, p_r instead of the traditional two primes p and q.

a. Key generation: The key generation algorithm[5] receives as parameter the integer r, indicating the number of primes to be used. The key pairs are generated as in the following steps.

a) Choose r distinct primes $p_1,p_2,$--------, p_r each one $\left\lfloor \frac{Logn}{r} \right\rfloor$ bits in length and

n = $\prod_{i=1}^{r} p_i = p_1.p_2.--------p_r$

b) Compute E and D such that d = e^{-1} (mod $\phi(n)$) where gcd (e, $\phi(n)$) = 1,

where $\varphi(n) = \prod_{i=1}^{r}(p_i-1)$

$=(p_1-1)(p_2-1).....(p_r-1)$

c) for $1 \leq i \leq r,$ compute $d_i \equiv d \pmod{p_i-1}$

Public Key = (n, e)

Private Key = (n, d_1, d_2, ---- d_r)

Encryption: Given a Public Key (n, e) and a message $M \in Z_n,$ encrypt M exactly as in the original RSA, thus

$C \equiv M^e \pmod{n}$

Decryption: The decryption is an extension of the quisquater – couvreur method. To decrypt a ciphertext C, first calculate

$M_i \equiv C^{d_i} \pmod{p_i}$ for each i = 1, 2, …….. r

Next apply the Chinese Remainder Theorem to the M_i's to get $M \equiv C^d \pmod{n}$

IV. M-PRIMEJ$_2$–RSA CRYPTOSYSTEM WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS

By replacing $\varphi(n)$ by $J_2(n)$ with the same property we can generate a new variant cryptosystem. Threshold key generation, encryption and decryption are given below.

A. *Key Generation:*

a. Choose r distinct primes $p_1, p_2, \ldots\ldots\ldots p_r$ each one

$\left\lfloor \frac{Logn}{r} \right\rfloor$ bits in length and

n = $\prod_{i=1}^{r} p_i = p_1.p_2 \ldots\ldots\ldots\ldots p_r$

b. compute E and D such that $D = E^{-1} \pmod{J_2(n)}$ i.e., $ED \equiv 1 \pmod{J_2(n)}$ where gcd (E, $J_2(n)$) = 1 and

$J_2(n) = n^k \prod_{p|k} (1 - p^{-k})$

$= (p_1^k - 1)(p_2^k - 1) - - - - - (p_r^k - 1)$

$= \prod_{i=1}^{r} (p_i^k - 1)$

c. for $1 \le i \le r$, compute $d_i \equiv d \pmod{p_i^k - 1}$

Public Key = (2, E, n)

Private Key = (2, D, n)

Encryption: Given a Public Key (2, E, n) and a message $M \in Z_n$, encrypt M exactly as in the original RSA, thus

$$C \equiv M^E \pmod{n}$$

Decryption: The decryption is an extension of the Quisquater Couvreur method. To decrypt a ciphertext C, first calculate $M_i \equiv C^{d_i} \pmod{p_i}$ for each $1 \le i \le r$, next apply Chinese Remainder Theorem to the M_i's to get $M \equiv C^D \pmod{n}$

V. ALGORITHM FOR M-PRIME J_2 – RSA CRYPTOSYSTEM WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS

Step 1: Start
Step 2: Generate primes $p_1, p_2, p_3 \ldots .p_r$ having Log n /r bits.
Step 3: [Compute N] $N \leftarrow p_1 * p_2 . \ldots p_r$
Step 4: [Compute E and D] $D \leftarrow E^{-1} \pmod{J_2(N)}$
Step 5: While i<= r
  Step 5.1: $J_2(n) \leftarrow J_k(n) * (p_i^k - 1)$
  Step 5.2: i←i++
Step 6: for 1<=i<=r
  Step 6.1: $D_i \leftarrow D \pmod{p_i^k - 1}$
  Step 6.2:i←i++
Step 7: [Compute Public key] Public Key← (k,E,N)
Step 8: [Compute Private key] Private Key ← (k,D,N)
Step 9: [read the plain text] read M
Step 9: [Compute Encryption cipher text C] $C \leftarrow M^E \pmod{n}$
Step 10: [Compute cipher text to Plain Text C] for 1<=i<=r
  Step 10.1: $M_i \leftarrow C^{D_i} \pmod{p_i}$
Step 11: $M \leftarrow C^D \pmod{N}$
Step 12: Stop

VI. IMPLEMENTATION OF MJ2-RSA JAVA CODE WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS FOR 128 BIT LENGTH

```
import java.io.*;
import java.util.Vector;
import java.math.BigInteger;
import java.util.Random;
import java.io.ByteArrayOutputStream;
import java.io.FileOutputStream;
import java.security.MessageDigest;
public class MJ2RSA {
        final BigInteger zero = new BigInteger("0") ;
        final BigInteger one = new BigInteger("1") ;
        final BigInteger two = new BigInteger("2") ;
        final BigInteger three = new BigInteger("3") ;
    int bitlength= 128;
    private BigInteger p1;
    private BigInteger p2;
    private BigInteger p3;
    private BigInteger p4;
    private BigInteger N;
    private BigInteger phi;
    private BigInteger e;
    private BigInteger d;
    private BigInteger d1;
    private BigInteger d2;
    private Random r;
    public MJ2RSA() {
            r = new Random(10);
          // get two big primes
            p1 = BigInteger.probablePrime(bitlength,
r);
            p2 = BigInteger.probablePrime(bitlength,
r);
            p3 = BigInteger.probablePrime(bitlength,
r);
            p4 = BigInteger.probablePrime(bitlength,
r);
          N =
p1.multiply(p2).multiply(p3).multiply(p4);
            phi =
p1.pow(2).subtract(BigInteger.ONE).multiply(p2.pow(2).su
btract(BigInteger.ONE)).multiply(p3.pow(2).subtract(BigInt
eger.ONE)).multiply(p4.pow(2).subtract(BigInteger.ONE));
          // compute the exponent necessary for encryption
(private key)
            e = BigInteger.probablePrime(bitlength/2,
r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0
&& e.compareTo(phi) < 0 )
        {
                e.add(BigInteger.ONE);
            }
        d = e.modInverse(phi);
    }
   public void privateFactors(BigInteger number)
        {
          boolean flag = false ;
          BigInteger limit = bigRoot(number).add(one);
      for (BigInteger i = three; i.compareTo(limit) <= 0;
i=i.add(two))
            {
        while (number.mod(i).compareTo(zero) == 0)
                {
        number=number.divide(i) ;
                    d1=i;
                    d2=number;
                    flag = true;
                    break;
            }
                if(flag == true)
                break ;
```

```
                }
        }
        public BigInteger bigRoot(BigInteger number)
    {
                BigInteger result = zero ;
                  BigInteger oldRoot ;
                  BigInteger newRoot ;
                  BigInteger zero = new BigInteger("0") ;
                  BigInteger two = new BigInteger("2") ;
                  BigInteger num = number ;
                  newRoot =
num.shiftRight(num.bitLength()/2) ;
                  do {
                     oldRoot = newRoot ;
                     newRoot =
oldRoot.multiply(oldRoot).add(num).divide(oldRoot).divide
(two) ;
                     }
while(newRoot.subtract(oldRoot).abs().compareTo(two)>0)
;
                  return newRoot;
        }
     public MJ2RSA(BigInteger e, BigInteger d, BigInteger
N) {
            this.e = e;
       this.d = d;
       this.N = N;
    }
    public static void main (String[] args) {
          BufferedReader br;
          long KGTime,ETime,DTime;
          long startTime = System.currentTimeMillis();
          MJ2RSA rsa = new MJ2RSA();
          System.out.println("The bitlength "+
rsa.bitlength);
         System.out.println("The value of P1  is "+rsa.p1);
         System.out.println("The value of P2  is "+rsa.p2);
         System.out.println("The value of P3  is "+rsa.p3);
         System.out.println("The value of P4  is "+rsa.p4);
         System.out.println("The value of N   is "+rsa.N);
         System.out.println("The value of J2N is "+rsa.phi);
         System.out.println("The Public Key E is "+ rsa.e);
         System.out.println("The Private Key D is"+ rsa.d);
         rsa.privateFactors(rsa.d);
         System.out.println("The Singer's Key D1 is    :\n"+
rsa.d1);
         System.out.println(" The Co-Singer's Key D2 is
:\n"+ rsa.d2);
         long endTime = System.currentTimeMillis();
         KGTime=endTime-startTime;
         System.out.println(" Key Generation Time (in
miliseconds):"+ KGTime);
         String teststring="";
       try{
         br=new BufferedReader(new
InputStreamReader(System.in));
         System.out.println("Enter the test string");
        teststring = br.readLine();
        System.out.println("Encrypting String: " +
teststring);
            }catch(Exception ex){}
        // encrypt
      long startEncyTime = System.currentTimeMillis();
        byte[] encrypted = rsa.encrypt(teststring.getBytes());
        System.out.println("Eincrypted String in Bytes: " +
bytesToString(encrypted));
        long endEncyTime = System.currentTimeMillis();
            ETime = endEncyTime-startEncyTime;
            System.out.println(" Encryption Time in
millSecond"+ ETime);
            String HashVal="";//null;
            String newMessage ="";
            String newMessageHashVal ="";
            String singMessage ="";
            String  encryptedhash ="";
// rsa.sigCreation(HashVal);
// decrypt
        long startDecyTime = System.currentTimeMillis();
            byte[] decrypted1 = rsa.decrypt1(encrypted);
            System.out.println("decryption with D1  gives the
string is:\n"+new String(decrypted1));
            byte[] decrypted = rsa.decrypt2(decrypted1);
           System.out.println("decryption with D2  gives the
string is: \n" + new String(decrypted));
        long endDecyTime = System.currentTimeMillis();
            DTime =endDecyTime-startDecyTime;
            System.out.println(" Decrypted Time in
millSecond"+DTime);
     }
      /**     * Converts a byte array into its String
representations  */
     private  static String bytesToString(byte[] encrypted) {
        String test = "";
        for (byte b : encrypted) {
           test += Byte.toString(b);
        }
        return test;
     }
     public byte[] encrypt(byte[] message) {
        return (new BigInteger(message)).modPow(e,
N).toByteArray();
     }
     /** * decrypt byte array for single public and single
private       */
     public byte[] decrypt(byte[] message) {
        return (new BigInteger(message)).modPow(d,
N).toByteArray();
     }
          /**      * decrypt byte array  for dual private keys
*/
     public byte[] decrypt1(byte[] message) {
        return (new BigInteger(message)).modPow(d1,
N).toByteArray();
     }
/** * decrypt byte array dual private keys        */
     public byte[] decrypt2(byte[] message) {
        return (new BigInteger(message)).modPow(d2,
N).toByteArray();
     }
          /**  encrypt string for single public key and single
private key */
     public String sigCreation(String message) {
        return (new BigInteger(message)).modPow(d,
N).toString();
     }
     /**  encrypt string dual private keys co-signer  */
```

```
    public String sigCreation1(String message) {
        return (new BigInteger(message)).modPow(d1,
N).toString();
    }
/**  encrypt string dual private keys verifier        */
    public String sigCreation2(String message) {
        return (new BigInteger(message)).modPow(d2,
N).toString();
    }
/**  decrypt string using single public key       */
    public String sigVerification(String message) {
        return (new BigInteger(message)).modPow(e,
N).toString();
    }
   // We are using MD5 hash function
    public String MD5HashFunction(String text)  throws
Exception
    {
      MessageDigest md;
           md = MessageDigest.getInstance("MD5");
           byte[] md5hash = new byte[32];
      md.update(text.getBytes("iso-8859-1"), 0,
text.length());
      md5hash = md.digest();
           String hashValue=convertToHex(md5hash);
           return hashValue;
   }
   public String convertToHex(byte[] data)
   {
      StringBuffer buf = new StringBuffer();
      for (int i = 0; i < data.length; i++) {
         int halfbyte = (data[i] >>> 4) & 0x0F;
         int two_halfs = 0;
         do {
            if ((0 <= halfbyte) && (halfbyte <= 9))
               buf.append((char) ('0' + halfbyte));
            else
               buf.append((char) ('a' + (halfbyte - 10)));
            halfbyte = data[i] & 0x0F;
         } while(two_halfs++ < 1);
      }
      //return buf.toString();
           return HextoBinary(buf.toString());
    }
   public String HextoBinary(String userInput)
   {
String[]hex={"0","1","2","3","4","5","6","7","8","9","A","B
","C","D","E","F"};
String[]binary={"0000","0001","0010","0011","0100","010
1","0110","0111","1000","1001","1010","1011","1100","11
01","1110","1111"};
           String result="";
           for(int i=0;i<userInput.length();i++)
           {
                 char temp=userInput.charAt(i);
           String temp2=""+temp+"";
                 for(int j=0;j<hex.length;j++)
                 {
                  if(temp2.equalsIgnoreCase(hex[j]))
                        {
                        result=result+binary[j];
                        }
                 }
           }
//System.out.println("IT'S BINARY IS : "+result);
           return result;
    }
     //end of class
   }
```

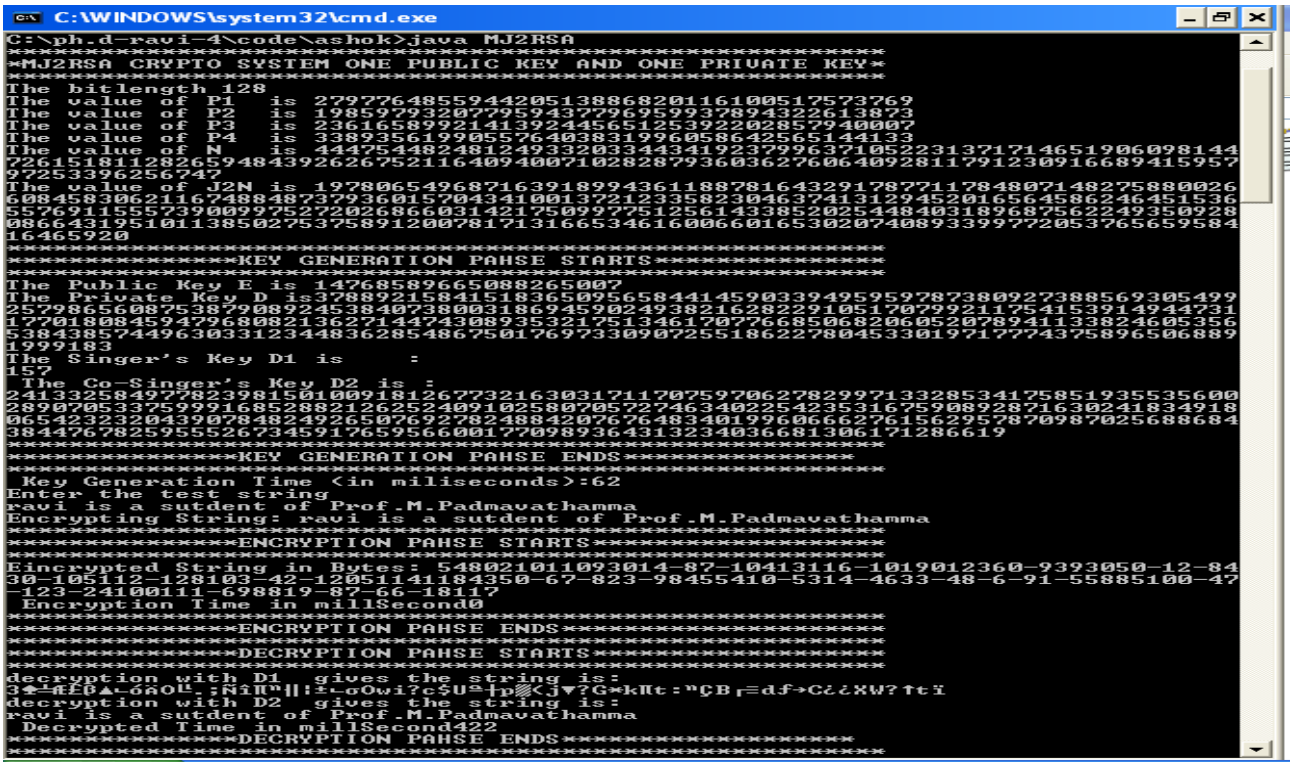

Figure 1. Test Results of MJ_2 -RSA with one public key and two private keys Java program

VII. GRAPHICAL PERFORMANCE ANALYSIS BETWEEN MRSA AND MJ_2 –RSA WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS

Table 1: Key Generation Time Performance

		MJ_2 -RSA
256	89	83
512	403	335
1024	3987	2791

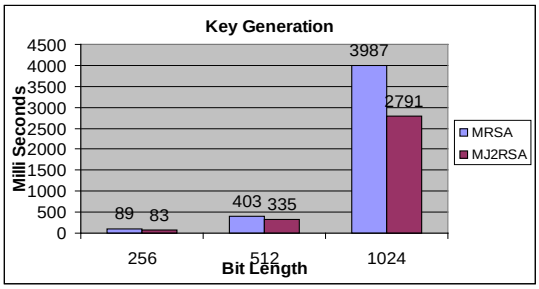

Figure 2: Key Generation

Table2:Encryption Time Performance

		MJ_2 -RSA
256	7	7
512	21	19
1024	107	85

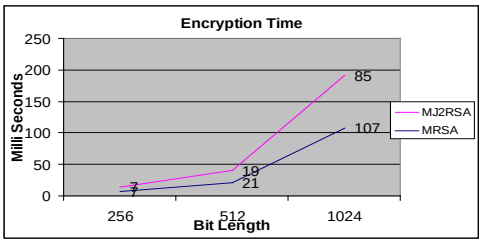

Figure 3:Encryption Time

Table 3:Decryption Time Performance

Bits	***MP-RSA-DSS***	***MPJ_2 –RSA-DSS***
256	24	24
512	43	37
1024	146	125

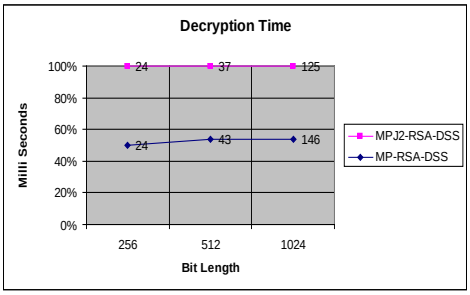

Figure 4: Decryption Time

Table 4:Comparison between MRSA and MJ_2 –RSA

		MJ_2 -RSA
KG	3987	2791
En Time	107	85
De Time	211	259

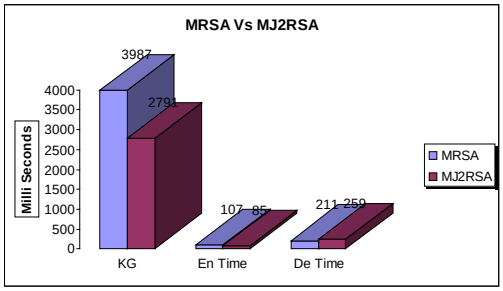

Figure 5:MRSA Vs MJ2RSA

VIII. CONCLUSION

In this article we presented design and development of Multi prime Jordan-Totient- RSA viz. MJ2-RSA cryptosystem with one public key and two private keys in Java and we analyzed the performance of our programs with the existing RSA cryptosystem and compared the performance of two systems key generation time, the performance of encryption time and decryption time respectively.

This result helps in enhancement of the block size for plaintext and enhances the range of public / private keys. The increase in the size of private key avoids the attacks on private key. This concludes that MJ2-RSA provides more security with low cost.

IX. REFERENCES

[1.] Apostal T.M, introduction to analytic number theory, Springer International Students Edition 1980.

[2.] E. Madhusudhana Reddy & M. Padmavathamma; Need for strong public key cryptography in electronic commerce, International journal of Computer Applications in Engineering Technology and Sciences. IJ-CA-ETS, Pages 58-68, March 2009.

[3.] E. Madhusudhana Reddy & M. Padmavathamma; An information security model for E-Business, The Technology World Quarterly Journal, Volume V, Issue I, Pages 203-206, December 2009.

[4.] E. Madhusudhana Reddy & M. Padmavathamma; A Middleware E-Commerce Security Solution using $RMPJ_K$ –RSA Cryptosystem and $RMPJ_K$ –RSA Digital Signature , International journal of knowledge Management and e-Learning, Volume 1, No 2, Pages 71-76, December 2009.

[5.] Thajoddin. S & Vangipuram S; A Note on Jordan's Totient function Indian J.Pure apple. Math. 19(12): 1156-1161, December, 1988.