I.J.E.M.S., VOL. 1(1) 2010: 1 8-25 ISSN 2229-600X

INTERNATIONAL JOURNAL OF ENGINEERING AND MANAGEMENT SCIENCES
© 2004-2010 Society for Science and Nature (SFSN) All rights reserved
www.scienceandnature.org

DEVELOPMENT OF NEW VARIANT MJ$_2$–RSA DIGITAL SIGNATURE SCHEME WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS

A. Ravi Prasad, M. Padmavathamma
Department of Computer Science, S.V.University, Tirupati

ABSTRACT
Digital Signature Scheme is very useful in communication between parties. We studied new applications of Jordon-Totient function and applied them to RSA Digital Signature Scheme and developed protocols for communication between two parties using java and shown the graphical performance analysis on test results for key generation time, encryption time and decryption time respectively.

KEY WORDS: Digital signature, Private Key, generation time, protocols etc.

INTRODUCTION

In this article we develop a new Digital Signature Schemes which are extension of some variants of the RSA Digital Signature Schemes. We extend new variant with the help of the properties of Jordan-Totient function [2]. 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 Digital Signature Schemes.

JORDAN–TOTIENT FUNCTION

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

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

We define the conjugate of this function as $\overline{J_k}(n)=n^k \prod_{p|n}\left(1+p^{-k}\right)$

Properties:

1) $J_k(1)=1, J_k(2)=2^k-1\equiv 1(\bmod 2)$

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

3) If p is a prime number Then

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

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

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

5) $J_1(n)=\phi(n)$

M - PRIME RSA DIGITAL SIGNATURE SCHEME

Multi Prime RSA digital Signature Scheme is constructed with r distinct prime numbers $p_1,p_2,……..,p_r$ instead of the traditional two primes p and q.

Key generation: 1. The Signer Choose sufficiently large distinct primes, $p_1,p_2,……..,p_r$ at random.

2. Compute N= p_1, p_2, ……..,p_r and $\phi(N)=(p_1-1)(p_2-1).......(p_r-1)$

$$\phi\text{ (n) } = \prod_{i=1}^{r}(p_i-1)$$

3. Choose a random integer e< $\phi(n)$ such that gcd (e, $\phi(n)$) = 1.

4. Compute an integer d which is the inverse of e such that $ed\equiv 1(\bmod\ \phi(n))$

5. For 1≤i≤r, compute $d_i\equiv d \quad (\bmod p_i-1)$

Public Key = (n, e)

Private Key =

($p_1,p_2,……..,p_r$, $d_1,d_2,……..,d_r$)

Signature generation: Using the Private Key ($p_1,p_2,……..,p_r$, $d_1,d_2,……..,d_r$) creates a signature 'σ' on the message M by the following method.

$$\sigma_1\equiv M^{d_1}(\bmod p_1)$$
$$\sigma_2\equiv M^{d_2}(\bmod p_2)$$
$$\vdots$$
$$\sigma_r\equiv M^{d_r}(\bmod p_r)$$

By Chinese Remainder Theorem, the above systems of congruence have unique solution.

$\sigma\equiv M^d(\bmod p_1.p_2......p_r)$

$\sigma\equiv M^d(\bmod n)$

Signature Verification: After obtaining the signature 'σ' and the message M.

Check whether $M\equiv\sigma^e(\bmod n)$

If the above equation holds then "Accept" the message otherwise "Reject" it.

M - Prime J$_2$ – RSA Digital Signature Scheme with one public key and two private keys

Here we replace $\varphi(n)$ by J$_2$(N) with the same property, we get a new variant digital Signature Scheme. Below we describe the modified key generation, signature generation and verification.

Key generation: 1. The Signer Choose sufficiently large distinct primes, $p_1, p_2, \ldots\ldots, p_r$ at random.

2. Compute $N = \prod_{i=1}^{r} p_i = p_1 \cdot p_2 \ldots\ldots\ldots\ldots p_r$ and

$$J_2(N) = n^k \prod_{p|k} (1 - p^{-k})$$
$$= (p_1^k - 1)(p_2^k - 1) - - - - - (p_r^k - 1)$$
$$= \prod_{i=1}^{r} (p_i^k - 1)$$

3. Choose a random integer E< J$_2$(N) such that gcd (E, J$_2$(N)) = 1.

4. Compute the integer D Which is the inverse of E i.e, $ED \equiv 1(\bmod J_k(N))$.

5. for $1 \le i \le r,$ compute $D_i \equiv D(\bmod p_i^k - 1)$

Public Key = (2, N, E)

Private Key = (2, ($p_1, p_2, \ldots\ldots, p_r$, $D_1, D_2, \ldots\ldots, D_r$))

Signature generation: Using the Private Key (N, E) creates a signature 'σ' on the message M by computing.

$$\sigma \equiv M^D (\bmod N)$$

Signature Verification**:** After obtaining the signature 'σ' and the message M.

Check whether $M \equiv \sigma^e (\bmod N)$

If the above equation holds then "Accept: the message otherwise "Reject" it.

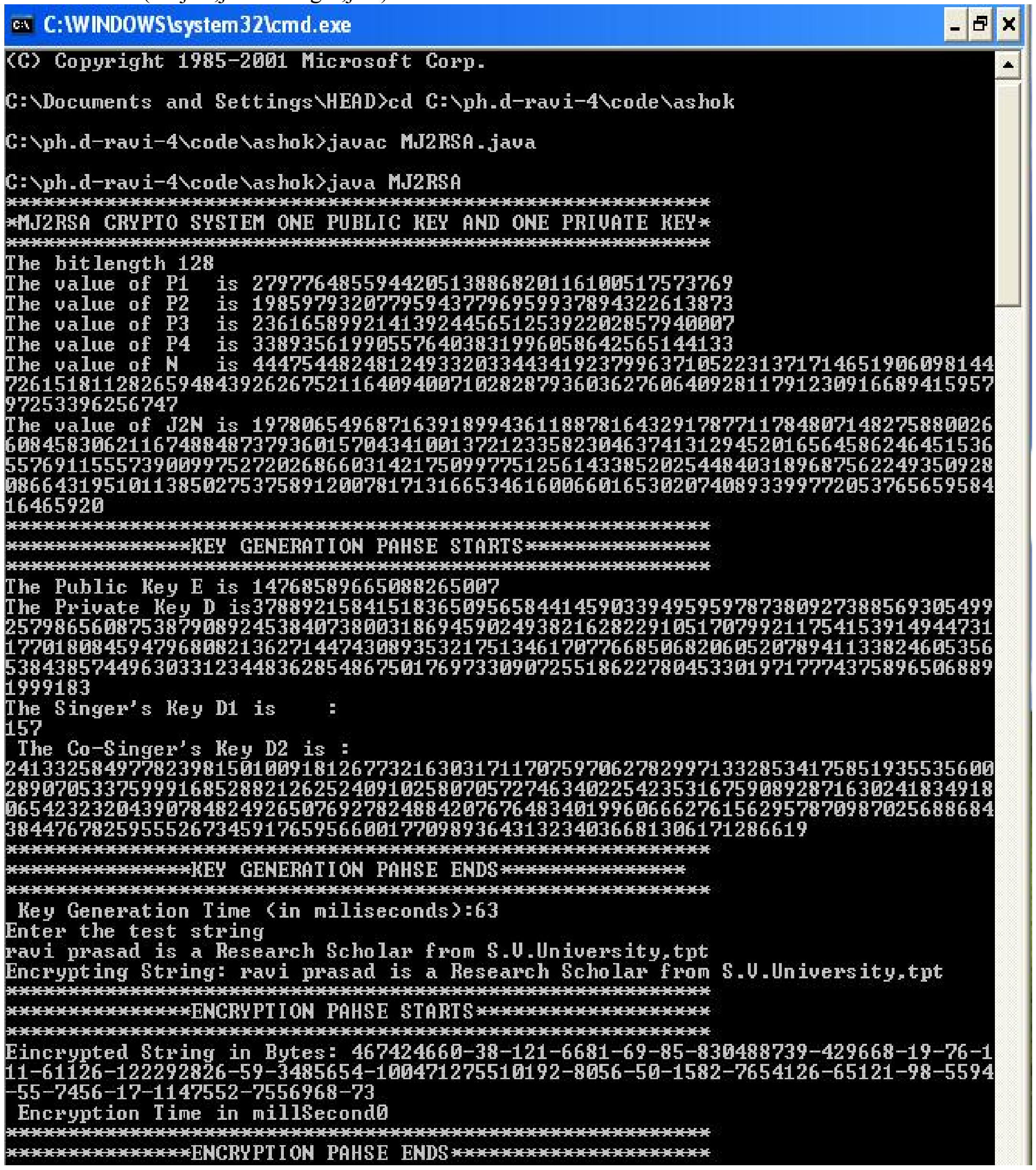
ALGORITHM FOR M-PRIME J$_2$ – RSA DIGITAL SIGNATURE SCHEME WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS

```
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   $p_1 * p_2. \ldots p_r$
Step 4:  [Compute E and D]         D E$^{-1}$ (mod J$_2$(N))

Step 5:  While i<= r
         Step 5.1:  J$_2$(n)   J$_k$(n)*($p_i^k$-1)
         Step 5.2:  i   i++
Step 6:     for 1<=i<=r
         Step 6.1:   Di   D          (mod $p_i^k$-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   M$^E$ (mod n)
Step 10: [Compute cipher text to Plain Text C]  for 1<=i<=r
Step 10.1: Mi   C$^{Di}$ (mod p$_i$)
Step 11:   M   C$^D$(mod N)
Step 12: Stop
```

IMPLEMENTATION OF MJ$_2$-RSA DIGITAL SIGNATURE SCHEME JAVA CODE WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS FOR 128 BIT LENGTH

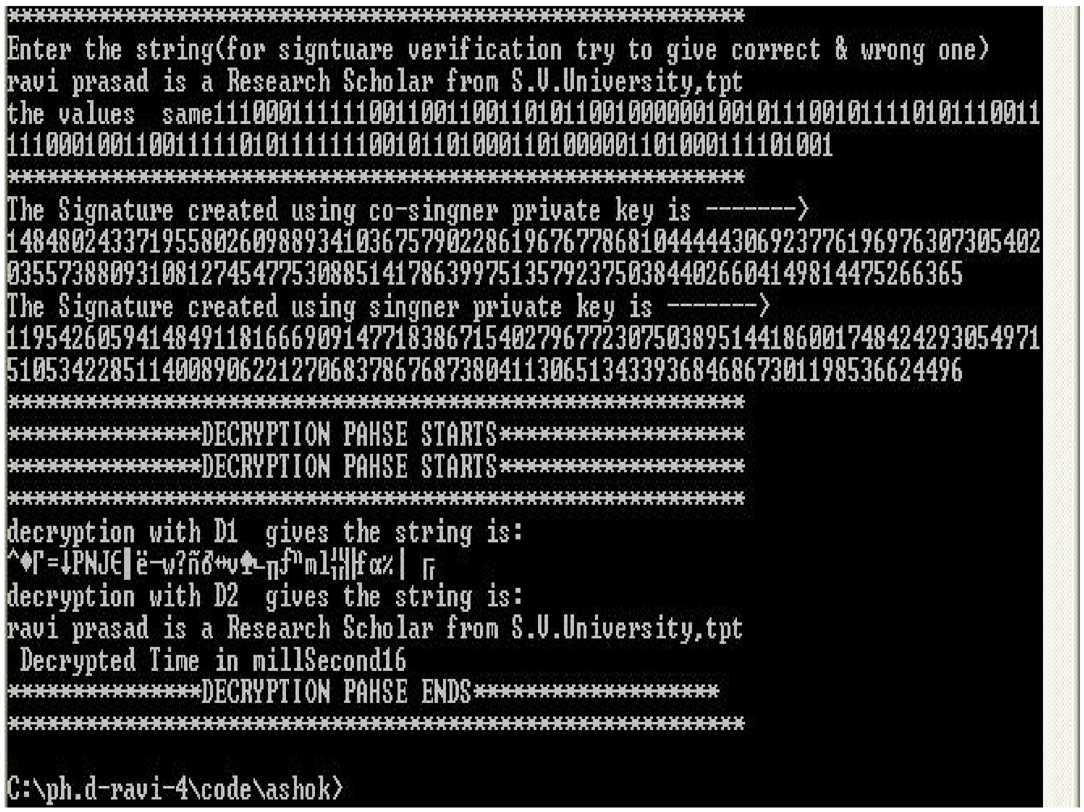
```
import java.io.*;
import java.util.Vector;
import java.math.BigInteger;
import java.util.Random;
import java.io.ByteArrayOutputStream;
import java.io.FileOutputStream;
import java.security.MessageDigest;

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).s
ubtract(BigInteger.ONE)).multiply(p3.pow(2).subtract(Bi
gInteger.ONE)).multiply(p4.pow(2).subtract(BigInteger.O
NE));
   // 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).divi
de(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);
          try{
         br=new BufferedReader(new
InputStreamReader(System.in));
                System.out.println("Enter the string(for
signtuare verification try to give correct & wrong one)");
         newMessage = br.readLine();
                HashVal =
rsa.MD5HashFunction(teststring);
                HashVal =
rsa.MD5HashFunction(teststring);

        newMessageHashVal=rsa.MD5HashFunction(ne
wMessage);

        if(!HashVal.equals(newMessageHashVal) )
          {
                      singMessage =
newMessageHashVal ;

        System.out.println("the values are not same
\n"+newMessageHashVal);
                  }
                else
                  {
                      singMessage = HashVal;
                System.out.println("the values
same"+HashVal);
```

```
            }
            }catch(Exception ex){System.out.println(ex);}
            String coSigner =
rsa.sigCreation1(singMessage);
/**/        System.out.println("The Signature created
using co-singner private key is ------->\n"+coSigner);
            encryptedhash =  rsa.sigCreation(singMessage);
            encryptedhash =  rsa.sigCreation2(coSigner);
            System.out.println("The Signature created
using singner private key is ------->\n"+encryptedhash);
// 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);
        //byte[] decrypted = rsa.decrypt(encrypted);
        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);
                    //end of main function
    }
     /**
     * 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;
    }
    /**
     * encrypt byte array
     */
    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","01
01","0110","0111","1000","1001","1010","1011","1100","
1101","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
  }
```



```
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\HEAD>cd C:\ph.d-ravi-4\code\ashok

C:\ph.d-ravi-4\code\ashok>javac MJ2RSA.java

C:\ph.d-ravi-4\code\ashok>java MJ2RSA
*********************************************************
*MJ2RSA CRYPTO SYSTEM ONE PUBLIC KEY AND ONE PRIVATE KEY*
*********************************************************
The bitlength 128
The value of P1  is 279776485594420513886820116100517573769
The value of P2  is 198597932077959437796959937894322613873
The value of P3  is 236165899214139244565125392202857940007
The value of P4  is 338935619905576403831996058642565144133
The value of N   is 444754482481249332033443419237996371052231371714651906098144
72615181128265948439262675211640940071028287936036276064092811791230916689415957
97253396256747
The value of J2N is 197806549687163918994361188781643291787711784807148275880026
60845830621167488487379360157043410013721233582304637413129452016564586246451536
55769115557390099752720268660314217509977512561433852025448403189687562249350928
08664319510113850275375891200781713166534616006601653020740893399772053765659584
16465920
*********************************************************
***************KEY GENERATION PAHSE STARTS***************
*********************************************************
The Public Key E is 14768589665088265007
The Private Key D is378892158415183650956584414590339495959787380927388569305499
25798656087538790892453840738003186945902493821628229105170799211754153914944731
17701808459479680821362714474308935321751346170776685068206052078941133824605356
53843857449630331234483628548675017697330907255186227804533019717774375896506889
1999183
The Singer's Key D1 is    :
157
 The Co-Singer's Key D2 is :
24133258497782398150100918126773216303171170759706278299713328534175851935535600
28907053375999168528821262524091025807057274634022542353167590892871630241834918
06542323204390784824926507692782488420767648340199606662761562957870987025688684
38447678259555267345917659566001770989364313234036681306171286619
*********************************************************
***************KEY GENERATION PAHSE ENDS***************
*********************************************************
 Key Generation Time (in miliseconds):63
Enter the test string
ravi prasad is a Research Scholar from S.V.University,tpt
Encrypting String: ravi prasad is a Research Scholar from S.V.University,tpt
*********************************************************
***************ENCRYPTION PAHSE STARTS*******************
*********************************************************
Eincrypted String in Bytes: 467424660-38-121-6681-69-85-830488739-429668-19-76-1
11-61126-122292826-59-3485654-100471275510192-8056-50-1582-7654126-65121-98-5594
-55-7456-17-1147552-7556968-73
 Encryption Time in millSecond0
*********************************************************
***************ENCRYPTION PAHSE ENDS*********************
```

TEST RESULTS OF MJ$_2$ -RSA DIGITAL SIGNATURE WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS

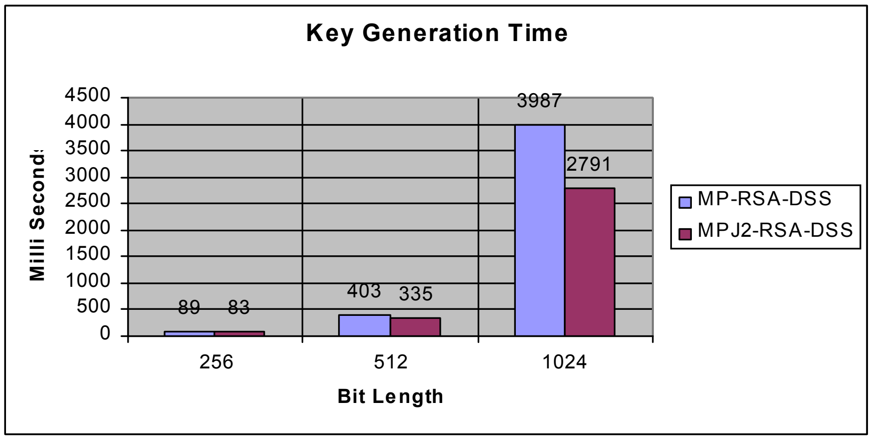


```
*********************************************************
Enter the string(for signtuare verification try to give correct & wrong one)
ravi prasad is a Research Scholar from S.V.University,tpt
the values  same1110001111110011001100110101100100000010010111001011110101110011
1110001001100111110101111111001011010001101000001101000111101001
*********************************************************
The Signature created using co-singner private key is ------->
1484802433719558026098893410367579022861967677868104444306923776196976307305402
03557388093108127454775308851417863997513579237503844026604149814475266365
The Signature created using singner private key is ------->
11954260594148491181666909147718386715402796772307503895144186001748424293054971
51053422851140089062212706837867687380411306513433936846867301198536624496
*********************************************************
***************DECRYPTION PAHSE STARTS*******************
***************DECRYPTION PAHSE STARTS*******************
*********************************************************
decryption with D1  gives the string is:
^♦Γ=↓PNJ€▌ë-w?ñδ↔v♠-╥ƒⁿml╬╣f α%| ╓
decryption with D2  gives the string is:
ravi prasad is a Research Scholar from S.V.University,tpt
 Decrypted Time in millSecond16
***************DECRYPTION PAHSE ENDS*******************
*********************************************************

C:\ph.d-ravi-4\code\ashok>
```

GRAPHICAL PERFORMANCE ANALYSIS BETWEEN M - PRIME RSA DIGITAL SIGNATURE SCHEME (MRSA-DSS) WITH M - PRIME J$_2$ – RSA DIGITAL SIGNATURE SCHEME (MPJ$_2$-RSA-DSS) WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS

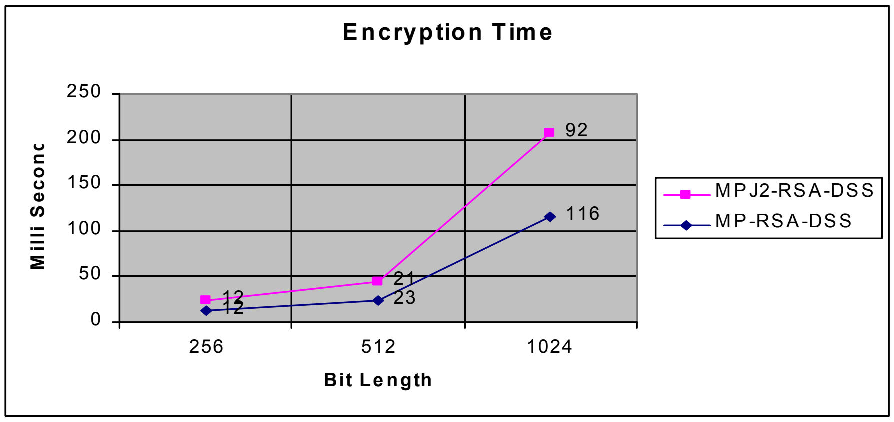
Key Generation Time Performance

Bits	MRSA-DSS	MPJ$_2$ –RSA-DSS
256	89	83
512	403	335
1024	3987	2791



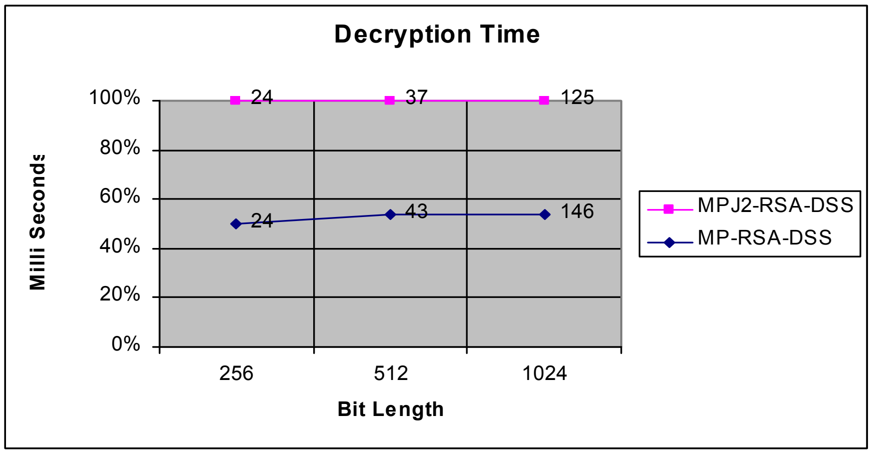
Encryption Time Performance

Bits	MRSA-DSS	MPJ$_2$ –RSA-DSS
256	12	12
512	23	21
1024	116	92



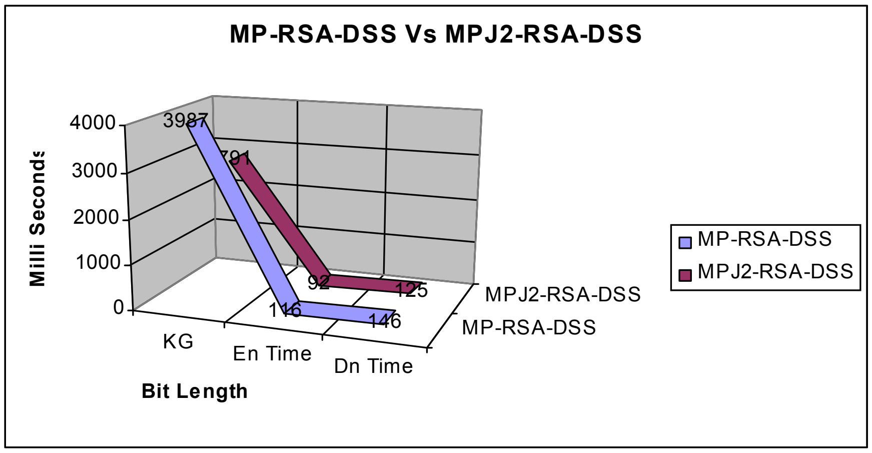
Decryption Time Performance

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



Comparison between MRSA-DSS and MPJ2-RSA-DSS

1024Bits	MRSA-DSS	MPJ_2 –RSA-DSS
KG	3987	2791
En Time	116	92
Dn Time	146	125



CONCLUSION

In this article we presented design and development of Multi prime Jordan-Totient- RSA viz. MJ2-RSA Digital signature scheme with one public key and two private keys in Java and we analyzed the performance of our programs with the existing RSA Digital signature schemes 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.

REFERENCES

[1]. D. Chaum. “Blind Signature for Untraceable Payments”.Advances in Cryptology – Crypto'82, pp.199-203, 1983.

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

[3]. SSL 3.0 Specification. http://wp.netscape.com/eng/ssl3/.1996

[4]. P. L. Yu and C. L. Lei. An User Efficient Fair E-cash Scheme with Anonymous Certificates. Electrical and Electronic Technology, 2001. TENCON. Proceedings of IEEE Region 10 International Conference on, Vol. 1, Aug 2001.

[5]. R. L. Rivest, A. Shamir, and L. Adleman. A Method For Obtaining Digital Signatures and Public-key Cryptosystems. Communications of ACM, Vol.21, No.2, pp.120-126, Feb 1978.

[6]. Proceedings of ACM CCS’s 93, pages 62-93, ACM, 1993

[7]. Quisquater J.J and Couvreur C, Fast Decipherment Algorithm for RSA Public-Key Cryptosystem. Electronic Lectures, Vol-18, 905-907, 1982.

[8]. Rivest R: Shamir A and Adleman L: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM 21 (2), pages 120-126,1978.