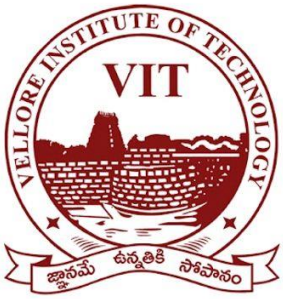
Cryptography Project Report



VIT-AP UNIVERSITY

Submitted to:
Dr. K. Ganesh Reddy,
Professor Senior grade-1,
School of Computer Science and Engineering,
VIT-AP University.

Submitted By:-
Madhav Rav Tripathi-20BCE7109
K. Akhil -20BCE7100
A. Rohith Reddy-18BCE7213
E. Sai Pavan-18BCE7220

CONTENT

Page No:-

Topic: The First Variant of Dependent RSA Encryption Scheme to Secure Text.

Abstract:

The First Variant of Dependent RSA Encryption Scheme to Secure Text

In digital communication, securing confidential messages is a sine qua non condition. All symmetric key algorithms have at least one main problem: the secret keys must be distributed to all communicating parties. The distribution of secret key must be done in a secure channel. A secure channel can be quite expensive to build and quite difficult to manage.

In 1976, the Diffie-Hellman key exchange protocol help in overcome the problem of exchanging secret keys. In this protocol, a new term called "public key" was introduced. It belongs to the class of asymmetric key algorithm or asymmetric cryptography. The RSA cryptosystem, which was published in 1978, was the first encryption algorithm to use Diffie-Hellman-Merkle's idea of asymmetric cryptography. The security of RSA depends on the hardness of factoring a big integer into its two big prime factors: the bigger the integer, the securer the RSA cryptosystem.

The RSA, however, has a problem of being not semantically secure. is a condition where only very little information regarding the original text can be practically pulled out by an adversary from the ciphertext (concealed text). To overcome this problem, Point cheval has proposed a variant of RSA cryptosystem, namely Dependent RSA (DRSA) encryption scheme. The DRSA encryption scheme has several variants. One of those variants is DRSA-1, a scheme that is proven to be semantically secure against the presentation of adaptive chosen-ciphertext attacks. This project shows a step-by-step application on how to secure text with DRSA-1.

Algorithm:-

Key Generation:-

- Using some test generates two large distinct prime numbers, p and q and verify it.
- calculates n = p*q
- calculates $\Phi(n) = (p - 1)(q - 1)$. Bob keeps the value of $\Phi(n)$.
- chooses one random value of e, such that e and $\Phi(n)$ are relatively prime.
- computes d, where $d \equiv e^{-1} (\text{mod } \Phi(n))$.
- Private key component:-d
- Public key component:-(e,n)

Encryption:-

- Alice chooses m, the message to be encrypted.
- Alice obtains the values of e and n, which are Bob's public key, and also obtain the hash function used by Bob.
- Alice chooses a random value of k, where k can be any integer between 1 and n – 1.
- Alice calculates $A = k^e \bmod n$.
- Alice calculates $B = m \times (k + 1)^e \bmod n$.
- Alice calculates H = h(m, k).
- Alice sends the ciphertext C = (A, B, H) to Bob.

Decryption:-

- Bob gets the ciphertext C = (A, B, H) from Alice.
- Bob calculates $k = A^d \bmod n$.
- Bob calculates $m = B / (k + 1)^e \bmod n$.
- Bob checks whether H = h(m, k).

Library Used:-

JavaFx:-

- **BigInteger Class**
- **TextField Class**
- **Label Class**
- **Buttons Class**
- **GridPane**
- **CSS**
- **EventHandler**
- **ActionEvent**
- **Application**
- **Stage**
- **Scene**

Source Code:

```
class Test{
    static BigInteger modulus(BigInteger p, BigInteger q){
        return p.multiply(q);
    }
    static BigInteger Euler(BigInteger p, BigInteger q){
        BigInteger j=new BigInteger ("1");
        return (p.subtract(j)).multiply(q.subtract(j));
    }
}
public class Drsa extends Application {
   @Override
   public void start(Stage stage) {
      Test t = new Test();
Text textKey = new Text("KEY GENERATION");
      textKey.setFont(Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 40));
      textKey.setFill(Color.BROWN);
      textKey.setStrokeWidth(2);
      textKey.setStroke(Color.BLUE);
Text textEnc = new Text("ENCRYPTION");
      textEnc.setFont(Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 40));
      textEnc.setFill(Color.RED);
      textEnc.setStrokeWidth(2);
      textEnc.setStroke(Color.BLUE);
      Text textDec = new Text("DECRYPTION");
      textDec.setFont(Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 40));
      textDec.setFill(Color.BROWN);
      textDec.setStrokeWidth(2);
      textDec.setStroke(Color.BLUE);
      Label lb1 = new Label("  Enter P:");
      lb1.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #ff0000;-fx-stroke: black;-fx-stroke-width: 1");
      TextField txt1 = new TextField();
Label lb2 = new Label("  Enter Q:");
      lb2.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #ff0000;-fx-stroke: black;-fx-stroke-width: 1");
      TextField txt2 = new TextField();
      Label lb3 = new Label();
      Label lb4 = new Label("  N Value:");
```

```
    lb4.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #ff0000;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txt4 = new TextField();
    Label lb5 = new Label("  Φ:");
    TextField txt5 = new TextField();
    lb5.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #ff0000;-fx-stroke: black;-fx-stroke-width: 1");
    Label lb6 = new Label("  E value:");
    lb6.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #ff0000;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txt6 = new TextField();
    Label lb7 = new Label("  D value:");
    lb7.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #ff0000;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txt7 = new TextField();
    Label lb8 = new Label("Public Key(E,N)");
    lb8.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #ff0000;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txte8 = new TextField();
    TextField txtn8 = new TextField();
     Label lb9 = new Label("Private Key(D):");
    lb9.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #ff0000;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txt9 = new TextField();
    Label lbMes = new Label(" Message:");
    lbMes.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtMes = new TextField();
    Label lbA = new Label(" A:");
    lbA.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtA = new TextField();
    Label lbB = new Label(" B:");
    lbB.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtB = new TextField();
    Label lbH = new Label(" H:");
    lbH.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtH = new TextField();
    Button button = new Button("Calculate Key's");
    button.setStyle("-fx-background-color: #ffff00;-fx-text-fill: #ff0000; -fx-border-color: #000000; -fx-border-width: 1px;");
    Button button1 = new Button("ENCRYPT");
    button1.setStyle("-fx-background-color: #ffff00;-fx-text-fill: #ff0000; -fx-border-color: #000000; -fx-border-width: 1px;");
    Label lbDecA = new Label("Enter A:");
    lbDecA.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtDecA = new TextField();
```

```
    Label lbDecB = new Label("Enter B:");
    lbDecB.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtDecB = new TextField();
    Label lbDecH = new Label("Enter H:");
    lbDecH.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtDecH = new TextField();
    Label lbDecD = new Label("Enter D:");
    lbDecD.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtDecD = new TextField();
    Label lbDecM = new Label("Decrypt Message:");
    lbDecM.setStyle("-fx-font: 15px Tahoma;-fx-text-fill: #00ff00;-fx-stroke: black;-fx-stroke-width: 1");
    TextField txtDecM = new TextField();
    Button button2 = new Button("DECRYPT");
   button2.setStyle("-fx-background-color: #ffff00;-fx-text-fill: #ff0000; -fx-border-color: #000000; -fx-border-width: 1px;");
    GridPane gridPane = new GridPane();
    gridPane.setStyle("-fx-background-image: url('https://wallpapercave.com/wp/wp1280676.jpg')");
    gridPane.add(textKey,2,0);
    gridPane.add(lb1, 0, 1);
    gridPane.add(txt1, 1, 1);
    gridPane.add(lb2, 0, 2);
    gridPane.add(txt2, 1, 2);
    gridPane.add(lb4, 0, 3);
    gridPane.add(txt4, 1, 3);
    gridPane.add(lb5, 0, 4);
    gridPane.add(txt5, 1, 4);
    gridPane.add(lb6, 0, 5);
    gridPane.add(txt6, 1, 5);
    gridPane.add(button, 1, 6);
    gridPane.add(lb8, 0, 7);
    gridPane.add(txte8, 1, 7);
    gridPane.add(txtn8, 2, 7);
    gridPane.add(lb9, 0, 8);
    gridPane.add(txt9, 1, 8);
    gridPane.add(textEnc, 1, 9);
    gridPane.add(textDec, 4, 9);
    gridPane.add(lbMes, 0, 10);
    gridPane.add(txtMes, 1, 10);
    gridPane.add(lbA, 0, 11);
```

```
        gridPane.add(txtA, 1, 11);
        gridPane.add(lbB, 0, 12);
        gridPane.add(txtB, 1, 12);
        gridPane.add(lbH, 0, 13);
        gridPane.add(txtH, 1, 13);
        gridPane.add(button1, 1, 14);
        gridPane.add(lbDecA, 3, 10);
        gridPane.add(txtDecA, 4, 10);
        gridPane.add(lbDecB, 3, 11);
        gridPane.add(txtDecB, 4, 11);
        gridPane.add(lbDecH, 3, 12);
        gridPane.add(txtDecH, 4, 12);
        gridPane.add(lbDecD, 3, 13);
        gridPane.add(txtDecD, 4, 13);
       gridPane.add(lbDecM, 3, 14);
        gridPane.add(txtDecM, 4, 14);
        gridPane.add(button2, 4, 15);
        gridPane.setHgap(10);
        gridPane.setVgap(10);
       button.setOnAction(new EventHandler<ActionEvent>() {
         public void handle(ActionEvent event) {
BigInteger p=new BigInteger(txt1.getText());
BigInteger q=new BigInteger(txt2.getText());
BigInteger n=t.modulus(p,q);
          txt4.setText(String.valueOf(n));
          BigInteger phin=t.Euler(p,q);
          txt5.setText(String.valueOf(phin));
          BigInteger E = new BigInteger("48156273336932132972531521463640697933355004948938665004152 3");
          txt6.setText(String.valueOf(E));
          BigInteger d = E.modInverse(phin);
          txt7.setText(String.valueOf(d));
          txte8.setText(String.valueOf(E));
          txtn8.setText(String.valueOf(n));
          txt9.setText(String.valueOf(d));
         }
       });
       button1.setOnAction(new EventHandler<ActionEvent>() {
         public void handle(ActionEvent event) {
```

```
BigInteger i = new BigInteger("1");
        BigInteger hun = new BigInteger("100");
        BigInteger k = new BigInteger("60136443429560609700948117591556143798834200971260967671767 4");
        BigInteger K = k.add(i);
BigInteger m=new BigInteger(txtMes.getText());
        BigInteger e=new BigInteger(txt6.getText());
BigInteger ne=new BigInteger(txt4.getText());
        BigInteger A=k.modPow(e,ne);
        txtA.setText(String.valueOf(A));
        BigInteger B=((m.modPow(i,ne)).multiply(K.modPow(e,ne))).modPow(i,ne);
        txtB.setText(String.valueOf(B));
BigInteger H = (m.modPow(k,hun));
        txtH.setText(String.valueOf(H));
       }
     });
     button2.setOnAction(new EventHandler<ActionEvent>() {
       public void handle(ActionEvent event) {
BigInteger i = new BigInteger("1");
BigInteger a=new BigInteger(txtDecA.getText());
BigInteger b=new BigInteger(txtDecB.getText());
BigInteger n=new BigInteger(txt4.getText());
BigInteger d=new BigInteger(txt7.getText());
BigInteger e=new BigInteger(txt6.getText());
BigInteger k = (a.modPow(d,n));
        BigInteger K = k.add(i);
BigInteger Einv = e.negate();
BigInteger M = ((b.modPow(i,n)).multiply(K.modPow(Einv,n))).modPow(i,n);
txtDecM.setText(String.valueOf(M));
       }
     });
      Scene scene = new Scene(gridPane, 200, 100, Color.BEIGE);
      stage.setTitle("DRSA CRYPTOSYSTEM");
      stage.setScene(scene);
      stage.show();
   }
   public static void main(String args[]){
      launch(args);
   }}
```

Code to check Prime(Verification):

```
import java.math.*;
import java.util.*;
public class Main{
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        System.out.println("RSA ENCRYPTOR/DECRYPTOR");
        System.out.println("********");

        System.out.print("Enter the Value of P:-");
        BigInteger p = sc.nextBigInteger();

        System.out.print("Enter the Value of Q:-");
        BigInteger q = sc.nextBigInteger();

        boolean check_p = p.isProbablePrime(1);
        boolean check_q = q.isProbablePrime(1);

        while(((check_p==false) || (check_q==false))){
            System.out.println("Warning!! Inputs are not prime");
            System.out.println();
            System.out.print("Enter the Value of P:-");
            p = sc.nextBigInteger();
            System.out.print("Enter the Value of Q:-");
            q = sc.nextBigInteger();
            check_p = p.isProbablePrime(1);
            check_q = q.isProbablePrime(1);
        }
        System.out.println();
        System.out.println("P adn Q are prime Number");
    }
}
```

Outputs:

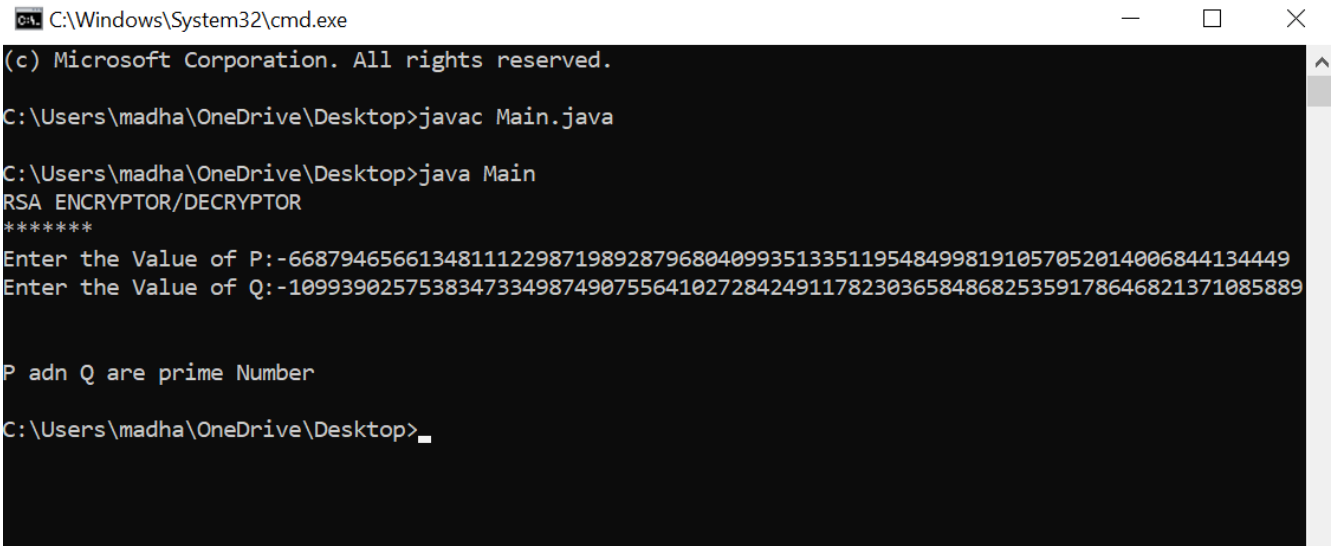
Compile and run command of main program:

KEY GENERATION:-

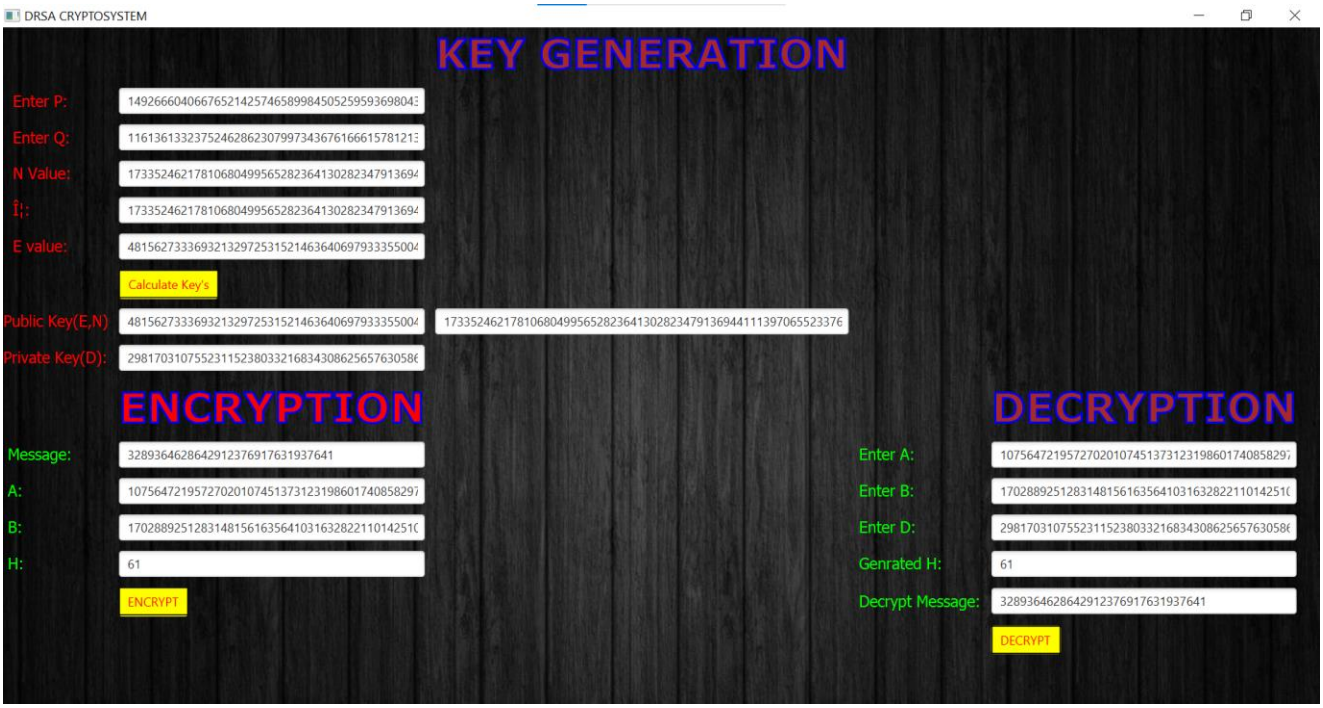
ENCRYPTION:-

DECRYPTION:-

Sub program Output:-

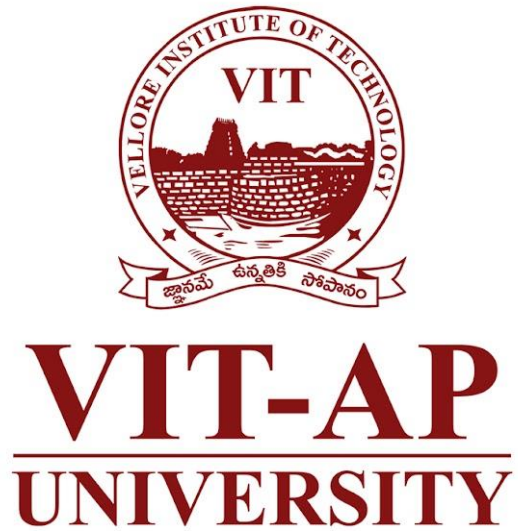


Overall Output:



Conclusion:-

- We have presented and developed the Program on to use the first variant of Dependent RSA (DRSA-1) encryption scheme to secure a simple text.
- The DRSA-1 algorithm has been implemented in the Java.
- In order to preserve higher security, we using longer digits of n, for example 512-digit or 1024 digit.
- To preserve integrity, we passed our message through hash function. This create a message digest and we verify it in receiver end.



Cryptography Project Review 1

Submitted to:
Dr. K. Ganesh Reddy,
Professor Senior grade-1,
School of Computer Science and Engineering,
VIT-AP University.

The First Variant of Dependent RSA Encryption Scheme to Secure Text

Submitted By:-

Madhav Rav Tripathi-20BCE7109

K. Akhil -20BCE7100

A. Rohith Reddy-18BCE7213

E. Sai Pavan-18BCE7220

VIT-AP UNIVERSITY

The First Variant of Dependent RSA Encryption Scheme to Secure Text

- In digital communication, securing confidential messages is a sine qua non condition. All symmetric key algorithms have at least one main problem: the secret keys must be distributed to all communicating parties. The distribution of secret key must be done in a secure channel. A secure channel can be quite expensive to build and quite difficult to manage.

- In 1976, the Diffie-Hellman key exchange protocol help in overcome the problem of exchanging secret keys. In this protocol, a new term called “public key” was introduced. It belongs to the class of asymmetric key algorithm or asymmetric cryptography. The RSA cryptosystem, which was published in 1978, was the first encryption algorithm to use Diffie-Hellman-Merkle’s idea of asymmetric cryptography. The security of RSA depends on the hardness of factoring a big integer into its two big prime factors: the bigger the integer, the securer the RSA cryptosystem.

- The RSA, however, has a problem of being not semantically secure. is a condition where only very little information regarding the original text can be practically pulled out by an adversary from the ciphertext (concealed text). To overcome this problem, Point cheval has proposed a variant of RSA cryptosystem, namely Dependent RSA (DRSA) encryption scheme. The DRSA encryption scheme has several variants. One of those variants is DRSA-1, a scheme that is proven to be semantically secure against the presentation of adaptive chosen-ciphertext attacks. This project shows a step-by-step application on how to secure text with DRSA-1.

Software Used:Java

- Data Structure(Array)
- File Handling
- Big Integer Class(take long digit input)
- We will even try to use Javafx as well(to add some more graphics)

VIT-AP UNIVERSITY

Contribution:

- Coding :

  Coding will be done by each and every member as per requirement.

- Documentation:

  Reports and all other documentation will be done by Pavan and Rohith.

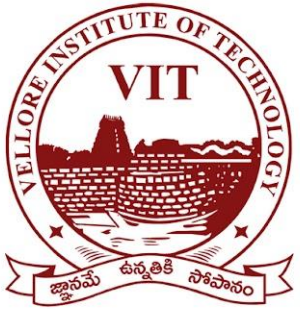
- Research Work:

  Madhav and Akhil did the required research on RSA and have briefed and explained the group.

VIT-AP UNIVERSITY

Thank You

VIT-AP UNIVERSITY



VIT-AP UNIVERSITY

Cryptography Project Review 2

Submitted to:
Dr. K. Ganesh Reddy,
Professor Senior grade-1,
School of Computer Science and Engineering,
VIT-AP University.

The First Variant of Dependent RSA Encryption Scheme to Secure Text

Submitted By:-

Madhav Rav Tripathi-20BCE7109

K. Akhil -20BCE7100

A. Rohith Reddy-18BCE7213

E. Sai Pavan-18BCE7220

VIT-AP UNIVERSITY

The First Variant of Dependent RSA Encryption Scheme to Secure Text

- In digital communication, securing confidential messages is a sine qua non condition. All symmetric key algorithms have at least one main problem: the secret keys must be distributed to all communicating parties. The distribution of secret key must be done in a secure channel. A secure channel can be quite expensive to build and quite difficult to manage.

- In 1976, the Diffie-Hellman key exchange protocol help in overcome the problem of exchanging secret keys. In this protocol, a new term called “public key” was introduced. It belongs to the class of asymmetric key algorithm or asymmetric cryptography. The RSA cryptosystem, which was published in 1978, was the first encryption algorithm to use Diffie-Hellman-Merkle’s idea of asymmetric cryptography. The security of RSA depends on the hardness of factoring a big integer into its two big prime factors: the bigger the integer, the securer the RSA cryptosystem.

- The RSA, however, has a problem of being not semantically secure. It is a condition where only very little information regarding the original text can be practically pulled out by an adversary from the ciphertext (concealed text). To overcome this problem, Point cheval has proposed a variant of RSA cryptosystem, namely Dependent RSA (DRSA) encryption scheme. The DRSA encryption scheme has several variants. One of those variants is DRSA-1, a scheme that is proven to be semantically secure against the presentation of adaptive chosen-ciphertext attacks. This project shows a step-by-step application on how to secure text with DRSA-1.

VIT-AP UNIVERSITY

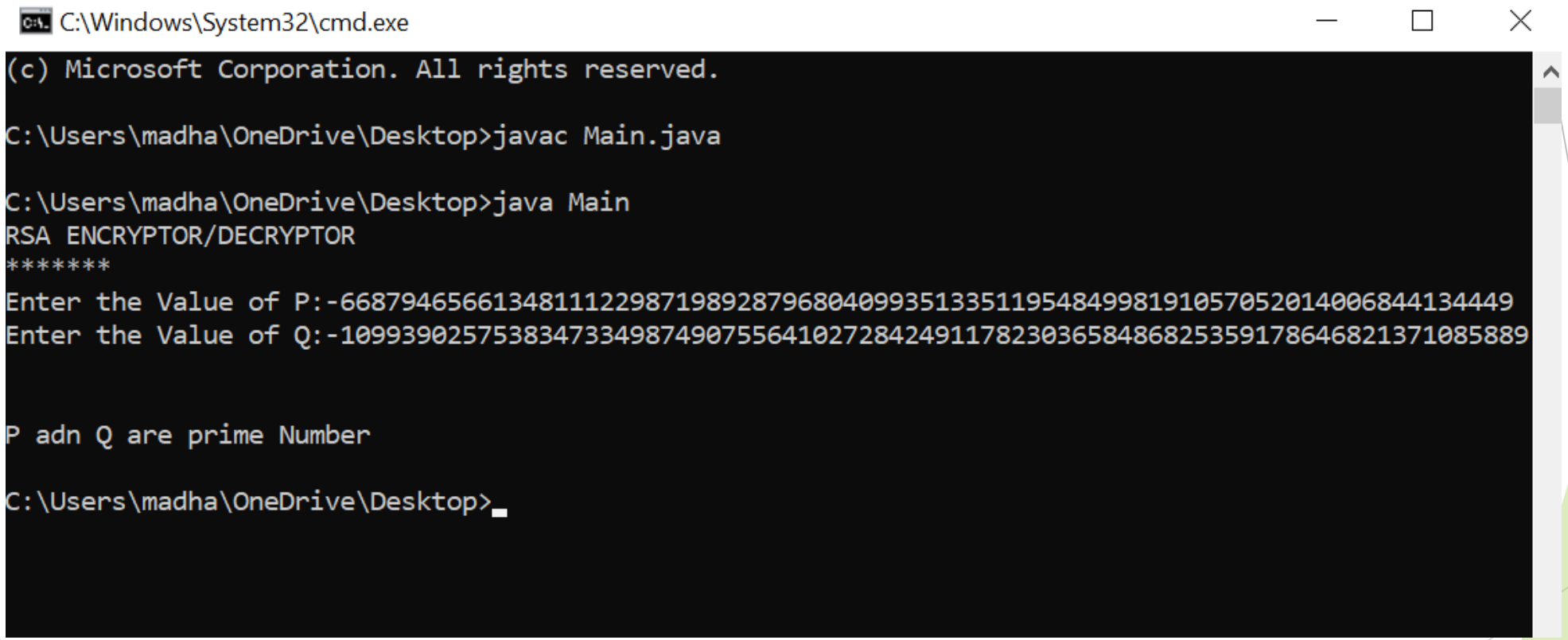
Prime Number(Generation):-

No of bits in prime is 256
Random n-bit Prime (p): 66879465661348111229871989287968040993513351195484998191057052014006844134449
Random n-bit Prime (q): 109939025753834733498749075564102728424911782303658486825359178646821371085889

No of bits in prime is 1024
Random N-bit Prime (p): 14926660406676521425746589984505259593698043308528112047243863356010910984506208081319567489713652594984018496531250529886994872297764946902308436155041298948606020791758054045408114058735386223444557752047687254367648616789244387230870502677846112126122432249532834663038348638666362887877283844908777012330 3
Random N-bit Prime (q): 11613613323752462862307997343676166615781213580255442213384399716278215827708188540430994158743163224360474004390260851035079396569070805436204141716645377206469931168305351122258807934047024235765278566582937247825531441295648260124631056178986340098086793666788683120626019654875802245983332214723863553333

Result

Prime Number Verification:-



Key Generation:-

- Using some test generates two large distinct prime numbers p and q and verify.
- To calculates n = p*q
- To calculates $\Phi(n) = (p - 1)(q - 1)$. Bob keeps the value of $\Phi(n)$.
- Choose one random value of e, such that e and $\Phi(n)$ are relatively prime.
- computes d, where $d \equiv e^{-1} (\text{mod } \Phi(n))$.
- Private key component:-d
- Public key component:-(e,n)

VIT-AP UNIVERSITY

Key Generation:-

Encryption:-

- Alice chooses m, the message to be encrypted.
- Alice obtains the values of e and n, which are Bob's public key, and also obtain the hash function used by Bob.
- Alice chooses a random value of k, where k can be any integer between 1 and n – 1.
- Alice calculates $A = k^e \bmod n$.
- Alice calculates $B = m \times (k + 1)^e \bmod n$.
- Alice calculates H = h(m, k).
- Alice sends the ciphertext C = (A, B, H) to Bob.

VIT-AP UNIVERSITY

Encryption:-


ENCRYPTION
Message:
3289364628642912376917631937641
A:
107564721957270201074513731231986017408582 97
B:
17028892512831481561635641031632822110142510
H:
61
ENCRYPT

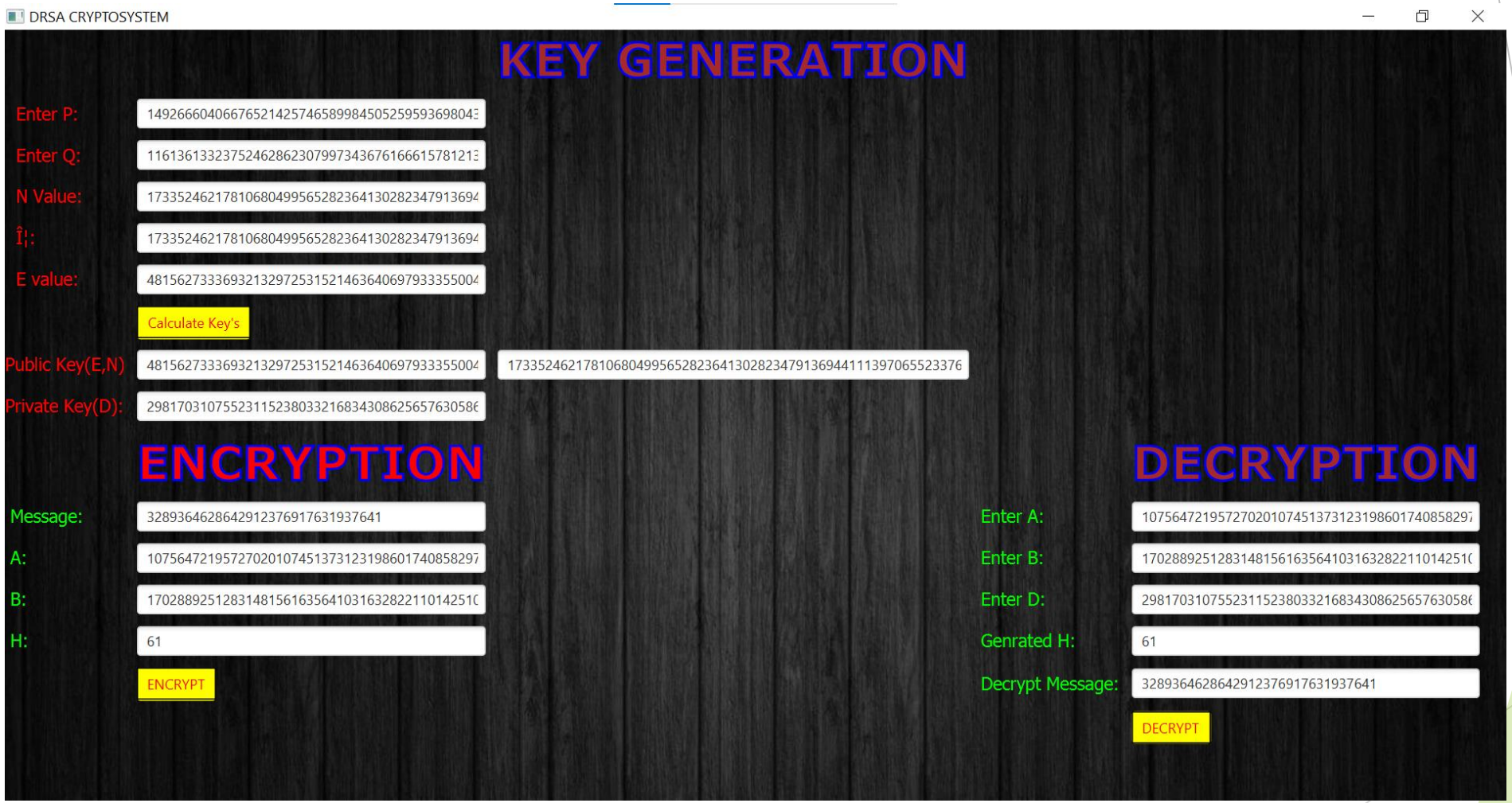
Decryption:-

- Bob gets the ciphertext C = (A, B, H) from Alice.
- Bob calculates $k = A^d \bmod n$.
- Bob calculates $m = B / (k + 1)^e \bmod n$.
- Bob checks whether H = h(m, k).

Decryption:-


DECRYPTION
Enter A:
1075647219572702010745137312319860174085829
Enter B:
17028892512831481561635641031632822110142510
Enter D:
29817031075523115238033216834308625657630586
Genrated H:
61
Decrypt Message:
32893646286429123769176319376 41
DECRYPT

Overall:-


DRSA CRYPTOSYSTEM
KEY GENERATION
Enter P:
Enter Q:
N Value:
Î¦:
E value:
Calculate Key's
Public Key(E,N)
Private Key(D):
ENCRYPTION
Message:
A:
B:
H:
ENCRYPT
DECRYPTION
Enter A:
Enter B:
Enter D:
Genrated H:
Decrypt Message:
DECRYPT

Software Used:Java

- Big Integer Class(take long digit input)
- Javafx(Graphics)

VIT-AP UNIVERSITY

Conclusion:-

- We have presented and developed the Program on to use the first variant of Dependent RSA (DRSA-1) encryption scheme to secure a simple text.
- The DRSA-1 algorithm has been implemented in the Java.
- In order to preserve higher security, we using longer digits of n, for example 512-digit or 1024 digit.
- To preserve integrity, we passed our message through hash function. This create a message digest and we verify it in receiver end.

Thank You

VIT-AP UNIVERSITY