DEVELOPMENT OF NEW VARIANT MJ2–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 RSA Digital Schemes which are extension of some variants of the RSA Digital Schemes. We extend new variant with the help of the properties of Jordon-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

\[ J_k(n) = \prod_{p \mid n} \left(1 - p^{-k}\right), \]

Where \( k, n \in \mathbb{Z}^+ \)

We define the conjugate of this function as

\[ \overline{J_k}(n) = n^k \prod_{p \mid n} \left(1 + p^{-k}\right) \]

Properties:
1) \( J_k(1) = 1, J_k(2) = 2^k - 1 \equiv 1 \pmod{2} \)
2) \( J_k(n) \) is even if and only if \( n \geq 3 \)
3) If \( p \) is a prime number Then
\[ J_k(p^r) = p^r \left(1 - p^{-k}\right) = \left(p^r - 1\right) \]
\[ J_k(p^r) = p^r \left(1 - p^{-k}\right) = \left(p^r - 1\right) \]
4) If \( n = p_1^{a_1}p_2^{a_2} \ldots \ldots \ldots p_r^{a_r} \) Then
\[ J_k(n) = \prod_{i=1}^{r} \left(p_i^{a_i} - 1\right) \]
5) \( J_k(n) = \phi(n) \)

\( \phi(n) = \prod_{i=1}^{r} \left(p_i - 1\right) \)

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 \( ed \equiv 1 \pmod{\phi(n)} \)
5. For \( 1 \leq i \leq r \), compute \( d_i \equiv d \pmod{p_i - 1} \)

Public Key = \( (n, e) \)
Private Key = \( (p_1, p_2, \ldots \ldots, p_r, d_1, d_2, \ldots \ldots, d_r) \)

Signature generation: Using the Private Key \( (p_1, p_2, \ldots \ldots, p_r, d_1, d_2, \ldots \ldots, d_r) \) creates a signature ‘\( \sigma \)’ on the message \( M \) by the following method.

\[ \sigma_i \equiv M^{d_i} \pmod{p_i} \]
\[ \sigma_j \equiv M^{d_j} \pmod{p_j} \]
\[ \ldots \ldots \ldots \]
\[ \sigma_r \equiv M^{d_r} \pmod{p_r} \]

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

\[ \sigma \equiv M^d \pmod{p_1p_2 \ldots \ldots p_r} \]

Signature Verification: After obtaining the signature ‘\( \sigma \)’ and the message \( M \).
Check whether \( M \equiv \sigma^e \pmod{n} \)

If the above equation holds then “Accept” the message otherwise “Reject” it.
M - Prime J_r – RSA Digital Signature Scheme with one public key and two private keys

Here we replace \( \varphi(n) \) by \( J_r(N) \) 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, \( p_1, p_2, \ldots, p_r \) at random.

2. Compute \( N = \prod_{i=1}^{r} p_i \), \( p_1 \neq p_2 \ldots \neq p_r \) and

\[
J_r(N) = \prod_{p_k} \left( 1 - \frac{1}{p_k^n} \right)
= \left( \prod_{p_k} \left( \frac{1}{p_k^n} - 1 \right) \right)
= \prod_{i=1}^{r} \left( p_i^{k_i} - 1 \right)
\]

3. Choose a random integer \( E < J_r(N) \) such that \( \gcd(E, J_r(N)) = 1 \).

4. Compute the integer \( D \) which is the inverse of \( E \) i.e., \( E \cdot D \equiv 1 \pmod{J_r(N)} \).

5. For \( 1 \leq i \leq r \), compute \( D_i = D \left( \mod p_i^{k_i} - 1 \right) \)

Public Key = \((2, N, E)\)

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

Signature generation: Using the Private Key \((N, E)\) creates a signature ‘\( \sigma \)’ on the message \( M \) by computing.

\[ \sigma = M^D \pmod{N} \]

Signature Verification: After obtaining the signature ‘\( \sigma \)’ and the message \( M \).

Check whether \( M \equiv \sigma^E \pmod{N} \)

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

Algorithm for M-Prime J_r – 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 \( N \)] \( N = p_1 \cdot p_2 \cdot \ldots \cdot p_r \)

Step 4: [Compute \( E \) and \( D \)] \( D_y E^{-1} \pmod{J_r(N)} \)

Step 5: While \( i < r \)

Step 5.1: \( J_r(n) = \prod_{p_k} \left( \frac{1}{p_k^n} - 1 \right) \)

Step 5.2: \( i++ \)

Step 6: For \( 1 \leq i < r \)

Step 6.1: \( D_i = D \left( \mod p_i^{k_i} - 1 \right) \)

Step 6.2: \( i++ \)

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 10: [Compute Encryption cipher text C] \( C = M^E \pmod{n} \)

Step 10.1: \( M_i \leftarrow C^{D_i} \pmod{p_i} \)

Step 11: \( M \leftarrow C^D \pmod{n} \)

Step 12: Stop
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 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);
Development of new variant MJ–RSA digital signature scheme with one public key and two private keys

```java
long startDecyTime = System.currentTimeMillis();
byte[] decrypted = rsa.decrypt1(encrypted);
System.out.println("decryption with D1  gives the string is:
" + new String(decrypted1));
//byte[] decrypted = rsa.decrypt2(encrypted);
System.out.println("decryption with D2 gives the string is:
" + new String(decrypted));
long endDecyTime = System.currentTimeMillis();
DTime =endDecyTime-startDecyTime;
System.out.println(" Decrypted Time in millSecond"+DTime);
```

```java
/**
 * 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(d, N).toByteArray();
}

/**
 * decrypt byte array for single public and single private
 */
public byte[] decrypt(byte[] message) {
    return (new BigInteger(message)).modPow(d1, 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();
}
```

```java
public String sigCreation1(String message) {
    return (new BigInteger(message)).modPow(d1, N).toString();
}

public String sigCreation2(String message) {
    return (new BigInteger(message)).modPow(d2, N).toString();
}

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);
        do {
            if (((0 <= halfbyte) && (halfbyte <= 9))
                buf.append((char) ('0' + halfbyte));
            else
                buf.append((char) ('a' + (halfbyte - 10)));
        } while(halfbyte+++ < 1);
    }
    //return buf.toString();
    return HextoBinary(buf.toString());
}
```
public String HextoBinary(String userInput) {
    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

TEST RESULTS OF MJ, -RSA DIGITAL SIGNATURE WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS
Development of new variant MJ–RSA digital signature scheme with one public key and two private keys

The Signature created using co-signer private key is -------
1464062437195589269088341036757922861967877868184444306923776196973073642
03557388931081275475353985141786399751357923758384426664149814475266365

The Signature created using signer private key is -------
1195426859414911816669994477183967154027967230750389514418600174942493054971
5105342285114008562212706837867587380443065134333646067891190536624496

GRAPHICAL PERFORMANCE ANALYSIS BETWEEN M - PRIME RSA DIGITAL SIGNATURE SCHEME (MRSA-DSS) WITH M - PRIME J – RSA DIGITAL SIGNATURE SCHEME (MPJ2-RSA-DSS) WITH ONE PUBLIC KEY AND TWO PRIVATE KEYS

Key Generation Time Performance

<table>
<thead>
<tr>
<th>Bits</th>
<th>MRSA-DSS</th>
<th>MPJ2–RSA-DSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>89</td>
<td>83</td>
</tr>
<tr>
<td>512</td>
<td>403</td>
<td>335</td>
</tr>
<tr>
<td>1024</td>
<td>3987</td>
<td>2791</td>
</tr>
</tbody>
</table>

Encryption Time Performance

<table>
<thead>
<tr>
<th>Bits</th>
<th>MRSA-DSS</th>
<th>MPJ2–RSA-DSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>512</td>
<td>23</td>
<td>21</td>
</tr>
<tr>
<td>1024</td>
<td>116</td>
<td>92</td>
</tr>
</tbody>
</table>
Decryption Time Performance

<table>
<thead>
<tr>
<th>Bits</th>
<th>MRSA-DSS</th>
<th>MPJ2-RSA-DSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>512</td>
<td>43</td>
<td>37</td>
</tr>
<tr>
<td>1024</td>
<td>146</td>
<td>125</td>
</tr>
</tbody>
</table>

Decryption Time

<table>
<thead>
<tr>
<th>Bit Length</th>
<th>KG</th>
<th>En Time</th>
<th>Dn Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>3987</td>
<td>116</td>
<td>146</td>
</tr>
<tr>
<td>512</td>
<td>116</td>
<td>92</td>
<td>37</td>
</tr>
<tr>
<td>1024</td>
<td>146</td>
<td>125</td>
<td>125</td>
</tr>
</tbody>
</table>

Comparison between MRSA-DSS and MPJ2-RSA-DSS

MP-RSA-DSS Vs MPJ2-RSA-DSS
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


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