

## Faculty of Computer Science

Semester 71 – Spring 2020

**Computer Security (CS401)**

**Final Course Project**

**Project Title:**

**S-DES**

**RC4**

**Diffie-Hellman**

**RSA**

**ElGamal**

**Done by:**

Name	ID
<b>Yahia Mahmoud Abdelrahman Basyouny</b>	<b>173269</b>

**Submitted to:**

**Dr. Ehab Emam**

**Eng. Nourhan Khalaf**

## **Problem Statement:**

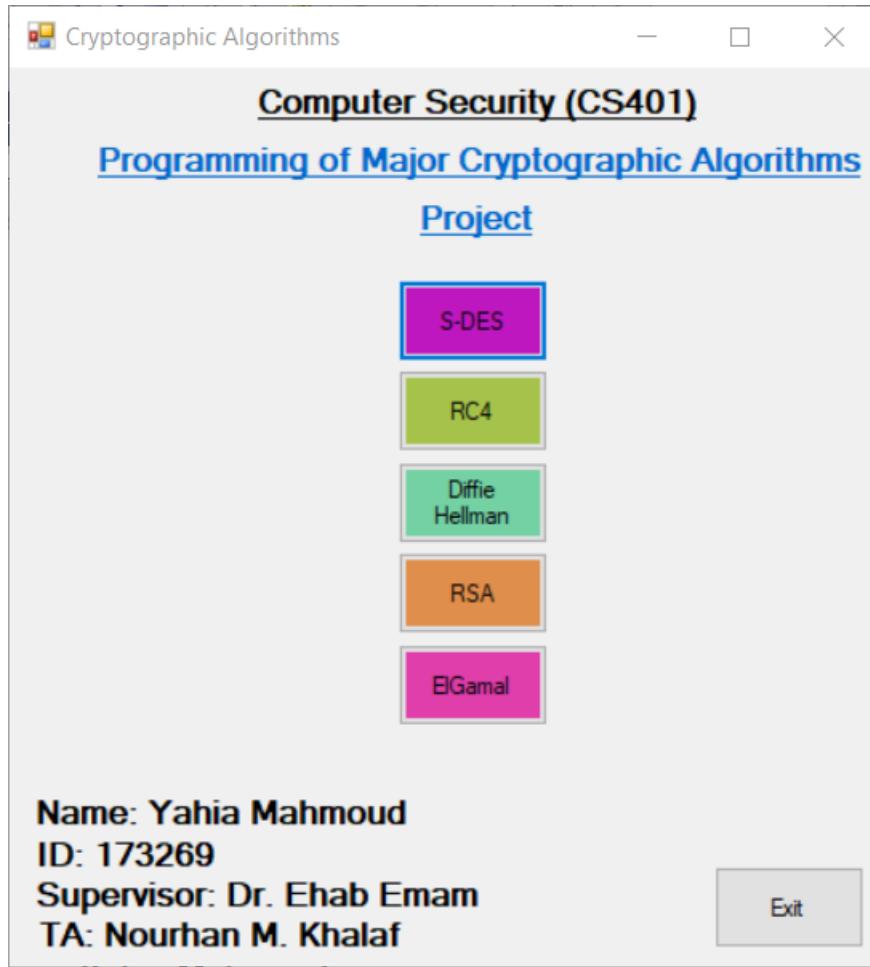
Network security consist of strategies and practices implemented to prevent and stop unapproved or unauthorized access, DOF (denial of network service), misuse, and resources. This days, communication between people depend on computer network that offers valuable services and supports, where we can send and receive any form of data like texts, audio, video, and more all over the world in no time. . So the scope of network security is to secure the data rapidly to prevent any attack. Goals of Network Security are confidentiality, integrity and availability. Cryptography is the study of encryption principles and methods for data. So we use a cipher which is an algorithm for transiting data (plain text) into encrypted data (cypher text) by using a key which is info used in cipher known by sender and receiver. Encrypt is converting plain text into cypher text and decrypt is the vice versa. So to secure the data, we must encrypt data then transmit them over the network and the receiver is the only one who can decrypt the data by a specific key. Our system will be a package contains multiple cypher algorithms with a GUI. The system will be a package software contains the cipher algorithms (S-DES, RC4, Diffie-Helman, RSA, and ELGAMAL) with a GUI. The software used to implement the algorithm with GUI is Microsoft visual studio and the language is C#. It will be a multiple window forms that contains text field and labels and buttons. The first algorithm is the simplified encryption standard that is called S-DES, S-DES is a simplified algorithm for DES algorithm, where it uses the same implementation steps of DES except number of bits is different (10 bits of key) and number of rounds are 2. S-DES and DES are a symmetric block cipher. The second algorithm is RC4, RC4 is a stream cipher algorithm and uses a variable length key from 1 to 256 bytes. A key information is pseudorandom bit generator that is going to create a stream of 8 bit number that cannot be predicted without information on input key, The produced output of the generator is known as key stream, by using XOR operation, the key stream is combined each byte with the plaintext stream cipher at a time. The third algorithm is Diffie-Hellman, Diffie-Hellman key exchange algorithm is a technique for safely exchanging cryptographic keys over a public communications channel and get the secret key using parameters. But in reality Keys are not really exchange, they are derived. So all user agree on some parameters ( $q$  &  $a$ ), where  $q$  is a large prime integer and  $a$  is a primitive root mod  $q$ . Then each user generate their keys. The fourth algorithm is RSA, RSA algorithm is asymmetric cryptography algorithm. The meaning of Asymmetric is that the algorithm works on two different keys Public Key and Private Key. Since the Public Key is known and given to all the users, where the Private Key is private. For example a user A uses a public key of user B, and user A want to send data for user B, so User A encrypts the data by user B public key then user B receives this data and decrypt it. By this way nobody else can decrypt the data. The public key consists of two numbers where the first number ( $n$ ) is based on the multiplication of two large prime numbers ( $P$  &  $Q$ ) and the second number is  $e$  where  $1 < e < \Phi(n)$  and  $\Phi(n) = (P-1)(Q-1)$ . And private key is also calculated from the same two

prime numbers. The encryption strength increases and depend on the size of the key, so by increasing the key size the strength of encryption increases. The fifth algorithm is ElGamal, ElGamal algorithm is a public key cryptosystem. It is asymmetric cryptography algorithm, where it uses a public key cryptosystem related to Diffie-Helman. Its security based on difficulty of calculating the discrete logarithms as in Diffie-Helman. In ElGamal each user generates their key by choosing their secret key and compute their public key. For example, user A generates public key and private key then user B encrypts data using user A public key then user A decrypts the data that is sent by user B.

**Inputs:**

Choose one of the cryptographic algorithms to run by clicking on one of them button.

**Note:** Exit button is for closing the program.



*Figure 1: Choose one of the cryptographic algorithms*

## S-DES:

The inputs are a Text as plain text or cipher text in a text box and a Key in binary of 10 bits Key in a text box too. After inserting the text click on Encrypt check box if the text is a plaintext or Decrypt check box if the text is a cipher text and choose the type of the text (Text, Binary, or Hex). Then click on En/Decrypt button to get the result in the “Result” text box.

The screenshot shows the 'S-DES Algorithm' application window. It has a title bar 'S DES Form'. The main area contains a 'Text: (Plain Text or Cipher Text)' label and a text input box containing 'Hitler attacked russia.'. Below it are three checkboxes: 'Encrypt' (checked), 'Decrypt' (unchecked), 'Text' (checked), 'Binary' (unchecked), and 'Hex' (unchecked). A 'Key (In Binary[10 bits]):' label is followed by a text input box containing '1010011010'. A 'Result:' label is followed by an empty text input box. At the bottom is a 'En/Decrypt' button with three checkboxes below it: 'Text' (unchecked), 'Binary' (unchecked), and 'Hex' (unchecked).

Figure 2: Encrypting

The screenshot shows the 'S-DES Algorithm' application window. It has a title bar 'S DES Form'. The main area contains a 'Text: (Plain Text or Cipher Text)' label and a text input box containing 'ZÆwlliß(ww(çÉißøèèÆ(i'. Below it are three checkboxes: 'Encrypt' (unchecked), 'Decrypt' (checked), 'Text' (checked), 'Binary' (unchecked), and 'Hex' (unchecked). A 'Key (In Binary[10 bits]):' label is followed by a text input box containing '1010011010'. A 'Result:' label is followed by an empty text input box. At the bottom is a 'En/Decrypt' button with three checkboxes below it: 'Text' (unchecked), 'Binary' (unchecked), and 'Hex' (unchecked).

Figure 3: Decrypting

## RC4:

The inputs are a Text as plain text or cipher text in a text box and a Key in a text box too. After inserting the text, click on Encrypt check box if the text is a plaintext or Decrypt check box if the text is a cipher text and choose the type of the text (Text, Binary, or Hex) and choose the type of the key (Binary or Hex). Then click on En/Decrypt button to get the result in the “Result” text box.

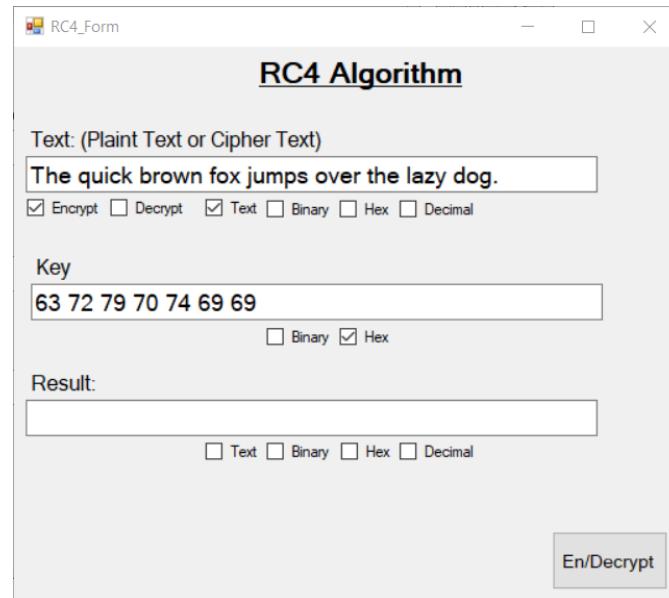


Figure 4: Encrypting

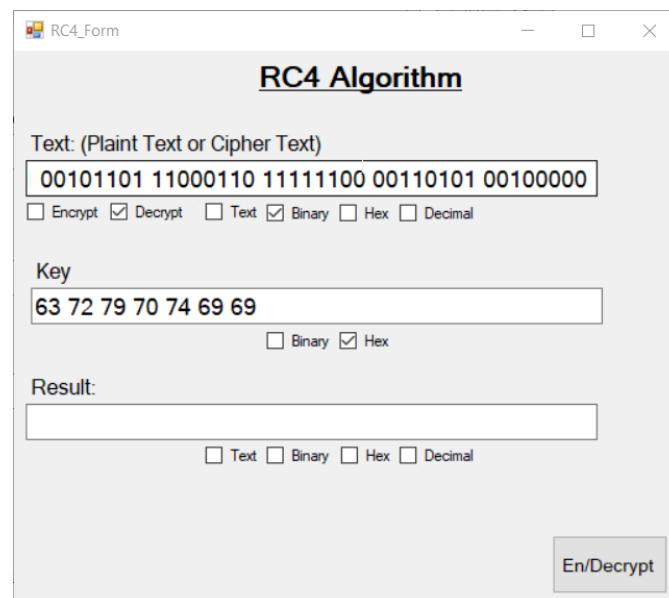


Figure 5: Decrypting

### **Diffie-Hellman:**

The inputs are a Large Prime integer (q), a Primitive root integer (a), a Secret Key for user A, and a Secret Key for user B where each one of them is inserted in its own text box. After inserting click on “Generate Keys” button to get the results.

The screenshot shows a Microsoft Word document window titled "Diffie\_Hellman\_Form". The main content is a form titled "Diffie-Hellman Keys Generator Algorithm". The form contains four input fields for user inputs:

- Large Prime Integer (q): 353
- Primitive Root (a): 3
- Secret Key of User A ( $X_A < q$ ): 97
- Secret Key of User B ( $X_B < q$ ): 233

Below these input fields is a section titled "Results" which contains three empty text boxes for the generated keys:

- Public Key of User A ( $Y_A$ )
- Public Key of User B ( $Y_B$ )
- Shared Private Key of User A and B ( $K_{AB}$ )

In the bottom right corner of the form area, there is a blue-outlined button labeled "Generate Keys".

Figure 6: Inserting Data

## RSA:

The inputs are a Text as plain text or cipher text, a large prime integer (P), a large prime integer (Q) where each one of them is inserted in its own text box. After inserting the text, click on Encrypt check box if the text is a plaintext or Decrypt check box if the text is a cipher text and choose the type of the text (Text, Binary, Hex, or Decimal). Then click on En/Decrypt button to get the result in the “Result” text box.

The screenshot shows the RSA Algorithm application window titled "RSA Algorithm". In the "Text: (Plain Text or Cipher Text)" section, the input "We found golden statues and stones, so find an hones" is displayed. Below it, there are checkboxes for "Encrypt" (checked) and "Decrypt" (unchecked), and radio buttons for "Text" (checked), "Binary", "Hex", and "Decimal". Under "Large Prime Number (P)", the value "97" is entered. Under "Large Prime Number (Q)", the value "89" is entered. At the bottom, there is a "Result:" label followed by an empty text box, and a row of radio buttons for "Text", "Binary", "Hex", and "Decimal". A central "En/Decrypt" button is located at the bottom right.

Figure 7: Encrypting

The screenshot shows the RSA Algorithm application window titled "RSA Algorithm". In the "Text: (Plain Text or Cipher Text)" section, the input "We found golden statues and stones, so find an hones" is displayed, which appears to be encrypted text. Below it, there are checkboxes for "Encrypt" (unchecked) and "Decrypt" (checked), and radio buttons for "Text" (checked), "Binary", "Hex", and "Decimal". Under "Large Prime Number (P)", the value "97" is entered. Under "Large Prime Number (Q)", the value "89" is entered. At the bottom, there is a "Result:" label followed by an empty text box, and a row of radio buttons for "Text", "Binary", "Hex", and "Decimal". A central "En/Decrypt" button is located at the bottom right.

Figure 8: Decrypting

## ElGamal:

For Encryption:

The inputs are plain text, a large prime integer ( $q$ ), a primitive root ( $a$ ), and a Secret Key for user A where each one of them is inserted in its own text box. After inserting the plain text choose the type of the text (Text, Binary, Hex, or Decimal). Then click on “Encrypt” button to get the result in the “CipherText” text box.

The screenshot shows a Windows application window titled "ELGamal Algorithm". The interface is divided into two main sections: "PlainText" on the left and "CipherText (C1, C2)" on the right. In the "PlainText" section, the input "Computer Security is Important." is displayed in a text box, with a checked checkbox for "Text" and unchecked boxes for "Binary", "Hex", and "Decimal". Below it, the "Large Prime Number (q)" is set to 1613, and the "Primitive Root (a)" is set to 1119. The "Secret Key of User A (XA < q - 1)" is set to 1229. In the "CipherText (C1, C2)" section, there is an empty text box for the cipher text, with an unchecked checkbox for "Text" and checked checkboxes for "Binary", "Hex", and "Decimal". Below it, the "Large Prime Number (q)" is set to 1, and the "Secret Key of User A (XA < q - 1)" is set to 1. At the bottom center is an "Encrypt" button, and at the bottom right is a "Decrypt" button.

Figure 9: Encrypting

For Decryption:

The inputs are cipher text, a large prime integer ( $q$ ), and a Secret Key of user A where each one of them is inserted in its own text box. After inserting the cipher text choose the type of the cipher text (Text, Binary, Hex, or Decimal). Then click on “Decrypt” button to get the result in the “PlainText” text box.

The screenshot shows the 'ELGamal Algorithm' application window. It has two main sections: 'Encrypt' (left) and 'Decrypt' (right).

**Encrypt Section (Left):**

- PlainText:** Text input field.
- CipherText (C1, C2):** Text input field containing the value **۱۶۱۳**.
- Large Prime Number (q):** Text input field containing the value **1613**.
- Primitive Root (a):** Text input field.
- Secret Key of User A (XA < q - 1):** Text input field.
- CipherText (C1, C2):** Text input field.

Each input field has a corresponding row of checkboxes for Text, Binary, Hex, and Decimal formats. The 'Text' checkbox is checked for the CipherText (C1, C2) field.

**Decrypt Section (Right):**

- PlainText:** Text input field.
- CipherText (C1, C2):** Text input field containing the value **۱۲۲۹**.
- Large Prime Number (q):** Text input field containing the value **1613**.
- Secret Key of User A (XA < q - 1):** Text input field containing the value **1229**.
- CipherText (C1, C2):** Text input field.

Each input field has a corresponding row of checkboxes for Text, Binary, Hex, and Decimal formats. The 'Text' checkbox is checked for the CipherText (C1, C2) field.

**Buttons:**

- Encrypt** button (in the Encrypt section).
- Decrypt** button (in the Decrypt section).

Figure 10: Decrypting

## Implementation:

### Main Form:

### Screen Shots:

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
Main Form.cs [Design]
Main Form.cs < Main Form.cs [Design]
Cryptographic_Algorithms.Main
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Cryptographic_Algorithms
{
    public partial class Main : Form
    {
        Timer T = new Timer();
        public Main()
        {
            InitializeComponent();

            T.Start();
            T.Tick += new EventHandler(T_Tick);
            T.Interval = 500;
        }

        void T_Tick(object sender, EventArgs e)
        {
            Random R = new Random();

            S_DES_Button.BackColor = Color.FromArgb(R.Next(255), R.Next(255), R.Next(255));
            RC4_Button.BackColor = Color.FromArgb(R.Next(255), R.Next(255), R.Next(255));
            Diffie_Hellman_Button.BackColor = Color.FromArgb(R.Next(255), R.Next(255), R.Next(255));
            RSA_Button.BackColor = Color.FromArgb(R.Next(255), R.Next(255), R.Next(255));
            ElGamal_Button.BackColor = Color.FromArgb(R.Next(255), R.Next(255), R.Next(255));
        }

        private void S_DES_Button_Click(object sender, EventArgs e)
        {
            S_DES_Form S_DESForm = new S_DES_Form();
            S_DESForm.Show();
            this.Visible = false;
        }
    }
}
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
Main Form.cs [Design]
Main Form.cs < Main Form.cs [Design]
Cryptographic_Algorithms.Main
private void S_DES_Button_Click(object sender, EventArgs e)
{
    S_DES_Form S_DESForm = new S_DES_Form();
    S_DESForm.Show();
    this.Visible = false;
    S_DESForm.FormClosed += new FormClosedEventHandler(S_DESForm_FormClosed);
}

void S_DESForm_FormClosed(object sender, FormClosedEventArgs e)
{
    this.Visible = true;
}

private void RC4_Button_Click(object sender, EventArgs e)
{
    RC4_Form RC4Form = new RC4_Form();
    RC4Form.Show();
    this.Visible = false;
    RC4Form.FormClosed += new FormClosedEventHandler(RC4Form_FormClosed);
}

void RC4Form_FormClosed(object sender, FormClosedEventArgs e)
{
    this.Visible = true;
}

private void Diffie_Hellman_Button_Click(object sender, EventArgs e)
{
    Diffie_Hellman_Form DHForm = new Diffie_Hellman_Form();
    DHForm.Show();
    this.Visible = false;
    DHForm.FormClosed += new FormClosedEventHandler(DHForm_FormClosed);
}

void DHForm_FormClosed(object sender, FormClosedEventArgs e)
{
    this.Visible = true;
}

private void RSA_Button_Click(object sender, EventArgs e)
{
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
Main Form.cs X Main Form.cs [Design]
Cryptographic_Algorithms.Main RSA_Button_Click(object sender, EventArgs e)
{
    RSA_Form RSAForm = new RSA_Form();
    RSAForm.Show();
    this.Visible = false;
    RSAForm.FormClosed += new FormClosedEventHandler(RSAForm_FormClosed);
}

void RSAForm_FormClosed(object sender, FormClosedEventArgs e)
{
    this.Visible = true;
}

private void ElGmail_Button_Click(object sender, EventArgs e)
{
    ELGmail_Form ELGmailForm = new ELGmail_Form();
    ELGmailForm.Show();
    this.Visible = false;
    ELGmailForm.FormClosed += new FormClosedEventHandler(ELGmailForm_FormClosed);
}

void ELGmailForm_FormClosed(object sender, FormClosedEventArgs e)
{
    this.Visible = true;
}

private void Exit_Button_Click(object sender, EventArgs e)
{
    this.Close();
}
}

100 % Error List Ready Ln 76 Col 9 Ch 9 INS
```

## **S-DES:**

The inputs are a Text as plain text or cipher text in a text box and a Key in binary of 10 bits Key in a text box too. After inserting the text click on Encrypt check box if the text is a plaintext or Decrypt check box if the text is a cipher text and choose the type of the text (Text, Binary, or Hex). Then click on En/Decrypt button to get the result in the “Result” text box.

### **Steps:**

#### **Step 1: Keys Generation**

Generate 2 Keys from the inserted key by using P10 array on both keys. Then shift the first key to the left by 1 and the second key to the left by 3. Then using P8 array on both keys.

#### **Step 2: Initial Permutation**

Perform Initial Permutation function on Plain Text, if you are Encrypting. Or on Cipher Text, if you are decrypting.

#### **Step 3: Encryption or Decryption**

Perform Encryption or Decryption Function according to the inserted input. If you are encrypting, you will perform the encryption on the IP plain text. If you are decrypting, you will perform the decryption on the IP cipher text

#### **In Encryption function:**

Round 1: Use EP array on IP plain text. Then perform Xor operation on the result with the first key. Then use S Boxes on the result, then use P4 on the result. Then the last result swap it with the right part of the IP plain text.

Round 2: Round 2 take the result of Round 1 and do the same steps as Round 1 but use the second key instead of the first one and do not swap the last result in the last step.

#### **In Decryption function:**

Decryption function do the same 2 Round but in round 1 use the second key and in Round 2 use the first key.

#### **Step 4: Final Permutation**

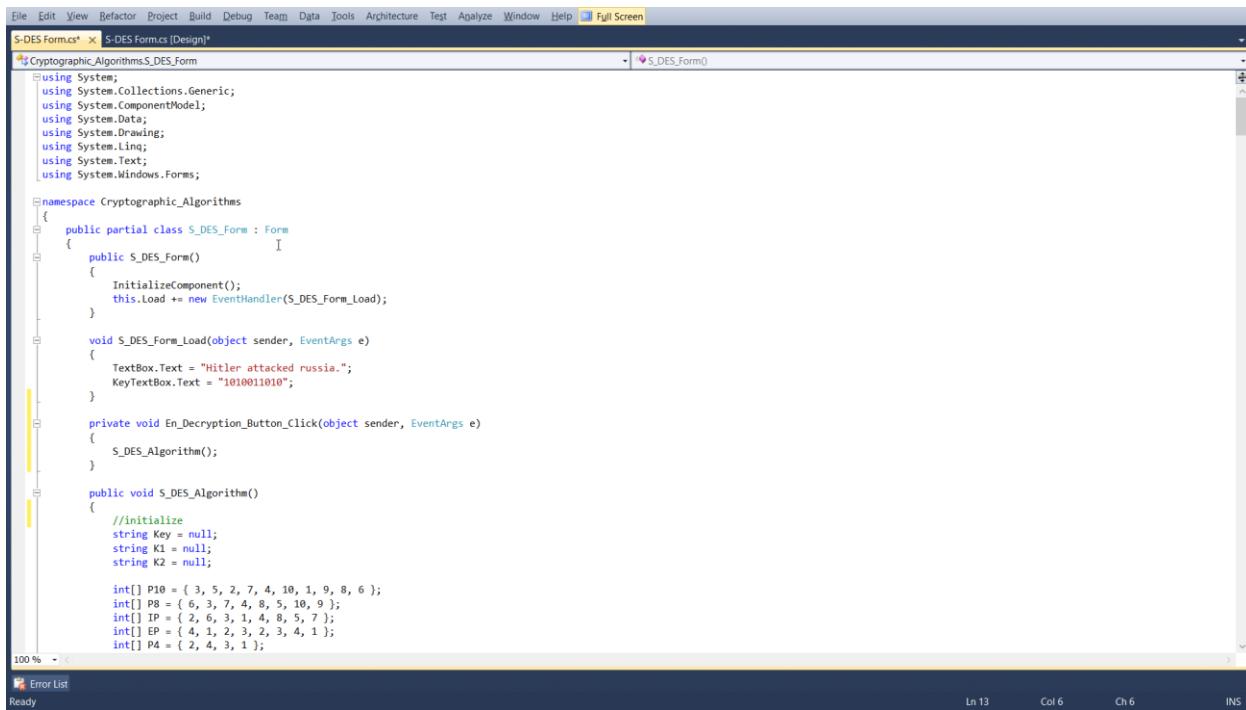
In Final Permutation, initial permutation inverse is created using IP array. Then by using the initial permutation inverse (IP-1), the result will be the plain text if you are decrypting or will be the cipher text if you are encrypting.

#### **Step 5: Result**

Then add result of the Final Permutation to the “Result” text box.

Functions used: Converting Function as Converting Plaintext or Ciphertext into String, Binary, Hexadecimal, and Decimal.

### Screen Shots:



```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
S-DES Form.cs [Design]
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

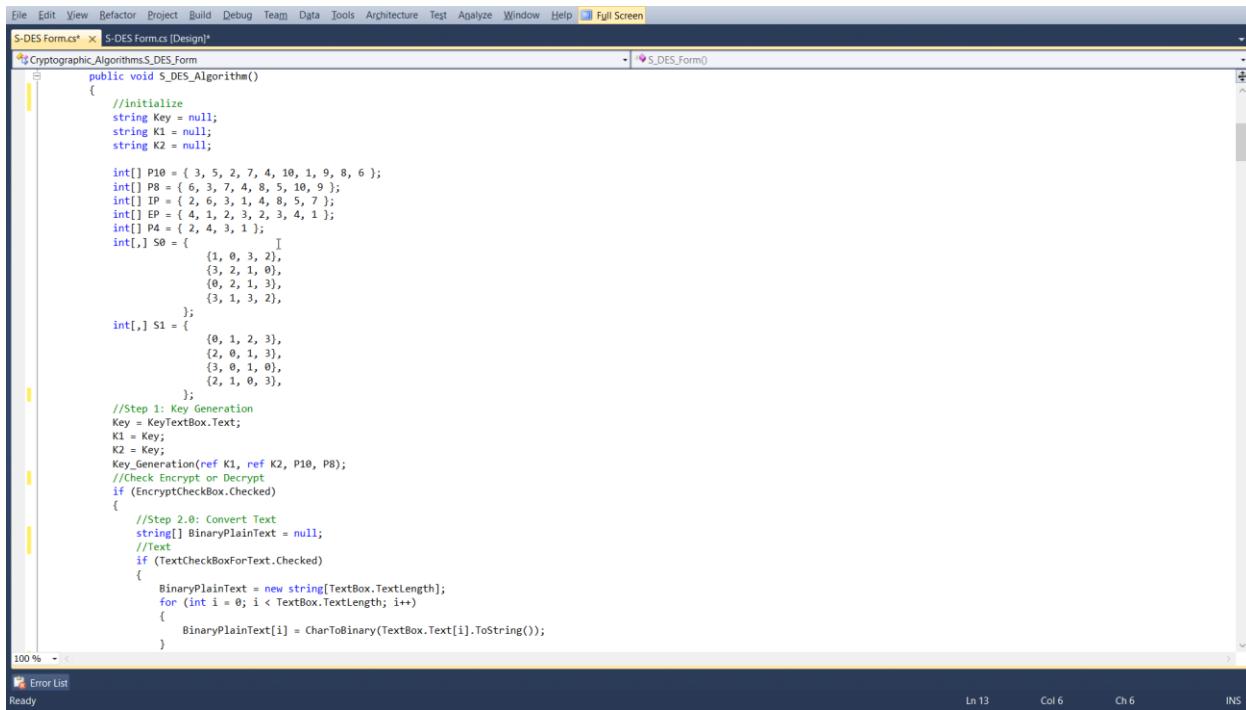
namespace Cryptographic_Algorithms
{
    public partial class S_DES_Form : Form
    {
        public S_DES_Form()
        {
            InitializeComponent();
            this.Load += new EventHandler(S_DES_Form_Load);
        }

        void S_DES_Form_Load(object sender, EventArgs e)
        {
            TextBox.Text = "Hitler attacked russia.";
            KeyTextBox.Text = "1010011010";
        }

        private void En_Decryption_Button_Click(object sender, EventArgs e)
        {
            S_DES_Algorithm();
        }

        public void S_DES_Algorithm()
        {
            //initialize
            string Key = null;
            string K1 = null;
            string K2 = null;

            int[] P10 = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6 };
            int[] P8 = { 6, 3, 7, 4, 8, 5, 10, 9 };
            int[] IP = { 2, 6, 3, 1, 4, 8, 5, 7 };
            int[] EP = { 4, 1, 2, 3, 2, 3, 4, 1 };
            int[] P4 = { 2, 4, 3, 1 };
        }
    }
}
```



```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
S-DES Form.cs [Design]
public void S_DES_Algorithm()
{
    //initialize
    string Key = null;
    string K1 = null;
    string K2 = null;

    int[] P10 = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6 };
    int[] P8 = { 6, 3, 7, 4, 8, 5, 10, 9 };
    int[] IP = { 2, 6, 3, 1, 4, 8, 5, 7 };
    int[] EP = { 4, 1, 2, 3, 2, 3, 4, 1 };
    int[] P4 = { 2, 4, 3, 1 };

    int[,] S0 =
    {
        {1, 0, 3, 2},
        {3, 2, 1, 0},
        {0, 2, 1, 3},
        {3, 1, 3, 2},
    };
    int[,] S1 =
    {
        {0, 1, 2, 3},
        {2, 0, 1, 3},
        {3, 0, 1, 0},
        {2, 1, 0, 3},
    };

    //Step 1: Key Generation
    Key = KeyTextBox.Text;
    K1 = Key;
    K2 = Key;
    Key_Generation(ref K1, ref K2, P10, P8);

    //Check Encrypt or Decrypt
    if (EncryptCheckBox.Checked)
    {
        //Step 2.0: Convert Text
        string[] BinaryPlainText = null;
        //Text
        if (TextCheckBoxForText.Checked)
        {
            BinaryPlainText = new string[TextBox.TextLength];
            for (int i = 0; i < TextBox.TextLength; i++)
            {
                BinaryPlainText[i] = CharToBinary(TextBox.Text[i].ToString());
            }
        }
    }
}
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
S Cryptographic_AlgorithmsS_DES_Form
if (EncryptCheckBox.Checked)
{
    //Step 2.0: Convert Text
    string[] BinaryPlainText = null;
    //Text
    if (TextCheckBoxForText.Checked)
    {
        BinaryPlainText = new string[TextBox.TextLength];
        for (int i = 0; i < TextBox.TextLength; i++)
        {
            BinaryPlainText[i] = CharToBinary(TextBox.Text[i].ToString());
        }
    }//Binary
    else if (BinaryCheckBoxForText.Checked)
    {
        string[] SplitBinaryPlainText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        BinaryPlainText = SplitBinaryPlainText;
    }//Hex
    else if (HexCheckBoxForText.Checked)
    {
        string[] SplitHexPlainText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        BinaryPlainText = new string[SplitHexPlainText.Length];
        for (int i = 0; i < SplitHexPlainText.Length; i++)
        {
            BinaryPlainText[i] = HexToBin(SplitHexPlainText[i]);
        }
    }//Error
    else
    {
        MessageBox.Show("You did not choose the type of the 'PlainText' !!!");
        return;
    }
    //Step 2: Initial Permutation
    string[] IP_PlainText = Initial_Permutation(BinaryPlainText, IP);
    //Step 3: Encryption
    string[] OutputOfEncryption = Encryption(IP_PlainText, EP, K1, K2, S0, S1, P4);
    //Step 4: Final Permutation
    string[] CipherText = Final_Permutation(OutputOfEncryption, IP);

    string Temp = "";
    for (int i = 0; i < CipherText.Length; i++)
    {

```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
S Cryptographic_AlgorithmsS_DES_Form
string Temp = "";
for (int i = 0; i < CipherText.Length; i++)
{
    if (TextCheckBoxForResult.Checked)//Result As Text
        Temp += BinaryToChar(CipherText[i]);
    else if (BinaryCheckBoxForResult.Checked)//Result As Binary
        Temp += CipherText[i] + " ";
    else if (HexCheckBoxForResult.Checked)//Result As Hex
        Temp += BinToHex(CipherText[i]) + " ";
    else
    {
        MessageBox.Show("You did not choose the type of Result or Result is Wrong !!!");
        return;
    }
}
//Result CipherText
ResultBox.Text = Temp;
}
else if (DecryptCheckBox.Checked)
{
    //Step 2.0: Convert Text
    string[] BinaryCipherText = null;
    //Text
    if (TextCheckBoxForText.Checked)
    {
        BinaryCipherText = new string[TextBox.TextLength];
        for (int i = 0; i < TextBox.TextLength; i++)
        {
            BinaryCipherText[i] = CharToBinary(TextBox.Text[i].ToString());
        }
    }//Binary
    else if (BinaryCheckBoxForText.Checked)
    {
        string[] SplitBinaryPlainText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        BinaryCipherText = SplitBinaryPlainText;
    }//Hex
    else if (HexCheckBoxForText.Checked)
    {
        string[] SplitHexPlainText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        BinaryCipherText = new string[SplitHexPlainText.Length];
        for (int i = 0; i < SplitHexPlainText.Length; i++)
        {

```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
S-DES Form.cs [Design]
string[] SplitBinaryPlainText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
BinaryCipherText = SplitBinaryPlainText;
} //Hex
else if (HexCheckBoxForText.Checked)
{
    string[] SplitHexPlainText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    BinaryCipherText = new string[SplitHexPlainText.Length];
    for (int i = 0; i < SplitHexPlainText.Length; i++)
    {
        BinaryCipherText[i] = HexToBin(SplitHexPlainText[i]);
    }
} //Error
else
{
    MessageBox.Show("You did not choose the type of the 'CipherText' !!!");
    return;
}
//Step 2: Initial Permutation
string[] IP_CipherText = Initial_Permutation(BinaryCipherText, IP);
//Step 3: Decryption
string[] OutputOfDecryption = Decryption(IP_CipherText, EP, K1, K2, S0, S1, P4);
//Step 4: Final Permutation
string[] PlainText = Final_Permutation(OutputOfDecryption, IP);

string Temp = "";
for (int i = 0; i < PlainText.Length; i++)
{
    if (TextCheckBoxForResult.Checked)
        Temp += BinaryToChar(PlainText[i]);
    else if (BinaryCheckBoxForResult.Checked)
        Temp += PlainText[i] + " ";
    else if (HexCheckBoxForResult.Checked)
        Temp += BinToHex(PlainText[i]) + " ";
    else
    {
        MessageBox.Show("You did not choose the type of Result or Result is Wrong !!!");
        return;
    }
}
//Result PlainText
ResultBox.Text = Temp;
} //Error
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
S-DES Form.cs [Design]
//Result PlainText
ResultBox.Text = Temp;
} //Error
else
{
    MessageBox.Show("You did not choose to Encrypt or Decrypt !!!");
    return;
}

public void Key_Generation(ref string K1, ref string K2, int[] P10, int[] P8)
{
    //Key 1
    //P10
    string TempKey = "";
    for (int i = 0; i < P10.Length; i++)
    {
        TempKey += K1[P10[i] - 1];
    }
    K1 = TempKey;
    K2 = TempKey;

    //Shift To Left By 1
    string LeftK = K1.Substring(1, (K1.Length / 2) - 1) + K1[0];
    string RightK = K1.Substring((K1.Length / 2) + 1, (K1.Length / 2) - 1) + K1[K1.Length / 2];
    K1 = LeftK + RightK;

    //P8
    TempKey = "";
    for (int i = 0; i < P8.Length; i++)
    {
        TempKey += K1[P8[i] - 1];
    }
    K1 = TempKey;

    //Key 2
    //P10 is Done
    //Shift To Left By 3
    LeftK = K2.Substring(3, (K2.Length / 2) - 3) + K2[0] + K2[1] + K2[2];
    RightK = K2.Substring((K2.Length / 2) + 3, (K2.Length / 2) - 3) + K2[K2.Length / 2] + K2[(K2.Length / 2) + 1] + K2[(K2.Length / 2) + 2];
    K2 = LeftK + RightK;
} //P8
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
S-DES Form.cs [Design]
RightK = K2.Substring((K2.Length / 2) + 3, (K2.Length / 2) - 3) + K2[K2.Length / 2] + K2[(K2.Length / 2) + 1] + K2[(K2.Length / 2) + 2];
K2 = LeftK + RightK;
//PB
TempKey = "";
for (int i = 0; i < P8.Length; i++)
{
    TempKey += K2[P8[i] - 1];
}
K2 = TempKey;

public string[] Initial_Permutation(string[] PlainText, int[] IP)
{
    string[] TempPlainText = new string[PlainText.Length];
    for (int i = 0; i < PlainText.Length; i++)
    {
        for (int j = 0; j < IP.Length; j++)
        {
            TempPlainText[i] += PlainText[i][IP[j] - 1];
        }
    }
    return TempPlainText;
}

public string[] Encryption(string[] IP_PlainText, int[] EP, string K1, string K2, int[,] S0, int[,] S1, int[] P4)
{
    string[] Encrypted = new string[IP_PlainText.Length];
    for (int i = 0; i < IP_PlainText.Length; i++)
    {
        string Round1 = Round_1(IP_PlainText[i], EP, K1, S0, S1, P4);
        string Round2 = Round_2(Round1, EP, K2, S0, S1, P4);
        Encrypted[i] = Round2;
    }
    return Encrypted;
}

public string[] Decryption(string[] IP_CipherText, int[] EP, String K1, string K2, int[,] S0, int[,] S1, int[] P4)
{
    string[] Decrypted = new string[IP_CipherText.Length];
    for (int i = 0; i < IP_CipherText.Length; i++)
    {
        string Round1 = Round_1(IP_CipherText[i], EP, K2, S0, S1, P4);
    }
}

100 %
Error List
Ready
Ln 13 Col 6 Ch 6 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
S-DES Form.cs [Design]
public string[] Decryption(string[] IP_CipherText, int[] EP, String K1, string K2, int[,] S0, int[,] S1, int[] P4)
{
    string[] Decrypted = new string[IP_CipherText.Length];
    for (int i = 0; i < IP_CipherText.Length; i++)
    {
        string Round1 = Round_1(IP_CipherText[i], EP, K2, S0, S1, P4);
        string Round2 = Round_2(Round1, EP, K1, S0, S1, P4);
        Decrypted[i] = Round2;
    }
    return Decrypted;
}

public string Round_1(string Text, int[] EP, String K1, int[,] S0, int[,] S1, int[] P4)
{
    //Using EP //With IP_PlainTextRight
    string IP_PlainTextRight = null;
    string Temp = Text.Substring(Text.Length / 2, Text.Length / 2);
    for (int i = 0; i < EP.Length; i++)
    {
        IP_PlainTextRight += Temp[EP[i] - 1];
    }
    //PlainTextRight Xor K1
    string XorResult = null;
    for (int i = 0; i < IP_PlainTextRight.Length; i++)
    {
        if (IP_PlainTextRight[i] == K1[i])
            XorResult += "0";
        else
            XorResult += "1";
    }

    //S Boxes
    string SLeft = XorResult.Substring(0, XorResult.Length / 2);
    string SRight = XorResult.Substring(XorResult.Length / 2, XorResult.Length / 2);
    //S0
    Temp = null;
    Temp += SLeft[0];
    Temp += SLeft[3];
    int Row = BinaryToDecimal(Temp);
    Temp = null;
    Temp += SLeft[1];
    Temp += SLeft[2];
}

100 %
Error List
Ready
Ln 13 Col 6 Ch 6 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Forms.cs [Design]
Cryptographic_AlgorithmsS_DES_Form S_DES_Form
//S Boxes
string SLeft = XorResult.Substring(0, XorResult.Length / 2);
string SRight = XorResult.Substring(XorResult.Length / 2, XorResult.Length / 2);
//S0
Temp = null;
Temp += SLeft[0];
Temp += SLeft[3];
int Row = BinaryToDecimal(Temp);
Temp = null;
Temp += SLeft[1];
Temp += SLeft[2];
int Col = BinaryToDecimal(Temp);

IP_PlainTextRight = null;
IP_PlainTextRight += DecimalToBinary(S0[Row, Col]).PadLeft(2, '0');
//S1
Temp = null;
Temp += SRight[0];
Temp += SRight[3];
Row = BinaryToDecimal(Temp);
Temp = null;
Temp += SRight[1];
Temp += SRight[2];
Col = BinaryToDecimal(Temp);

IP_PlainTextRight += DecimalToBinary(S1[Row, Col]).PadLeft(2, '0');
//Using P4
Temp = null;
for (int i = 0; i < P4.Length; i++)
{
    Temp += IP_PlainTextRight[P4[i] - 1];
}

//Temp Xor with the Left of IP_PlainText
XorResult = null;
for (int i = 0; i < Temp.Length; i++)
{
    if (Temp[i] == Text[i])
        XorResult += "0";
    else
        XorResult += "1";
}
100 %
Error List
Ready Ln 13 Col 6 Ch 6 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Forms.cs [Design]
Cryptographic_AlgorithmsS_DES_Form S_DES_Form
//Temp Xor with the Left of IP_PlainText
XorResult = null;
for (int i = 0; i < Temp.Length; i++)
{
    if (Temp[i] == Text[i])
        XorResult += "0";
    else
        XorResult += "1";
}
//Swap New Left(XorResult) With the Old Right of IP_PlainText
Text = Text.Substring(Text.Length / 2, Text.Length / 2) + XorResult;
}
return Text;
}

public string Round_2(string Text, int[] EP, string K2, int[,] S0, int[,] S1, int[] P4)
{
    //Using EP
    string Round1Right = null;
    string Temp = Text.Substring(Text.Length / 2, Text.Length / 2);
    for (int i = 0; i < EP.Length; i++)
    {
        Round1Right += Temp[EP[i] - 1];
    }

    //Round1Right Xor K1
    string XorResult = null;
    for (int i = 0; i < Round1Right.Length; i++)
    {
        if (Round1Right[i] == K2[i])
            XorResult += "0";
        else
            XorResult += "1";
    }

    //S Boxes
    string SLeft = XorResult.Substring(0, XorResult.Length / 2);
    string SRight = XorResult.Substring(XorResult.Length / 2, XorResult.Length / 2);
    //S0
    Temp = null;
    Temp += SLeft[0];
    100 %
Error List
Ready Ln 13 Col 6 Ch 6 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Forms.cs S-DES Form.cs [Design]
Cryptographic_AlgorithmsS_DES_Form
//S Boxes
string SLeft = XorResult.Substring(0, XorResult.Length / 2);
string SRight = XorResult.Substring(XorResult.Length / 2, XorResult.Length / 2);
//S0
Temp = null;
Temp += SLeft[0];
Temp += SLeft[3];
int Row = BinaryToDecimal(Temp);
Temp = null;
Temp += SLeft[1];
Temp += SLeft[2];
int Col = BinaryToDecimal(Temp);
Round1Right = null;
Round1Right += DecimalToBinary(S0[Row, Col]).PadLeft(2, '0');
//S1
Temp = null;
Temp += SRight[0];
Temp += SRight[3];
Row = BinaryToDecimal(Temp);
Temp = null;
Temp += SRight[1];
Temp += SRight[2];
Col = BinaryToDecimal(Temp);
Round1Right += DecimalToBinary(S1[Row, Col]).PadLeft(2, '0');
//Using P4
Temp = null;
for (int i = 0; i < P4.Length; i++)
{
    Temp += Round1Right[P4[i] - 1];
}

//Temp Xor with the Left of Round1
XorResult = null;
for (int i = 0; i < Temp.Length; i++)
{
    if (Temp[i] == Text[i])
        XorResult += "0";
    else
        XorResult += "1";
}
// New Left(XorResult) With the Old Right of Round1
100 %
Error List
Ready
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Forms.cs S-DES Form.cs [Design]
Cryptographic_AlgorithmsS_DES_Form
}

// New Left(XorResult) With the Old Right of Round1
Text = XorResult + Text.Substring(Text.Length / 2, Text.Length / 2);

return Text;
}

public string[] Final_Permutation(string[] Round2, int[] IP)
{
    //Create IP_Inverse
    int[] IP_Inverse = new int[IP.Length];
    for (int i = 0; i < IP.Length; i++)
    {
        IP_Inverse[IP[i] - 1] = i + 1;
    }
    // Create FP of Round2
    string[] Output = new string[Round2.Length];
    for (int i = 0; i < Round2.Length; i++)
    {
        for (int j = 0; j < IP_Inverse.Length; j++)
        {
            Output[i] += Round2[i][IP_Inverse[j] - 1];
        }
    }
    return Output;
}

public string DecimalToBinary(int data)
{
    return Convert.ToString(data, 2);
}

public int BinaryToDecimal(string data)
{
    return Convert.ToInt16(data, 2);
}

public string BinToHex(string S)
{
    string Result = "";
    Dictionary<string, char> hexCharacterToBinary = new Dictionary<string, char> {
100 %
Error List
Ready
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
Cryptographic_AlgorithmsS_DES_Form
public string BinToHex(string S)
{
    string Result = "";
    Dictionary<string, char> hexCharacterToBinary = new Dictionary<string, char> {
        { "0000", '0' },
        { "0001", '1' },
        { "0010", '2' },
        { "0011", '3' },
        { "0100", '4' },
        { "0101", '5' },
        { "0110", '6' },
        { "0111", '7' },
        { "1000", '8' },
        { "1001", '9' },
        { "1010", 'A' },
        { "1011", 'B' },
        { "1100", 'C' },
        { "1101", 'D' },
        { "1110", 'E' },
        { "1111", 'F' },
    };
    for (int i = 0; i < S.Length; i += 4)
    {
        string BinaryOf4 = "";
        BinaryOf4 += S[i];
        BinaryOf4 += S[i + 1];
        BinaryOf4 += S[i + 2];
        BinaryOf4 += S[i + 3];
        Result += hexCharacterToBinary[BinaryOf4];
    }
    return Result;
}
public string HexToBin(string S)
{
    string Result = "";
    Dictionary<char, string> hexCharacterToBinary = new Dictionary<char, string> {
        { '0', "0000" },
        { '1', "0001" },
        { '2', "0010" },
        { '3', "0011" },
        { '4', "0100" },
        { '5', "0101" },
        { '6', "0110" },
        { '7', "0111" },
        { '8', "1000" },
        { '9', "1001" },
        { 'A', "1010" },
        { 'B', "1011" },
        { 'C', "1100" },
        { 'D', "1101" },
        { 'E', "1110" },
        { 'F', "1111" },
    };
    for (int i = 0; i < S.Length; i++)
    {
        Result += hexCharacterToBinary[S[i]];
    }
    return Result;
}
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs [Design]
Cryptographic_AlgorithmsS_DES_Form
}
public string HexToBin(string S)
{
    string Result = "";
    Dictionary<char, string> hexCharacterToBinary = new Dictionary<char, string> {
        { '0', "0000" },
        { '1', "0001" },
        { '2', "0010" },
        { '3', "0011" },
        { '4', "0100" },
        { '5', "0101" },
        { '6', "0110" },
        { '7', "0111" },
        { '8', "1000" },
        { '9', "1001" },
        { 'A', "1010" },
        { 'B', "1011" },
        { 'C', "1100" },
        { 'D', "1101" },
        { 'E', "1110" },
        { 'F', "1111" },
    };
    for (int i = 0; i < S.Length; i++)
    {
        Result += hexCharacterToBinary[S[i]];
    }
    return Result;
}
public string CharToBinary(string data)
{
    return Convert.ToString(Convert.ToChar(data), 2).PadLeft(8, '0');
}
public string BinaryToChar(string data)
{
    int val = BinaryToDecimal(data);
    return Convert.ToChar(val).ToString();
}

/// <summary>
/// Note: unneeded to be Explained
/// </summary>
```

S-DES Form.cs [Design]

```
/// <summary>
/// Note: unneeded to be Explained
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
/// 
//Text
private void TextCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (TextCheckBoxForText.Checked)
    {
        BinaryCheckBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
    }
}
private void BinaryCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForText.Checked)
    {
        TextCheckBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
    }
}
private void HexCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForText.Checked)
    {
        TextCheckBoxForText.Checked = false;
        BinaryCheckBoxForText.Checked = false;
    }
}
//En/Decrypt
private void EncryptCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (EncryptCheckBox.Checked)
    {
        DecryptCheckBox.Checked = false;
    }
}
private void DecryptCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (DecryptCheckBox.Checked)
```

S-DES Form.cs [Design]

```
private void DecryptCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (DecryptCheckBox.Checked)
    {
        EncryptCheckBox.Checked = false;
    }
}
//Result
private void TextCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (TextCheckBoxForResult.Checked)
    {
        BinaryCheckBoxForResult.Checked = false;
        HexCheckBoxForResult.Checked = false;
    }
}
private void BinaryCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForResult.Checked)
    {
        TextCheckBoxForResult.Checked = false;
        HexCheckBoxForResult.Checked = false;
    }
}
private void HexCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForResult.Checked)
    {
        TextCheckBoxForResult.Checked = false;
        BinaryCheckBoxForResult.Checked = false;
    }
}
```

## **RC4:**

The inputs are a Text as plain text or cipher text in a text box and a Key in a text box too. After inserting the text, click on Encrypt check box if the text is a plaintext or Decrypt check box if the text is a cipher text and choose the type of the text (Text, Binary, or Hex) and choose the type of the key (Binary or Hex). Then click on En/Decrypt button to get the result in the “Result” text box.

### **Steps:**

Step 1:

Generate State Vector S[] and Temporary Vector T[] where both of them are arrays of integers.

Step 2:

Preform Initial Permutation function on State Vector array S[].

Step 3:

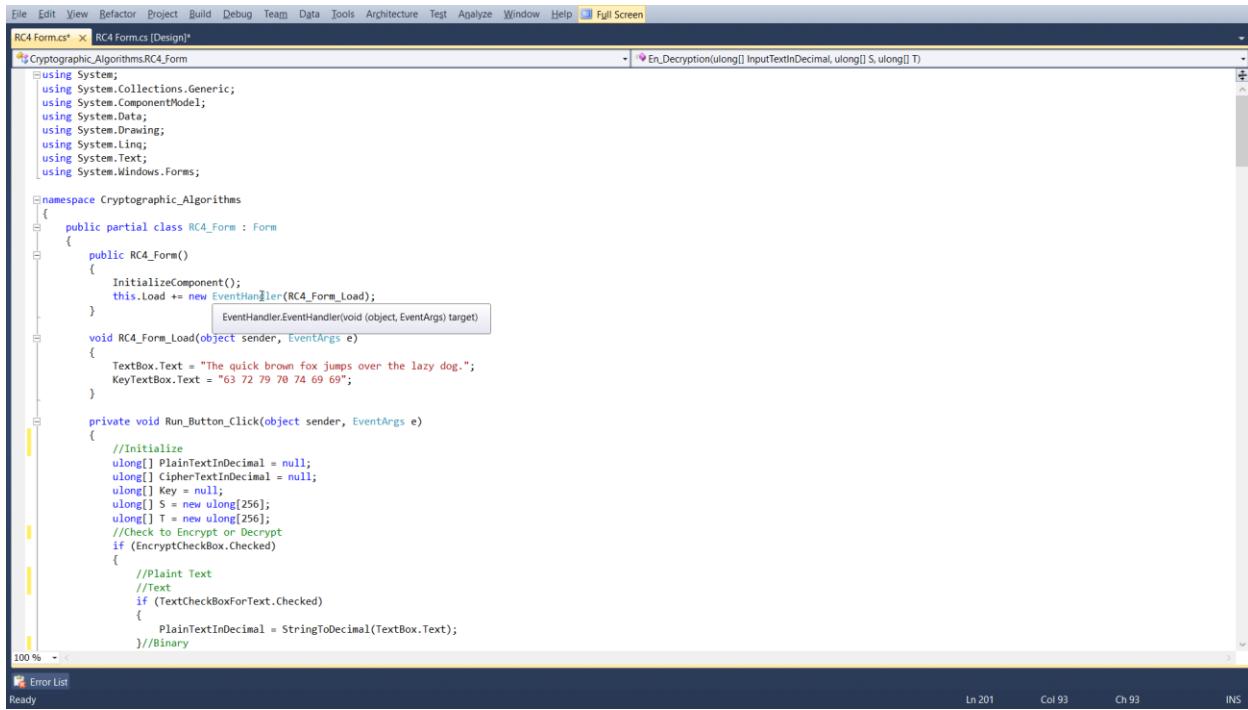
Perform Encryption or Decryption Function according to the inserted input. If you are encrypting, you will perform the encryption on the inserted plain text in decimal form. If you are decrypting, you will perform the decryption on the inserted cipher text in decimal form.

Step 4:

Then add result of the encryption or the decryption to the “Result” text box.

Functions used: Converting Function as Converting Plaintext or Ciphertext into String, Binary, Hexadecimal, and Decimal. And converting Key into Binary from Hexadecimal.

## Screen Shots:



RC4 Form.cs [Design]\*

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace Cryptographic_Algorithms
{
    public partial class RC4_Form : Form
    {
        public RC4_Form()
        {
            InitializeComponent();
            this.Load += new EventHandler(RC4_Form_Load);
        }

        void RC4_Form_Load(object sender, EventArgs e)
        {
            TextBox.Text = "The quick brown fox jumps over the lazy dog.";
            KeyTextBox.Text = "63 72 79 78 74 69 69";
        }

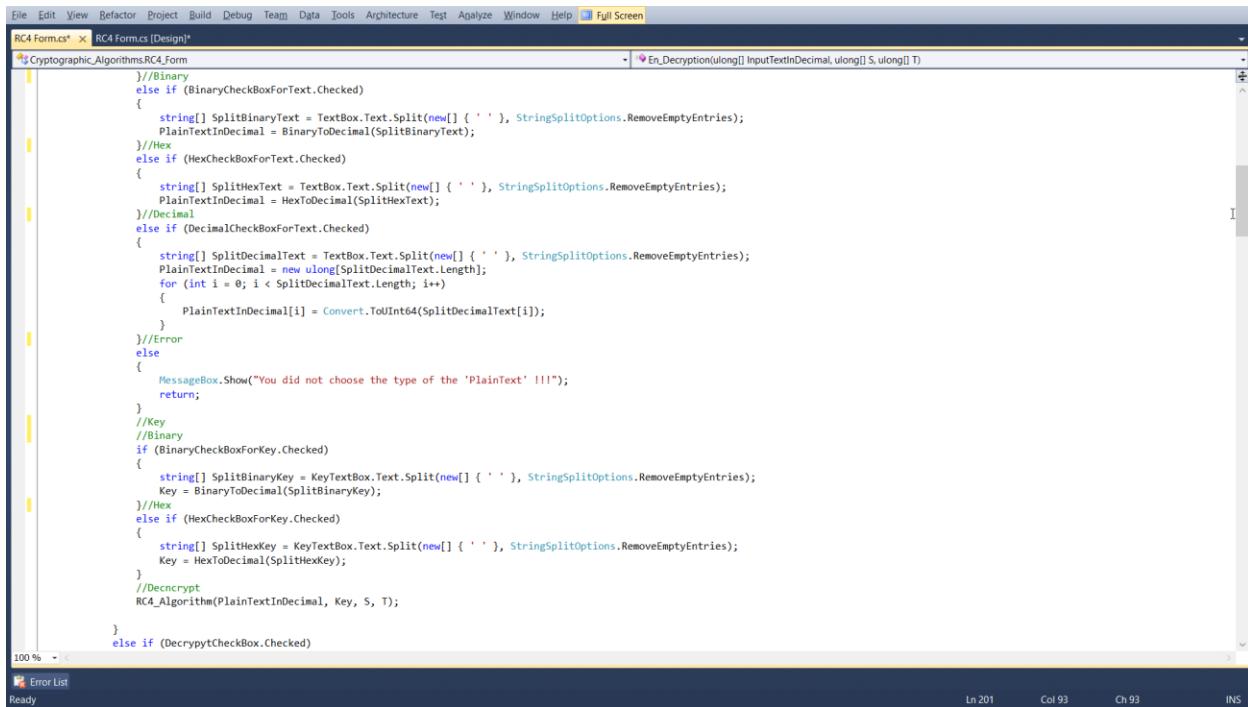
        private void Run_Button_Click(object sender, EventArgs e)
        {
            //Initialize
            ulong[] PlainTextInDecimal = null;
            ulong[] CipherTextInDecimal = null;
            ulong[] Key = null;
            ulong[] S = new ulong[256];
            ulong[] T = new ulong[256];
            //Check to Encrypt or Decrypt
            if (EncryptCheckBox.Checked)
            {
                //Plaint Text
                //Text
                if (TextCheckBoxForText.Checked)
                {
                    PlainTextInDecimal = StringToDecimal(textBox.Text);
                }
                //Binary
            }
            else
            {
                //Binary
            }
        }
    }
}
```

100 %

Error List

Ready

Ln 201 Col 93 Ch 93 INS



```
RC4 Form.cs [Design]*
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
//Binary
else if (BinaryCheckBoxForText.Checked)
{
    string[] SplitBinaryText = textBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    PlainTextInDecimal = BinaryToDecimal(SplitBinaryText);
}
//Hex
else if (HexCheckBoxForText.Checked)
{
    string[] SplitHexText = textBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    PlainTextInDecimal = HexToDecimal(SplitHexText);
}
//Decimal
else if (DecimalCheckBoxForText.Checked)
{
    string[] SplitDecimalText = textBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    PlainTextInDecimal = new ulong[SplitDecimalText.Length];
    for (int i = 0; i < SplitDecimalText.Length; i++)
    {
        PlainTextInDecimal[i] = Convert.ToInt64(SplitDecimalText[i]);
    }
}
//Error
else
{
    MessageBox.Show("You did not choose the type of the 'PlainText' !!!");
    return;
}
//Key
//Binary
if (BinaryCheckBoxForKey.Checked)
{
    string[] SplitBinaryKey = keyTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    Key = BinaryToDecimal(SplitBinaryKey);
}
//Hex
else if (HexCheckBoxForKey.Checked)
{
    string[] SplitHexKey = keyTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    Key = HexToDecimal(SplitHexKey);
}
//Decnrypt
RC4_Algorithm(PlainTextInDecimal, Key, S, T);

}
else if (DecryptcheckBox.Checked)
```

100 %

Error List

Ready

Ln 201 Col 93 Ch 93 INS

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RCA Form.cs* RC4 Form.cs [Design]* En_Decrypt(ulong[] InputTextInDecimal, ulong[] S, ulong[] T)
C:\Cryptographic_Algorithms\RC4\Form.cs
}
else if (DecryptptCheckBox.Checked)
{
    //Cipher Text
    //Text
    if (TextCheckBoxForText.Checked)
    {
        CipherTextInDecimal = StringToDecimal(textBox.Text);
    }
    //Binary
    else if (BinaryCheckBoxForText.Checked)
    {
        string[] SplitBinaryText = textBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        CipherTextInDecimal = BinaryToDecimal(SplitBinaryText);
    }
    //Hex
    else if (HexCheckBoxForText.Checked)
    {
        string[] SplitHexText = textBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        CipherTextInDecimal = HexToDecimal(SplitHexText);
    }
    //Decimal
    else if (DecimalCheckBoxForText.Checked)
    {
        string[] SplitDecimalText = textBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        CipherTextInDecimal = new ulong[SplitDecimalText.Length];
        for (int i = 0; i < SplitDecimalText.Length; i++)
        {
            CipherTextInDecimal[i] = Convert.ToInt64(SplitDecimalText[i]);
        }
    }
    //Error
    else
    {
        MessageBox.Show("You did not choose the type of the 'CipherText' !!!");
        return;
    }
}
//Key
//Binary
if (BinaryCheckBoxForKey.Checked)
{
    string[] SplitBinaryKey = KeyTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    Key = BinaryToDecimal(SplitBinaryKey);
}
//Hex
else if (HexCheckBoxForKey.Checked)
{
    MessageBox.Show("You did not choose the type of the 'Key'!!!");
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RCA Form.cs* RC4 Form.cs [Design]* En_Decrypt(ulong[] InputTextInDecimal, ulong[] S, ulong[] T)
C:\Cryptographic_Algorithms\RC4\Form.cs
}
//Hex
else if (HexCheckBoxForKey.Checked)
{
    string[] SplitHexKey = KeyTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    Key = HexToDecimal(SplitHexKey);
}
//Error
else
{
    MessageBox.Show("You did not choose the type of the 'Key'!!!");
    return;
}
//Encrypt
RC4_Algorithm(CipherTextInDecimal, Key, S, T);
}
else
{
    MessageBox.Show("You did not choose to Encrypt or Decrypt !!!");
    return;
}
}

public void RC4_Algorithm(ulong[] Text, ulong[] Key, ulong[] S, ulong[] T)
{
    //Step 1: Generate State Vector S[] and Temporary Vector T[]
    Generate_SandT(ref S, ref T, Key);

    //Step 2: Initial Permutation On S[]
    Initial_Permutation(ref S, ref T);

    //Step 3: Encryption Or Decryption
    En_Decryption(Text, S, T);
}

public void Generate_SandT(ref ulong[] S, ref ulong[] T, ulong[] Key)
{
    //Generating State Vector
    for (ulong i = 0; i < 256; i++)
    {
        S[i] = i;
        T[i] = Key[i % Convert.ToInt64(Key.Length)];
    }
}
```

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RC4 Form.cs [Design]\*

```
using System;
using System.Text;
using System.Security.Cryptography;

public class RC4_Algorithm
{
    public void Initial_Permutation(ref ulong[] S, ref ulong[] T)
    {
        ulong j = 0;
        for (ulong i = 0; i < 256; i++)
        {
            j = (j + S[i] + T[i]) % 256;
            ulong Swap = S[i];
            S[i] = S[j];
            S[j] = Swap;
        }
    }

    public void En_Decryption(ulong[] InputTextInDecimal, ulong[] S, ulong[] T)
    {
        ulong i = 0, j = 0;
        int TextIndex = 0;
        string ResultTextAsString = "";
        while (TextIndex < InputTextInDecimal.Length)
        {
            i = (i + 1) % 256;
            j = (j + S[i]) % 256;

            ulong Swap = S[i];
            S[i] = S[j];
            S[j] = Swap;

            ulong t = (S[i] + S[j]) % 256;

            ulong K = S[t];
            //Kor
            ulong ResultTextInDecimal = K ^ InputTextInDecimal[TextIndex];

            if (EncryptCheckBox.Checked)
            {
                if (TextCheckBoxForResult.Checked)
                    ResultTextAsString += DecimalToChar(ResultTextInDecimal);
                else if (BinaryCheckBoxForResult.Checked)
                    ResultTextAsString += DecimalToBinary(ResultTextInDecimal).PadLeft(8, '0') + " ";
                else if (HexCheckBoxForResult.Checked)
                    ResultTextAsString += DecimalToHex(ResultTextInDecimal) + " ";
                else if (DecimalCheckBoxForResult.Checked)
                    ResultTextAsString += ResultTextInDecimal + " ";
            }
        }
    }
}
```

100 %

Error List Ready

Ln 201 Col 93 Ch 93 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RC4 Form.cs [Design]\*

```
using System;
using System.Text;
using System.Security.Cryptography;

public class RC4_Algorithm
{
    if (EncryptCheckBox.Checked)
    {
        if (TextCheckBoxForResult.Checked)
            ResultTextAsString += DecimalToChar(ResultTextInDecimal);
        else if (BinaryCheckBoxForResult.Checked)
            ResultTextAsString += DecimalToBinary(ResultTextInDecimal).PadLeft(8, '0') + " ";
        else if (HexCheckBoxForResult.Checked)
            ResultTextAsString += DecimalToHex(ResultTextInDecimal) + " ";
        else if (DecimalCheckBoxForResult.Checked)
            ResultTextAsString += ResultTextInDecimal + " ";
    }
    else
    {
        MessageBox.Show("You did not choose the type of Result or Result is Wrong !!!");
        return;
    }
}
else if (DecrypttCheckBox.Checked)
{
    if (TextCheckBoxForResult.Checked)
        ResultTextAsString += DecimalToChar(ResultTextInDecimal);
    else if (BinaryCheckBoxForResult.Checked)
        ResultTextAsString += DecimalToBinary(ResultTextInDecimal).PadLeft(8, '0') + " ";
    else if (HexCheckBoxForResult.Checked)
        ResultTextAsString += DecimalToHex(ResultTextInDecimal) + " ";
    else if (DecimalCheckBoxForResult.Checked)
        ResultTextAsString += ResultTextInDecimal + " ";
    else
    {
        MessageBox.Show("You did not choose the type of Result or Result is Wrong !!!");
        return;
    }
}
TextIndex++;
}
ResultBox.Text = ResultTextAsString;
}

public ulong[] StringToDecimal(string data)
{
    ulong[] Total = new ulong[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = Convert.ToUInt64(data[i].ToString());
    }
    return Total;
}
```

100 %

Error List Ready

Ln 201 Col 93 Ch 93 INS

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Refactor, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, Help, and Full Screen. The title bar displays "RC4 Form.cs" and "RC4 Form.cs [Design]\*". The main code editor window contains C# code for a class named "Cryptographic\_algorithms.RC4\_Form". The code defines several methods for converting between binary, decimal, and hex values:

```
public ulong[] StringToDecimal(string data)
{
    ulong[] Total = new ulong[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = data[i];
    }
    return Total;
}

public string DecimalToString(ulong data)
{
    return Convert.ToString(data);
}

public string DecimalToChar(ulong data)
{
    return Convert.ToChar(data).ToString();
}

public ulong CharToDecimal(string data)
{
    return Convert.ToInt64(Convert.ToChar(data));
}

public string DecimalToBinary(ulong data)
{
    return Convert.ToString((long)data, 2);
}

public ulong[] BinaryToDecimal(string[] data)
{
    ulong[] Total = new ulong[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = Convert.ToInt64(data[i], 2);
    }
    return Total;
}

public ulong[] HexToDecimal(string[] data)
{
    ulong[] Total = new ulong[data.Length];
```

```
public ulong[] HexToDecimal(string[] data)
{
    ulong[] Total = new ulong[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = Convert.ToInt64(data[i], 16);
    }
    return Total;
}
public string DecimalToHex(ulong data)
{
    return Convert.ToString((long)data, 16);
}

/// <summary>
/// Note: unneeded to be Explained
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
/// 
//Text
private void TextBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (TextBoxForText.Checked)
    {
        BinaryCheckBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
        DecimalCheckBoxForText.Checked = false;
    }
}
private void BinaryCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForText.Checked)
    {
        TextBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
        DecimalCheckBoxForText.Checked = false;
    }
}
private void HexCheckBoxForText_CheckedChanged(object sender, EventArgs e)
```

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RC4 Form.cs [Design]\*

```
private void HexCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForText.Checked)
    {
        TextCheckBoxForText.Checked = false;
        BinaryCheckBoxForText.Checked = false;
        DecimalCheckBoxForText.Checked = false;
    }
}

private void DecimalCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (DecimalCheckBoxForText.Checked)
    {
        TextCheckBoxForText.Checked = false;
        BinaryCheckBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
    }
}

//En/Decrypt
private void EncryptCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (EncryptCheckBox.Checked)
    {
        DecryptptCheckBox.Checked = false;
    }
}

private void DecryptptCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (DecryptptCheckBox.Checked)
    {
        EncryptCheckBox.Checked = false;
    }
}

//Key
private void BinaryCheckBoxForKey_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForKey.Checked)
    {
        HexCheckBoxForKey.Checked = false;
    }
}
```

100 %

Error List Ready Ln 201 Col 93 Ch 93 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RC4 Form.cs [Design]\*

```
//Key
private void BinaryCheckBoxForKey_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForKey.Checked)
    {
        HexCheckBoxForKey.Checked = false;
    }
}

private void HexCheckBoxForKey_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForKey.Checked)
    {
        BinaryCheckBoxForKey.Checked = false;
    }
}

//Result
private void TextCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (TextCheckBoxForResult.Checked)
    {
        BinaryCheckBoxForResult.Checked = false;
        HexCheckBoxForResult.Checked = false;
        DecimalCheckBoxForResult.Checked = false;
    }
}

private void BinaryCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForResult.Checked)
    {
        TextCheckBoxForResult.Checked = false;
        HexCheckBoxForResult.Checked = false;
        DecimalCheckBoxForResult.Checked = false;
    }
}

private void HexCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForResult.Checked)
    {
        TextCheckBoxForResult.Checked = false;
        BinaryCheckBoxForResult.Checked = false;
        DecimalCheckBoxForResult.Checked = false;
    }
}
```

100 %

Error List Ready Ln 201 Col 93 Ch 93 INS

A screenshot of a Windows application window, likely a developer's environment. The window title is "RC4 Form.cs [Design]\*". The main area displays C# code for a class named "Cryptographic\_Algorithms.RC4\_Form". The code includes a private method "DecimalCheckBoxForResult\_CheckedChanged" which checks if a checked state has changed and updates other checkboxes if necessary. The status bar at the bottom shows "100 %", "Error List", "Ready", "Ln 201", "Col 93", "Ch 93", and "INS".

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RC4 Form.cs* RC4 Form.cs [Design]*
Cryptographic_Algorithms.RC4_Form
    private void DecimalCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
    {
        if (DecimalCheckBoxForResult.Checked)
        {
            TextCheckBoxForResult.Checked = false;
            BinaryCheckBoxForResult.Checked = false;
            HexCheckBoxForResult.Checked = false;
        }
    }
}

100 %
Error List
Ready
Ln 201 Col 93 Ch 93 INS
```

## **Diffie-Hellman:**

The inputs are a Large Prime integer (q), a Primitive root integer (a) that must be primitive on q, a Secret Key for user A, and a Secret Key for user B where each one of them is inserted in its own text box. After inserting click on “Generate Keys” button to get the results.

Steps:

Step 1:

2 Users will choose insert the Large Prime integer (q), the Primitive root integer (a) that must be primitive on q.

Step 2:

User A will choose a Secret key and user B will choose a Secret key too.

Step 3:

Each user will calculate a Public key using their Secret keys and the Primitive root integer (a), then they will share the 2 Public keys.

Step 4:

Each user will calculate a Private Key using their Public keys, Secret keys the Large Prime integer (q), where the 2 Private keys will be equal.

The used functions are: Prime and Primitive Function for checking.

## Screen Shots:

Screenshot of Microsoft Visual Studio showing the code editor for `Diffie_Hellman_Form.cs`. The code implements a simple Diffie-Hellman key exchange application. It includes imports for System, System.Collections.Generic, System.ComponentModel, System.Data, System.Drawing, System.Linq, System.Text, and System.Windows.Forms. The class `Diffie_Hellman_Form` contains a constructor that initializes components and loads the form. The `Diffie_Hellman_Form_Load` event handler sets initial values for text boxes: `q_TextBox.Text = "53";`, `a_TextBox.Text = "3";`, `Xa_TextBox.Text = "97";`, and `Xb_TextBox.Text = "23";`. The `Diffie_Button_Click` event handler performs step 1: it checks if the input `q` is a prime number. If not, it shows a message box and returns. If it is prime, it checks if it is a primitive root mod `q`. If not, it shows a message box and returns. If it is a primitive root, it proceeds to step 2.

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
Diffie Hellman Form.cs* Diffie Hellman Form.cs [Design]
Cryptographic_Algorithms.Diffie_Hellman_Form
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Cryptographic_Algorithms
{
    public partial class Diffie_Hellman_Form : Form
    {
        public Diffie_Hellman_Form()
        {
            InitializeComponent();
            this.Load += new EventHandler(Diffie_Hellman_Form_Load);
        }

        void Diffie_Hellman_Form_Load(object sender, EventArgs e)
        {
            q_TextBox.Text = "53";
            a_TextBox.Text = "3";
            Xa_TextBox.Text = "97";
            Xb_TextBox.Text = "23";
        }

        private void Diffie_Button_Click(object sender, EventArgs e)
        {
            //Step 1:
            //Large Prime Integer
            ulong q;
            if (!ulong.TryParse(q_TextBox.Text, out q))
            {
                MessageBox.Show("Insert Large Prime Integer");
                return;
            }
            else
            {
                if (!isPrime(q))
                {
                    MessageBox.Show("Insert Large Prime Integer");
                    return;
                }
                else
                {
                    if (!isPrimitiveRoot(q))
                    {
                        MessageBox.Show(q + " is not a Primitive Root Integer");
                        return;
                    }
                    //being a primitive root mod q
                    //for n=1 to q-1
                    // a^n mod q = [1 to q-1]
                    ulong a;
                    if (!ulong.TryParse(a_TextBox.Text, out a))
                    {
                        MessageBox.Show("Insert Primitive Root Integer");
                        return;
                    }
                    else
                    {
                        //Check if a is primitive root to q or not
                        if (!isPrimitive(a, q))
                        {
                            MessageBox.Show(q + " is not a Primitive root");
                            return;
                        }
                    }
                }
            }
        }
    }
}

100 %
Error List Ready Ln 13 Col 6 Ch 6 INS
```

Screenshot of Microsoft Visual Studio showing the continuation of the `Diffie_Button_Click` event handler. The code now handles step 2: it checks if the secret keys `a` and `b` are less than `q`. It then calculates the shared secret `key` using the formula  $a^b \mod q$ . Finally, it displays the result in the `key_TextBox`.

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
Diffie Hellman Form.cs* Diffie Hellman Form.cs [Design]
Cryptographic_Algorithms.Diffie_Hellman_Form
private void Diffie_Button_Click(object sender, EventArgs e)
{
    //Step 1:
    //Large Prime Integer
    ulong q;
    if (!ulong.TryParse(q_TextBox.Text, out q))
    {
        MessageBox.Show("Insert Large Prime Integer");
        return;
    }
    else
    {
        if (!isPrime(q))
        {
            MessageBox.Show(q + " is not a Prime Integer");
            return;
        }
        else
        {
            if (!isPrimitiveRoot(q))
            {
                MessageBox.Show(q + " is not a Primitive Root Integer");
                return;
            }
            //being a primitive root mod q
            //for n=1 to q-1
            // a^n mod q = [1 to q-1]
            ulong a;
            if (!ulong.TryParse(a_TextBox.Text, out a))
            {
                MessageBox.Show("Insert Primitive Root Integer");
                return;
            }
            else
            {
                //Check if a is primitive root to q or not
                if (!isPrimitive(a, q))
                {
                    MessageBox.Show(q + " is not a Primitive root");
                    return;
                }
            }
        }
    }
}

//Step 2:
//Secret Key < q for user A and user B
ulong Xa;
ulong Xb;
if (!ulong.TryParse(Xa_TextBox.Text, out Xa))
{
    MessageBox.Show("Insert Secret Key for User A");
    return;
}
if (!ulong.TryParse(Xb_TextBox.Text, out Xb))
{
    MessageBox.Show("Insert Secret Key for User B");
    return;
}

//Calculation
key_TextBox.Text = ((a * Xb) % q).ToString();

Ln 13 Col 6 Ch 6 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
Diffie Hellman Form.cs  Diffie Hellman Form.cs [Design]
C:\Cryptographic_Algorithms\Diffie_Hellman_Form
}
//Step 2:
//Secret Key < q for user A and user B
ulong XA;
ulong XB;
if (!ulong.TryParse(XA_TextBox.Text, out XA))
{
    MessageBox.Show("Insert Secret Key for User A");
    return;
}
else
{
    if (XA >= q)
    {
        MessageBox.Show("Insert Secret Key for User A (XA) Less than the Large Prime Integer (q)");
        return;
    }
}
if (!ulong.TryParse(XB_TextBox.Text, out XB))
{
    MessageBox.Show("Insert Secret Key for User B (XB)");
    return;
}
else
{
    if (XB >= q)
    {
        MessageBox.Show("Insert Secret Key for User B (XB) Less than the Large Prime Integer (q)");
        return;
    }
}
//Step 3:
//Public Keys for user A and user B
ulong YA = 1, YB = 1;
for (ulong n = 0; n < XA; n++)
{
    YA = (YA * a) % q;
}
YA_TextBox.Text = YA.ToString();
for (ulong n = 0; n < XB; n++)
{
    YB = (YB * a) % q;
}
100 %
Error List
Ready
Ln 13 Col 6 Ch 6 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
Diffie Hellman Form.cs  Diffie Hellman Form.cs [Design]
C:\Cryptographic_Algorithms\Diffie_Hellman_Form
}
//Step 3:
//Public Keys for user A and user B
ulong YA = 1, YB = 1;
for (ulong n = 0; n < XA; n++)
{
    YA = (YA * a) % q;
}
YA_TextBox.Text = YA.ToString();
for (ulong n = 0; n < XB; n++)
{
    YB = (YB * a) % q;
}
YB_TextBox.Text = YB.ToString();
//Step 4: Share Public Keys
//Step 5: Private Key
//KAB = a^(XA*XB) mod q
//      = ya^XB mod q = (a^Xa)^XB mod q = (a^XB mod q)^XA (which A can compute)
//KAB = KA = KB
ulong KA = 1, KB = 1;
//Private key for user A
for (ulong n = 0; n < XA; n++)
{
    KA = (KA * YB) % q;
}
//Private key for user B
for (ulong n = 0; n < XB; n++)
{
    KB = (KB * YA) % q;
}
//Private key for user A must be equal to Private key for user B
if (KA == KB)
{
    KAB_TextBox.Text = KA.ToString();
}
else
{
    MessageBox.Show("Error Try again !!!");
    return;
}
100 %
Error List
Ready
Ln 13 Col 6 Ch 6 INS
```

The screenshot shows the Microsoft Visual Studio IDE interface. The menu bar includes File, Edit, View, Refactor, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, Help, and Full Screen. The title bar shows "Diffie Hellman Form.cs" and "Diffie Hellman Form.cs [Design]\*". The main code editor window displays C# code for a class named Cryptographic\_Algorithms.Diffie\_Hellman\_Form. The code includes two methods: `isPrime` and `isPrimitive`. A tooltip for the `sort` variable in the `isPrimitive` method is visible, stating "void Diffie\_Hellman\_Form.Diffie\_Hellman\_Form\_Load(object sender, EventArgs e)". The status bar at the bottom shows "100 %", "Error List", "Ready", "Ln 13", "Col 6", "Ch 6", and "INS".

```
public bool isPrime(ulong n)
{
    // Corner case
    if (n <= 1)
        return false;

    // Check from 2 to n-1
    for (ulong i = 2; i < n; i++)
        if (n % i == 0)
            return false;

    return true;
}

public bool isPrimitive(ulong a, ulong q)
{
    ulong Total = 1;
    ulong[] sort = new ulong[q - 1];
    for (ulong n = 0; n < q - 1; n++)
    {
        Total = (Total * a) % q;
        sort[n] = total;
    }
    Array.Sort(sort);
    for (int n = 0; n < sort.Length; n++)
    {
        if (sort[n] != Convert.ToInt64(n) + 1)
        {
            return false;
        }
    }
    return true;
}
```

## **RSA:**

The inputs are a Text as plain text or cipher text, a large prime integer (P), a large prime integer (Q) where each one of them is inserted in its own text box. After inserting the text, click on Encrypt check box if the text is a plaintext or Decrypt check box if the text is a cipher text and choose the type of the text (Text, Binary, Hex, or Decimal). Then click on En/Decrypt button to get the result in the “Result” text box.

Steps:

### **Step 1: Generate Public encryption and Private Decryption keys**

The public encryption keys are divided into 2 variables: The first variable of the Public Key "System Modulus" is n ( $n = P * Q$ ) and the second variable of the public key is E ( $1 < E < \phi(n)$ ), where E must be co-prime to  $\phi(n)$  and smaller than it using GCD. The private decryption keys "Eular function" are also divided into 2 variables: The first variable of the private key is  $\phi(n)$  where  $\phi(n) = (P - 1) * (Q - 1)$  and the second variable of the private key is d where d is the mod inverse of E and  $\phi(n)$  where  $d = E^{-1} \text{ mod } \phi(n)$ .

### **Step2: Encryption or Decryption**

In Encryption:

The cipher text is equal to the plaintext power of E modules N ( $C = M^E \text{ mod } N$ ).

In Decryption:

The plaintext is equal to the cipher text power of D modules N ( $P = C^D \text{ mod } N$ ).

The used functions are: CGD, Modules Inverse and Prime functions.

## Screen Shots:

A screenshot of a Windows application window titled "RSA Form.cs". The window contains C# code for a Windows Form named RSA\_Form. The code includes imports for System, Collections.Generic, ComponentModel, Data, Drawing, Linq, Text, and Windows.Forms. It defines a class RSA\_Form that initializes itself and handles the Load event to set text boxes. The Run\_Button\_Click event handles prime number input and checks if P and Q are prime. The RSA\_Algorithm method generates keys and performs encryption or decryption based on user input.

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RSA Form.cs* X RSA Form()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Cryptographic_Algorithms
{
    public partial class RSA_Form : Form
    {
        public RSA_Form()
        {
            InitializeComponent();
            this.Load += new EventHandler(RSA_Form_Load);
        }

        void RSA_Form_Load(object sender, EventArgs e)
        {
            TextBox.Text = "We found golden statues and stones, so find an honest person to sell them for us";
            P_TextBox.Text = "97";
            Q_TextBox.Text = "89";
        }

        private void Run_Button_Click(object sender, EventArgs e)
        {
            ulong P = 0, Q = 0, N = 0, PhiOfN = 0, E = 2, D = 0;

            //Large Prime Integer (P)
            if (!ulong.TryParse(P_TextBox.Text, out P))
            {
                MessageBox.Show("Insert Large Prime Integer (P)");
                return;
            }
            else
            {
                //Check if P is prime or not
                if (!isPrime(P))
                {
                    MessageBox.Show(P + " is not a Prime Integer");
                }
            }
        }

        //Large Prime Integer (Q)
        if (!ulong.TryParse(Q_TextBox.Text, out Q))
        {
            MessageBox.Show("Insert Large Prime Integer (Q)");
            return;
        }
        else
        {
            //Check if Q is prime or not
            if (!isPrime(Q))
            {
                MessageBox.Show(P + " is not a Prime Integer");
            }
        }
        RSA_Algorithm(P, Q, N, PhiOfN, E, D);
    }

    public void RSA_Algorithm(ulong P, ulong Q, ulong N, ulong PhiOfN, ulong E, ulong D)
    {
        //Generate Public encryption and Private Decryption keys
        GenerateKeys(P, Q, ref N, PhiOfN, ref E, ref D);
        //Check to Encrypt or Decrypt
        if (EncryptCheckBox.Checked)
        {
            ulong[] PlainTextInDecimal = null;
            //Encryption
            //Text to Decimal
            if (TextCheckBoxForText.Checked)
            {
                PlainTextInDecimal = StringToDecimal(textBox.Text);
            }
            //Binary to Decimal
            else if (BinaryCheckBoxForText.Checked)
        }
    }
}
```

100 % Error List Ready Ln 10 Col 35 Ch 35 INS

A screenshot of a Windows application window titled "RSA Form.cs". The window contains C# code for a Windows Form named RSA\_Form. The code includes imports for System, Collections.Generic, ComponentModel, Data, Drawing, Linq, Text, and Windows.Forms. It defines a class RSA\_Form that initializes itself and handles the Load event to set text boxes. The Run\_Button\_Click event handles prime number input and checks if P and Q are prime. The RSA\_Algorithm method generates keys and performs encryption or decryption based on user input.

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RSA Form.cs* X RSA Form()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Cryptographic_Algorithms
{
    public partial class RSA_Form : Form
    {
        public RSA_Form()
        {
            InitializeComponent();
            this.Load += new EventHandler(RSA_Form_Load);
        }

        void RSA_Form_Load(object sender, EventArgs e)
        {
            //Check if P is prime or not
            if (!isPrime(P))
            {
                MessageBox.Show(P + " is not a Prime Integer");
                return;
            }
            //Large Prime Integer (Q)
            if (!ulong.TryParse(Q_TextBox.Text, out Q))
            {
                MessageBox.Show("Insert Large Prime Integer (Q)");
                return;
            }
            else
            {
                //Check if Q is prime or not
                if (!isPrime(Q))
                {
                    MessageBox.Show(P + " is not a Prime Integer");
                }
            }
        }

        RSA_Algorithm(P, Q, N, PhiOfN, E, D);
    }

    public void RSA_Algorithm(ulong P, ulong Q, ulong N, ulong PhiOfN, ulong E, ulong D)
    {
        //Generate Public encryption and Private Decryption keys
        GenerateKeys(P, Q, ref N, PhiOfN, ref E, ref D);
        //Check to Encrypt or Decrypt
        if (EncryptCheckBox.Checked)
        {
            ulong[] PlainTextInDecimal = null;
            //Encryption
            //Text to Decimal
            if (TextCheckBoxForText.Checked)
            {
                PlainTextInDecimal = StringToDecimal(textBox.Text);
            }
            //Binary to Decimal
            else if (BinaryCheckBoxForText.Checked)
        }
    }
}
```

100 % Error List Ready Ln 10 Col 35 Ch 35 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RSA Form.cs\* RSA Form

```
using System;
using System.Text;
using System.Security.Cryptography;
using System.Linq;
using System.Collections.Generic;
using System.Windows.Forms;

public partial class RSAForm : Form
{
    public RSAForm()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        string plainText = textBox1.Text;
        string cipherText = textBox2.Text;
        string publicKeyN = textBox3.Text;
        string publicKeyE = textBox4.Text;
        string privateKeyD = textBox5.Text;

        if (string.IsNullOrEmpty(plainText) || string.IsNullOrEmpty(cipherText) || string.IsNullOrEmpty(publicKeyN) || string.IsNullOrEmpty(publicKeyE) || string.IsNullOrEmpty(privateKeyD))
        {
            MessageBox.Show("All fields must be filled!");
            return;
        }

        try
        {
            long n = long.Parse(publicKeyN);
            long e = long.Parse(publicKeyE);
            long d = long.Parse(privateKeyD);

            string decryptedText = Decrypt(cipherText, n, d);
            string encryptedText = Encrypt(plainText, n, e);

            textBox6.Text = decryptedText;
            textBox7.Text = encryptedText;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private string Encrypt(string plainText, long n, long e)
    {
        byte[] plainTextBytes = Encoding.UTF8.GetBytes(plainText);
        byte[] cipherTextBytes = new byte[plainTextBytes.Length];

        for (int i = 0; i < plainTextBytes.Length; i++)
        {
            cipherTextBytes[i] = (byte)(plainTextBytes[i] ^ e);
        }

        return Convert.ToBase64String(cipherTextBytes);
    }

    private string Decrypt(string cipherText, long n, long d)
    {
        byte[] cipherTextBytes = Convert.FromBase64String(cipherText);
        byte[] plainTextBytes = new byte[cipherTextBytes.Length];

        for (int i = 0; i < cipherTextBytes.Length; i++)
        {
            plainTextBytes[i] = (byte)(cipherTextBytes[i] ^ d);
        }

        return Encoding.UTF8.GetString(plainTextBytes);
    }
}
```

100 %

Error List Ready

Ln 10 Col 35 Ch 35 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RSA Form.cs\* RSA Form

```
using System;
using System.Text;
using System.Security.Cryptography;
using System.Linq;
using System.Collections.Generic;
using System.Windows.Forms;

public partial class RSAForm : Form
{
    public RSAForm()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        string plainText = textBox1.Text;
        string cipherText = textBox2.Text;
        string publicKeyN = textBox3.Text;
        string publicKeyE = textBox4.Text;
        string privateKeyD = textBox5.Text;

        if (string.IsNullOrEmpty(plainText) || string.IsNullOrEmpty(cipherText) || string.IsNullOrEmpty(publicKeyN) || string.IsNullOrEmpty(publicKeyE) || string.IsNullOrEmpty(privateKeyD))
        {
            MessageBox.Show("All fields must be filled!");
            return;
        }

        try
        {
            long n = long.Parse(publicKeyN);
            long e = long.Parse(publicKeyE);
            long d = long.Parse(privateKeyD);

            string decryptedText = Decrypt(cipherText, n, d);
            string encryptedText = Encrypt(plainText, n, e);

            textBox6.Text = decryptedText;
            textBox7.Text = encryptedText;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private string Encrypt(string plainText, long n, long e)
    {
        byte[] plainTextBytes = Encoding.UTF8.GetBytes(plainText);
        byte[] cipherTextBytes = new byte[plainTextBytes.Length];

        for (int i = 0; i < plainTextBytes.Length; i++)
        {
            cipherTextBytes[i] = (byte)(plainTextBytes[i] ^ e);
        }

        return Convert.ToBase64String(cipherTextBytes);
    }

    private string Decrypt(string cipherText, long n, long d)
    {
        byte[] cipherTextBytes = Convert.FromBase64String(cipherText);
        byte[] plainTextBytes = new byte[cipherTextBytes.Length];

        for (int i = 0; i < cipherTextBytes.Length; i++)
        {
            plainTextBytes[i] = (byte)(cipherTextBytes[i] ^ d);
        }

        return Encoding.UTF8.GetString(plainTextBytes);
    }

    public void GenerateKeys(long p, long q, ref long n, ref long phiOfN, ref long e, ref long d)
    {
        //Step 1: Generate Public Key
        //The First Part of the Public Key "System Modulus"
        n = p * q;

        //Step 2: Calculate the totient of n
        phiOfN = (p - 1) * (q - 1);

        //Step 3: Find the modular inverse of e (mod phi(n))
        d = ExtendedGCD(e, phiOfN);
    }

    private long ExtendedGCD(long a, long b)
    {
        if (b == 0)
        {
            return a;
        }
        else
        {
            return ExtendedGCD(b, a % b);
        }
    }
}
```

100 %

Error List Ready

Ln 10 Col 35 Ch 35 INS

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RSA Form.cs* X RSA Form
Cryptographic_AlgorithmsRSA.Form
public void GenerateKeys(ulong P, ulong Q, ref ulong N, ulong PhiOfN, ref ulong E, ref ulong D)
{
    //Step 1: Generate Public Key
    //The First Part of the Public Key "System Modulus"
    N = P * Q;

    //Step 2: Generate Private Key "Eular function"
    PhiOfN = (P - 1) * (Q - 1);
    //The Second Part of the Public Key is E (1 < E < PhiOfN)
    while (E < PhiOfN)
    {
        // e must be co-prime to phi and
        // smaller than phi.
        if (GCD(E, PhiOfN) == 1)
            break;
        else
            E++;
    }
    //Public Keys = (N, E)
    D = modInverse(E, PhiOfN);
    //Private Keys = (N, D)
}

public void Encryption(ulong[] PlainTextInDecimal, ulong N, ulong E)
{
    //CipherText = Math.Pow(PlainText, E) % N;
    string CipherText = "";
    for (int i = 0; i < PlainTextInDecimal.Length; i++)
    {
        ulong Total = 1;
        for (ulong n = 0; n < E; n++)
        {
            Total = (Total * PlainTextInDecimal[i]) % N;
        }
        if (TextCheckBoxForResult.Checked)
            CipherText += DecimalToChar(Total) + " ";
        else if (BinaryCheckBoxForResult.Checked)
            CipherText += DecimalToBinary(Total) + " ";
        else if (HexCheckBoxForResult.Checked)
            CipherText += DecimalToHex(Total) + " ";
    }
}
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RSA Form.cs* X RSA Form
Cryptographic_AlgorithmsRSA.Form
public void Encryption(ulong[] PlainTextInDecimal, ulong N, ulong E)
{
    //CipherText = Math.Pow(PlainText, E) % N;
    string CipherText = "";
    for (int i = 0; i < PlainTextInDecimal.Length; i++)
    {
        ulong Total = 1;
        for (ulong n = 0; n < E; n++)
        {
            Total = (Total * PlainTextInDecimal[i]) % N;
        }
        if (TextCheckBoxForResult.Checked)
            CipherText += DecimalToChar(Total) + " ";
        else if (BinaryCheckBoxForResult.Checked)
            CipherText += DecimalToBinary(Total) + " ";
        else if (HexCheckBoxForResult.Checked)
            CipherText += DecimalToHex(Total) + " ";
        else if (DecimalCheckBoxForResult.Checked)
            CipherText += Total + " ";
        else
        {
            MessageBox.Show("You did not choose the type of Result CipherText or Result is Wrong !!!");
            return;
        }
    }
    ResultBox.Text = CipherText;
}

public void Decryption(ulong[] CipherTextInDecimal, ulong N, ulong D)
{
    //PlainText = Math.Pow(CipherText, D) % N;
    string PlainText = "";
    for (int i = 0; i < CipherTextInDecimal.Length; i++)
    {
        ulong Total = 1;
        for (ulong n = 0; n < D; n++)
        {
            Total = (Total * CipherTextInDecimal[i]) % N;
        }
        if (TextCheckBoxForResult.Checked)
            PlainText += DecimalToChar(Total);
```

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RSA Form.cs\* RSA Form

```
public void Decryption(ulong[] CipherTextInDecimal, ulong N, ulong D)
{
    //PlainText = Math.Pow(CipherText, D) % N;
    string PlainText = "";
    for (int i = 0; i < CipherTextInDecimal.Length; i++)
    {
        ulong Total = 1;
        for (ulong n = 0; n < D; n++)
        {
            Total = (Total * CipherTextInDecimal[i]) % N;
        }
        if (TextCheckBoxForResult.Checked)
            PlainText += DecimalToChar(Total);
        else if (BinaryCheckBoxForResult.Checked)
            PlainText += DecimalToBinary(Total) + " ";
        else if (HexCheckBoxForResult.Checked)
            PlainText += DecimalToHex(Total) + " ";
        else if (DecimalCheckBoxForResult.Checked)
            PlainText += Total + " ";
        else
        {
            MessageBox.Show("You did not choose the type of Result PlainText or Result is Wrong !!!");
            return;
        }
    }
    ResultBox.Text = PlainText;
}

public ulong GCD(ulong a, ulong b)
{
    ulong Reminder;
    while (b != 0)
    {
        Reminder = a % b;
        a = b;
        b = Reminder;
    }
    return a;
}

public ulong modInverse(ulong a, ulong m)
```

100% Error List Ready Ln 10 Col 35 Ch 35 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RSA Form.cs\* RSA Form

```
public ulong modInverse(ulong a, ulong m)
{
    a = a % m;
    for (ulong x = 1; x < m; x++)
        if ((a * x) % m == 1)
            return x;
    return 1;
}

public ulong[] StringToDecimal(string data)
{
    ulong[] Total = new ulong[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = data[i];
    }
    return Total;
}

public string DecimalToString(ulong data)
{
    return Convert.ToString(data);
}

public string DecimalToChar(ulong data)
{
    return Convert.ToChar(data).ToString();
}

public ulong CharToDecimal(string data)
{
    return Convert.ToInt64(data);
}

public string DecimalToBinary(ulong data)
{
    return Convert.ToString((long)data, 2);
}

public ulong[] BinaryToDecimal(string[] data)
{
    ulong[] Total = new ulong[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = Convert.ToInt64(data[i], 2);
    }
}
```

100% Error List Ready Ln 10 Col 35 Ch 35 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RSA Form.cs\* RSA Form

```
public ulong[] BinaryToDecimal(string[] data)
{
    ulong[] Total = new ulong[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = Convert.ToInt64(data[i], 2);
    }

    return Total;
}

public ulong[] HexToDecimal(string[] data)
{
    ulong[] Total = new ulong[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = Convert.ToInt64(data[i], 16);
    }

    return Total;
}

public string DecimalToHex(ulong data)
{
    return Convert.ToString((long)data, 16);
}

public bool isPrime(ulong n)
{
    // Corner case
    if (n <= 1)
        return false;

    // Check from 2 to n-1
    for (ulong i = 2; i < n; i++)
        if (n % i == 0)
            return false;

    return true;
}

/// <summary>
/// Note: unneeded to be Explained

```

100% Error List Ready Ln 10 Col 35 Ch 35 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RSA Form.cs\* RSA Form

```
/// <summary>
/// Note: unneeded to be Explained
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
/// </Text>
private void TextCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (TextCheckBoxForText.Checked)
    {
        BinaryCheckBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
        DecimalCheckBoxForText.Checked = false;
    }
}

private void BinaryCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForText.Checked)
    {
        TextCheckBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
        DecimalCheckBoxForText.Checked = false;
    }
}

private void HexCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForText.Checked)
    {
        TextCheckBoxForText.Checked = false;
        BinaryCheckBoxForText.Checked = false;
        DecimalCheckBoxForText.Checked = false;
    }
}

private void DecimalCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (DecimalCheckBoxForText.Checked)
    {
        TextCheckBoxForText.Checked = false;
        BinaryCheckBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
    }
}
```

100% Error List Ready Ln 10 Col 35 Ch 35 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RSA Form.cs\* RSA Form

```
private void DecimalCheckBoxForText_CheckedChanged(object sender, EventArgs e)
{
    if (DecimalCheckBoxForText.Checked)
    {
        TextCheckBoxForText.Checked = false;
        BinaryCheckBoxForText.Checked = false;
        HexCheckBoxForText.Checked = false;
    }
}
//En/Decrypt
private void EncryptCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (EncryptCheckBox.Checked)
    {
        DecryptCheckBox.Checked = false;
    }
}
private void DecryptCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (DecryptCheckBox.Checked)
    {
        EncryptCheckBox.Checked = false;
    }
}
//Result
private void TextCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (TextCheckBoxForResult.Checked)
    {
        BinaryCheckBoxForResult.Checked = false;
        HexCheckBoxForResult.Checked = false;
        DecimalCheckBoxForResult.Checked = false;
    }
}
private void BinaryCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForResult.Checked)
    {
        TextCheckBoxForResult.Checked = false;
        HexCheckBoxForResult.Checked = false;
        DecimalCheckBoxForResult.Checked = false;
    }
}
```

100 %

Error List Ready Ln 10 Col 35 Ch 35 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

RSA Form.cs\* RSA Form

```
HexCheckBoxForResult.Checked = false;
DecimalCheckBoxForResult.Checked = false;
}
private void HexCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForResult.Checked)
    {
        TextCheckBoxForResult.Checked = false;
        BinaryCheckBoxForResult.Checked = false;
        DecimalCheckBoxForResult.Checked = false;
    }
}
private void DecimalCheckBoxForResult_CheckedChanged(object sender, EventArgs e)
{
    if (DecimalCheckBoxForResult.Checked)
    {
        TextCheckBoxForResult.Checked = false;
        BinaryCheckBoxForResult.Checked = false;
        HexCheckBoxForResult.Checked = false;
    }
}
```

100 %

Error List Ready Ln 10 Col 35 Ch 35 INS

## **ElGamal:**

### **For Encryption:**

The inputs are plain text, a large prime integer ( $q$ ), a primitive root ( $a$ ), and a Secret Key ( $X_A$ ) for user A where each one of them is inserted in its own text box. After inserting the plain text choose the type of the text (Text, Binary, Hex, or Decimal). Then click on “Encrypt” button to get the result in the “CipherText” text box.

Steps:

#### **Step 1:**

We have user A and user B. In step 1, user A chooses large prime integer ( $q$ ), a primitive root ( $a$ ), and a Secret Key ( $X_A$ ) then user A compute a public key using the primitive root ( $a$ ), the Secret Key ( $X_A$ ), and the large prime integer ( $q$ ), where public key ( $Y_A$ )  $Y_A = a^{X_A} \text{ mod } q$ . The chosen large prime integer and the Secret Key ( $X_A$ ) are shared between the two users.

#### **Step 2: Encryption**

User B select random integer  $k$  ( $1 \leq k < q$ ), then compute one-time key ( $K = Y_A^k \text{ mod } q$ ). Then compute the cipher text, where the cipher text is equal to  $C_1$  and  $C_2$ , since  $C_1 = a^k \text{ mod } q$  and  $C_2 = KM \text{ mod } q$ , where  $M$  is the plain text and cipher text is equal to  $(C_1, C_2)$ .

### **For Decryption:**

The inputs are cipher text, a large prime integer ( $q$ ), and a Secret Key of user A where each one of them is inserted in its own text box. After inserting the cipher text choose the type of the cipher text (Text, Binary, Hex, or Decimal). Then click on “Decrypt” button to get the result in the “PlainText” text box.

Steps:

#### **Step 1:**

User A uses the same large prime integer ( $q$ ) and the Secret Key of user A ( $X_A$ ).

#### **Step 2: Decryption**

User A compute one-time key using  $C_1$  of the cipher text, his secret key and the large prime integer ( $q$ ) ( $K = C_1^{X_A} \text{ mod } q$ ). Then compute the  $K$  inverse by using  $K$  and  $q$  in Modules inverse function. Then compute the plain text using  $C_2$ ,  $K$  inverse, and  $q$  where ( $P = M = C_2 * K^{-1} \text{ mod } q$ ).

The used Functions are: Modules inverse, Prime, Primitive and GCD functions.

## Screen Shots:

This screenshot shows the initial state of the RSA Form.cs code in Visual Studio. The code defines a partial class ELGamil\_Form that initializes components and handles the Load event to set initial text values for four text boxes. It also contains an Encryption\_Click event handler that performs key generation and encryption logic.

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGamil Form.cs RSA Form.cs
Cryptographic_Algorithms.ELGamil_Form
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Cryptographic_Algorithms
{
    public partial class ELGamil_Form : Form
    {
        public ELGamil_Form()
        {
            InitializeComponent();
            this.Load += new EventHandler(ELGamil_Form_Load);
        }

        void ELGamil_Form_Load(object sender, EventArgs e)
        {
            PlainTextBox.Text = "Computer Security is Important.";
            q_TextBox.Text = "1613";
            a_TextBox.Text = "1119";
            XA_TextBox.Text = "1229";
        }

        private void Encryption_Button_Click(object sender, EventArgs e)
        {
            int q = 0, a = 0, XA = 0, YA = 0, C1 = 1;
            int[] C2 = null;
            //Keys Generation by user A
            bool isReady = UserA_GenerateKeys(ref q, ref a, ref XA, ref YA);

            if (isReady)
            {
                int[] PlainTextInDecimal = null;
                //Text to Decimal
                if (TextCheckBoxForTextEncryption.Checked)
                {
                    PlainTextInDecimal = StringToDecimal(PlainTextBox.Text);
                } //Binary to Decimal
            }
        }
    }
}
```

This screenshot shows the continuation of the RSA Form.cs code, specifically the Encryption\_Click event handler. It includes logic for handling different input types (Binary, Hex, Decimal) and performing the encryption process using the generated keys.

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGamil Form.cs RSA Form.cs
Cryptographic_Algorithms.ELGamil_Form
//Binary to Decimal
else if (BinaryCheckBoxForTextEncryption.Checked)
{
    string[] SplitBinaryText = PlainTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    PlainTextInDecimal = Binary.ToDecimal(SplitBinaryText);
} //Hex to Decimal
else if (HexCheckBoxForTextEncryption.Checked)
{
    string[] SplitHexText = PlainTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    PlainTextInDecimal = Hex.ToDecimal(SplitHexText);
} //Decimal to Decimal
else if (DecimalCheckBoxForTextEncryption.Checked)
{
    string[] SplitDecimalText = PlainTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    PlainTextInDecimal = new int[SplitDecimalText.Length];
    for (int i = 0; i < SplitDecimalText.Length; i++)
    {
        PlainTextInDecimal[i] = Convert.ToInt32(SplitDecimalText[i]);
    }
} //Error
else
{
    MessageBox.Show("You did not choose the type of PlainText or PlainText is Wrong !!!");
    return;
}
//Encryption
Encryption(YA, q, a, PlainTextInDecimal, ref C1, ref C2);
}

public bool UserA_GenerateKeys(ref int q, ref int a, ref int XA, ref int YA)
{
    //Step 1:
    //Large Prime Integer
    if (!int.TryParse(q_TextBox.Text, out q))
    {
        MessageBox.Show("Insert Large Prime Integer (q)");
        return false;
    }
    else
    {
        if (!isPrime(q))

```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGamal_Form.cs* RSA_Form.cs*
Cryptographic_Algorithms\ElGamal_Form
{
    if (!isPrime(q))
    {
        MessageBox.Show(q + " is not a Prime Integer");
        return false;
    }
    //being a primitive root mod q
    //for n=1 to q-1
    // a^n mod q = [1 to q-1]
    if (!int.TryParse(a_TextBox.Text, out a))
    {
        MessageBox.Show("Insert a Primitive root (a)");
        return false;
    }
    else
    {
        if (!isPrimitive(a, q))
        {
            MessageBox.Show(a + " is not a Primitive root");
            return false;
        }
    }
    //Step 2:
    // we have 2 users A & B
    //User A Select Secret Key ( 1 < XA < q - 1)
    if (!int.TryParse(XA_TextBox.Text, out XA))
    {
        MessageBox.Show("Insert Secret Key for User A");
        return false;
    }
    else
    {
        if (!(1 < XA && XA < q - 1))
        {
            MessageBox.Show("Insert Secret Key for User A Less than the Large Prime Integer (q)");
            return false;
        }
    }
    //Step 3:
    //User A Compute Public Key
}
100 %
Error List
Ready Ln 5 Col 22 Ch 22 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGamal_Form.cs* RSA_Form.cs*
Cryptographic_Algorithms\ElGamal_Form
{
    //Step 3:
    //User A Compute Public Key
    VA = 1;
    for (int n = 0; n < XA; n++)
    {
        YA = (YA * a) % q;
    }
    return true;
}

public void Encryption(int YA, int q, int a, int[] PlainTextInDecimal, ref int C1, ref int[] C2)
{
    //Step 3:
    //User B select random inter k (1 <= k < q)
    Random R = new Random();
    int k = R.Next(1, Convert.ToInt32(q));
    //User B compute one-time Key K = YA^k mod q
    int K = 1;
    for (int n = 0; n < k; n++)
    {
        K = (K * YA) % q;
    }
    //User B compute C1 and C2
    //C1 = a^k mod q
    string CipherText = "";// CipherText = ( ";
    for (int n = 0; n < k; n++)
    {
        C1 = (C1 * a) % q;
        if (TextCheckBoxForResultEncryption.Checked)
            CipherText += DecimalToChar(C1) + " ";
        else if (BinaryCheckBoxForResultEncryption.Checked)
            CipherText += DecimalToBinary(C1) + " ";
        else if (HexCheckBoxForResultEncryption.Checked)
            CipherText += DecimalToHex(C1) + " ";
        else if (DecimalCheckBoxForResultEncryption.Checked)
            CipherText += C1 + " ";
    }
    if (TextCheckBoxForResultEncryption.Checked)
        CipherText += DecimalToChar(C1) + " ";
    else if (BinaryCheckBoxForResultEncryption.Checked)
        CipherText += DecimalToBinary(C1) + " ";
    else if (HexCheckBoxForResultEncryption.Checked)
        CipherText += DecimalToHex(C1) + " ";
    else if (DecimalCheckBoxForResultEncryption.Checked)
        CipherText += C1 + " ";
    else
    {
        MessageBox.Show("You did not choose the type of Result CipherText or Result is Wrong !!!");
        return;
    }
}
100 %
Error List
Ready Ln 5 Col 22 Ch 22 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGAMAL FORM.cs X RSA FORM.cs*
Cryptographic_Algorithms.ELGAMAL_Form
}
//C2 = KM mod q
C2 = new int[plainTextInDecimal.Length];
for (int i = 0; i < plainTextInDecimal.Length; i++)
{
    C2[i] = (K * plainTextInDecimal[i]) % q;
    if (textCheckBoxForResultEncryption.Checked)
        CipherText += DecimalToString(C2[i]) + " ";
    else if (BinaryCheckBoxForResultEncryption.Checked)
        CipherText += DecimalToBinary(C2[i]) + " ";
    else if (HexCheckBoxForResultEncryption.Checked)
        CipherText += DecimalToHex(C2[i]) + " ";
    else if (DecimalCheckBoxForResultEncryption.Checked)
        CipherText += C2[i] + " ";
    else
    {
        MessageBox.Show("You did not choose the type of Result CipherText or Result is Wrong !!!");
        return;
    }
}
// CipherText += "";
encryptResultTextBox.Text = CipherText;

//User B sends (C1, C2) to User A (returned by reference)
}

public void Decryption_Button_Click(object sender, EventArgs e)
{
    int q = 0, XA = 0, C1 = 1;
    int[] C2 = null;

    //Step 1:
    //Large Prime Integer
    if (!int.TryParse(q_TextBox2.Text, out q))
    {
        MessageBox.Show("Insert Large Prime Integer");
        return;
    }
    else
    {
        if (!isPrime(q))
        {
            if (!isPrime(q))
            {
                MessageBox.Show("Insert a Prime Integer");
                return;
            }
        }
        //Step 2:
        //we have 2 users A & B
        //User A Select Secret Key ( 1 < XA < q - 1)
        if (!int.TryParse(XA_TextBox2.Text, out XA))
        {
            MessageBox.Show("Insert Secret Key for User A");
            return;
        }
        else
        {
            if (!(1 < XA && XA < q - 1))
            {
                MessageBox.Show("Insert Secret Key for User A Less than the Large Prime Integer");
                return;
            }
        }
        int[] CipherTextInDecimal = null;
        //Text to Decimal
        if (textCheckBoxForTextDecryption.Checked)
        {
            string[] SplitText = cipherTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
            CipherTextInDecimal = new int[SplitText.Length];
            for (int i = 0; i < SplitText.Length; i++)
            {
                CipherTextInDecimal[i] = CharToDecimal(SplitText[i]);
            }
        }
        //Binary to Decimal
        else if (binaryCheckBoxForTextDecryption.Checked)
        {
            string[] SplitBinaryText = cipherTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
            CipherTextInDecimal = BinaryToDecimal(SplitBinaryText);
        }
        //Hex to Decimal
        else if (hexCheckBoxForTextDecryption.Checked)
        {
    }
}
100 %
Error List
Ready Ln 5 Col 22 Ch 22 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGAMAL FORM.cs X RSA FORM.cs*
Cryptographic_Algorithms.ELGAMAL_Form
}
else
{
    if (!isPrime(q))
    {
        MessageBox.Show("Insert a Prime Integer");
        return;
    }
}
//Step 2:
//we have 2 users A & B
//User A Select Secret Key ( 1 < XA < q - 1)
if (!int.TryParse(XA_TextBox2.Text, out XA))
{
    MessageBox.Show("Insert Secret Key for User A");
    return;
}
else
{
    if (!(1 < XA && XA < q - 1))
    {
        MessageBox.Show("Insert Secret Key for User A Less than the Large Prime Integer");
        return;
    }
}
int[] CipherTextInDecimal = null;
//Text to Decimal
if (textCheckBoxForTextDecryption.Checked)
{
    string[] SplitText = cipherTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    CipherTextInDecimal = new int[SplitText.Length];
    for (int i = 0; i < SplitText.Length; i++)
    {
        CipherTextInDecimal[i] = CharToDecimal(SplitText[i]);
    }
}
//Binary to Decimal
else if (binaryCheckBoxForTextDecryption.Checked)
{
    string[] SplitBinaryText = cipherTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    CipherTextInDecimal = BinaryToDecimal(SplitBinaryText);
}
//Hex to Decimal
else if (hexCheckBoxForTextDecryption.Checked)
{
}
100 %
Error List
Ready Ln 217 Col 14 Ch 14 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGmail Form.cs X RSA Form.cs
C:\Cryptographic_Algorithms\ElGmail_Form
    //Hex to Decimal
    else if (HexCheckBoxForTextDecryption.Checked)
    {
        string[] SplitHexText = CipherTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        CipherTextInDecimal = HexToDecimal(SplitHexText);
    }
    //Decimal to Decimal
    else if (DecimalCheckBoxForTextDecryption.Checked)
    {
        string[] SplitDecimalText = CipherTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        CipherTextInDecimal = new int[SplitDecimalText.Length];
        for (int i = 0; i < SplitDecimalText.Length; i++)
        {
            CipherTextInDecimal[i] = Convert.ToInt32(SplitDecimalText[i]);
        }
    }
    //Error
    else
    {
        MessageBox.Show("You did not choose the type of CipherText or CipherText is Wrong !!!");
        return;
    }
    //Inserted CipherText (C1, C2)
    // C1
    C1 = CipherTextInDecimal[0];
    //C2
    C2 = new int[cipherTextInDecimal.Length - 1];
    for (int i = 1; i < cipherTextInDecimal.Length; i++)
    {
        C2[i - 1] = Convert.ToInt32(CipherTextInDecimal[i]);
    }
    //Decryption
    Decryption(C1, C2, XA, q);
}

public void Decryption( int C1, int[] C2, int XA, int q)
{
    //Calculate K = C1^XA mod q
    int K = 1;
    for (int n = 0; n < XA; n++)
    {
        K = (K * C1) % q;
    }
    //Calculate K inverse (K^-1)
    int InversedK = modInverse(K, q);
    string PlainText = "";
    int Total = 0;
    //Calculate Plain Text
    for (int i = 0; i < C2.Length; i++)
    {
        Total = (C2[i] * InversedK) % q;
        if (TextCheckBoxForResultDecryption.Checked)
            PlainText += DecimalToChar(Total);
        else if (BinaryCheckBoxForResultDecryption.Checked)
            PlainText += DecimalToBinary(Total) + " ";
        else if (HexCheckBoxForResultDecryption.Checked)
            PlainText += DecimalToHex(Total) + " ";
        else if (DecimalcheckboxForResultDecryption.Checked)
            PlainText += Total + " ";
    }
    DecryptResultTextBox.Text = PlainText;
}

public int modInverse(int a, int m)
{
    a = a % m;
    for (int x = 1; x < m; x++)
        if ((a * x) % m == 1)
            return x;
    return 1;
}

public int GCD(int a, int b)
{
    int Reminder;
    while (b != 0)
    {
        Reminder = a % b;
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGmail Form.cs X RSA Form.cs
C:\Cryptographic_Algorithms\ElGmail_Form
    }
    //Calculate K inverse (K^-1)
    int InversedK = modInverse(K, q);
    string PlainText = "";
    int Total = 0;
    //Calculate Plain Text
    for (int i = 0; i < C2.Length; i++)
    {
        Total = (C2[i] * InversedK) % q;
        if (TextCheckBoxForResultDecryption.Checked)
            PlainText += DecimalToChar(Total);
        else if (BinaryCheckBoxForResultDecryption.Checked)
            PlainText += DecimalToBinary(Total) + " ";
        else if (HexCheckBoxForResultDecryption.Checked)
            PlainText += DecimalToHex(Total) + " ";
        else if (DecimalcheckboxForResultDecryption.Checked)
            PlainText += Total + " ";
    }
    DecryptResultTextBox.Text = PlainText;
}

public int modInverse(int a, int m)
{
    a = a % m;
    for (int x = 1; x < m; x++)
        if ((a * x) % m == 1)
            return x;
    return 1;
}

public int GCD(int a, int b)
{
    int Reminder;
    while (b != 0)
    {
        Reminder = a % b;
```

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

ELGamal Form.cs RSA Form.cs

Cryptographic\_Algorithms.ElGamal\_Form

```
public int GCD(int a, int b)
{
    int Reminder;
    while (b != 0)
    {
        Reminder = a % b;
        a = b;
        b = Reminder;
    }
    return a;
}

public bool isPrime(int n)
{
    // Corner case
    if (n <= 1)
        return false;

    // Check from 2 to n-1
    for (int i = 2; i < n; i++)
        if (n % i == 0)
            return false;

    return true;
}

public bool isPrimitive(int a, int q)
{
    int Total = 1;
    int[] sort = new int[q - 1];
    for (int n = 0; n < q - 1; n++)
    {
        Total = (Total * a) % q;
        sort[n] = Total;
    }
    Array.Sort(sort);
    for (int n = 0; n < sort.Length; n++)
    {
        if (sort[n] != n + 1)
        {
            return false;
        }
    }
}
```

100 %

Error List Ready Ln 217 Col 14 Ch 14 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

ELGamal Form.cs RSA Form.cs

Cryptographic\_Algorithms.ElGamal\_Form

```
public int[] StringToDecimal(string data)
{
    int[] Total = new int[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = data[i];
    }
    return Total;
}

public string DecimalToString(int data)
{
    return Convert.ToString(data);
}

public string DecimalToChar(int data)
{
    return Convert.ToChar(data).ToString();
}

public int CharToDecimal(string data)
{
    return Convert.ToInt32(Convert.ToChar(data));
}

public string DecimalToBinary(int data)
{
    return Convert.ToString((long)data, 2);
}

public int[] BinaryToDecimal(string[] data)
{
    int[] Total = new int[data.Length];
    for (int i = 0; i < data.Length; i++)
    {
        Total[i] = Convert.ToInt32(data[i], 2);
    }
    return Total;
}

public int[] HexToDecimal(string[] data)
{
    int[] Total = new int[data.Length];
```

100 %

Error List Ready Ln 217 Col 14 Ch 14 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

ELGmail Form.cs\* RSA Form.cs\*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Cryptographic_Algorithms;

public partial class ElGmail_Form : Form
{
    public ElGmail_Form()
    {
        InitializeComponent();
    }

    private void HexToDecimal(string[] data)
    {
        int[] Total = new int[data.Length];
        for (int i = 0; i < data.Length; i++)
        {
            Total[i] = Convert.ToInt32(data[i], 16);
        }
        return Total;
    }

    private string DecimalToHex(int data)
    {
        return Convert.ToString((long)data, 16);
    }

    //<summary>
    //<Note: unneeded to be Explained>
    //</summary>
    //<param name="sender"></param>
    //<param name="e"></param>
    //</>
    //Encryption
    //<text>
    private void TextCheckBoxForTextEncryption_CheckedChanged(object sender, EventArgs e)
    {
        if (TextCheckBoxForTextEncryption.Checked)
        {
            BinaryCheckBoxForTextEncryption.Checked = false;
            HexCheckBoxForTextEncryption.Checked = false;
            DecimalCheckBoxForTextEncryption.Checked = false;
        }
    }

    private void BinaryCheckBoxForTextEncryption_CheckedChanged(object sender, EventArgs e)
    {
        if (BinaryCheckBoxForTextEncryption.Checked)
        {
            TextCheckBoxForTextEncryption.Checked = false;
            HexCheckBoxForTextEncryption.Checked = false;
            DecimalCheckBoxForTextEncryption.Checked = false;
        }
    }

    private void HexCheckBoxForTextEncryption_CheckedChanged(object sender, EventArgs e)
    {
    }
}
```

100 %

Error List Ready Ln 217 Col 14 Ch 14 INS

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen

ELGmail Form.cs\* RSA Form.cs\*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Cryptographic_Algorithms;

public partial class RSA_Form : Form
{
    private void HexCheckBoxForTextEncryption_CheckedChanged(object sender, EventArgs e)
    {
        if (HexCheckBoxForTextEncryption.Checked)
        {
            TextCheckBoxForTextEncryption.Checked = false;
            BinaryCheckBoxForTextEncryption.Checked = false;
            DecimalCheckBoxForTextEncryption.Checked = false;
        }
    }

    private void DecimalCheckBoxForTextEncryption_CheckedChanged(object sender, EventArgs e)
    {
        if (DecimalCheckBoxForTextEncryption.Checked)
        {
            TextCheckBoxForTextEncryption.Checked = false;
            BinaryCheckBoxForTextEncryption.Checked = false;
            HexCheckBoxForTextEncryption.Checked = false;
        }
    }

    //Result
    private void TextCheckBoxForResultEncryption_CheckedChanged(object sender, EventArgs e)
    {
        if (TextCheckBoxForResultEncryption.Checked)
        {
            BinaryCheckBoxForResultEncryption.Checked = false;
            HexCheckBoxForResultEncryption.Checked = false;
            DecimalCheckBoxForResultEncryption.Checked = false;
        }
    }

    private void BinaryCheckBoxForResultEncryption_CheckedChanged(object sender, EventArgs e)
    {
        if (BinaryCheckBoxForResultEncryption.Checked)
        {
            TextCheckBoxForResultEncryption.Checked = false;
            HexCheckBoxForResultEncryption.Checked = false;
            DecimalCheckBoxForResultEncryption.Checked = false;
        }
    }

    private void HexCheckBoxForResultEncryption_CheckedChanged(object sender, EventArgs e)
    {
        if (HexCheckBoxForResultEncryption.Checked)
        {
            TextCheckBoxForResultEncryption.Checked = false;
    }}
```

100 %

Error List Ready Ln 217 Col 14 Ch 14 INS

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGmail Form.cs X RSA Form.cs
C:\Cryptographic_Algorithms\ElGmail\Form.cs - | Decryption_Button_Click(object sender, EventArgs e)
if (HexCheckBoxForResultEncryption.Checked)
{
    TextCheckBoxForResultEncryption.Checked = false;
    BinaryCheckBoxForResultEncryption.Checked = false;
    DecimalCheckBoxForResultEncryption.Checked = false;
}
private void DecimalCheckBoxForResultEncryption_CheckedChanged(object sender, EventArgs e)
{
    if (DecimalCheckBoxForResultEncryption.Checked)
    {
        TextCheckBoxForResultEncryption.Checked = false;
        BinaryCheckBoxForResultEncryption.Checked = false;
        HexCheckBoxForResultEncryption.Checked = false;
    }
}
//Decryption
//Text
private void TextCheckBoxForTextDecryption_CheckedChanged(object sender, EventArgs e)
{
    if (TextCheckBoxForTextDecryption.Checked)
    {
        BinaryCheckBoxForTextDecryption.Checked = false;
        HexCheckBoxForTextDecryption.Checked = false;
        DecimalCheckBoxForTextDecryption.Checked = false;
    }
}
private void BinaryCheckBoxForTextDecryption_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForTextDecryption.Checked)
    {
        TextCheckBoxForTextDecryption.Checked = false;
        HexCheckBoxForTextDecryption.Checked = false;
        DecimalCheckBoxForTextDecryption.Checked = false;
    }
}
private void HexCheckBoxForTextDecryption_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForTextDecryption.Checked)
    {
        TextCheckBoxForTextDecryption.Checked = false;
        BinaryCheckBoxForTextDecryption.Checked = false;
    }
}
100 %
Ready Error List Ln 217 Col 14 Ch 14 INS
```

```
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGmail Form.cs X RSA Form.cs*
C:\Cryptographic_Algorithms\ElGmail\Form.cs - | Decryption_Button_Click(object sender, EventArgs e)
BinaryCheckBoxForTextDecryption.Checked = false;
DecimalCheckBoxForTextDecryption.Checked = false;
}
private void DecimalCheckBoxForTextDecryption_CheckedChanged(object sender, EventArgs e)
{
    if (DecimalCheckBoxForTextDecryption.Checked)
    {
        TextCheckBoxForTextDecryption.Checked = false;
        BinaryCheckBoxForTextDecryption.Checked = false;
        HexCheckBoxForTextDecryption.Checked = false;
    }
}
//Result
private void TextCheckBoxForResultDecryption_CheckedChanged(object sender, EventArgs e)
{
    if (TextCheckBoxForResultDecryption.Checked)
    {
        BinaryCheckBoxForResultDecryption.Checked = false;
        HexCheckBoxForResultDecryption.Checked = false;
        DecimalCheckBoxForResultDecryption.Checked = false;
    }
}
private void BinaryCheckBoxForResultDecryption_CheckedChanged(object sender, EventArgs e)
{
    if (BinaryCheckBoxForResultDecryption.Checked)
    {
        TextCheckBoxForResultDecryption.Checked = false;
        HexCheckBoxForResultDecryption.Checked = false;
        DecimalCheckBoxForResultDecryption.Checked = false;
    }
}
private void HexCheckBoxForResultDecryption_CheckedChanged(object sender, EventArgs e)
{
    if (HexCheckBoxForResultDecryption.Checked)
    {
        TextCheckBoxForResultDecryption.Checked = false;
        BinaryCheckBoxForResultDecryption.Checked = false;
        DecimalCheckBoxForResultDecryption.Checked = false;
    }
}
private void DecimalCheckBoxForResultDecryption_CheckedChanged(object sender, EventArgs e)
100 %
Ready Error List Ln 217 Col 14 Ch 14 INS
```

The screenshot shows a code editor window with two tabs: "ElGamal Form.cs\*" and "RSA Form.cs\*". The "ElGamal Form.cs\*" tab is active, displaying the following C# code:

```
    }
    private void DecimalCheckBoxForResultDecryption_CheckedChanged(object sender, EventArgs e)
    {
        if (DecimalCheckBoxForResultDecryption.Checked)
        {
            TextCheckBoxForResultDecryption.Checked = false;
            BinaryCheckBoxForResultDecryption.Checked = false;
            HexCheckBoxForResultDecryption.Checked = false;
        }
    }
}
```

The code is part of a class named "Cryptographic\_Algorithms.ElGamal\_Form". A tooltip "Decryption\_Button\_Click(object sender, EventArgs e)" is visible near the bottom right of the code area. The status bar at the bottom shows "100 %", "Ready", "Ln 217", "Col 14", "Ch 14", and "INS".

## Outputs:

### S-DES:

The output is the plain text or the cipher text in a text box. Before Clicking on the “En/Decrypt” button choose the type of the text (Text, Binary, or Hex). Then click on En/Decrypt button to get the result in the “Result” text box.

This screenshot shows the S-DES Algorithm application window. The title bar says "S DES Form". The main title is "S-DES Algorithm".  
Text: (Plain Text or Cipher Text)  
Input: Hitler attacked russia.  
Encryption settings:  
 Encrypt  Decrypt  Text  Binary  Hex  
Key (In Binary[10 bits]):  
Input: 1010011010  
Result:  
Output: ZÆwliß(ww(cÉißøèèÆí  
Decryption settings:  
 Text  Binary  Hex  
En/Decrypt button

Figure 11: Encrypting Result

This screenshot shows the S-DES Algorithm application window. The title bar says "S DES Form". The main title is "S-DES Algorithm".  
Text: (Plain Text or Cipher Text)  
Input: ZÆwliß(ww(cÉißøèèÆí  
Decryption settings:  
 Encrypt  Decrypt  Text  Binary  Hex  
Key (In Binary[10 bits]):  
Input: 1010011010  
Result:  
Output: Hitler attacked russia.  
Encryption settings:  
 Text  Binary  Hex  
En/Decrypt button

Figure 12: Decrypting Result

## RC4:

The output is the plain text or the cipher text in a text box. Before Clicking on the “En/Decrypt” button choose the type of the text (Text, Binary, Hex, or Decimal). Then click on En/Decrypt button to get the result in the “Result” text box.

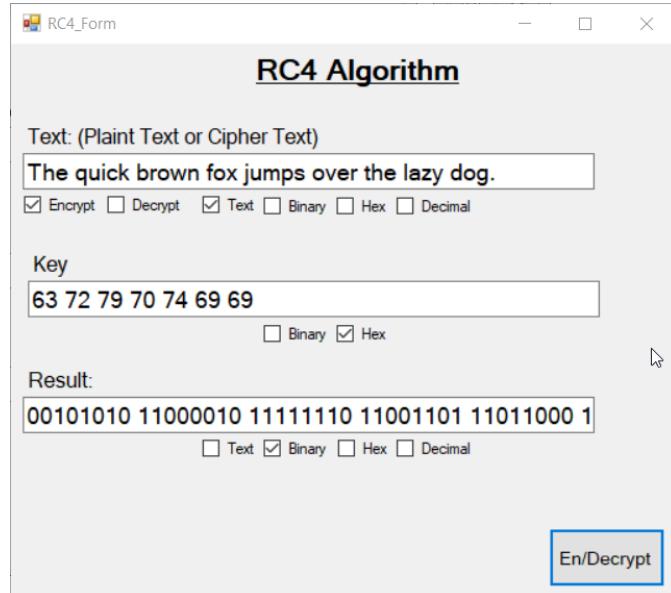


Figure 13: Encrypting Result

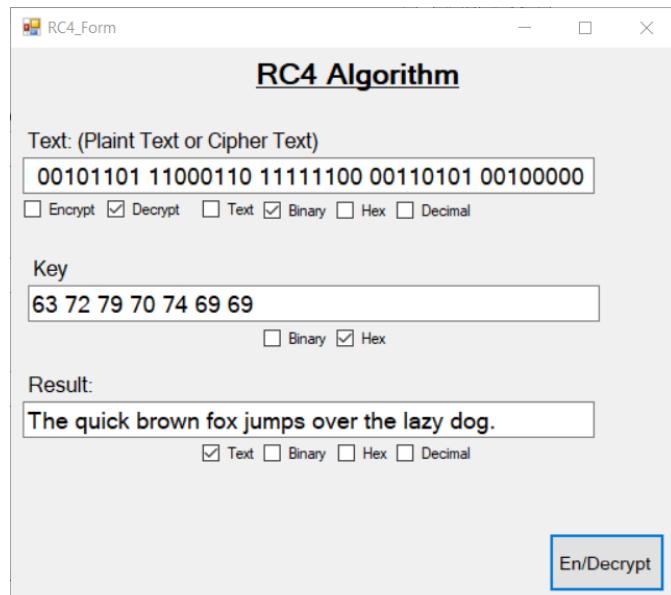


Figure 14: Decrypting Result

### Diffie-Hellman:

The output is a public key for user A, public key for user B, and the shared private key of user A and B, where they will appear when you click on “Generate Keys” button.

The screenshot shows a Windows application window titled "Diffie\_Hellman\_Form". The main title bar reads "Diffie-Hellman Keys Generator Algorithm". Inside the window, there are four input fields for parameters and three output fields for results. Below the results is a "Generate Keys" button.

Parameter	Value
Large Prime Integer (q):	353
Primitive Root (a):	3
Secret Key of User A ( $X_A < q$ ):	97
Secret Key of User B ( $X_B < q$ ):	233
<b>Results</b>	
Public Key of User A ( $Y_A$ ):	40
Public Key of User B ( $Y_B$ ):	248
Shared Private Key of User A and B ( $K_{AB}$ ):	160

**Generate  
Keys**

Figure 15: Results of Public Keys and Private Keys

**RSA:**

The output is the plain text or the cipher text in a text box. Before Clicking on the “En/Decrypt” button choose the type of the text (Text, Binary, Hex, or Decimal). Then click on En/Decrypt button to get the result in the “Result” text box.

*Figure 16: Encrypting Result*

*Figure 17: Decrypting Result*

## ElGamal:

For Encryption:

The output is the cipher text in a text box. Before Clicking on the “Encrypt” button choose the type of the cipher text (Text, Binary, Hex, or Decimal). Then click on Encrypt button to get the result in the “CipherText” text box.

The screenshot shows the 'ELGamal Algorithm' application window. On the left, under 'PlainText:', the input 'Computer Security is Important.' is shown in a text box with a checked 'Text' checkbox. Below it are fields for 'Large Prime Number (q)' containing '1613', 'Primitive Root (a)' containing '1119', and 'Secret Key of User A (XA < q - 1)' containing '1229'. On the right, under 'CipherText (C1, C2):', the output is displayed as a string of characters: 'Ç ð ã L Æ Æ ã + Œ ã , + z Æ Õ ã ä'. There are also text boxes for 'Large Prime Number (q)', 'Secret Key of User A (XA < q - 1)', and 'PlainText:' (empty), each with its own set of 'Text', 'Binary', 'Hex', and 'Decimal' checkboxes. At the bottom center is an 'Encrypt' button, and on the far right is a 'Decrypt' button.

Figure 18: Encrypting Result (CipherText)

For Decryption:

The output is the plain text in a text box. Before Clicking on the “Decrypt” button choose the type of the plain text (Text, Binary, Hex, or Decimal). Then click on Decrypt button to get the result in the “PlainText” text box.

The screenshot shows the ELGamal Algorithm application window titled "ELGamal Algorithm".

**PlainText:** (Input field is empty)

Text  Binary  Hex  Decimal

**CipherText (C1, C2):** **1ā\_ś\_ṛ\_ṛ\_ś\_Ԫ\_L\_θ\_ë\_ӹ\_ā\_É\_ᢃ\_ā\_g**

Text  Binary  Hex  Decimal

**Large Prime Number (q):** **1613**

**Primitive Root (a):** (Input field is empty)

**Secret Key of User A (XA < q - 1):** **1229**

**CipherText (C1, C2):** (Input field is empty)

Text  Binary  Hex  Decimal

**Encrypt** **Decrypt**

**PlainText:** **Computer Security is Important.**

Text  Binary  Hex  Decimal

Figure 19: Decrypting Result (PlainText)