Chapter 3

Threshold selective secure field RSA ($S^2$FRSA) Public Key Cryptosystem
THRESHOLD SELECTIVE SECURE FIELD RSA (S²FRSA) PUBLIC KEY CRYPTOSYSTEM

This chapter discusses about the various security techniques and proposed cryptosystem. In section 3.1 we briefly discuss about security concerns to data and its effects on health insurance. Section 3.2 gives idea about RSA technique for encryption and. In Section 3.3 we briefly discuss about the proposed S²FRSA technique along with key generation, encryption and decryption. The implementation of S²FRSA is given in Section 3.4 and the results are given in Section 3.5. In Section 3.6 performance analysis of S²FRSA and RSA with sample values are given.

3.1 Introduction

Currently, huge amount of data and information are available for everyone, and the data can now be stored in many different kinds of databases and information repositories in various formats, besides being available on the Internet or in printed form. With such a huge amount of data, there is a need for powerful data mining techniques for better interpretation of these data that exceeds the human's ability for comprehension and making decision in an improved manner. In order to reveal the best tools for dealing with the classification task that helps in decision making, various researches has conducted a comparative study between some of the free available data mining and knowledge discovery tools and software packages. Results have showed that the performance of the tools for the classification task is affected by the kind of dataset used and by the way the classification algorithms were implemented within the toolkits.

In this chapter, we present the design and development of our threshold public key cryptosystem viz. S²FRSA cryptosystem with one public key and one private key, MASDMA with one public key and one private key. The S²FRSA Cryptosystem allows user to select specified fields from encrypted data to be exposed from database securely. The MASDMA is designed by combining with S²FRSA cryptosystem, it helps users to make decision for selecting the best hospital and best insurance company in e-Medical Health Insurance System.
During this research, various existing algorithms for classifications were studied, which is considered as basic step in order to propose new advance algorithm or logic that extends the work previously done. Hence examined and compared the previous work with their short comings /limitations.

The Scope of the research is developing fast and secure decision making algorithm and to develop an appropriate e-Medical Health Insurance Application. With the new approaches we will be able to solve secure decision problem effectively and can generate decisions with speed up and size up perception.

The complexity of decision-making in e-Medical Health Insurance system is rapidly increasing, as the world is becoming data-driven. To cope with this increasing complexity in a changing environment, new modeling and computing paradigms are needed. The six salient features of the new modeling and computing paradigm are (i)Adaptability of decision-making models to the evolving decision environment, (ii) Ability to handle changing qualitative and quantitative data, (iii) Short decision response time, (iv)Large and overlapping problem domains, (v) Interpretability of the results, (vi)Process rather than problem orientation.

Considering this fact, the result of our research will be fast approach which will be applicable to various organizations.

3.2 RSA Cryptosystem

The RSA algorithm was described in 1977 by Ron Rivest, ShamirAdi, and Adleman Leonard, the letters RSA are the initials of their surnames [7,25]. The system includes a communications channel coupled to at least one terminal having an encoding device and to at least one terminal having a decoding device. A message-to-be-transferred is enciphered to cipher text at the encoding terminal by encoding the message as a number M in a predetermined set. That number is then raised to a first predetermined power (associated with the intended receiver) and finally computed. The remainder or residue, C, is... computed when the exponentiated number is divided by the product of two predetermined prime numbers (associated with the intended receiver).
The RSA algorithm involves three steps: key generation, encryption and decryption.

3.2.1 Key generation

RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers \( p \) and \( q \).
   - For security purposes, the integers \( p \) and \( q \) should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a primality test.

2. Compute \( n = pq \).
   - \( n \) is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.

3. Compute \( \phi(n) = \phi(p)\phi(q) = (p-1)(q-1) \), where \( \phi \) is Euler's Totient function.

4. Choose an integer \( e \) such that \( 1 < e < \phi(n) \) and \( \gcd(e, \phi(n)) = 1 \); i.e. \( e \) and \( \phi(n) \) are co-prime.
   - \( e \) is released as the public key exponent.
   - Having a short bit-length and small Hamming weight results in more efficient encryption – most commonly \( 2^{16} + 1 = 65,537 \). However, much smaller values of \( e \) (such as 3) have been shown to be less secure in some settings.

5. Determine \( d \) as \( d^{-1} = e \mod \phi(n) \), i.e., \( d \) is the multiplicative inverse of \( e \) (modulo \( \phi(n) \)).
   - This is more clearly stated as solve for \( d \) given \( d \cdot e \equiv 1 \mod \phi(n) \)
   - This is often computed using the extended Euclidean algorithm.
• $d$ is kept as the private key exponent.

By construction, $d \cdot e \equiv 1 \pmod{\varphi(n)}$. The public key consists of the modulus $n$ and the public (or encryption) exponent $e$. The private key consists of the modulus $n$ and the private (or decryption) exponent $d$, which must be kept secret. $p$, $q$, and $\varphi(n)$ must also be kept secret because they can be used to calculate $d$.

3.2.2 Encryption

User-A transmits her public key $(n, e)$ to User-B and keeps the private key secret. User-A then wishes to send message $M$ to User-B.

He first turns $M$ into an integer $m$, such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the cipher text $c$ corresponding to

$$c \equiv m^e \pmod{n}$$

This can be done quickly using the method of exponentiation by squaring. User-B then transmits $c$ to User-A.

3.2.3 Decryption

User-A can recover $m$ from $c$ by using her private key exponent $d$ via computing

$$m \equiv c^d \pmod{n}$$

Given $m$, she can recover the original message $M$ by reversing the padding scheme. Thus the RSA public key cryptosystem is used even today for security with increased key size which may not be suitable for some of the applications and also past twenty years it has been faced many attacks.

3.3 Proposed Threshold Selective Secure Field RSA ($S^2$FRSA)

$S^2$FRSA algorithm has three phases they are: Key generation, Encryption and Decryption phases.
Key generation Phase generates private and public keys. The private key generated is based on the selected fields in data to be viewed as remaining fields will be kept intact at data storage. Private Key is tokenized into keys parts with selected fields embedded into it to generate new key which can be used to view only the data corresponding to selected fields of data which are present in private key. In this way not only the private key security is enhanced but also selected fields are easily transferred to the receiver.

In number theory, the partition function \( p(n) \) represents the number of possible partitions of a natural number \( n \), which is to say the number of distinct ways of representing \( n \) as a sum of natural numbers (with order irrelevant)[43].

The general problem in the theory of partitions is to enumerate representations of a positive integer \( n \) as the sum

\[
n = \lambda_1 + \lambda_2 + \cdots + \lambda_k \text{ where each } \lambda_i \text{ comes from a specified multi set of integers.}
\]

The Selected Fields for each client is selected based on partitioning of random number \( n \) and selection of partition based on \( p(n) \). For security purpose fields are chosen by selecting partition that are not repeated and maximum of all partitions.

A) Key Generation Process

1. Generate two random large prime numbers \( r, s \) with equal size \( (n/2) \) bits.
2. Compute \( n = r \ast s \) and \( Eu = (r-1) \ast (s-1) \).
3. Compute \( e \) value such that \( \gcd(e, Eu) = 1 \) and \( 1 < e < Eu \).
4. Compute \( d \) value such that \( d = e^{-1} \mod(Eu) \).
5. Select a Random integer ‘\( z \)’ such that \( Eu > z > f \).
6. calculate \( x = (f^z)^f \mod(n) \).
7. Select a random positive integer \( P \) and by applying partition theory choose partition \( P = p_1 + p_2 + \ldots \) from possible partitions \( g(n) \) such that \( p_1 \) and \( p_2 \) are unique and maximum values in partitions. The field numbers are chosen as \( p_1 \) and \( p_2 \), and add Pinto the in middle of the private key \( d \) as

7.1. Randomly place \( P \) into \( d \) without specifying \( p_1 \) and \( p_2 \).
7.2 Assume \( P \) is placed in centre of \( d \) and therefore \( fnld=(d1+P+d2) \)

7.3 \( d = fnld \)

8. Publish Public key is \((z, e, n)\) and private key is \((x, d, n)\)

B) **Encryption Process**

The Sender uses public key to encrypt the entire data record while data in transit or at storage:

- Obtain the recipient public key \((z, e, n)\)
- Select a Random integer \( f \) such that \( f < z \)
- Message \( M \) will be represented as positive integer \( m < n \).
- Encrypt the message \( m \) with the public key \((e, n)\) like \( C = m^e \mod (n) + f^z \)
- Sends the cipher text \( C \) to recipient.

C) **Decryption Process**

The Receiver does the following to decrypt the selected field’s data from Cipher text \( C \) with the private key which encompasses the field numbers which have to be decrypted.

- Uses the private key \((x, d, n)\) to decrypt the message like
- To reverse the above process of calculate \( d \) in the key generation Phase.
- Before going to decryption process, the \( x \) will be decrypted as \( y = x^d \mod (n) \)
- Now finally decrypt the Cipher text as \( M=C^d \mod (n) - y \)

Below is the example for verification of our proposed Algorithm \( S^2\text{FRSA} \) where we are encrypting and decrypting the Message \( M=65 \).

1. Choose two distinct prime numbers, such as \( p=179 \) and \( q=229 \).
2. Select a Random number \( Z \) and according to partition theory \( Z \) is computed as \( Z=z1 + z2 \) where \( z1 \) and \( z2 \) are maximum unique numbers that yield \( Z \) in partitions fields.

\[ Z=7 \]
3. Partition $Z$ into $z1 + z2$ where $z1$ and $z2$ are two Fields numbers used for encryption or decryption.

For Field 1=3 (z1)

For Field 2=4 (z2)

Therefore we take $Z=F=7 (3+4)$.

4. Compute $n = p \cdot q$ giving

$$n=179 \times 229 = 40991.$$  

5. Compute the totient of the product as $\varphi(n) = (p - 1)(q - 1)$ giving

$$\varphi(40991) = (179 - 1)(229 - 1) = 40584.$$  

6. Compute the $f^e = 7^2 = 49$.

7. Compute the $x=(f^e)^x \mod (n)$

$$X=49^{131} \mod (40991)$$

8. Choose any number $1 < e < 40584$ that is coprime to 40584. Choosing a prime number for $e$ leaves us only to check that $e$ is not a divisor of 40584

Let $e=131$

9. Compute $d$, the modular multiplicative inverse of $e \mod \varphi(n)$ yielding

$$D=39035.$$  

The public key is $(z=2, e = 131, n = 40991)$. For a padded plaintext message $m$, the encryption function is

$$c(m) = m^{131} \mod (40991) + f^x$$

The private key is $(n = 40991, d = 39035)$. For an encrypted cipher text $c$, the decryption function is

$$m=C^d \mod (n) - y \quad \text{where} \ y = x^d \mod n$$

$$m=c^{39035} \mod (40991) - 49 \quad y=36940 \ mod \ 40991=49$$
For instance, in order to encrypt $m = 65$, we calculate
\[ C = 65^{131} \mod 40991 + 49 \]
\[ C = 17535 + 49 = 176734 \]
To decrypt $c = 17634$, we calculate
\[ M = 17634^{3035} \mod 40991 - 49 = 65 \]

### 3.4 Implementation of $S^2$FRSA Cryptosystem

The following is the code generated to implement $S^2$FRSA which works in all three phases of cryptosystem and tested with sample set of fields.

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

public class SpecialRSAS2 {
    static BigInteger cipher[];
    static int i, j = 0, k, count = 0, c = 0;

    static BigInteger g, e = new BigInteger("1"), o = new BigInteger("1"), n, d = new BigInteger("1"), Eu;
    static BigInteger p, q, u, eval, bi;
    static BigInteger Me, M1, C, tempe, x, y, ua, ewords[];
    static long startTime;
    static long stopTime, ptime;
    static Random rng;
    static int a, tlen, wno[], wnumber = 0, cp = 0;
    static String str = "", wstr = "";
    static String words[], swords[];
```
static String wc="",fans="";

    //static BigInteger b=new BigInteger("1"),U,x;

public static BigInteger[] genKeys(String p1,String q1) {
    rng=new Random();
tlen=8;
do
    {
p=new BigInteger(p1);
p=p.nextProbablePrime();
q=new BigInteger(q1);
q=q.nextProbablePrime();
n = p.multiply(q);
Eu = (p.subtract(o)).multiply(q.subtract(o));
for(i=0;i<20;i++)
    {
tempe=BigInteger.probablePrime(tlen,rng);
if((((Eu.gcd(tempe)).intValue())==1)
    {
e=tempe;
System.out.println("tempe:"+tempe);
brea};
    }

    e=new BigInteger("137");
d = e.modInverse(Eu);
}
while((p.equals(q)));

a=8;

u=new BigInteger("6");

BigInteger[] keys = new BigInteger[5];

keys[0] = n;

keys[1] = Eu;

keys[2] = e;

keys[3] = d;

keys[4]=u;

return keys;

}

public static BigInteger enCrypt(BigInteger e, BigInteger n, BigInteger M) {

returnM.modPow(e, n);

}

public static int deCrypt(BigInteger C, BigInteger n, BigInteger d) {

returnC.modPow(d, n).intValue();

}

public static void main(String[] args)

{

try

{

int size;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
Scanner cin = new Scanner(System.in);

String Msg, PlainTxt="", S, Ci;

int M[], Ptxt[], k;

String ans="";

System.out.print(" Enter the p value (must be five digits) : ");

String p1=br.readLine();

System.out.print(" Enter the q value (must be five digits) : ");

String q1=br.readLine();

System.out.print(" Enter the Plain Text : ");

Msg=br.readLine();

Ptxt = new int[Msg.length()];

System.out.println("nKey Generation Phase...\n");

BigInteger[] key = genKeys(p1,q1);

n = key[0];

Eu = key[1];

e = key[2];

d = key[3];

ua=u.pow(a);

x=ua.modPow(e,n);

//System.out.println("ua:\t"+ua);

System.out.println("n:\t"+n);

System.out.println("Eu:\t"+Eu);

System.out.println("e:\t"+e);

System.out.println("d:\t"+d);
StringTokenizer token = new StringTokenizer(Msg);

while (token.hasMoreTokens())
{
    count++;
    str = token.nextToken();
}

    words=new String[count];
swords=new String[count];
wno=new int[count];
i=0;
token = new StringTokenizer(Msg);
while (token.hasMoreTokens())
{
    str = token.nextToken();
    words[i]=str;
    //System.out.println((i+1)+".\t"+words[i]);
    i++;
}
System.out.println("\n\nEncryption phase..................");

try{
for(i=0;i<count;i++)
{
    Msg=words[i];
}
M = new int[Msg.length()];
cipher=new BigInteger[Msg.length()];
for (k = 0; k < M.length; k += 1)
{
    M[k] = (int) Msg.charAt(k);
    S = "" + M[k];
    Me = new BigInteger(S);
    // Ascci value of each Character
    C = (enCrypt(e, n, Me)); // Encrypted text for each Character based on Ascci value;
    C=C.add(ua);
    fans=fans+C.toString(16)+"h";
    ans=ans+C.toString(16)+"h";
    cipher[k]=C;
}
swords[j]=ans;
System.out.println(ans);
an="";
j++;
}
y=x.modPow(d,n);
}catch(Exception ex){System.out.println(ex.toString());}
System.out.println("\n\n");
for(i=0;i<count;i++)
{
    
    // System.out.println("Enter your Field numbers to decrypt: Total Number of Fields (" + count + ")");
    System.out.println("Enter your Field numbers to decrypt:");
    wnumber=Integer.parseInt(br.readLine());
    if(wnumber>count)
    {
        break;
    }
    else{
        cp++;
        wno[i]=wnumber;
        wstr=wstr+""+wno[i];
    }
}

BigIntegersf=new BigInteger(wstr);
String sd=d.toString();
int dlen=sd.length();
inhlen=dlen/2;
String fsubstr=sd.substring(0,hlen);
String ssubstr=sd.substring(hlen, sd.length());
String fstr=fsubstr.concat(wstr);
String ffstr=fstr.concat(ssubstr);
d=new BigInteger(ffstr);
System.out.println("nPrivate key :"+d);

sd=d.toString();

intrlen=cp+hlen;

fsbstr=sd.substring(0,hlen);

ssbstr=sd.substring(rlen,sd.length());

String ffsbstr=fsbstr.concat(ssbstr);

d=new BigInteger(ffsstr);

System.out.print("n
Enter Your Private Key :");

String uprkey=br.readLine();

int temp;

System.out.println("n
Decryption Phase...........
");

if(ya.equals(y))
{
    for(i=0;i<count;i++)
    {
        temp=wno[i];
        if(temp!=0)
        {
            temp=temp-1;

            StringTokenizer st=new StringTokenizer(swords[temp],"h");

            k=0;

            while(st.hasMoreTokens())
            {
                eval=new BigInteger(st.nextToken(),16);

                46
eval=(eval.subtract(y));
Ptxt[k] = deCrypt(eval, n, d);
k++;
}

PlainText = new String(Ptxt, 0, words[temp].length());
System.out.println("\nWord "+(temp+1)+": "+PlainText+"\n");
}
}

else
{
    System.out.println("\n'r'c1 and 'c1' are not equal");
    return ;
}

} catch (Exception ex) {

System.out.println(ex.toString());

}

}
3.5 Generation of Results for $S^2$FRSA Algorithm

The following figures from 3.1 to 3.7 depict the results generated by executing java code for $S^2$FRSA tested with sample data and all the three phases are verified.

Fig 3.1 $S^2$FRSA Key generation phase for Sample data -1
Fig 3.2 $S^2$FRSA Encrypted Phase for Sample input data
Fig 3.3 $S^2$FRSA Decryption phase showing results
Example-2 for Analysis of S²RSA with Sample Data set for Client-2

Fig 3.4 S²RSA Key generation phase for Sample data -2
Fig 3.5 $S^2$FRSA Encrypted Phase for Sample input data set -2
3.6 Performance Analysis between RSA and $S^2$RSA Crypto System

Graphical Analysis between Basic-RSA and $S^2$RSA Cryptosystem is accomplished by taking 256 bits, 512 bits, 1024 bits and 2048 bits. The key generation time is the same as in RSA schemes, because of $S^2$RSA follows the same mechanism of Basic-RSA. We have given sizes (256,512) as input, then it generates the r, s and n, $\varphi(n)$. We finally compute e and such that $d = e^{-1} \mod \varphi(n)$, where $\gcd(e, \varphi(n)) = 1$. The two Big-Integers $(n,e)$ are the public key and $(x,d)$ is the secret key. On comparison between Basic-RSA and $S^2$RSA Crypto systems, the time performance for key Generation is furnished below:
3.6.1 Key Generation Time Performance

<table>
<thead>
<tr>
<th>Bits(n)</th>
<th>( S^2 \text{FRSA Time(ms)} )</th>
<th>BASIC-RSA Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>512</td>
<td>93</td>
<td>94</td>
</tr>
<tr>
<td>1024</td>
<td>202</td>
<td>203</td>
</tr>
<tr>
<td>2048</td>
<td>2497</td>
<td>2496</td>
</tr>
</tbody>
</table>

Table 3.1 Sample data showing key generation time of \( S^2 \text{FRSA} \)

![Key Generation Process Time](image)

Fig 3.7 Graphical analysis of key generation phase of \( S^2 \text{FRSA} \)

3.6.2 Encryption Time Performance

We have performed the Graphical Analysis between Basic-RSA and \( S^2 \text{FRSA} \) Crypto System by taking 256 bits, 512 bits, 1024 bits and 2048 bits. Here for encryption we have given a public key \(<N, e>\) and a message \( M \in \mathbb{Z}_N \), encrypts \( M \) exactly as in the original RSA, thus \( C = M^e \mod N \). On comparison between RSA and \( S^2 \text{FRSA} \) Crypto systems, the time
Performance for Encryption is furnished below:

<table>
<thead>
<tr>
<th>Bits(n)</th>
<th>$S^2$FRSA Time(ms)</th>
<th>BASIC-RSA Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>512</td>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>1024</td>
<td>46</td>
<td>62</td>
</tr>
<tr>
<td>2048</td>
<td>180</td>
<td>266</td>
</tr>
</tbody>
</table>

*Table 3.2 Sample data showing encryption time of $S^2$FRSA*

![Encryption Process Time](chart)

*Fig 3.8 Graphical analysis of encryption phase of $S^2$FRSA*

### 3.6.3 Decryption Time Performance

Graphical Analysis between Basic-RSA and $S^2$FRSA Cryptosystem is performed by taking 256 bits, 512 bits, 1024 bits and 2048 bits. To decrypt a cipher text C, First consider the input from the user that specified fields or words will be decrypted as

\[
M_i = C_i^d \mod n.
\]

Where ‘i’ specifies the field number or word number.

Observe that this method considers reducing the time expense by decrypting the entire data of the row in database or total text. In this way, the remaining data will be kept in an encrypted format, and the required data only will be decrypted. On
comparison between Basic-RSA and $S^2$RSA Crypto systems, the time performance for Decryption is furnished below:

<table>
<thead>
<tr>
<th>Bits(n)</th>
<th>$S^2$RSA Time(ms)</th>
<th>BASIC-RSA Time(ms)</th>
<th>$S^2$RSA-Total words</th>
<th>Basic-RSA-Total words</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>46</td>
<td>78</td>
<td>3-words</td>
<td>7-words</td>
</tr>
<tr>
<td>512</td>
<td>187</td>
<td>265</td>
<td>4-words</td>
<td>7-words</td>
</tr>
<tr>
<td>1024</td>
<td>687</td>
<td>1779</td>
<td>3-words</td>
<td>7-words</td>
</tr>
<tr>
<td>2048</td>
<td>7410</td>
<td>12948</td>
<td>4-words</td>
<td>7-words</td>
</tr>
</tbody>
</table>

*Table 3.3 Sample data showing encryption time of $S^2$RSA*

![Decryption Process Time](image)

*Fig 3.9 Graphical analysis of decryption phase of $S^2$RSA*

### 3.7 Conclusion

In this chapter we have studied and analyzed about basic RSA algorithm which will normally encrypts and decrypts text with one private and public respectively. In our present work, we proposed related to insurance companies which require only some of the fields of patient's data, and we have proposed Selective Secure Field RSA which will encrypt only the specified fields in Patients/Clients data. $S^2$RSA and Basic RSA is analyzed with sample data and results are analyzed graphically which we have achieved during execution.