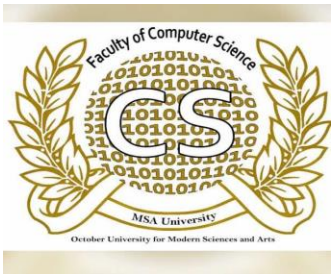


Faculty of Computer Science

Semester 71 – Spring 2020

Computer Security (CS401)

Final Course Project

Project Title:

S-DES

RC4

Diffie-Hellman

RSA

ElGamal

Done by:

Name	ID
Yahia Mahmoud Abdelrahman Basyouny	173269

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. It 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 decrypt 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.

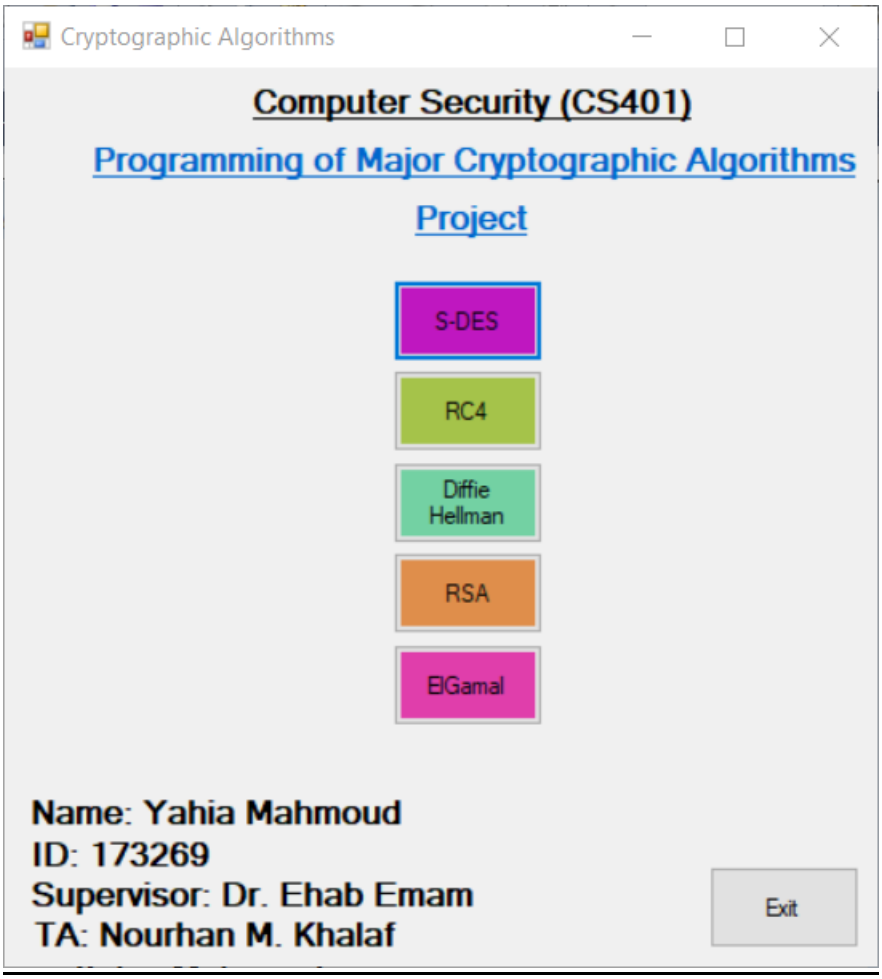


Figure 1: Choose one of the cryptographic algorithms

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.

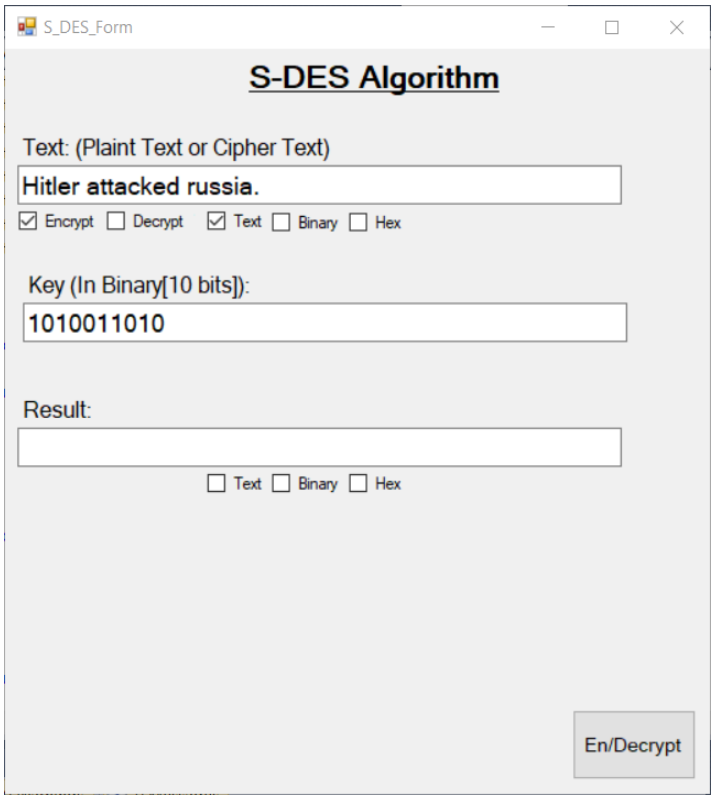


Figure 2: Encrypting

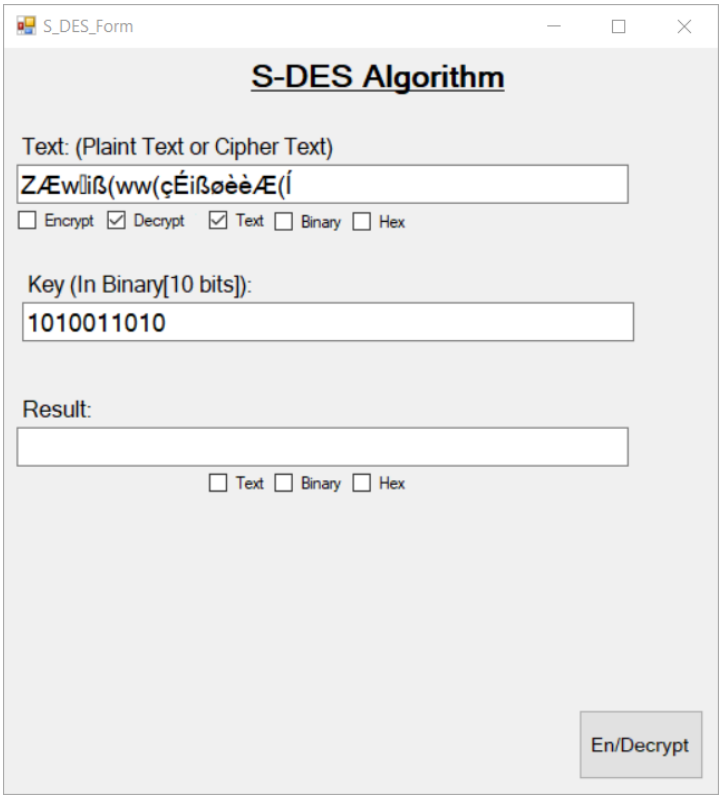


Figure 3: Decrypting

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.

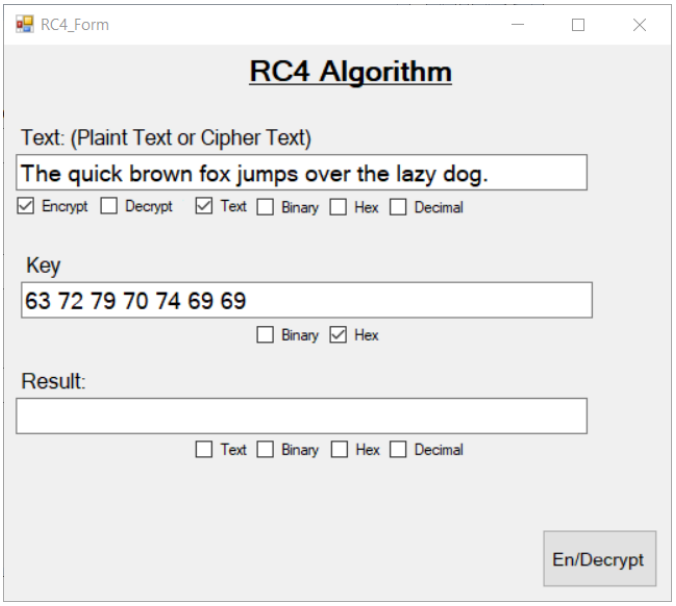


Figure 4: Encrypting

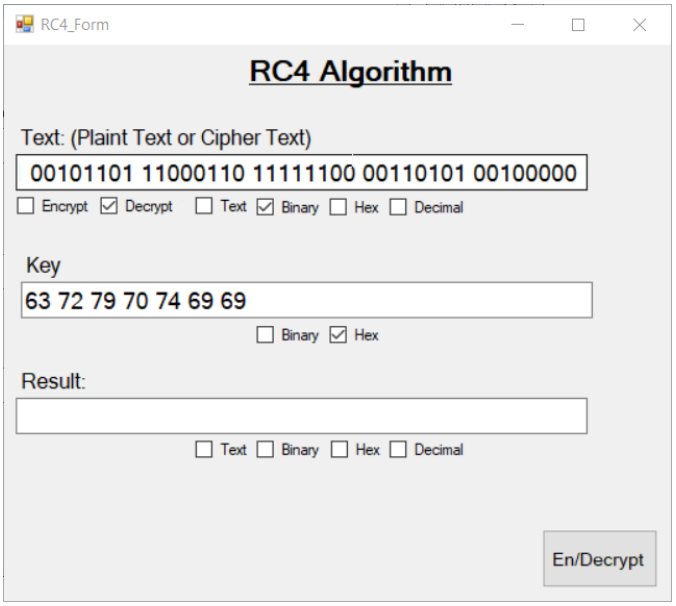


Figure 5: Decrypting

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.

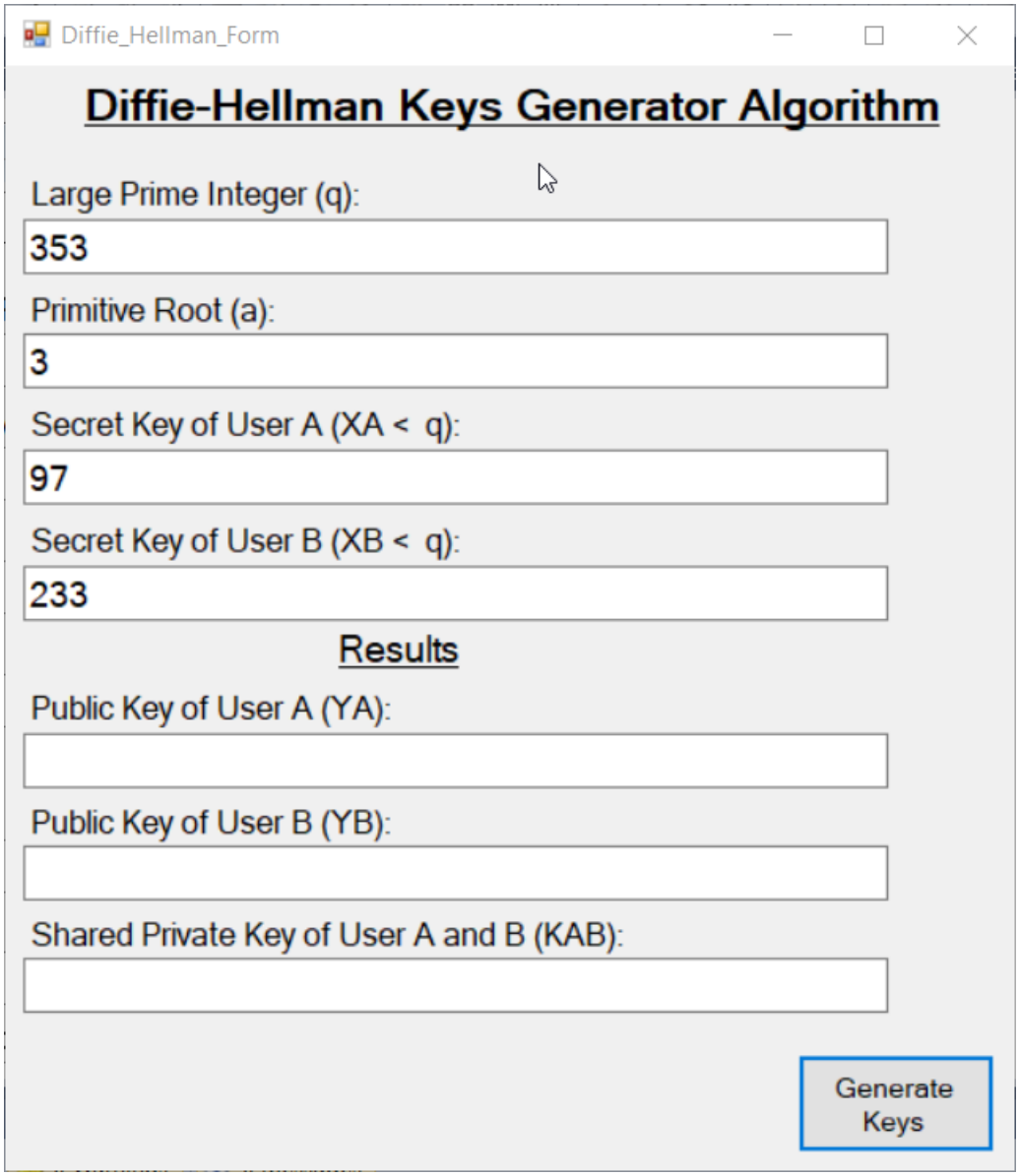


Figure 6: Inserting Data

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.

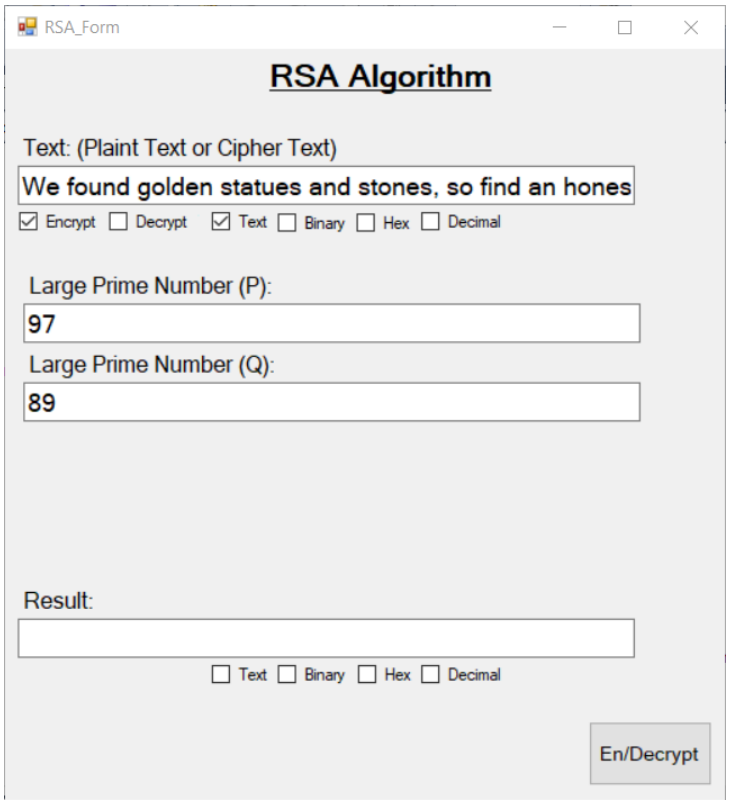


Figure 7: Encrypting

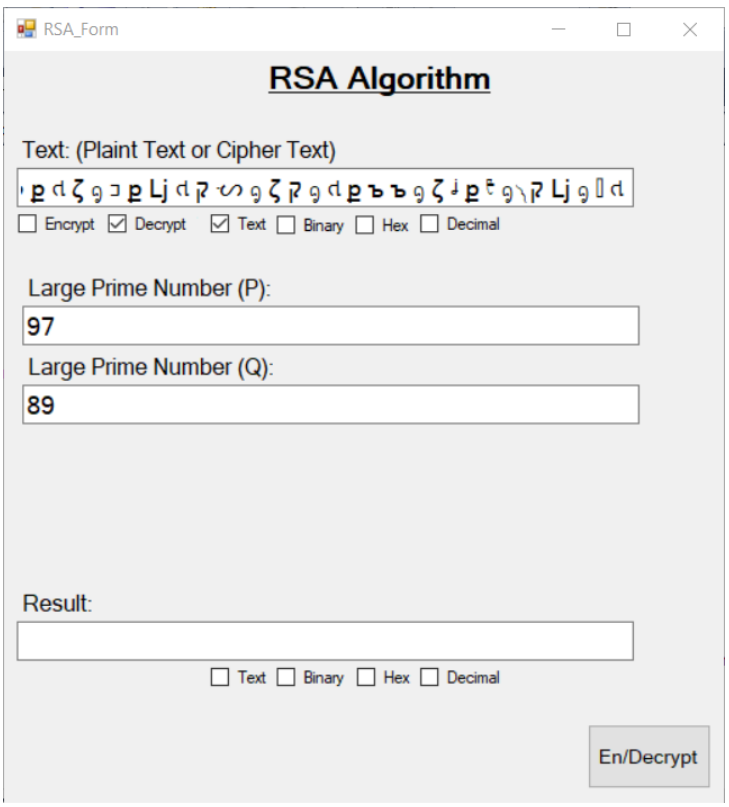


Figure 8: Decrypting

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.

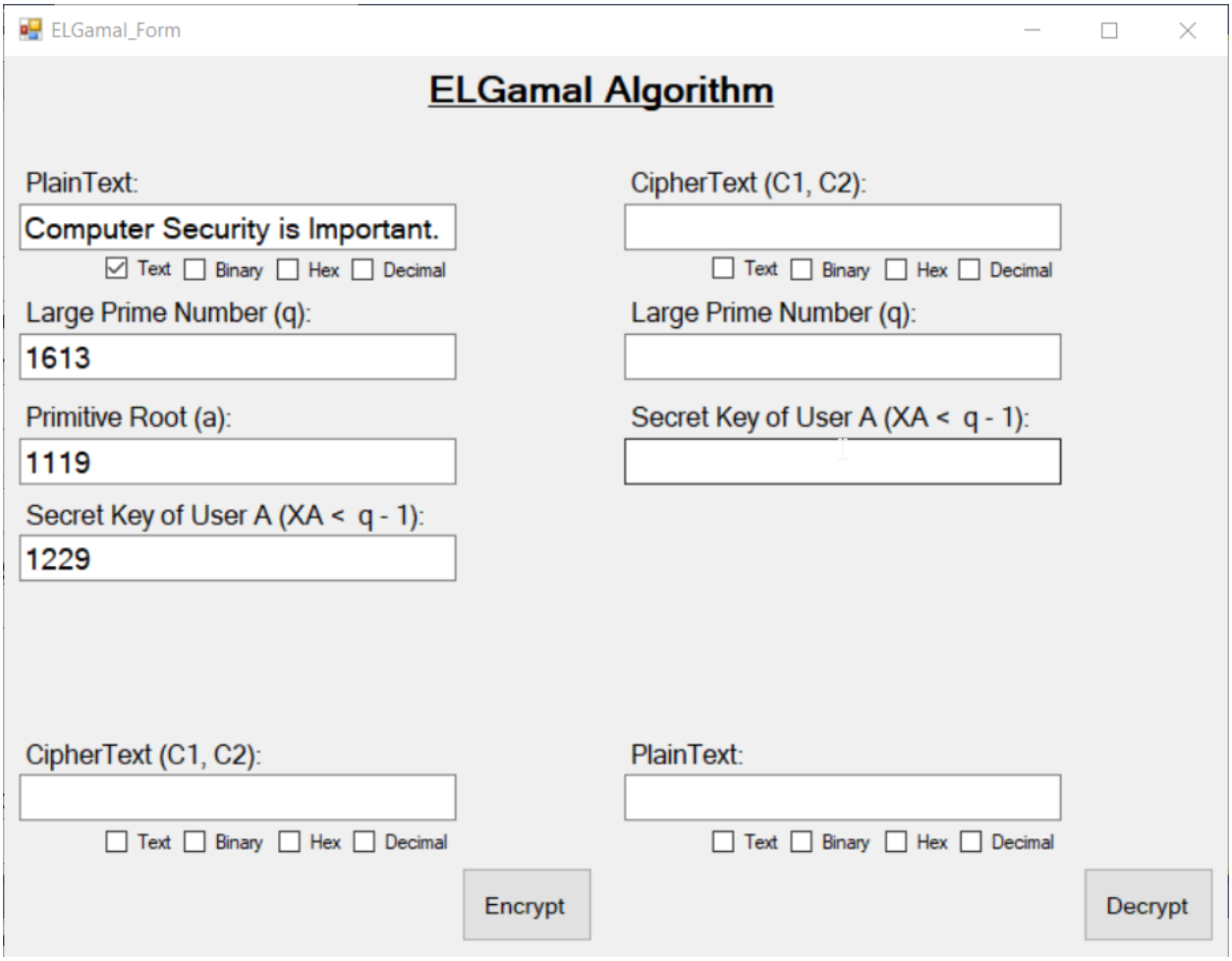


Figure 9: Encrypting

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.

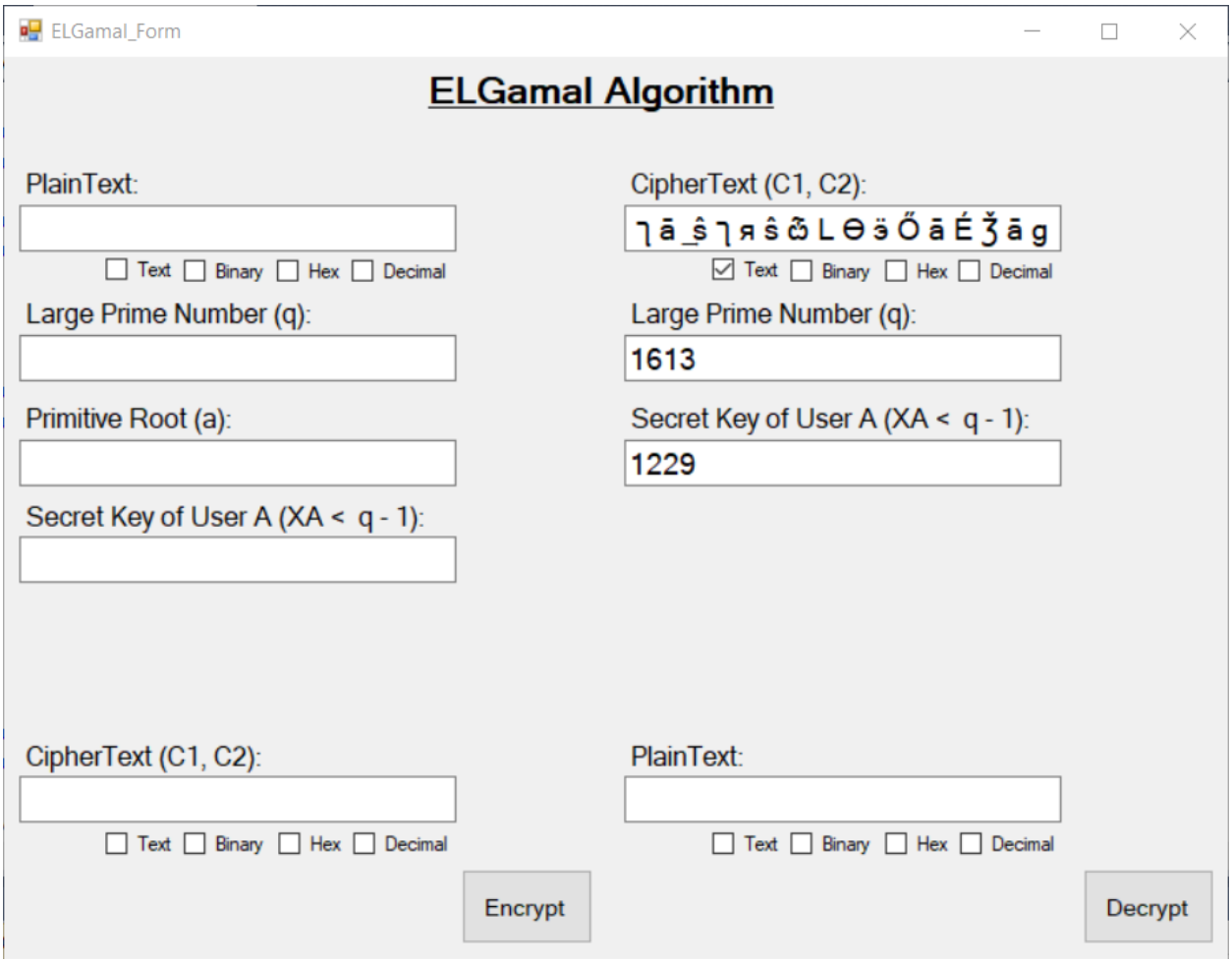
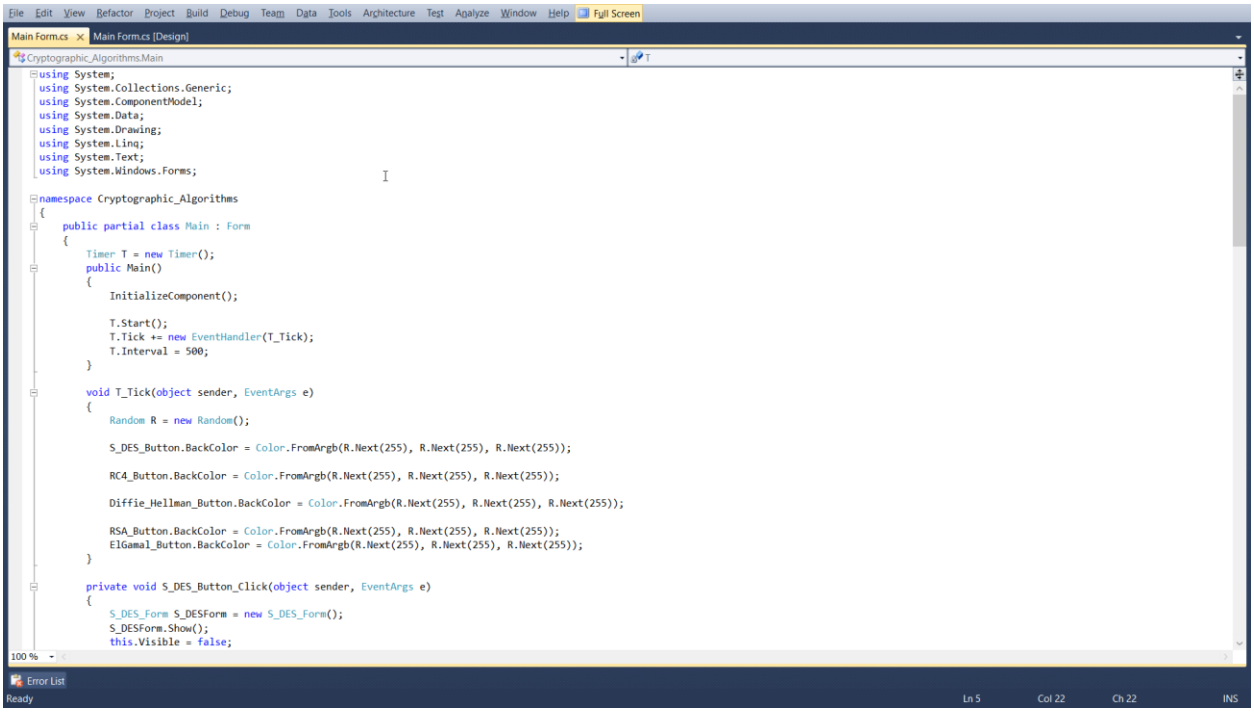


Figure 10: Decrypting

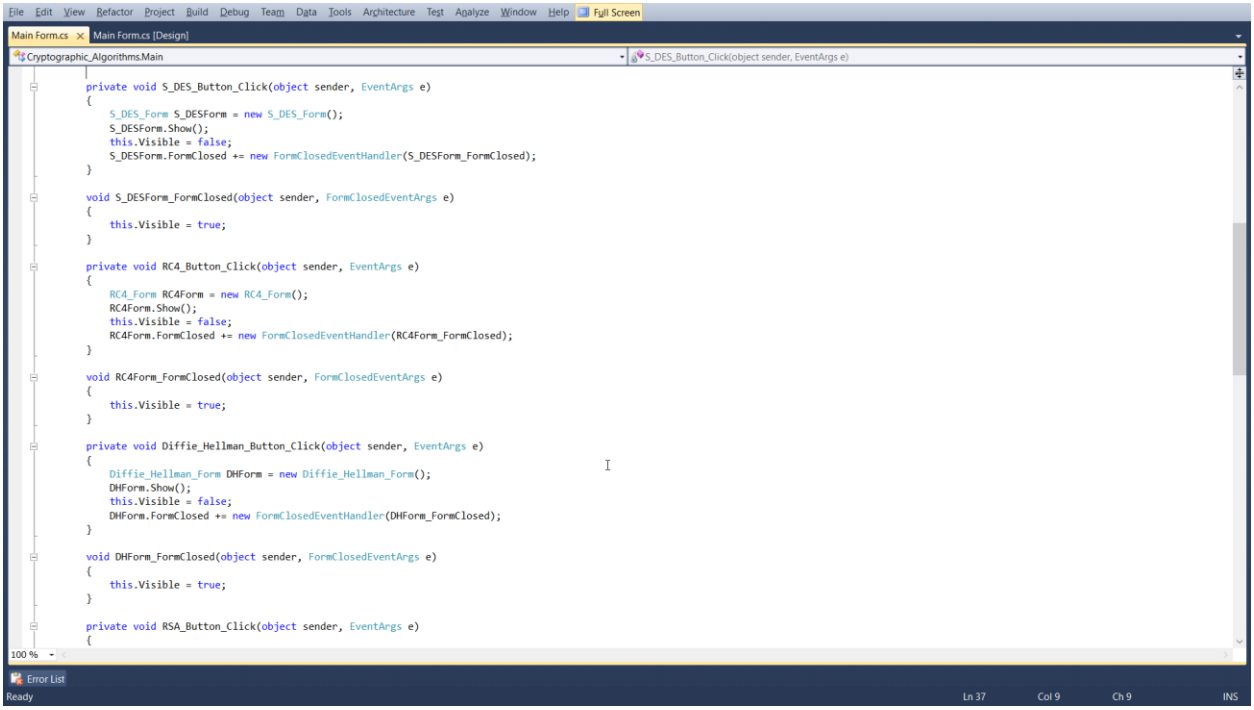
Implementation:

Main Form:

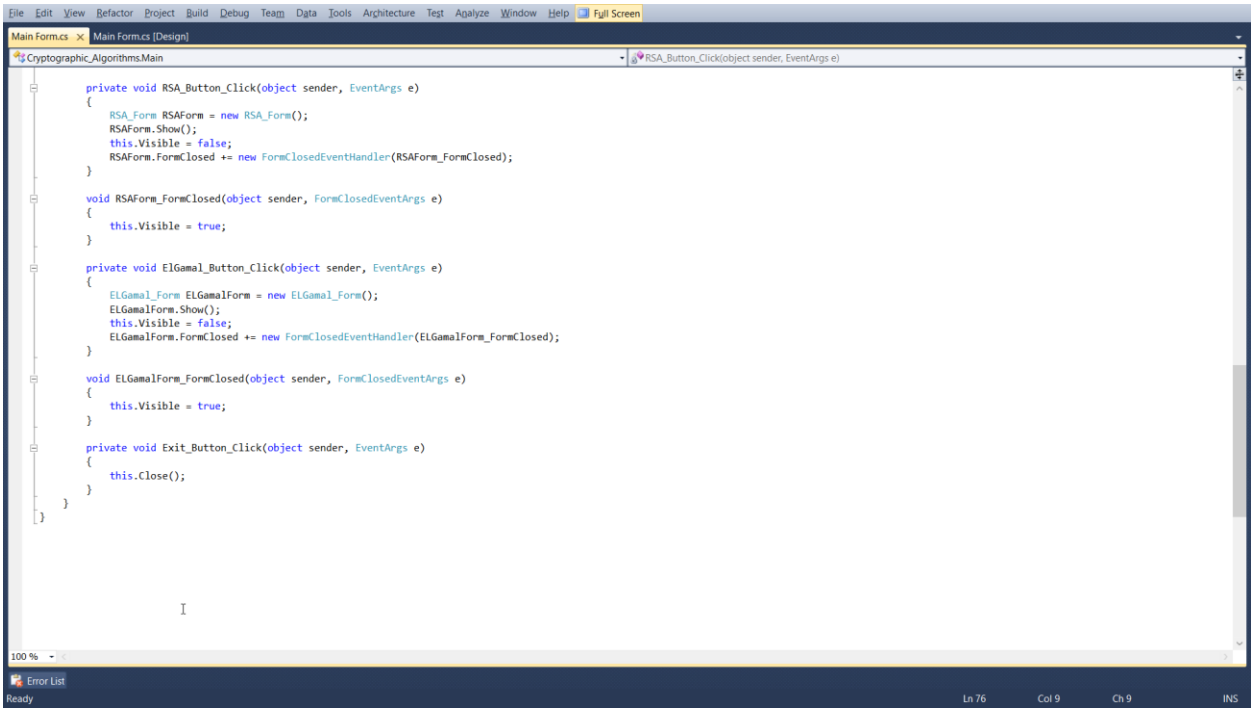
Screen Shots:



```
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;
```



```
        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
Main Form.cs [Design]
Cryptographic_Algorithms.Main
RSA_Button_Click(object sender, EventArgs e)
private void 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 ElGamal_Button_Click(object sender, EventArgs e)
{
ELGamal_Form ELGamalForm = new ELGamal_Form();
ELGamalForm.Show();
this.Visible = false;
ELGamalForm.FormClosed += new FormClosedEventHandler(ELGamalForm_FormClosed);
}
void ELGamalForm_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

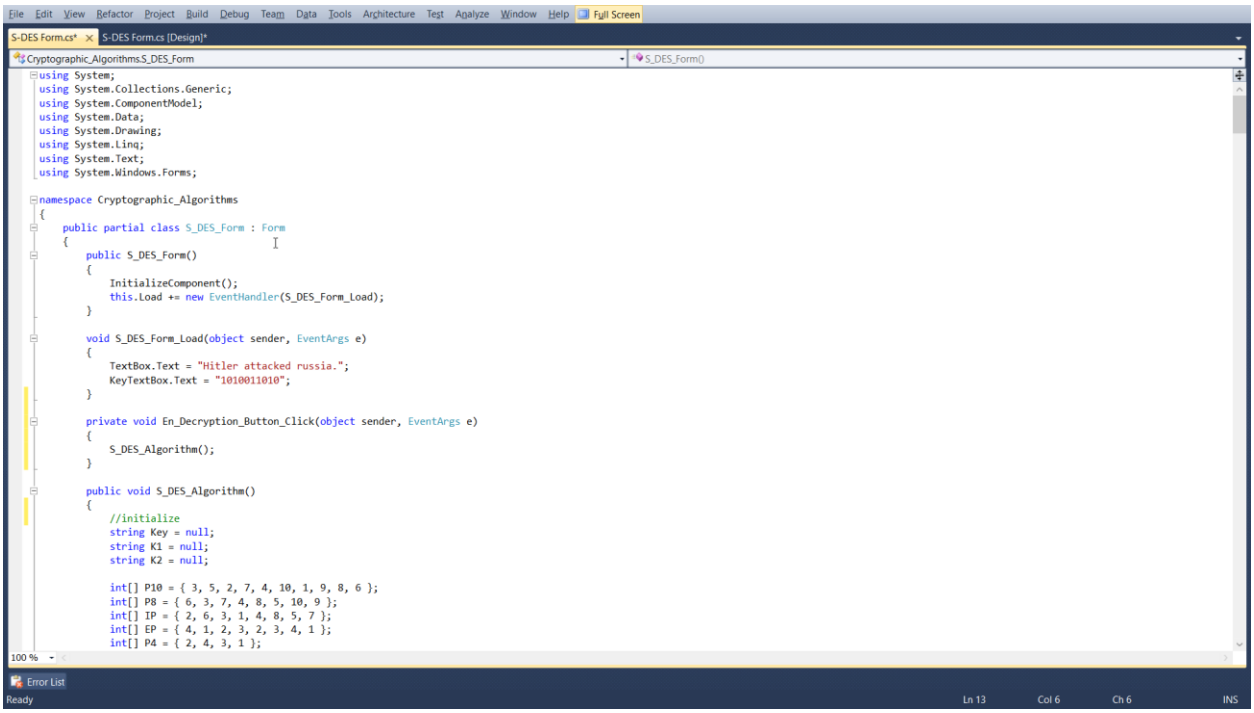
In Final Permutation, initial permutation inverse is created using IP array. Then by using the initial permutation inverse (IP-1), the result will be the plain text if you are decrypting or will be the cipher text if you are encrypting.

Step 5: Result

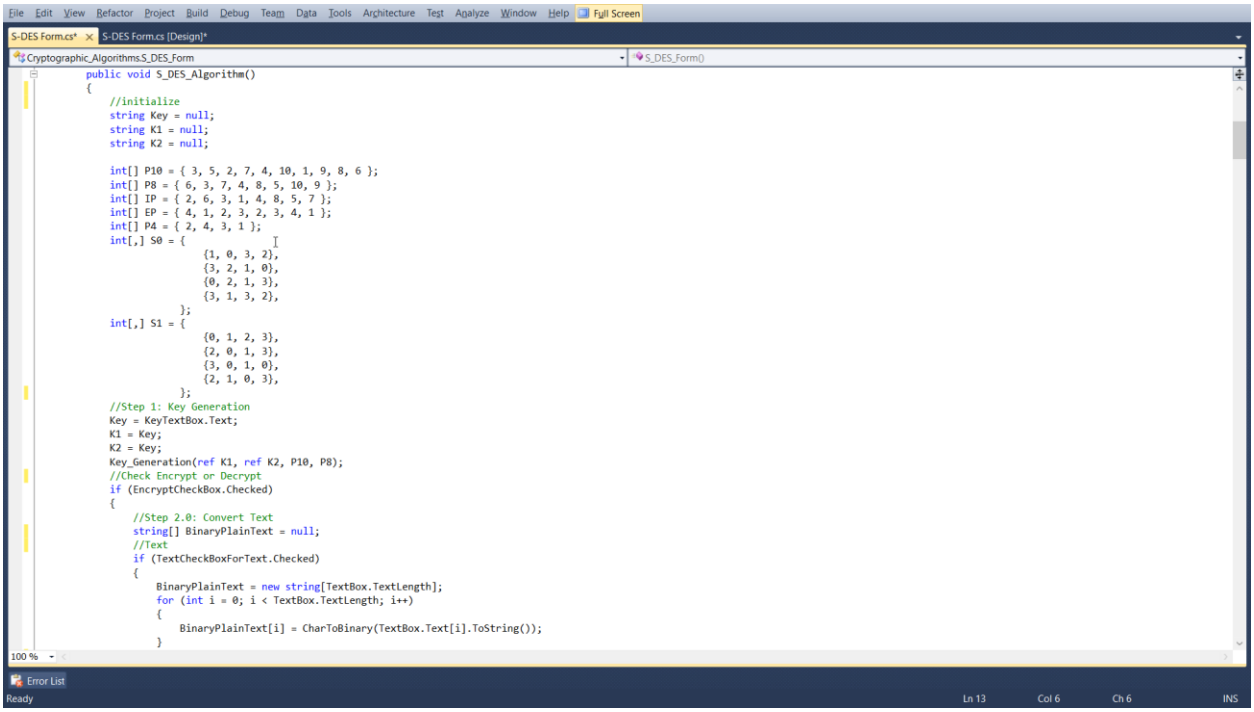
Then add result of the Final Permutation to the “Result” text box.

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

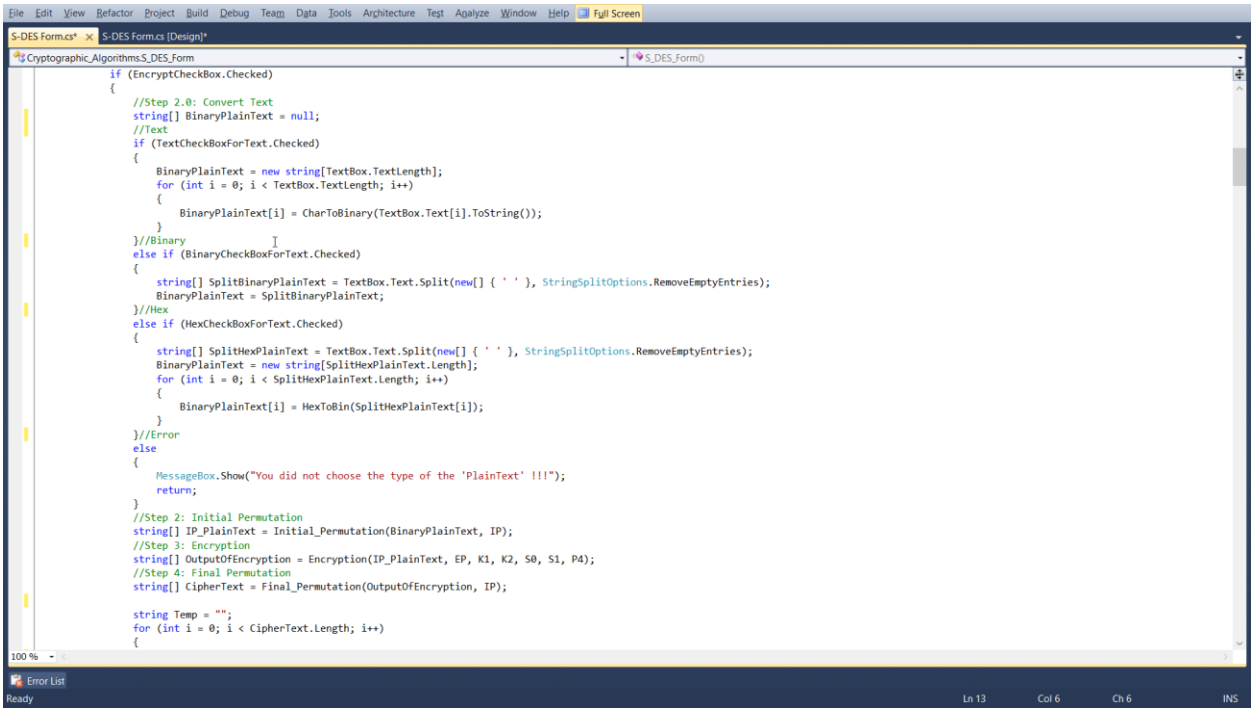
Screen Shots:



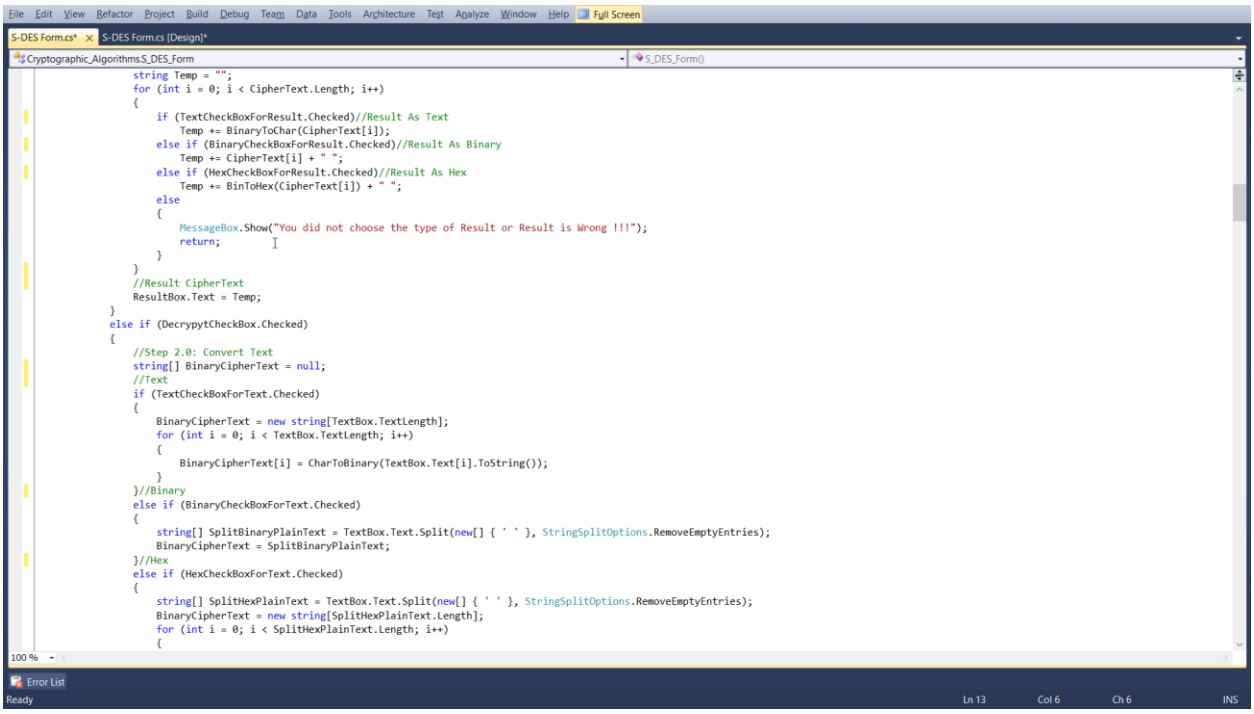
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

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 };
```



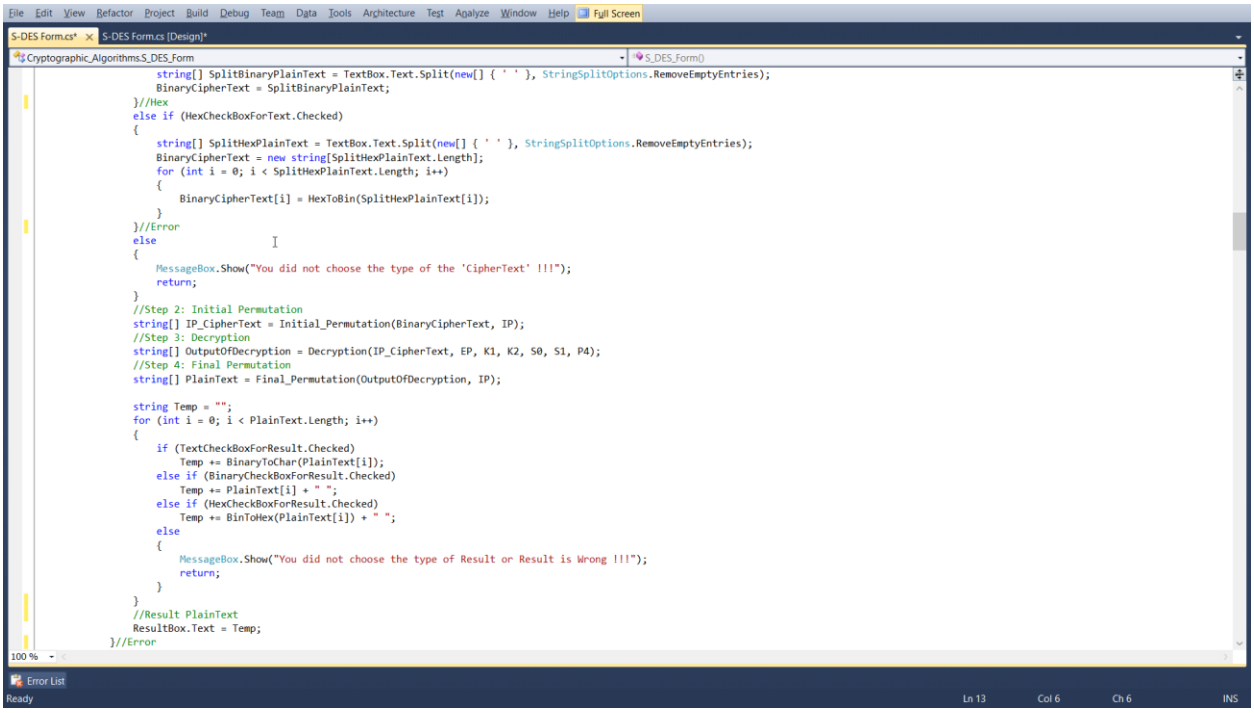
```
        public void S_DES_Algorithm()
        {
            //initialize
            string Key = null;
            string K1 = null;
            string K2 = null;

            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());
                    }
```



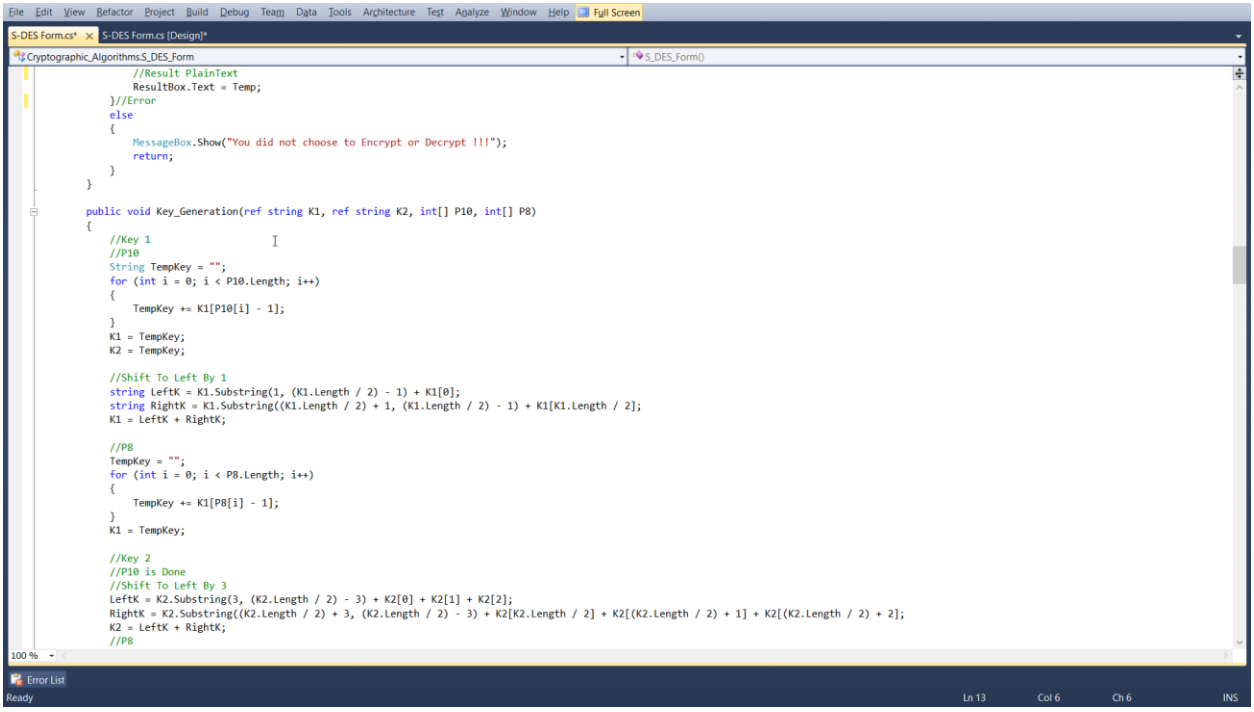
```
if (EncryptCheckBox.Checked)
{
    //Step 2.0: Convert Text
    string[] BinaryPlainText = null;
    //Text
    if (TextCheckBoxForText.Checked)
    {
        BinaryPlainText = new string[TextBox.TextLength];
        for (int i = 0; i < TextBox.TextLength; i++)
        {
            BinaryPlainText[i]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



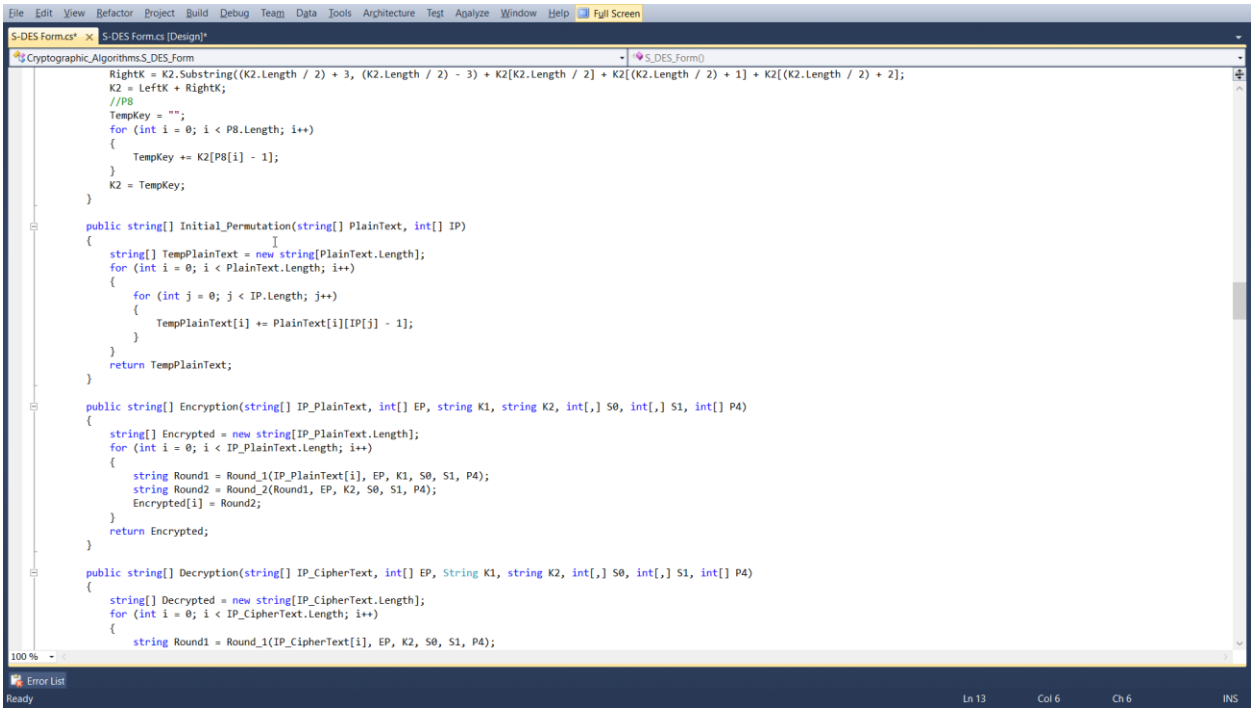
```
    string Temp = "";
    for (int i = 0; i < CipherText.Length; i++)
    {
        if (TextCheckBoxForResult.Checked)//Result As Text
            Temp += BinaryToChar(CipherText[i]);
        else if (BinaryCheckBoxForResult.Checked)//Result As Binary
            Temp += CipherText[i] + " ";
        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 (DecrypytCheckBox.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++)
        {
```



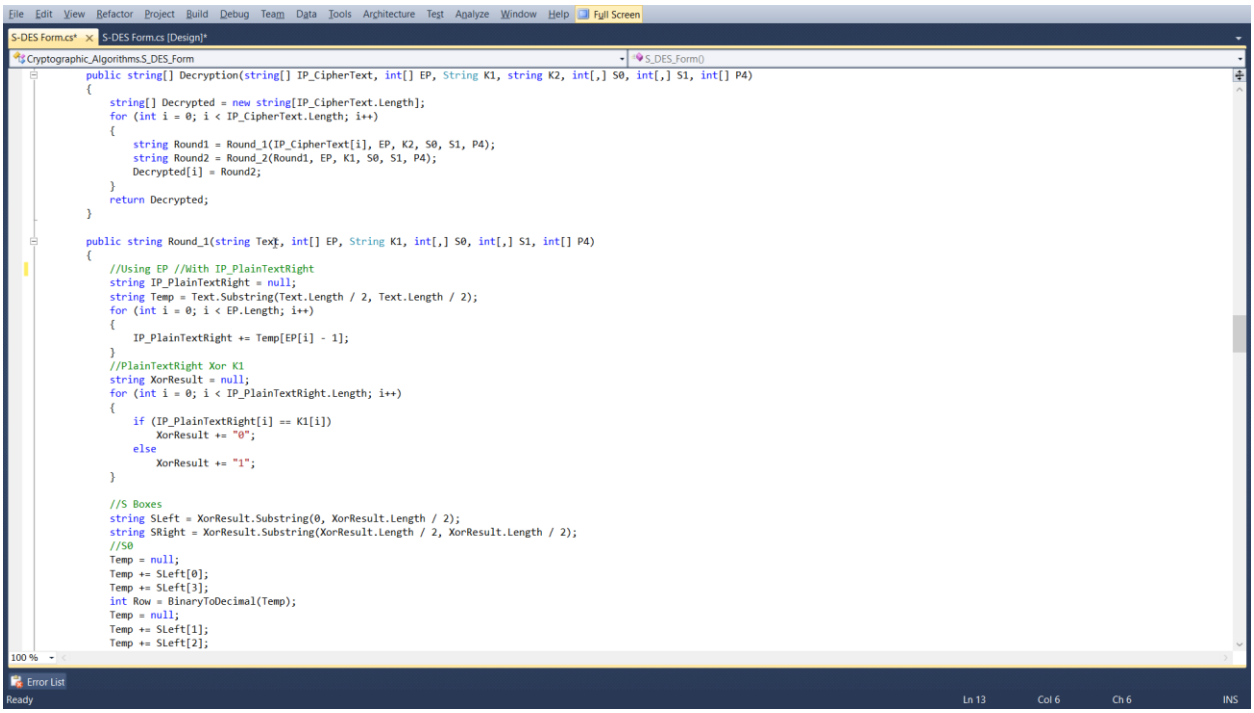
```
                string[] SplitBinaryPlainText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                BinaryCipherText = SplitBinaryPlainText;
            }//Hex
            else if (HexCheckBoxForText.Checked)
            {
                string[] SplitHexPlainText = 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
```



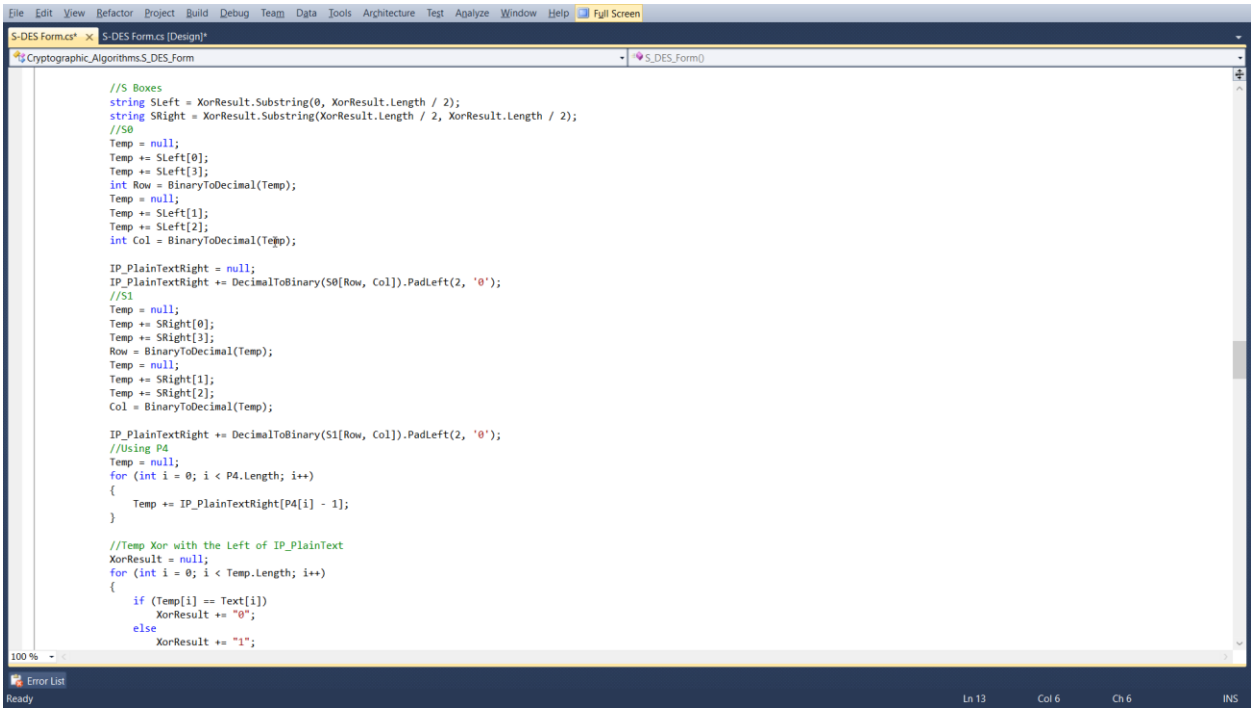
```
            //Result PlainText
            ResultBox.Text = Temp;
        }//Error
        else
        {
            MessageBox.Show("You did not choose to Encrypt or Decrypt !!!");
            return;
        }
    }

    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
```



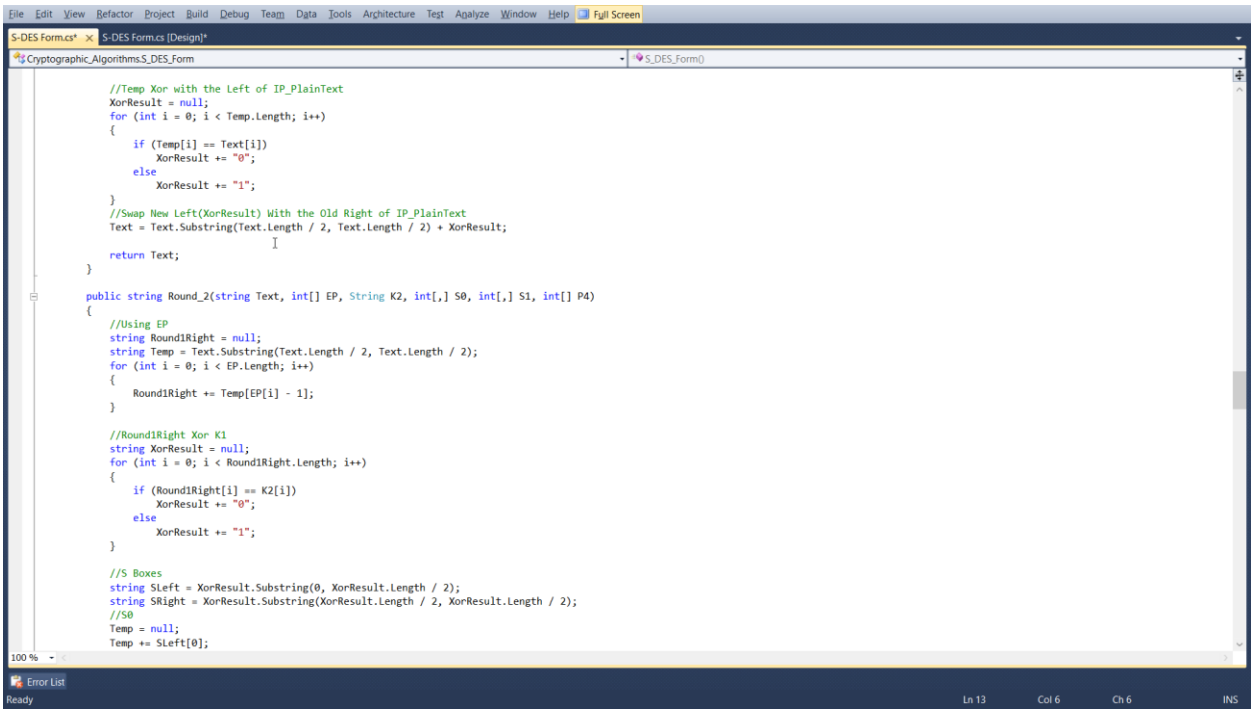
```
            RightK = K2.Substring((K2.Length / 2) + 3, (K2.Length / 2) - 3) + K2[K2.Length / 2] + K2[((K2.Length / 2) + 1] + K2[(K2.Length / 2) + 2];
            K2 = LeftK + RightK;
            //P8
            TempKey = "";
            for (int i = 0; i < P8.Length; i++)
            {
                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);
```



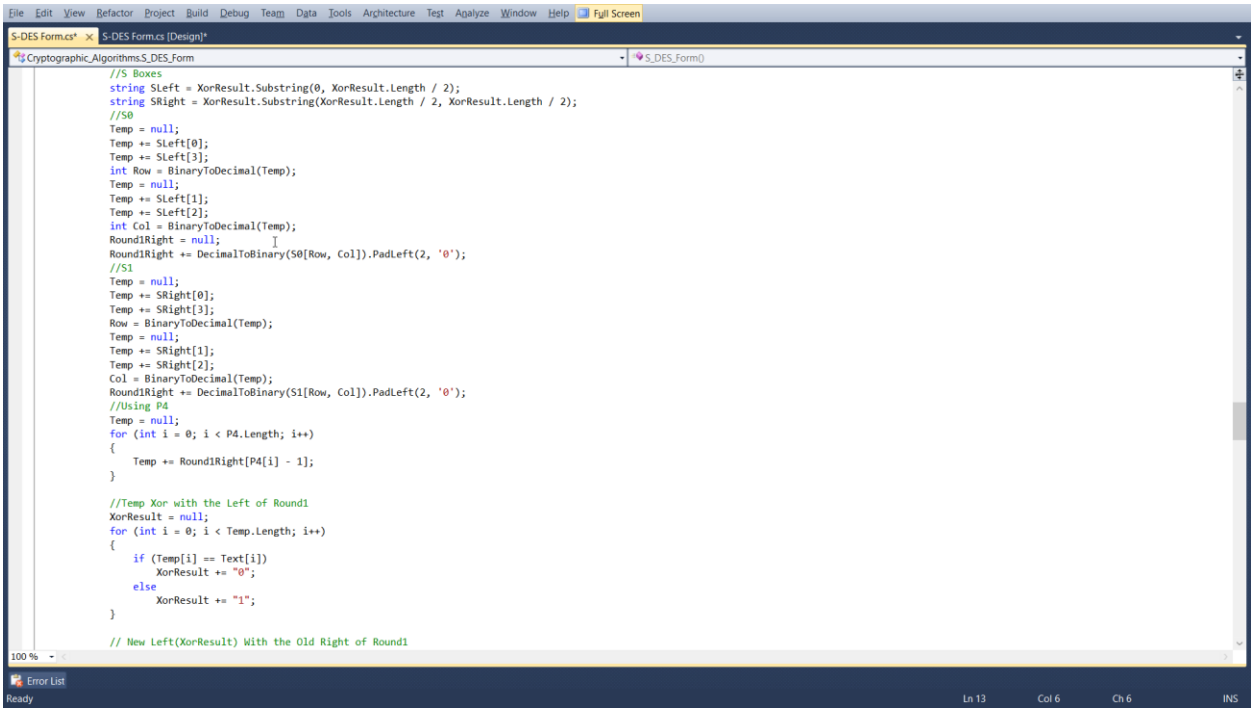
```
        public string[] Decryption(string[] IP_CipherText, int[] EP, String K1, string K2, int[,] S0, int[,] S1, int[] P4)
        {
            string[] Decrypted = new string[IP_CipherText.Length];
            for (int i = 0; i < IP_CipherText.Length; i++)
            {
                string Round1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



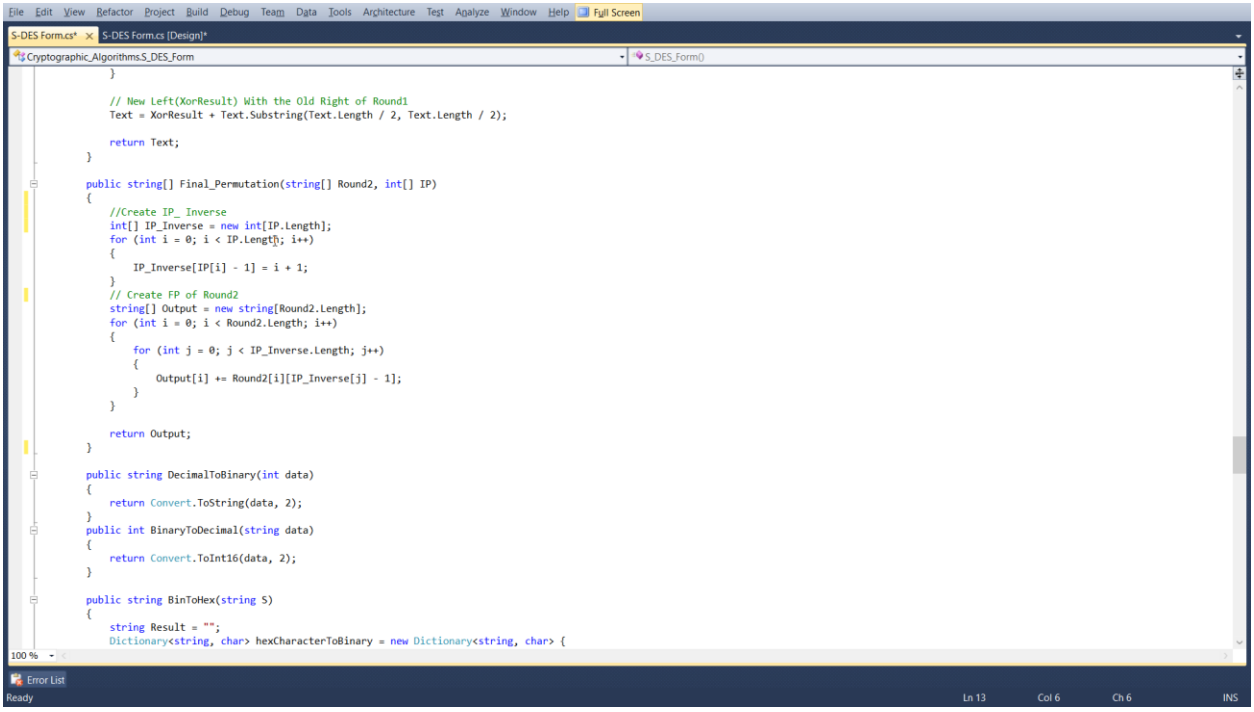
```
            //S Boxes
            string SLeft = XorResult.Substring(0, XorResult.Length / 2);
            string SRight = XorResult.Substring(XorResult.Length / 2, XorResult.Length / 2);
            //S0
            Temp = null;
            Temp += SLeft[0];
            Temp += SLeft[3];
            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";
```



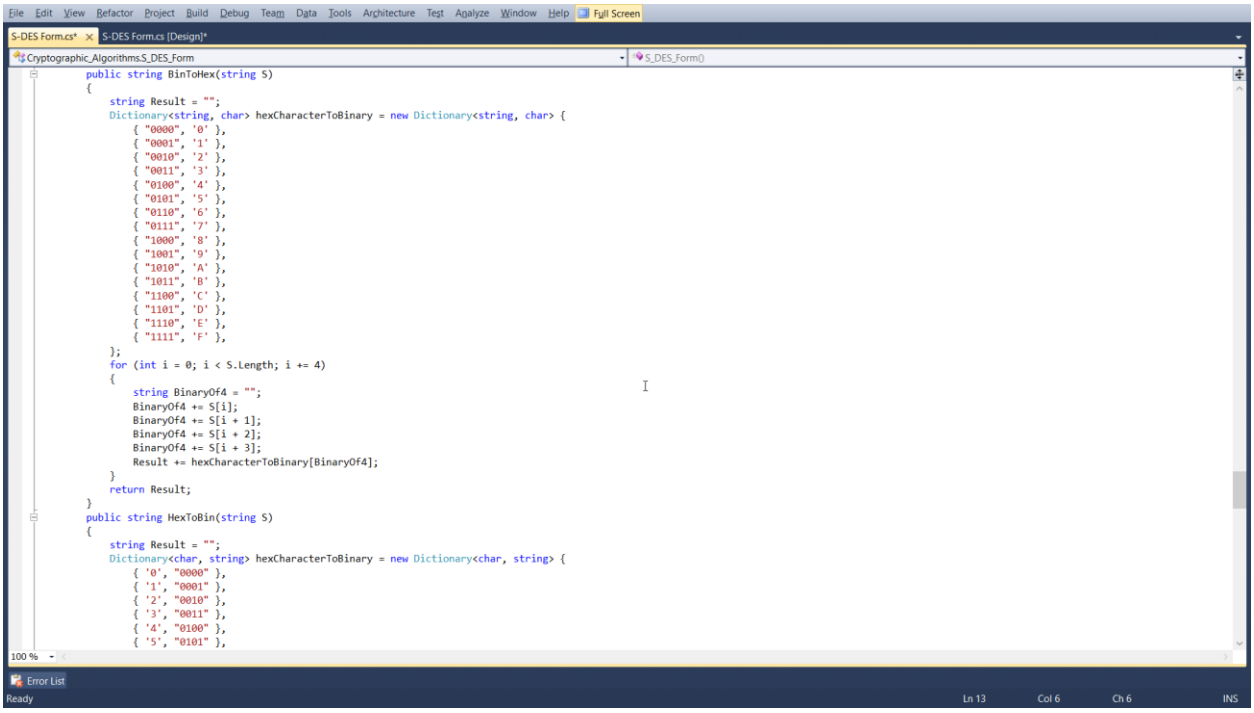
```
            //Temp Xor with the Left of IP_PlainText
            XorResult = null;
            for (int i = 0; i < Temp.Length; i++)
            {
                if (Temp[i] == Text[i])
                    XorResult += "0";
                else
                    XorResult += "1";
            }
            //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];
```

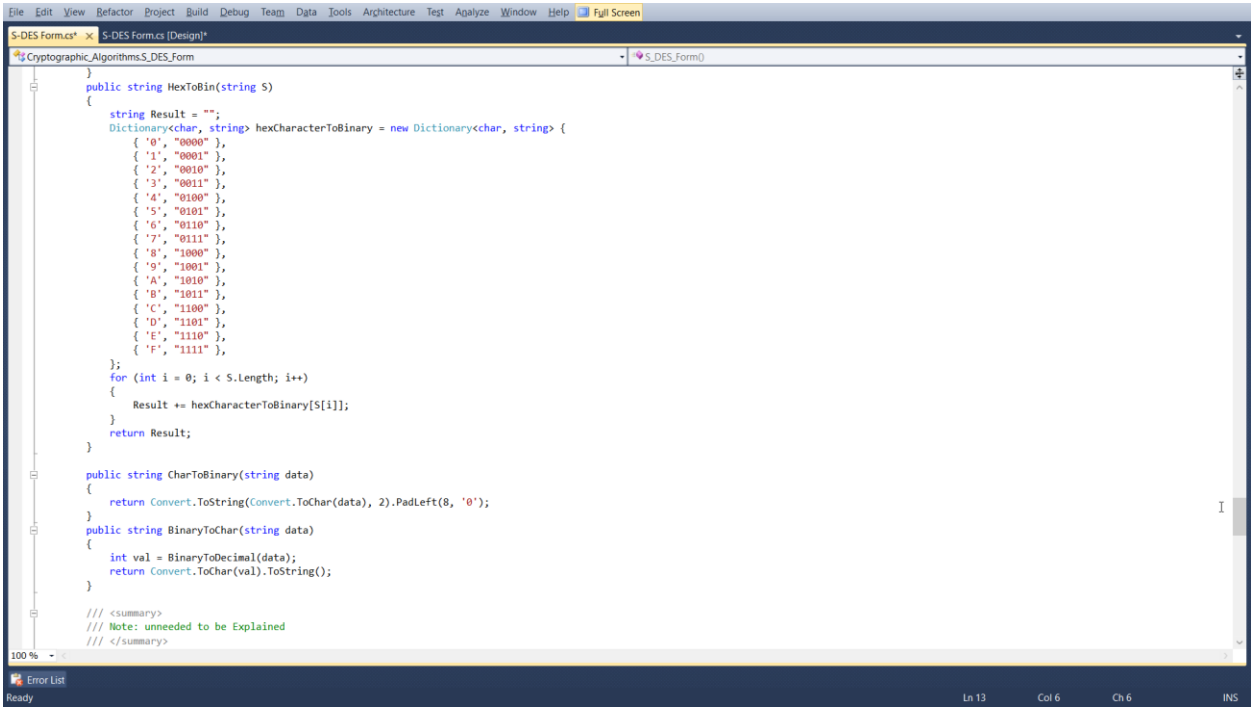


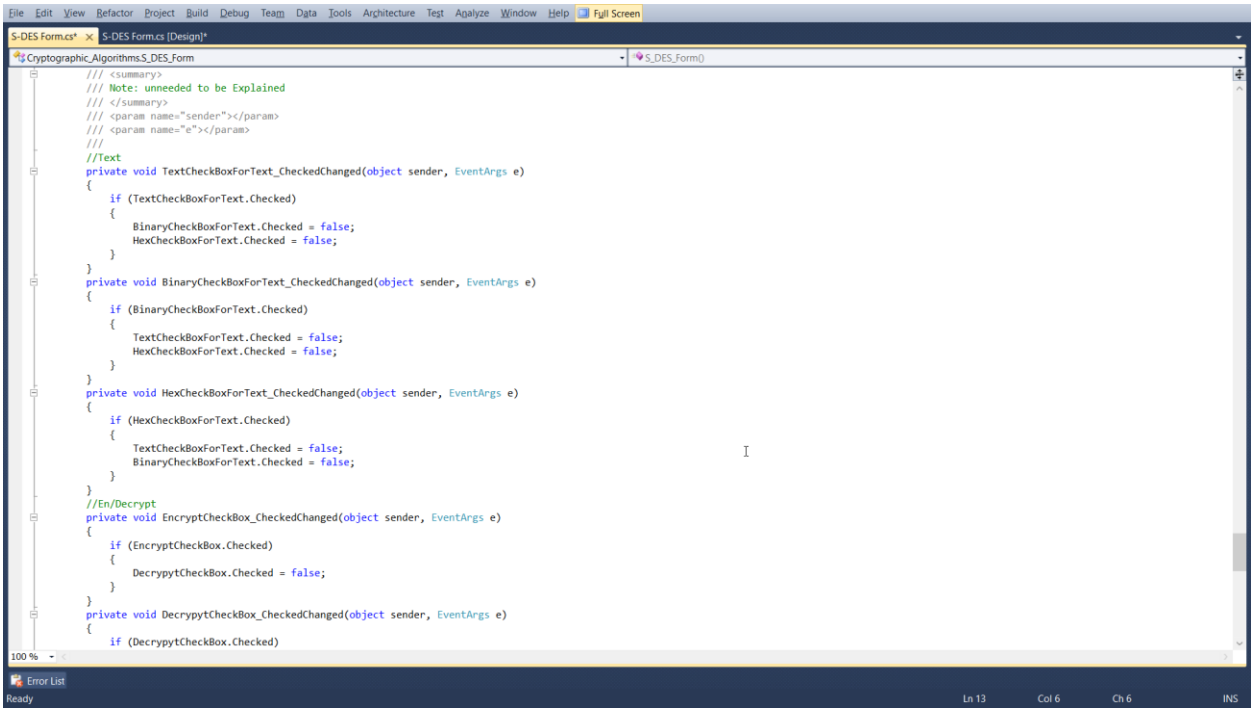
```
            //S Boxes
            string SLeft = XorResult.Substring(0, XorResult.Length / 2);
            string SRight = XorResult.Substring(XorResult.Length / 2, XorResult.Length / 2);
            //S0
            Temp = null;
            Temp += SLeft[0];
            Temp += SLeft[3];
      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

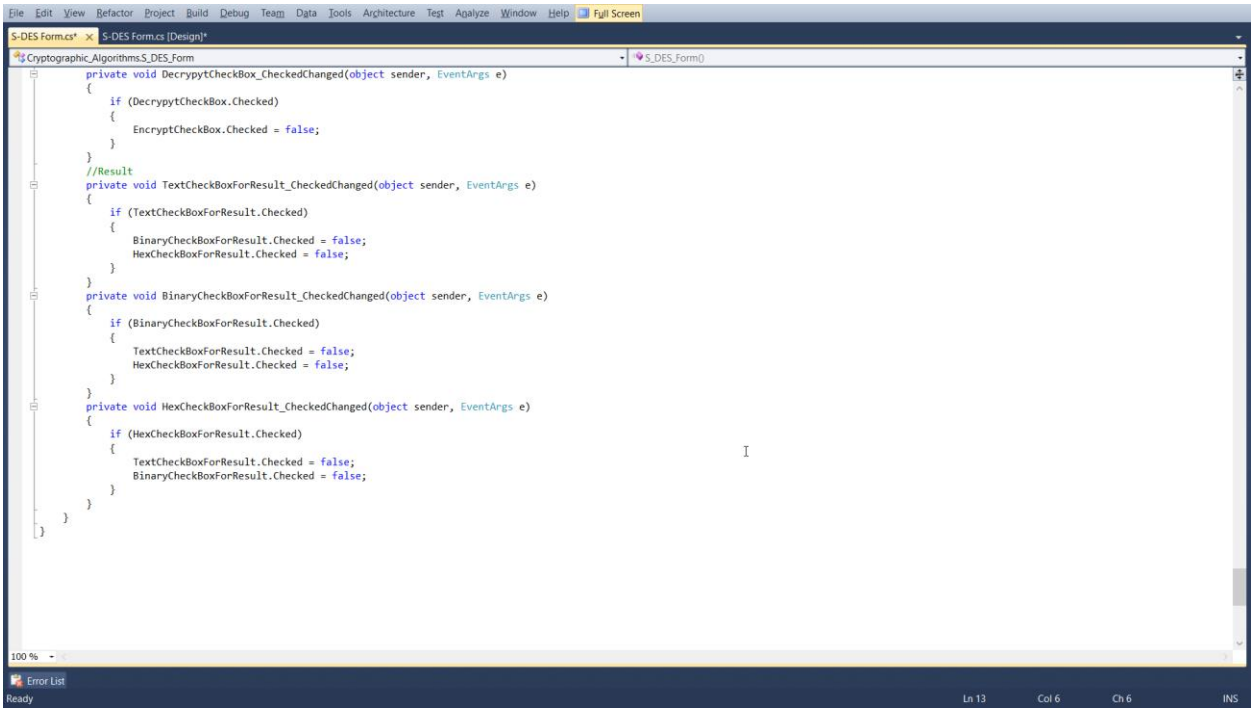


```
            }

            // 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> {
```


File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
S-DES Form.cs* S-DES Form.cs [Design]*
Cryptographic_Algorithms.S_DES_Form
S_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" },
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* S-DES Form.cs [Design]*
Cryptographic_Algorithms.S_DES_Form
S_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>
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* S-DES Form.cs [Design]*
Cryptographic_Algorithms.S_DES_Form
S_DES_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;
}
}
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)
{
DecrypytCheckBox.Checked = false;
}
}
private void DecrypytCheckBox_CheckedChanged(object sender, EventArgs e)
{
if (DecrypytCheckBox.Checked)
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* S-DES Form.cs [Design]*
Cryptographic_Algorithms.S_DES_Form
S_DES_Form()
private void DecrypytCheckBox_CheckedChanged(object sender, EventArgs e)
{
if (DecrypytCheckBox.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;
}
}
}
}
100 %
Error List
Ready
Ln 13
Col 6
Ch 6
INS

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:

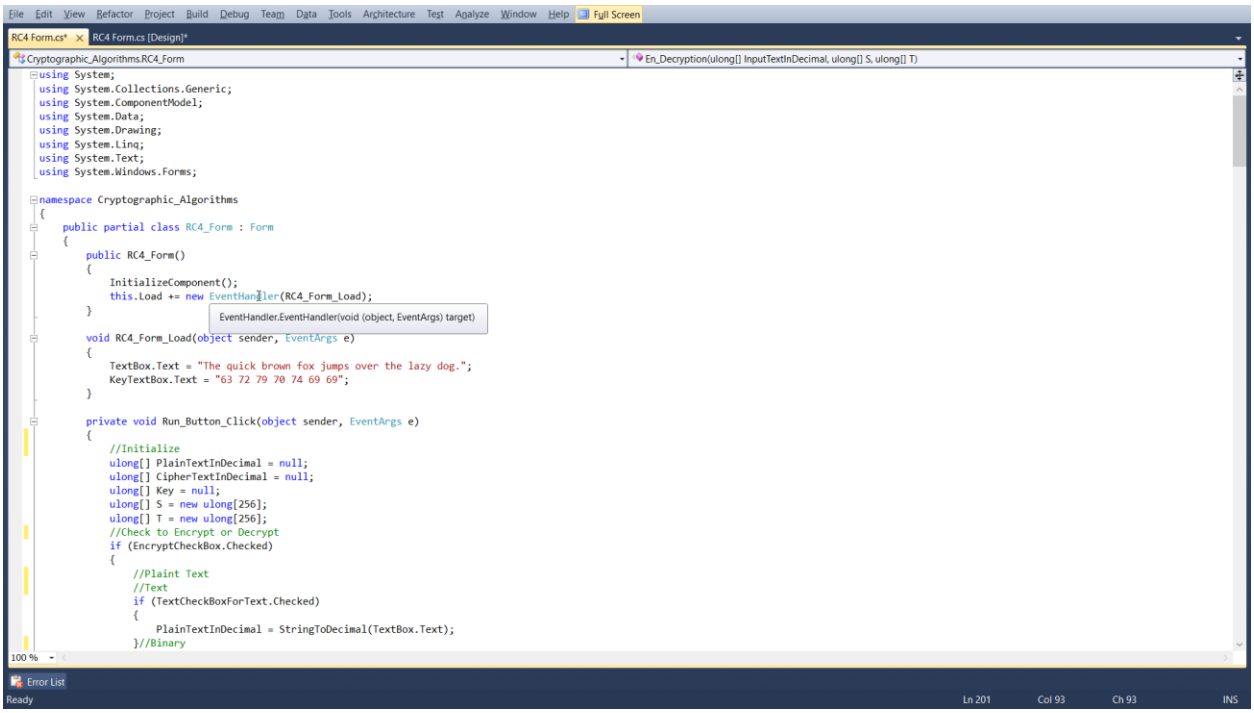
Perform Encryption or Decryption Function according to the inserted input. If you are encrypting, you will perform the encryption on the inserted plain text in decimal form. If you are decrypting, you will perform the decryption on the inserted cipher text in decimal form.

Step 4:

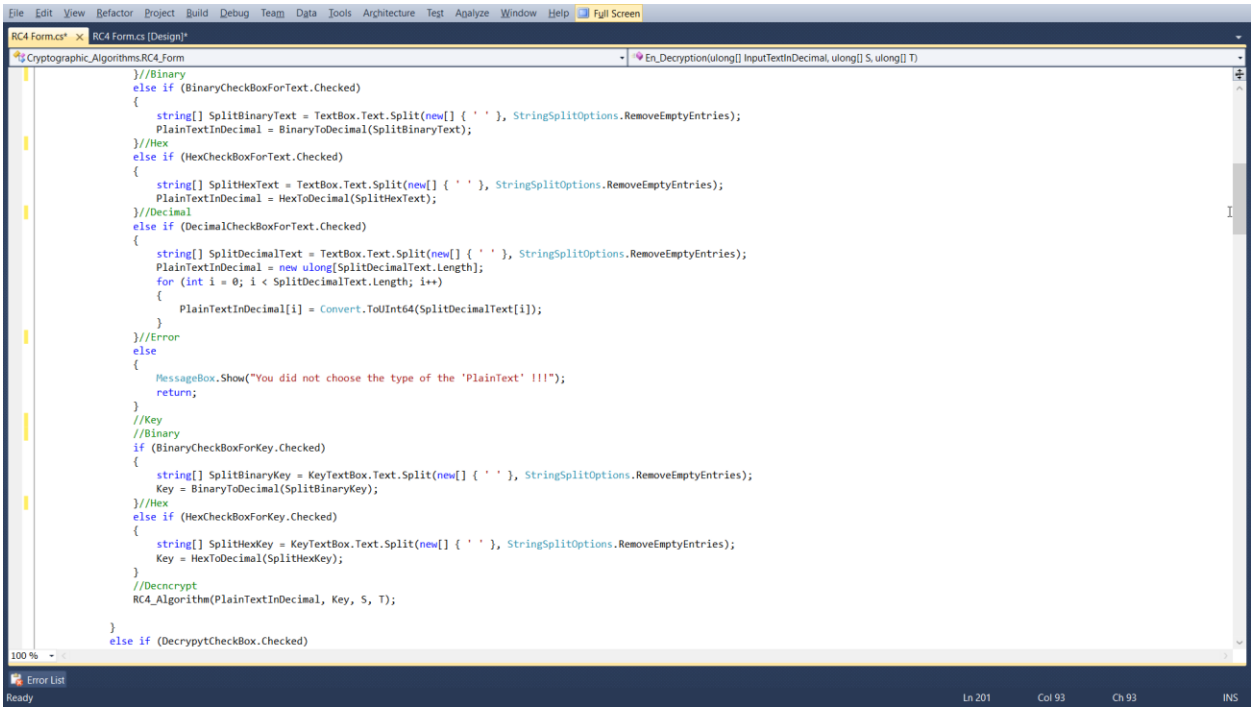
Then add result of the encryption or the decryption to the “Result” text box.

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

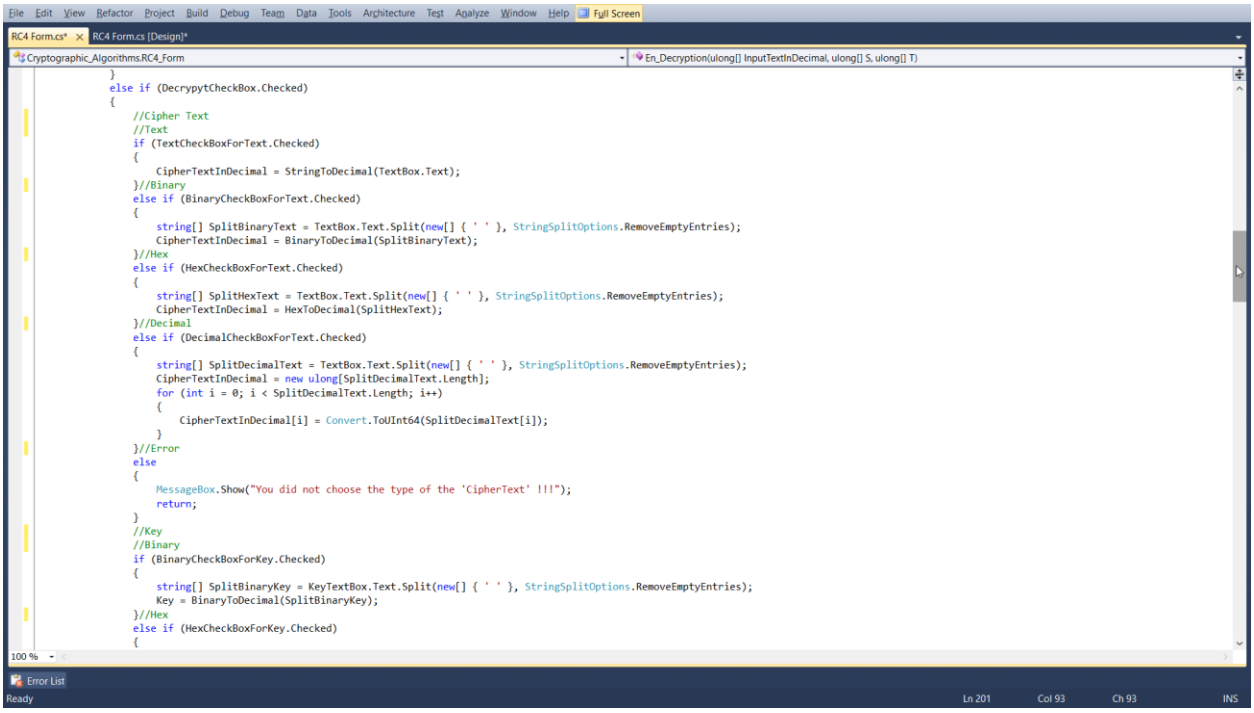
Screen Shots:



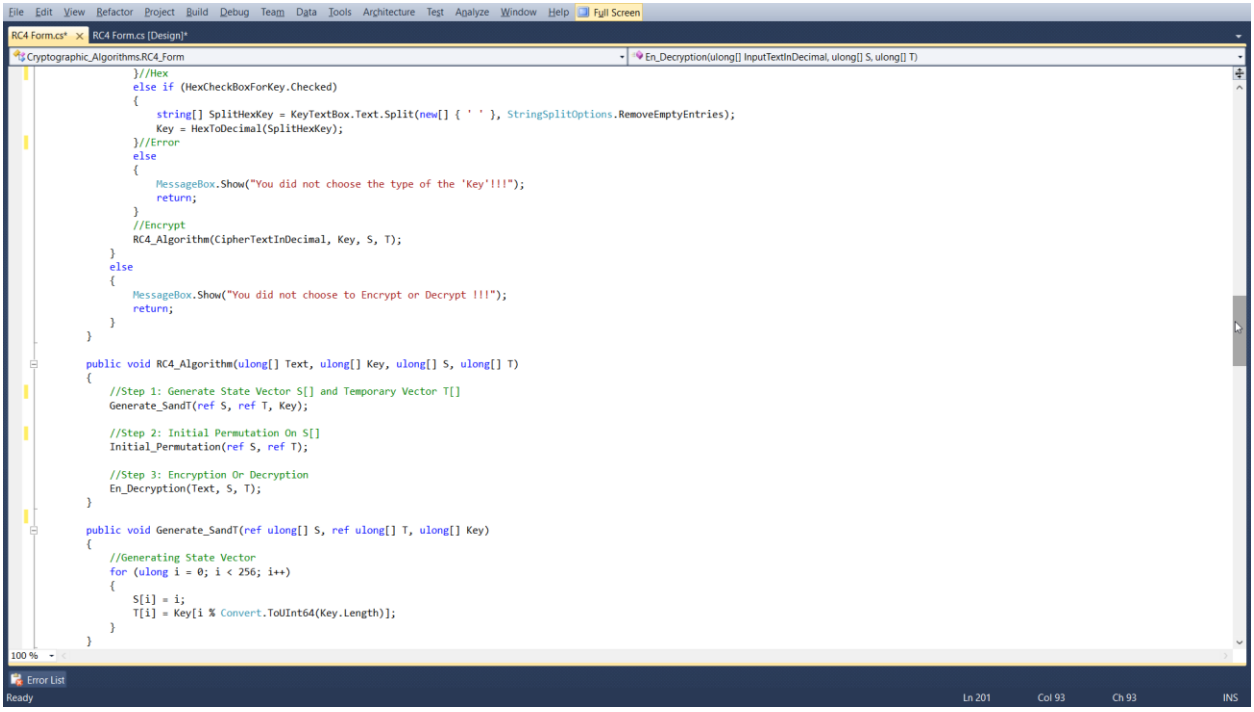
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Cryptographic_Algorithms
{
    public partial class RC4_Form : Form
    {
        public RC4_Form()
        {
            InitializeComponent();
            this.Load += new EventHandler(RC4_Form_Load);
        }
                                EventHandler.EventHandler(void (object, EventArgs) target)
        void RC4_Form_Load(object sender, EventArgs e)
        {
            TextBox.Text = "The quick brown fox jumps over the lazy dog.";
            KeyTextBox.Text = "63 72 79 70 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
```



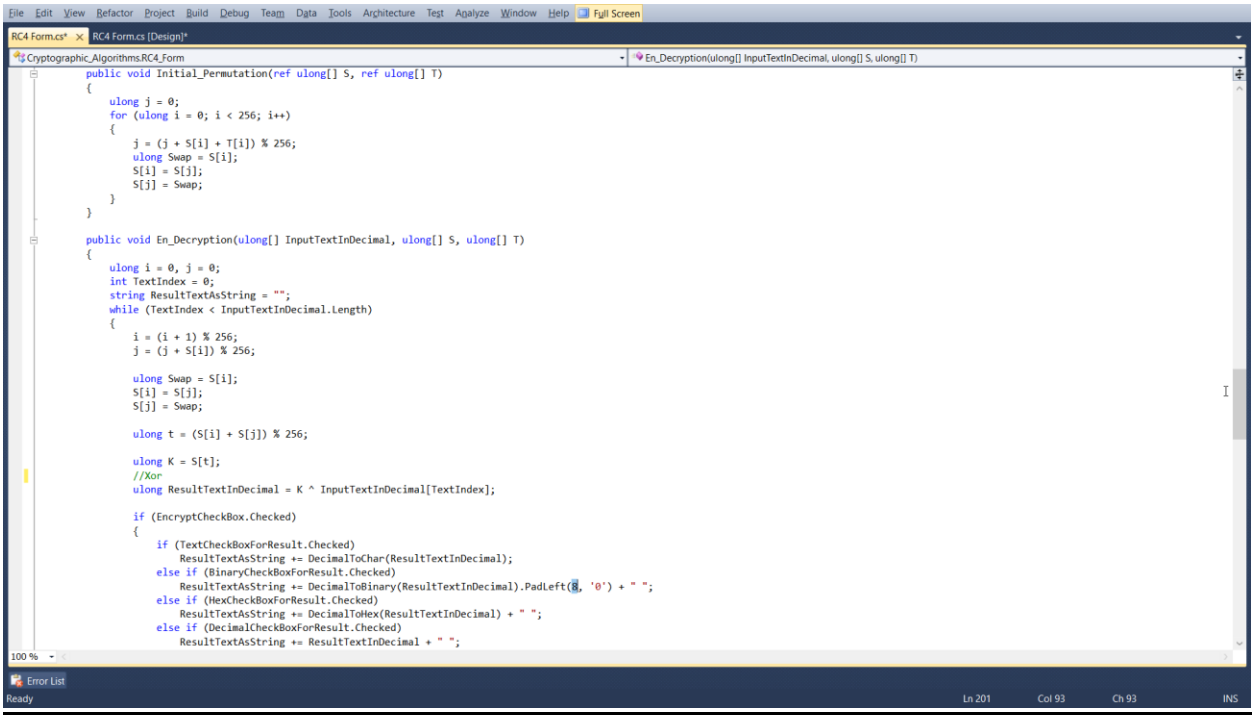
```
                }//Binary
                else if (BinaryCheckBoxForText.Checked)
                {
                    string[] SplitBinaryText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                    PlainTextInDecimal = BinaryToDecimal(SplitBinaryText);
                }//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.ToUInt64(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);
                }
                //Decncrypt
                RC4_Algorithm(PlainTextInDecimal, Key, S, T);

            }
            else if (DecrypytCheckBox.Checked)
```



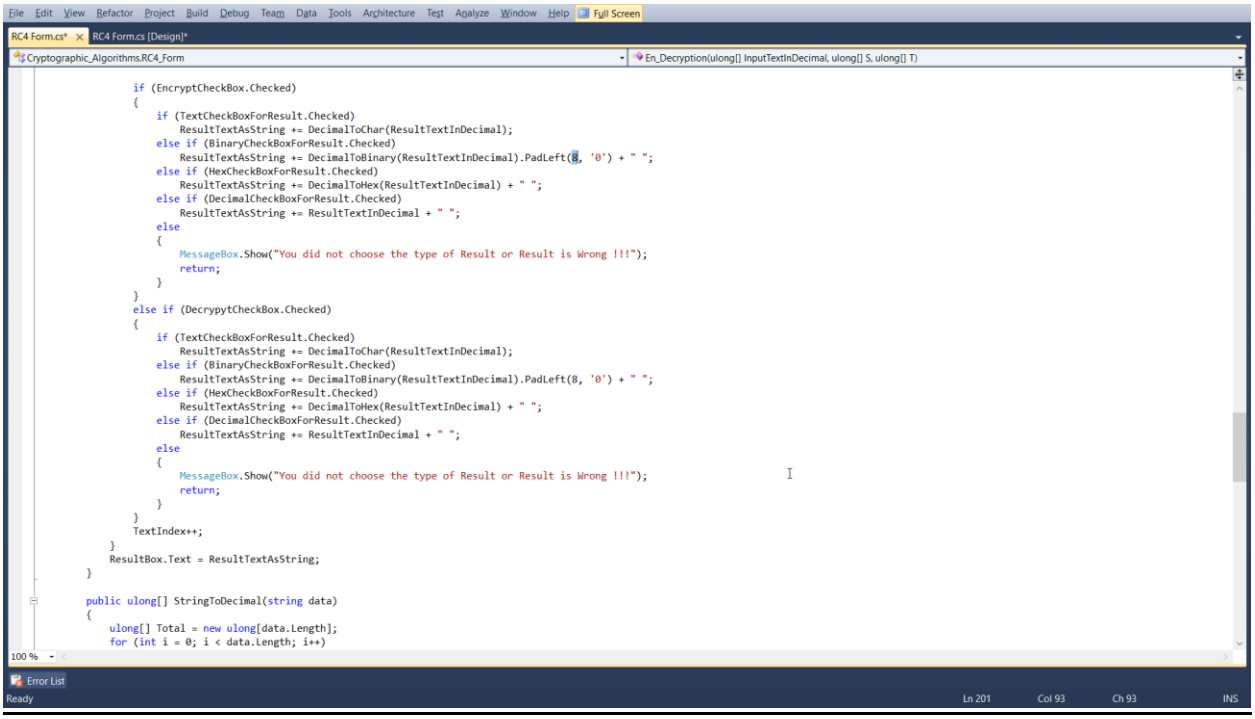
```
            }
            else if (DecrypytCheckBox.Checked)
            {
                //Cipher Text
                //Text
                if (TextCheckBoxForText.Checked)
                {
                    CipherTextInDecimal = StringToDecimal(TextBox.Text);
                }//Binary
                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.ToUInt64(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)
                {
```



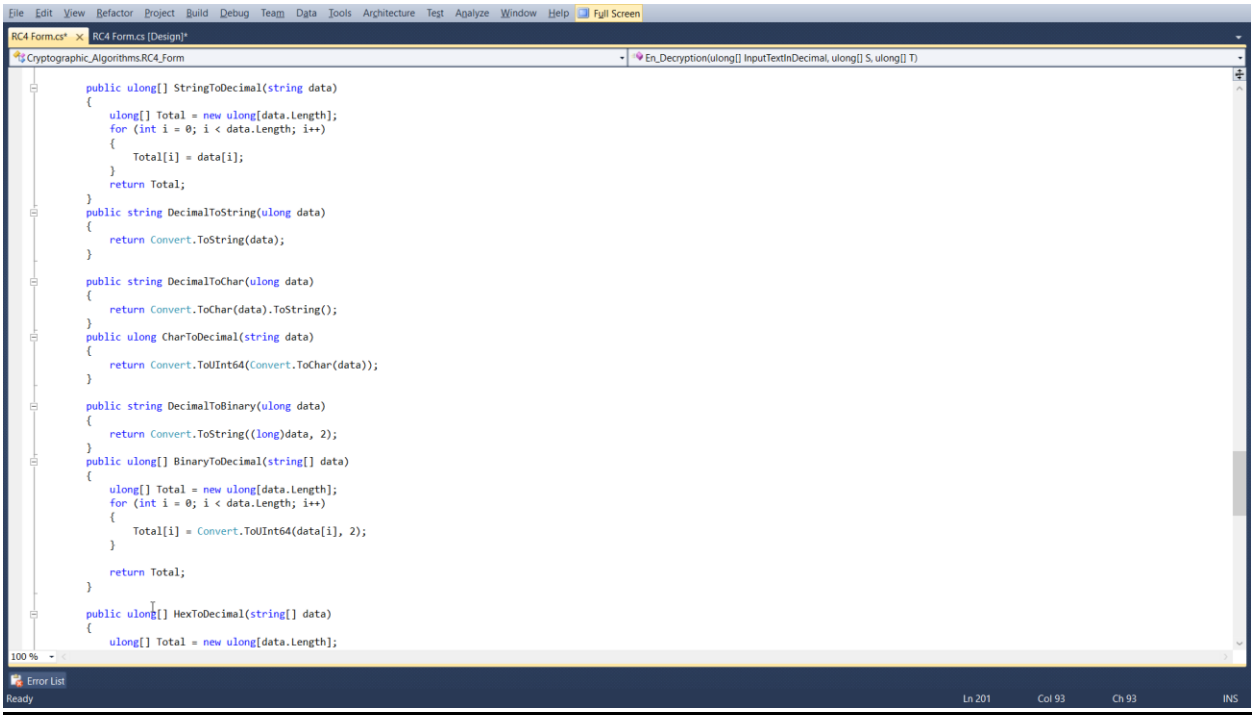
```
                }//Hex
                else if (HexCheckBoxForKey.Checked)
                {
                    string[] SplitHexKey = KeyTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                    Key = HexToDecimal(SplitHexKey);
                }//Error
                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.ToUInt64(Key.Length)];
            }
        }
```



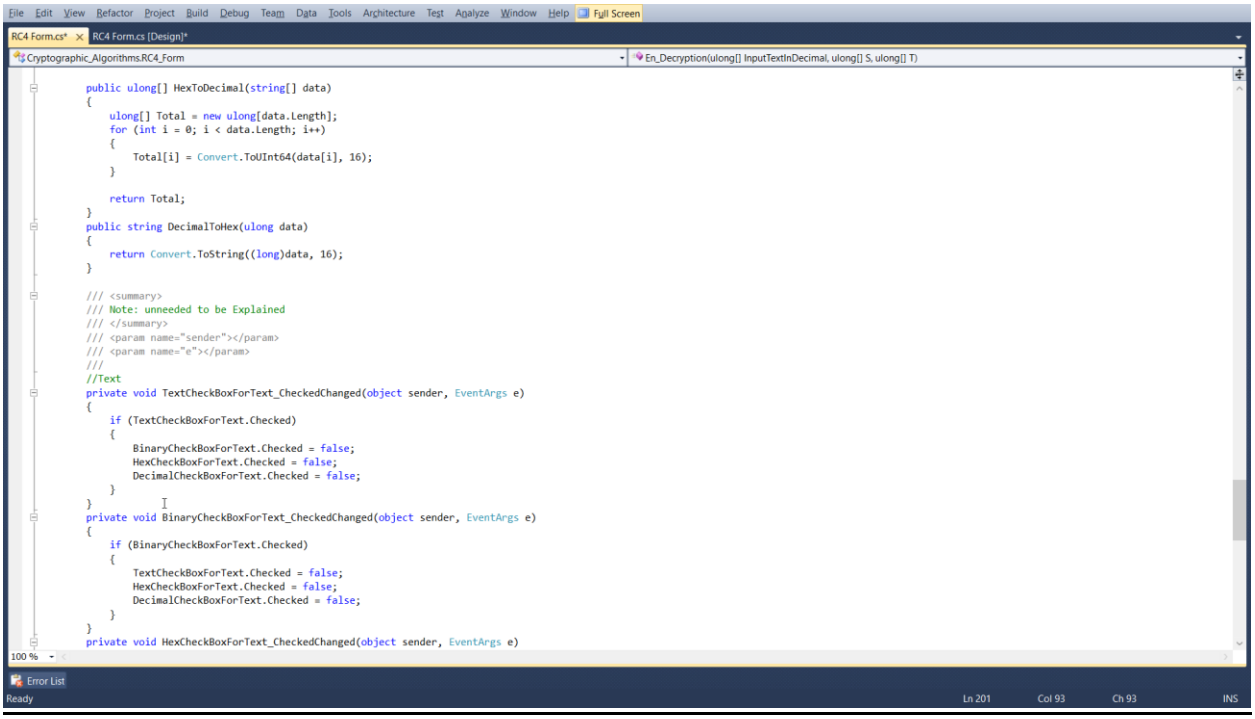
```
public void Initial_Permutation(ref ulong[] S, ref ulong[] T)
{
    ulong j = 0;
    for (ulong i = 0; i < 256; i++)
    {
        j = (j + S[i] + T[i]) % 256;
        ulong Swap = S[i];
        S[i] = S[j];
        S[j] = Swap;
    }
}

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];
        //Xor
        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 + " ";
```



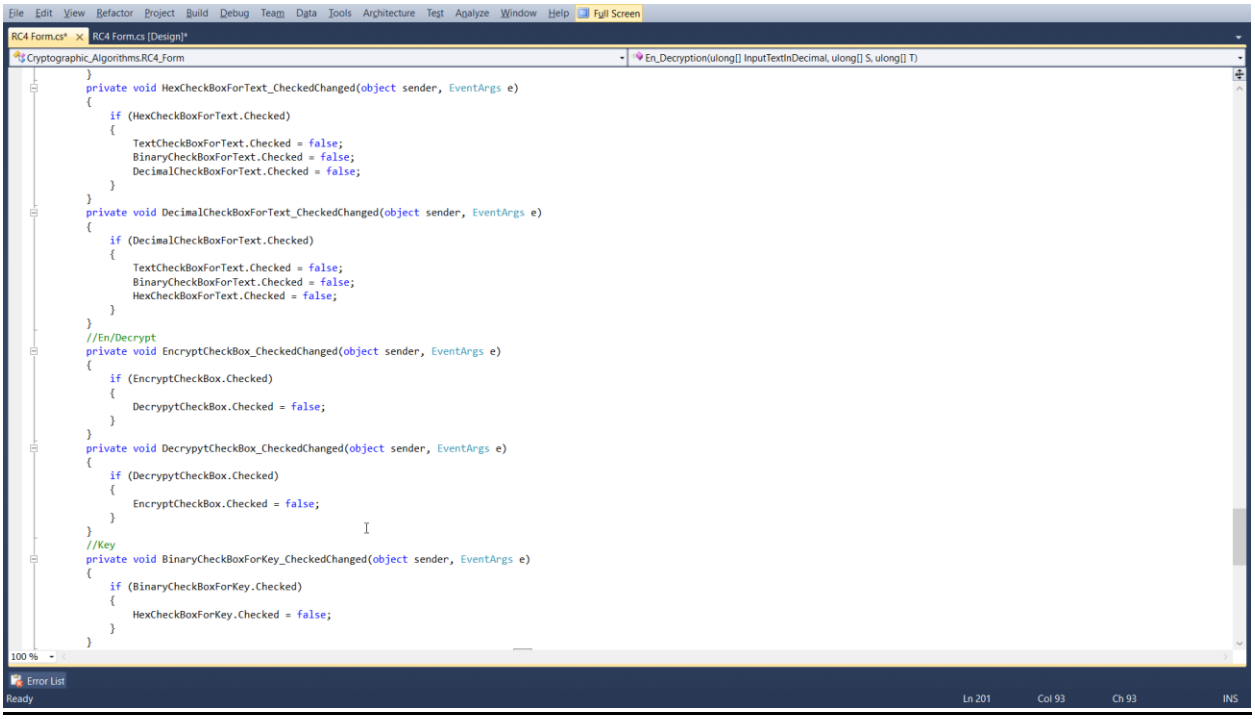
```
            if (EncryptCheckBox.Checked)
            {
                if (TextCheckBoxForResult.Checked)
                    ResultTextAsString += DecimalToChar(ResultTextInDecimal);
                else if (BinaryCheckBoxForResult.Checked)
                    ResultTextAsString += DecimalToBinary(ResultTextInDecimal).PadLeft(8, '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 (DecrypytCheckBox.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++)
```



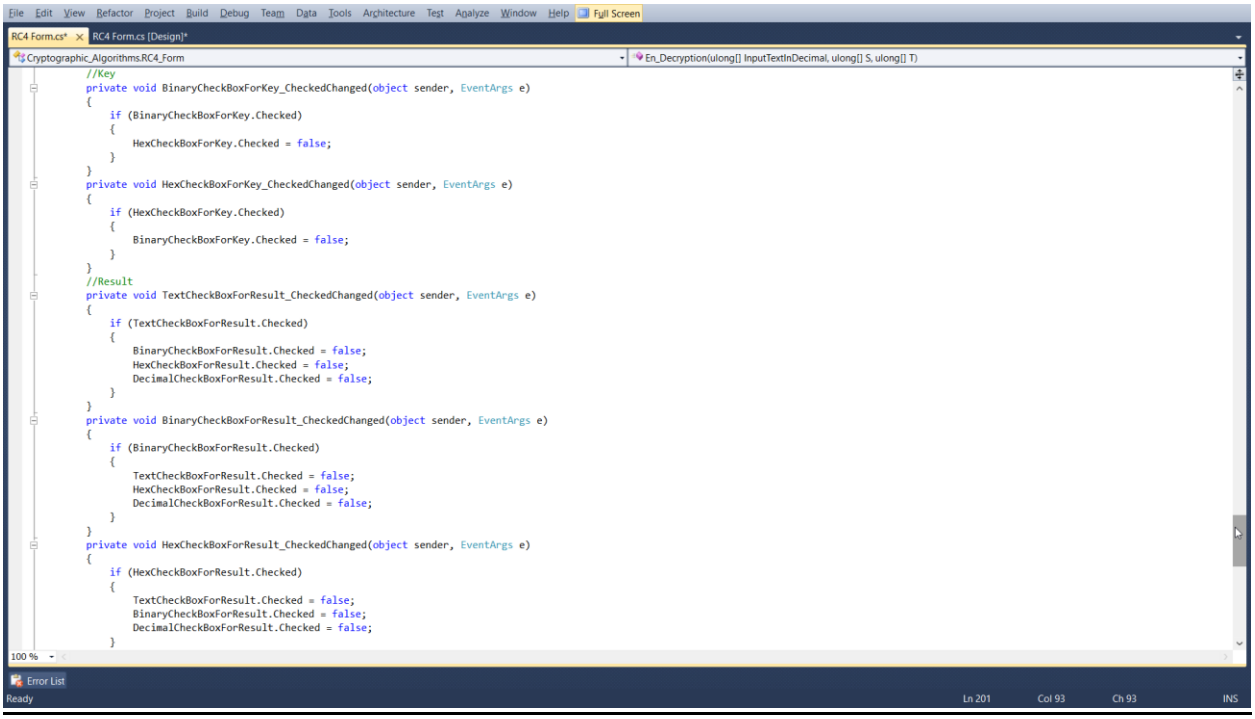
```
        public ulong[] StringToDecimal(string data)
        {
            ulong[] Total = new ulong[data.Length];
            for (int i = 0; i < data.Length; i++)
            {
                Total[i] = data[i];
            }
            return Total;
        }
        public string DecimalToString(ulong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

        public string DecimalToChar(ulong data)
        {
            return Convert.ToChar(data).ToString();
        }
        public ulong CharToDecimal(string data)
        {
            return Convert.ToUInt64(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.ToUInt64(data[i], 2);
            }

            return Total;
        }

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



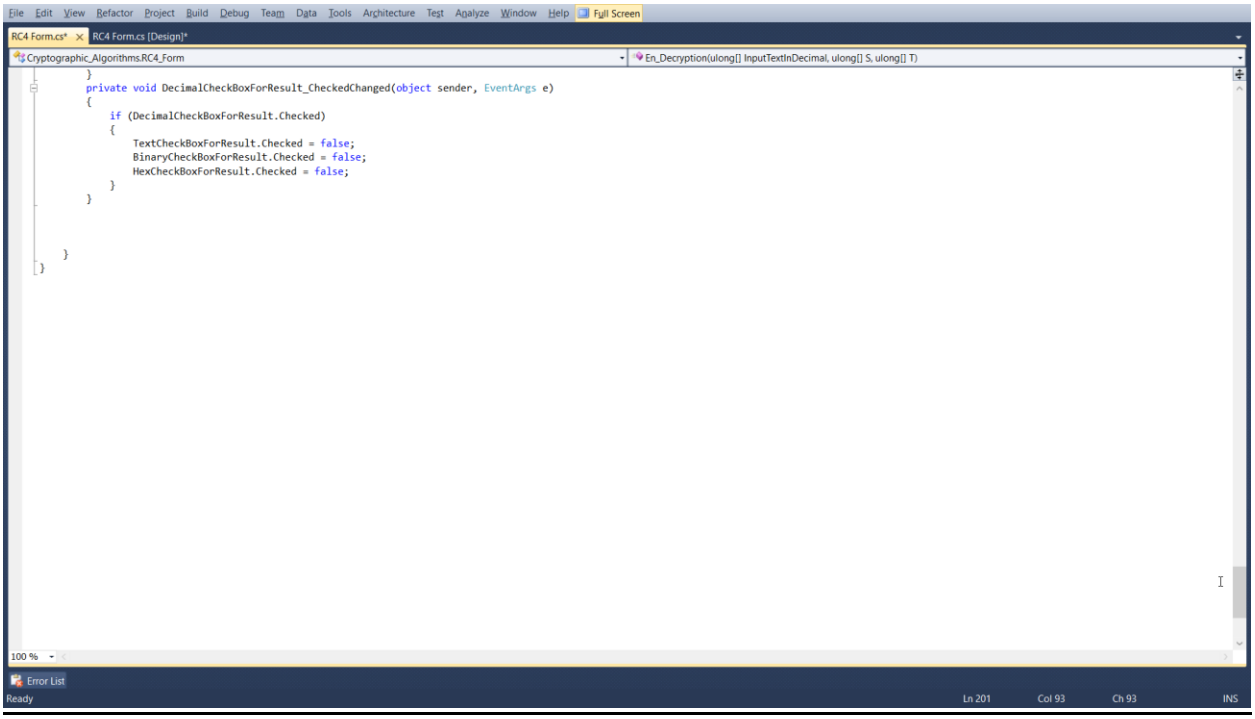
```
        public ulong[] HexToDecimal(string[] data)
        {
            ulong[] Total = new ulong[data.Length];
            for (int i = 0; i < data.Length; i++)
            {
                Total[i] = Convert.ToUInt64(data[i], 16);
            }

            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 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)
```



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



```
        //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;
            }
```


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
En_Decryption(ulong[] InputTextInDecimal, ulong[] S, ulong[] T)
}
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:

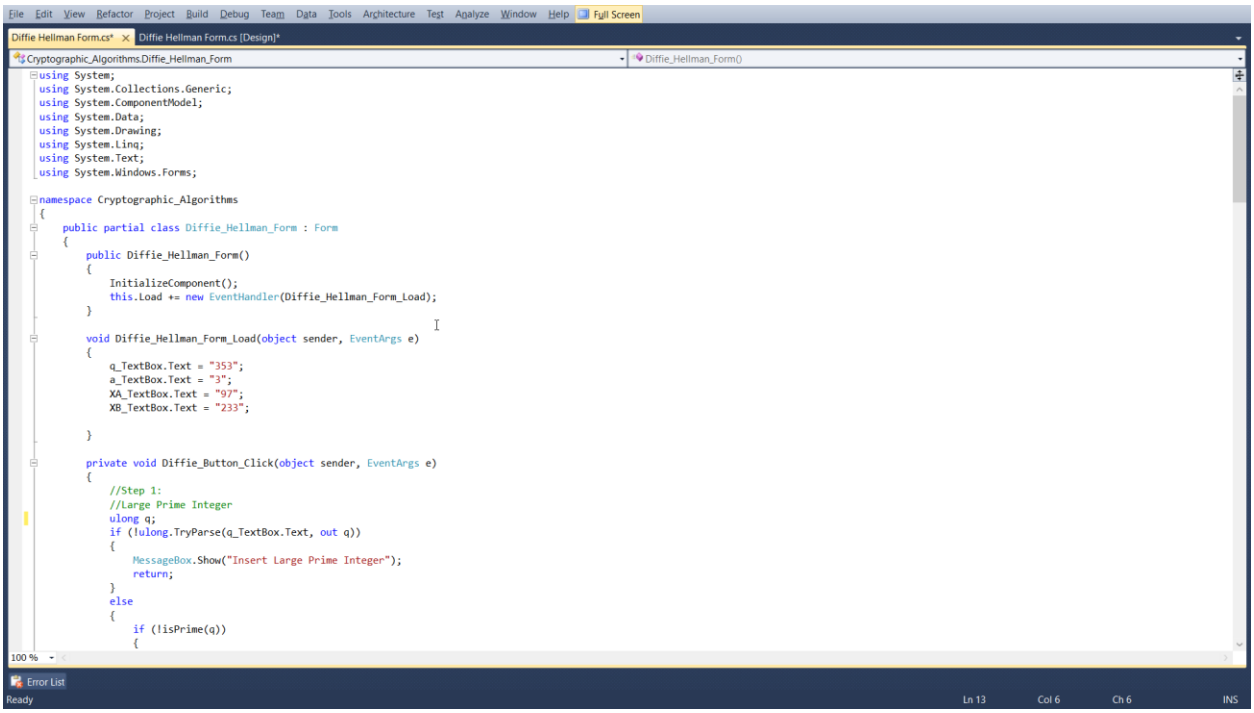
Each user will calculate a Public key using their Secret keys and the Primitive root integer (a), then they will share the 2 Public keys.

Step 4:

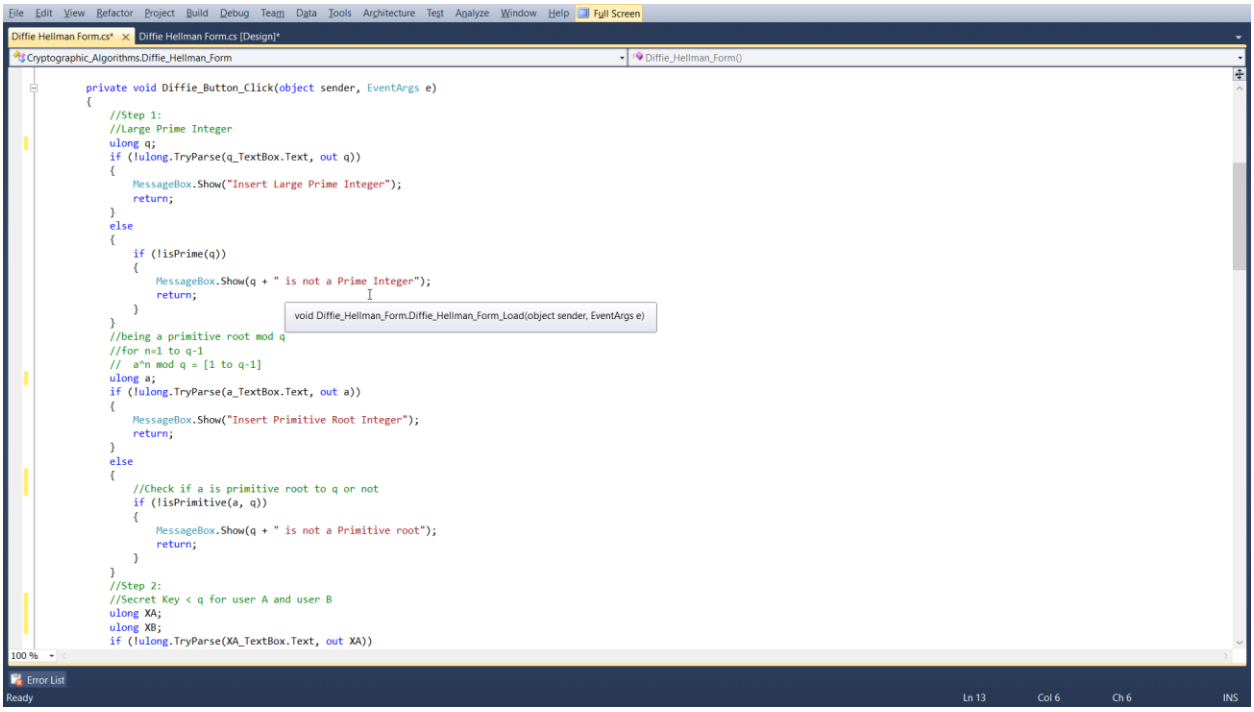
Each user will calculate a Private Key using their Public keys, Secret keys the Large Prime integer (q), where the 2 Private keys will be equal.

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

Screen Shots:

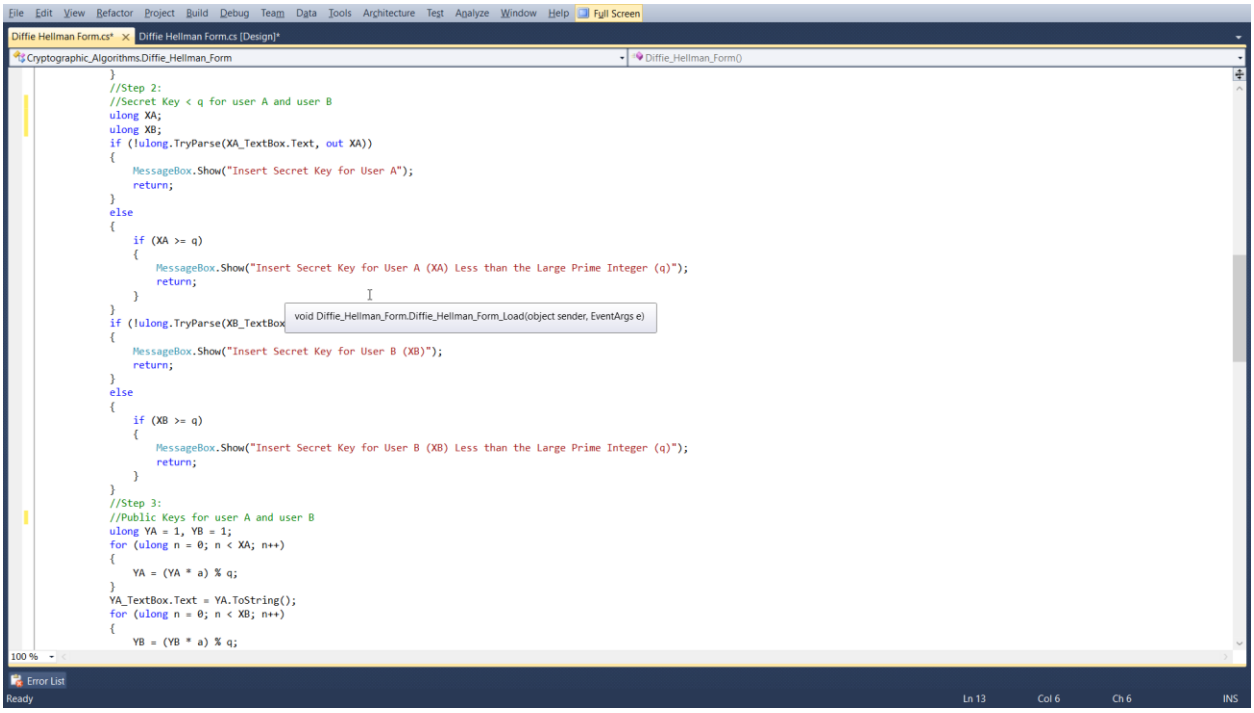


```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

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 = "353";
            a_TextBox.Text = "3";
            XA_TextBox.Text = "97";
            XB_TextBox.Text = "233";

        }

        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))
                {
```



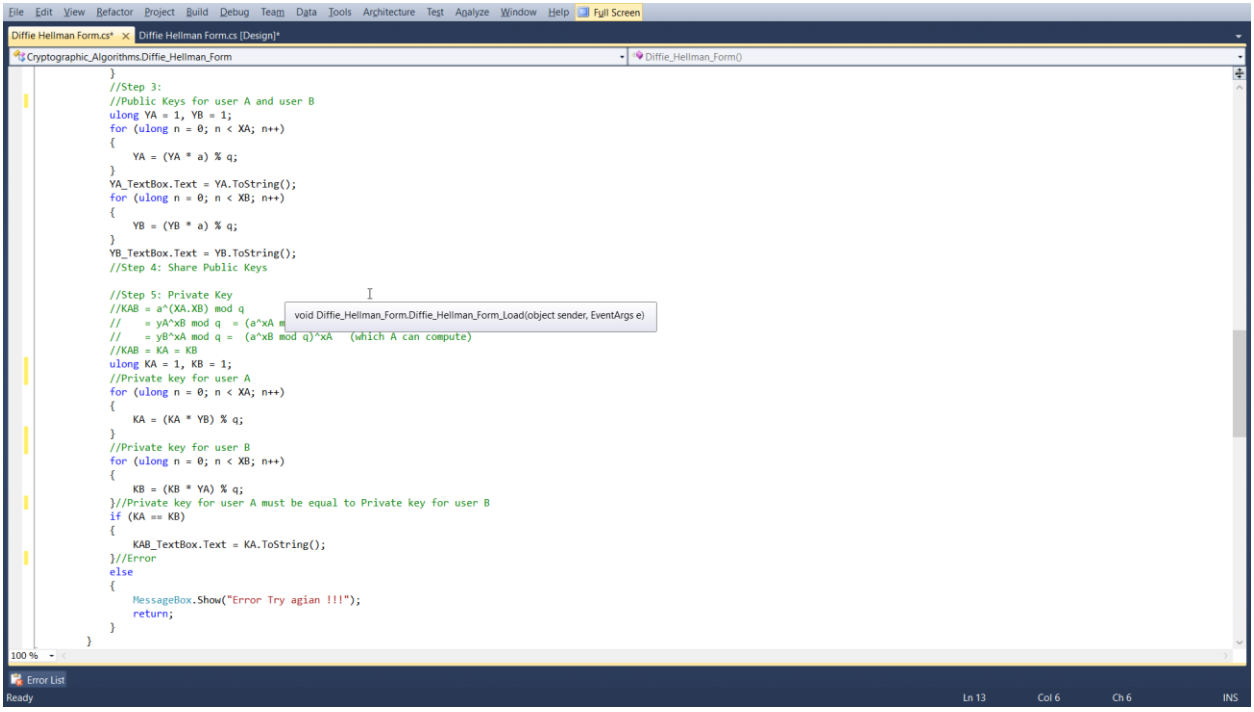
```
        private void Diffie_Button_Click(object sender, EventArgs e)
        {
            //Step 1:
            //Large Prime Integer
            ulong q;
            if (!ulong.TryParse(q_TextBox.Text, out q))
            {
                MessageBox.Show("Insert Large Prime Integer");
                return;
            }
            else
            {
                if (!isPrime(q))
                {
                    MessageBox.Show(q + " is not a Prime Integer");
                    return;
                }
            }
            //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))
```

void Diffie_Hellman_Form.Diffie_Hellman_Form_Load(object sender, EventArgs e)



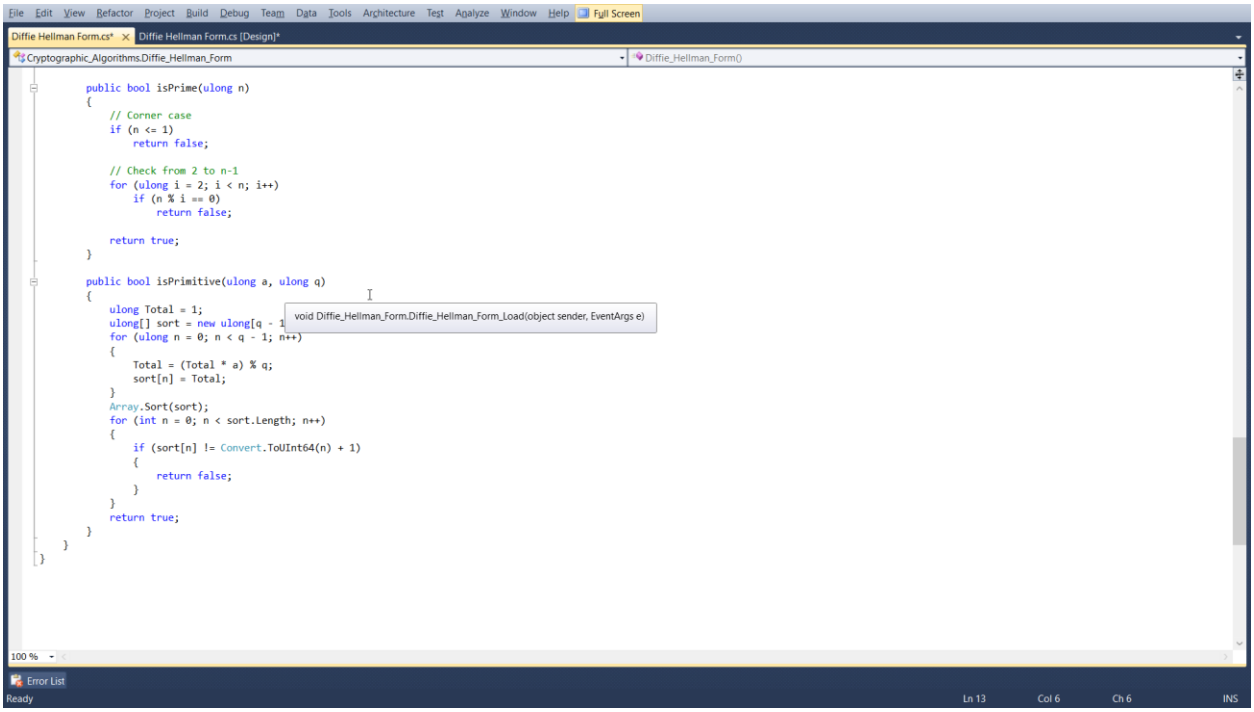
```
        }
        //Step 2:
        //Secret Key < q for user A and user B
        ulong XA;
        ulong XB;
        if (!ulong.TryParse(XA_TextBox.Text, out XA))
        {
            MessageBox.Show("Insert Secret Key for User A");
            return;
        }
        else
        {
            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
        {
            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;
```

void Diffie_Hellman_Form.Diffie_Hellman_Form_Load(object sender, EventArgs e)



```
        }
        //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 m
        //    = yB^xA 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();
        }//Error
        else
        {
            MessageBox.Show("Error Try agian !!!");
            return;
        }
    }
```

void Diffie_Hellman_Form.Diffie_Hellman_Form_Load(object sender, EventArgs e)


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
Diffie_Hellman_Form()
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
void Diffie_Hellman_Form.Diffie_Hellman_Form_Load(object sender, EventArgs e)
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.ToUInt64(n) + 1)
{
return false;
}
}
return true;
}
}
}
100 %
Error List
Ready
Ln 13
Col 6
Ch 6
INS

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.

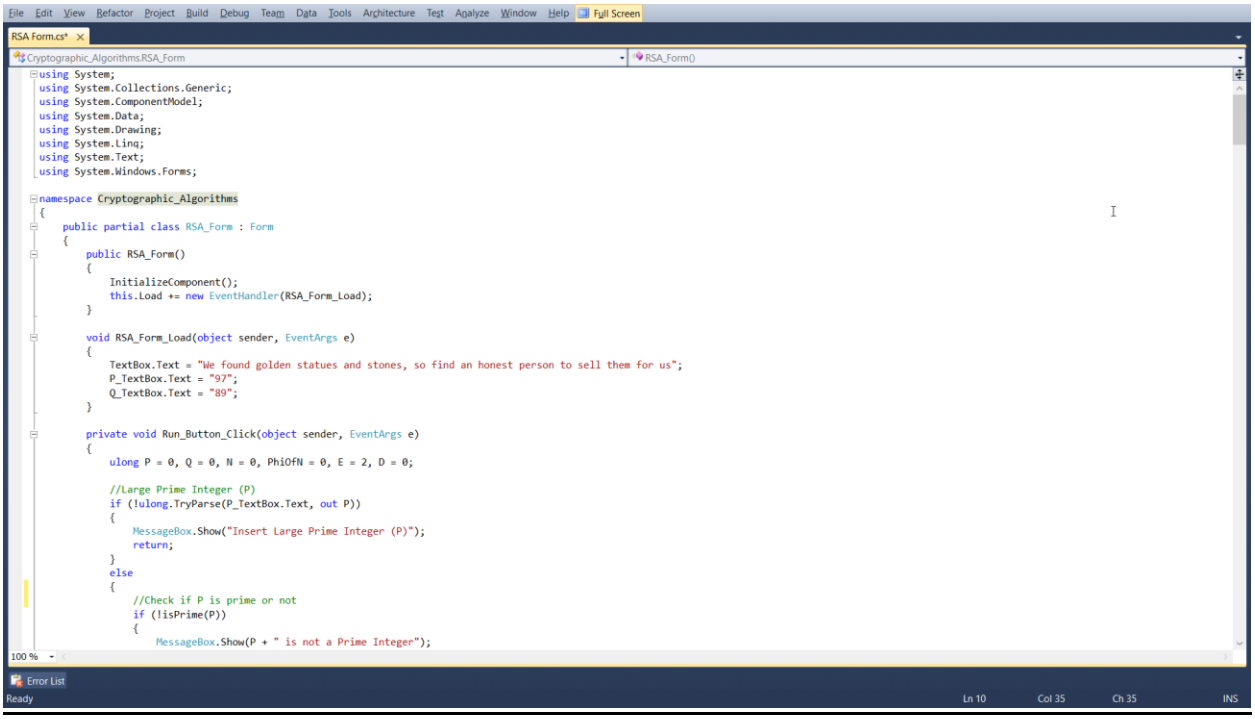
Steps:

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 <$\emptyset(n)$), where E must be co-prime to $\emptyset(n)$ 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 <$\emptyset(n)$ where <$\emptyset(n) = (P-1) * (Q-1)$ and the second variable of the private key is d where d is the mod inverse of E and $\emptyset(n)$ where d = $E^{-1} \, mod \, \emptyset(n)$.

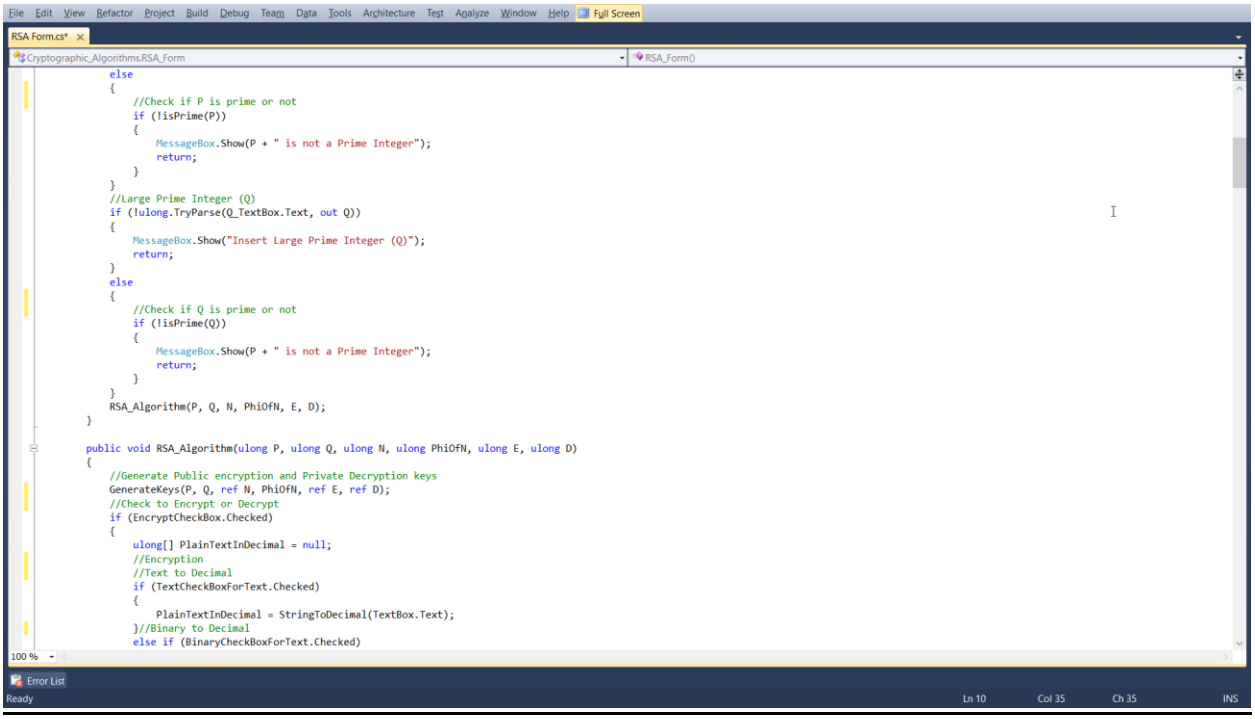
Step2: Encryption or Decryption
In Encryption:
The cipher text is equal to the plaintext power of E modules N ($C = M^E \, mod \, N$).
In Decryption:
The plaintext is equal to the cipher text power of D modules N ($P = C^D \, mod \, N$).

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

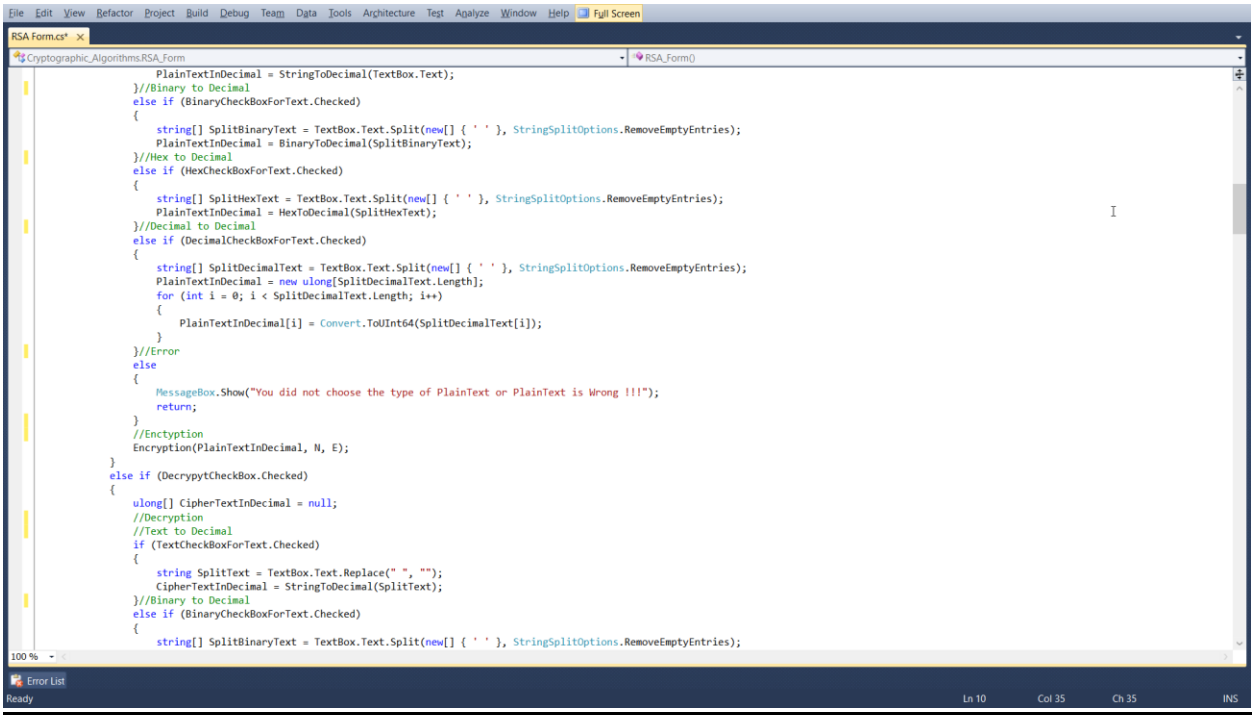
Screen Shots:



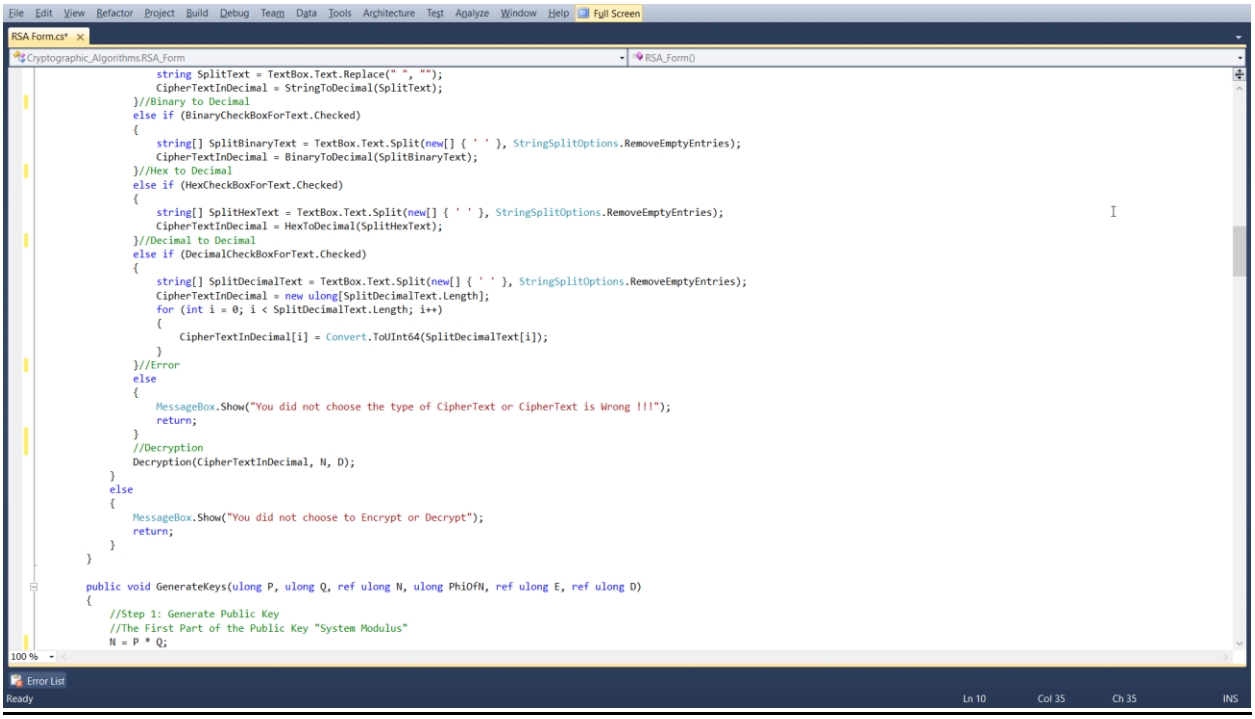
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

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");
```



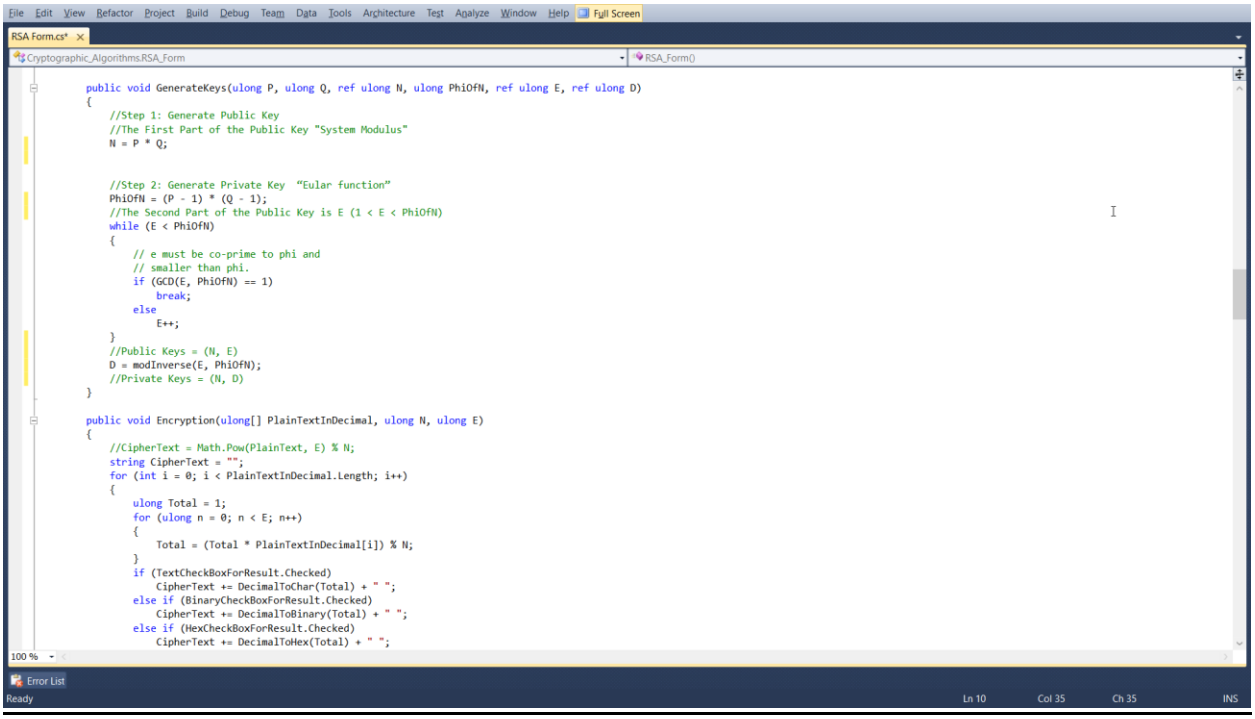
```
            else
            {
                //Check if P is prime or not
                if (!isPrime(P))
                {
                    MessageBox.Show(P + " is not a Prime Integer");
                    return;
                }
            }
            //Large Prime Integer (Q)
            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");
                    return;
                }
            }
            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)
```

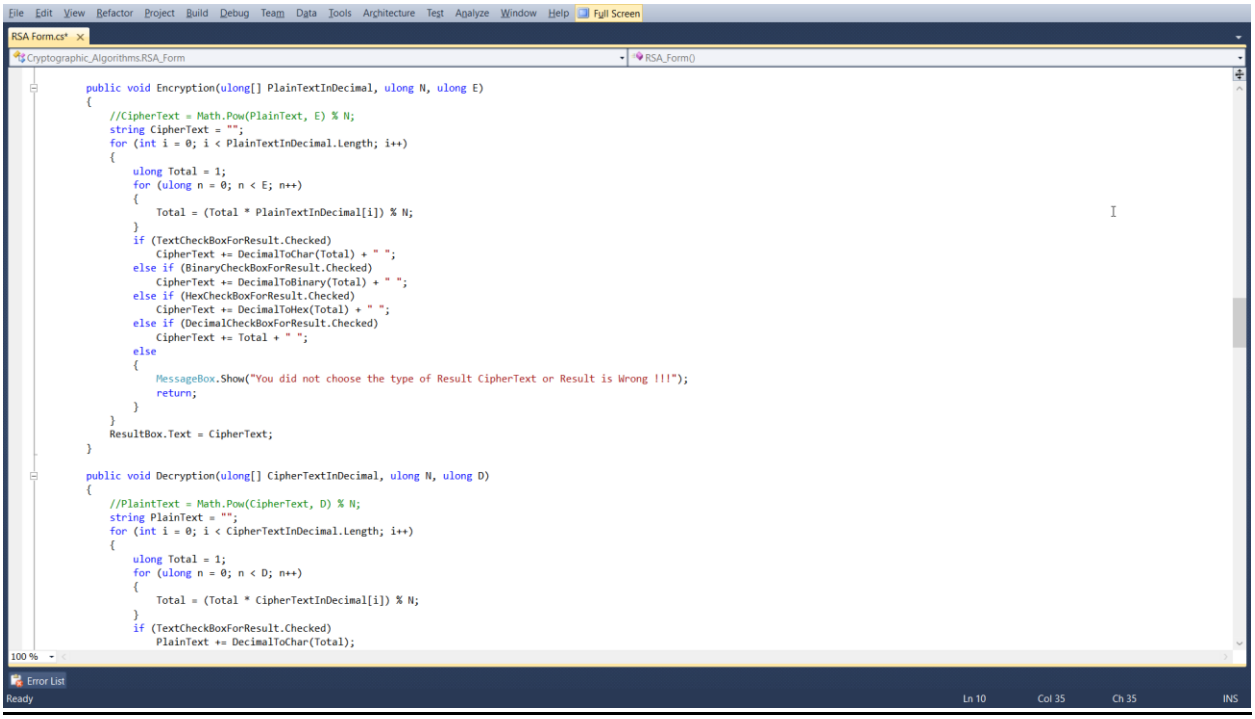


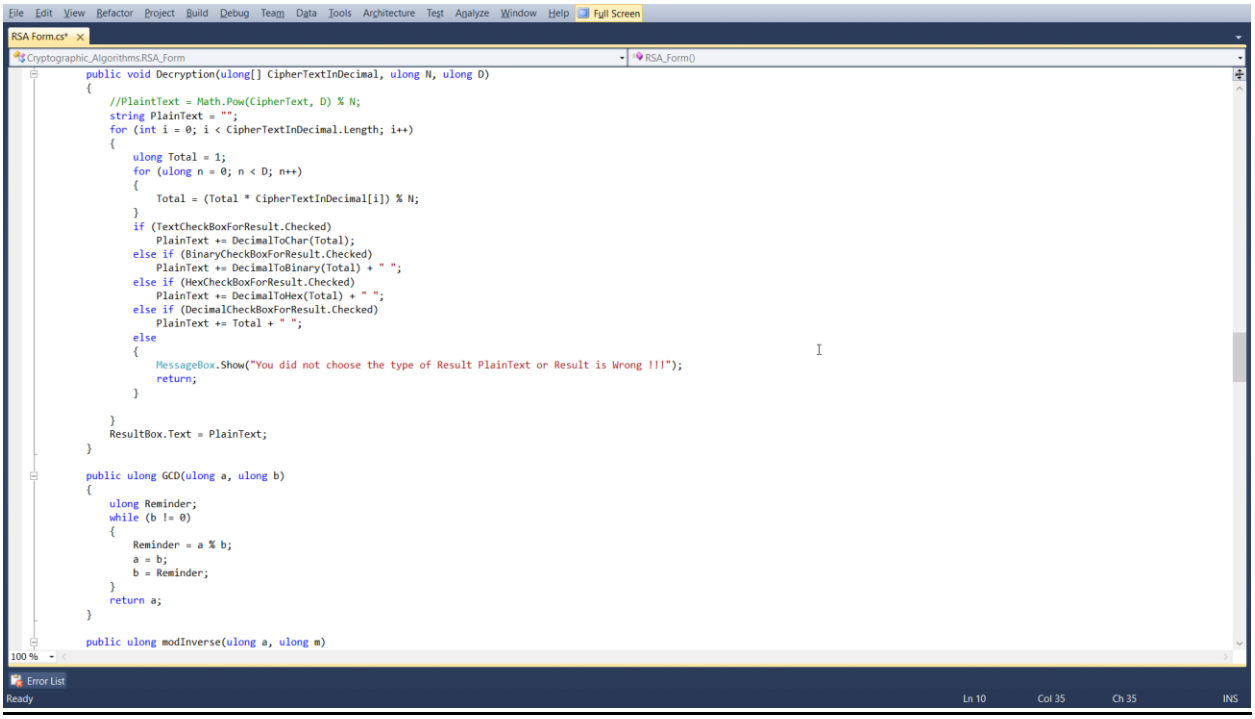
```
                    PlainTextInDecimal = StringToDecimal(TextBox.Text);
                }//Binary to Decimal
                else if (BinaryCheckBoxForText.Checked)
                {
                    string[] SplitBinaryText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                    PlainTextInDecimal = BinaryToDecimal(SplitBinaryText);
                }//Hex to Decimal
                else if (HexCheckBoxForText.Checked)
                {
                    string[] SplitHexText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                    PlainTextInDecimal = HexToDecimal(SplitHexText);
                }//Decimal to 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.ToUInt64(SplitDecimalText[i]);
                    }
                }//Error
                else
                {
                    MessageBox.Show("You did not choose the type of PlainText or PlainText is Wrong !!!");
                    return;
                }
                //Enctyption
                Encryption(PlainTextInDecimal, N, E);
            }
            else if (DecrypytCheckBox.Checked)
            {
                ulong[] CipherTextInDecimal = null;
                //Decryption
                //Text to Decimal
                if (TextCheckBoxForText.Checked)
                {
                    string SplitText = TextBox.Text.Replace(" ", "");
                    CipherTextInDecimal = StringToDecimal(SplitText);
                }//Binary to Decimal
                else if (BinaryCheckBoxForText.Checked)
                {
                    string[] SplitBinaryText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
```



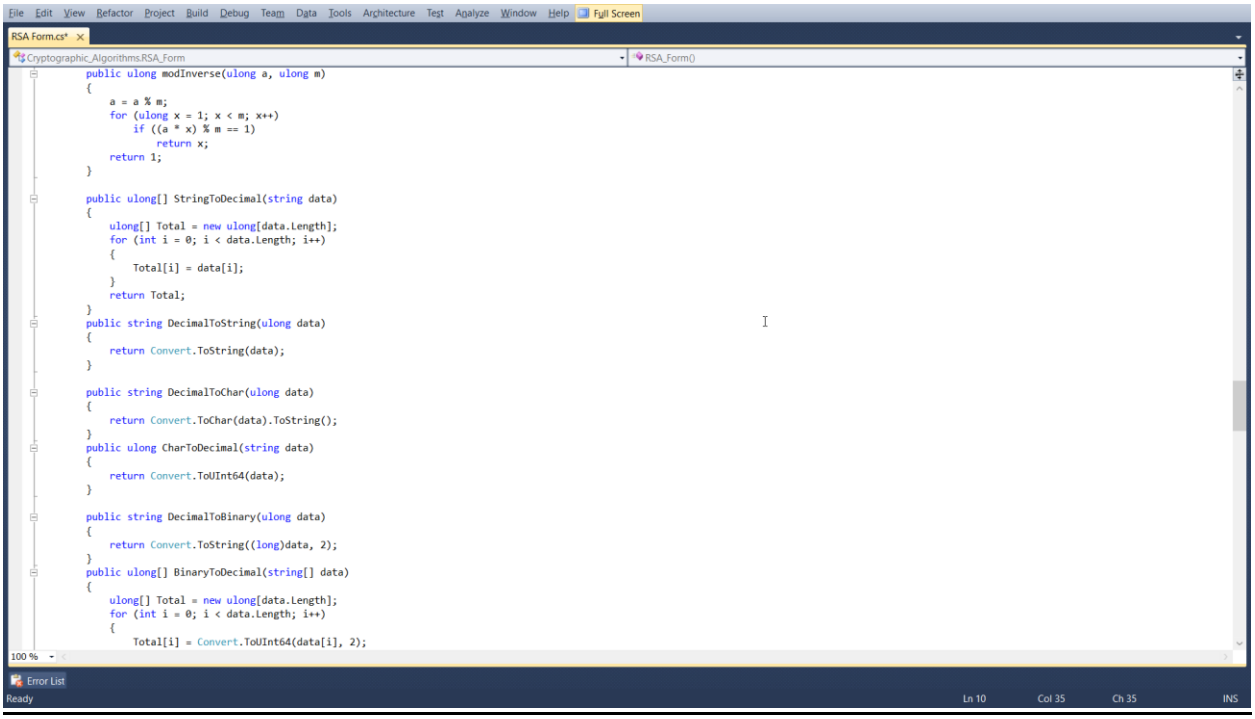
```
                    string SplitText = TextBox.Text.Replace(" ", "");
                    CipherTextInDecimal = StringToDecimal(SplitText);
                }//Binary to Decimal
                else if (BinaryCheckBoxForText.Checked)
                {
                    string[] SplitBinaryText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                    CipherTextInDecimal = BinaryToDecimal(SplitBinaryText);
                }//Hex to Decimal
                else if (HexCheckBoxForText.Checked)
                {
                    string[] SplitHexText = TextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                    CipherTextInDecimal = HexToDecimal(SplitHexText);
                }//Decimal to 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.ToUInt64(SplitDecimalText[i]);
                    }
                }//Error
                else
                {
                    MessageBox.Show("You did not choose the type of CipherText or CipherText is Wrong !!!");
                    return;
                }
                //Decryption
                Decryption(CipherTextInDecimal, N, D);
            }
            else
            {
                MessageBox.Show("You did not choose to Encrypt or Decrypt");
                return;
            }
        }

        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;
```


File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RSA Form.cs*
Cryptographic_Algorithms.RSA_Form
RSA_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) + " ";
100 %
Error List
Ready
Ln 10
Col 35
Ch 35
INS

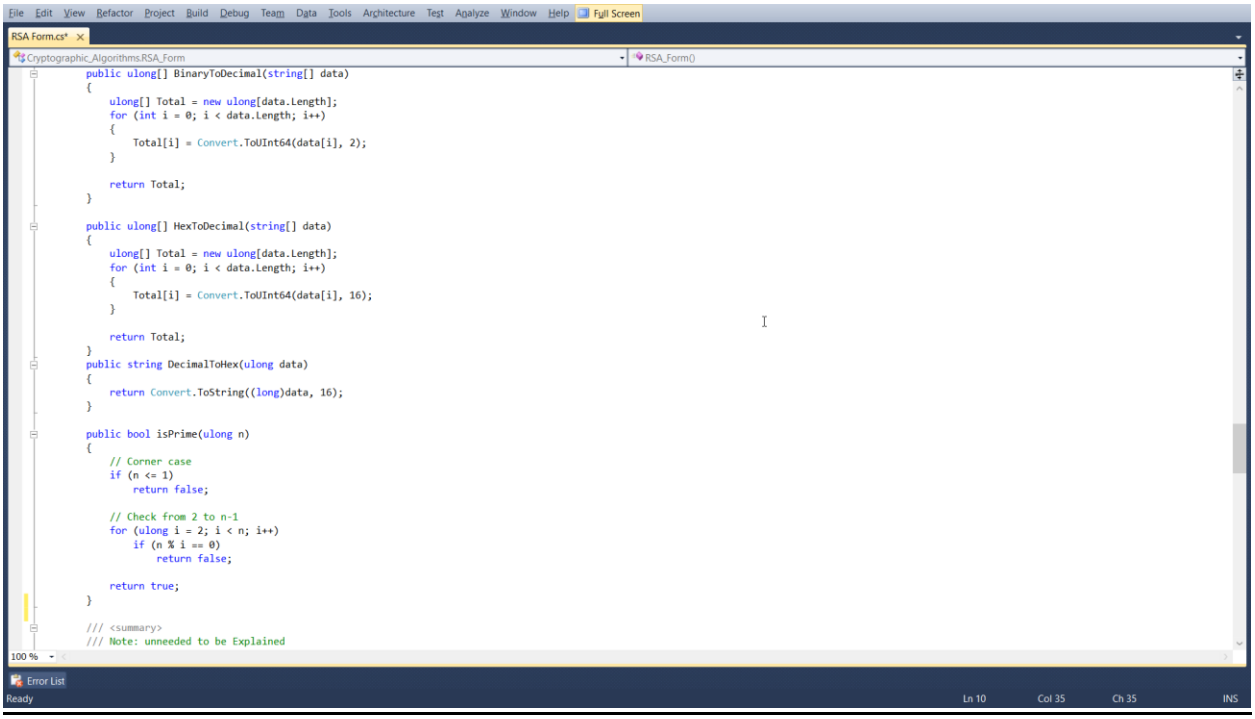

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RSA Form.cs*
Cryptographic_Algorithms.RSA_Form
RSA_Form()
public void Encryption(ulong[] PlainTextInDecimal, ulong N, ulong E)
{
//CipherText = Math.Pow(PlainText, E) % N;
string CipherText = "";
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)
{
//PlaintText = 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);
100 %
Error List
Ready
Ln 10
Col 35
Ch 35
INS

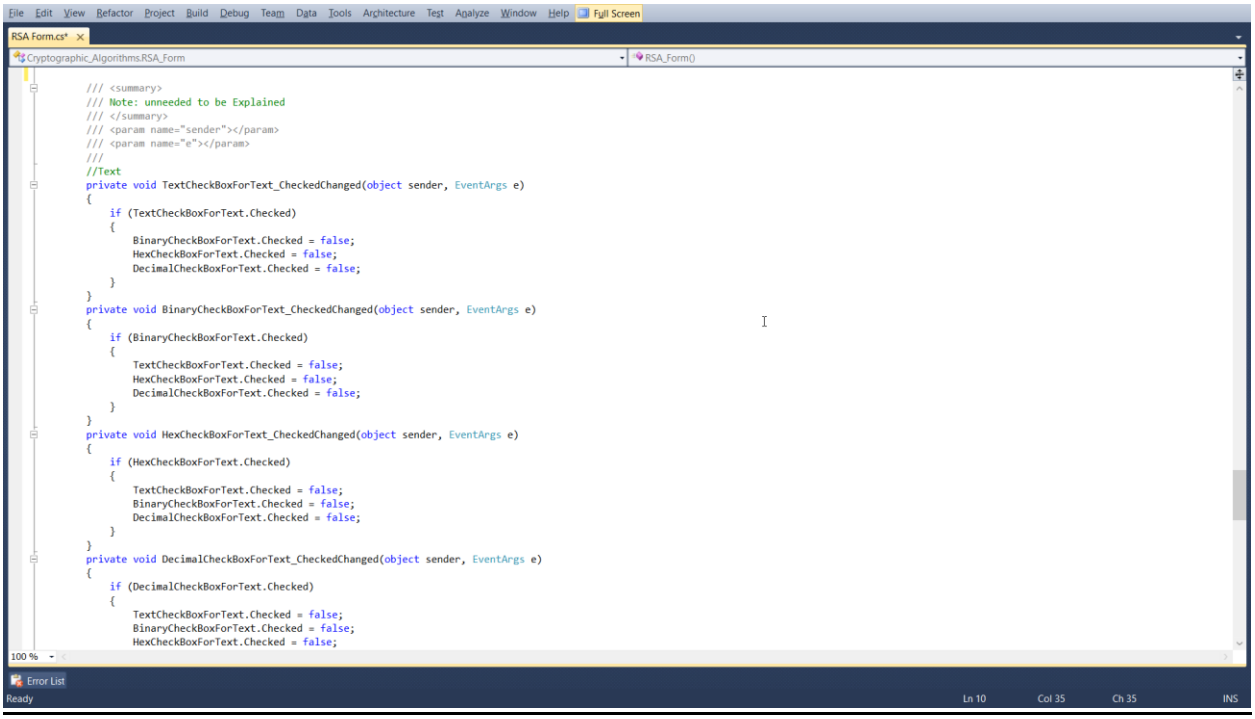


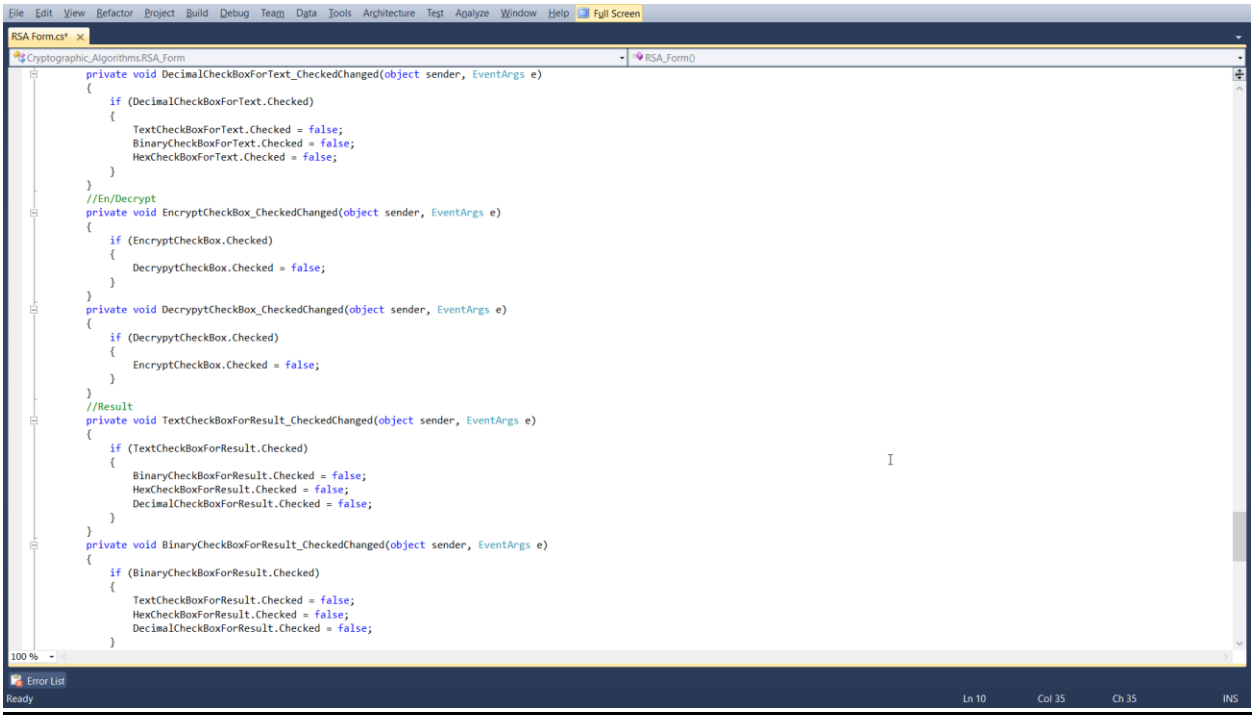
```
public void Decryption(ulong[] CipherTextInDecimal, ulong N, ulong D)
{
    //PlaintText = Math.Pow(CipherText, D) % N;
    string PlainText = "";
    for (int i = 0; i < CipherTextInDecimal.Length; i++)
    {
        ulong Total = 1;
        for (ulong n = 0; n < D; n++)
        {
            Total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

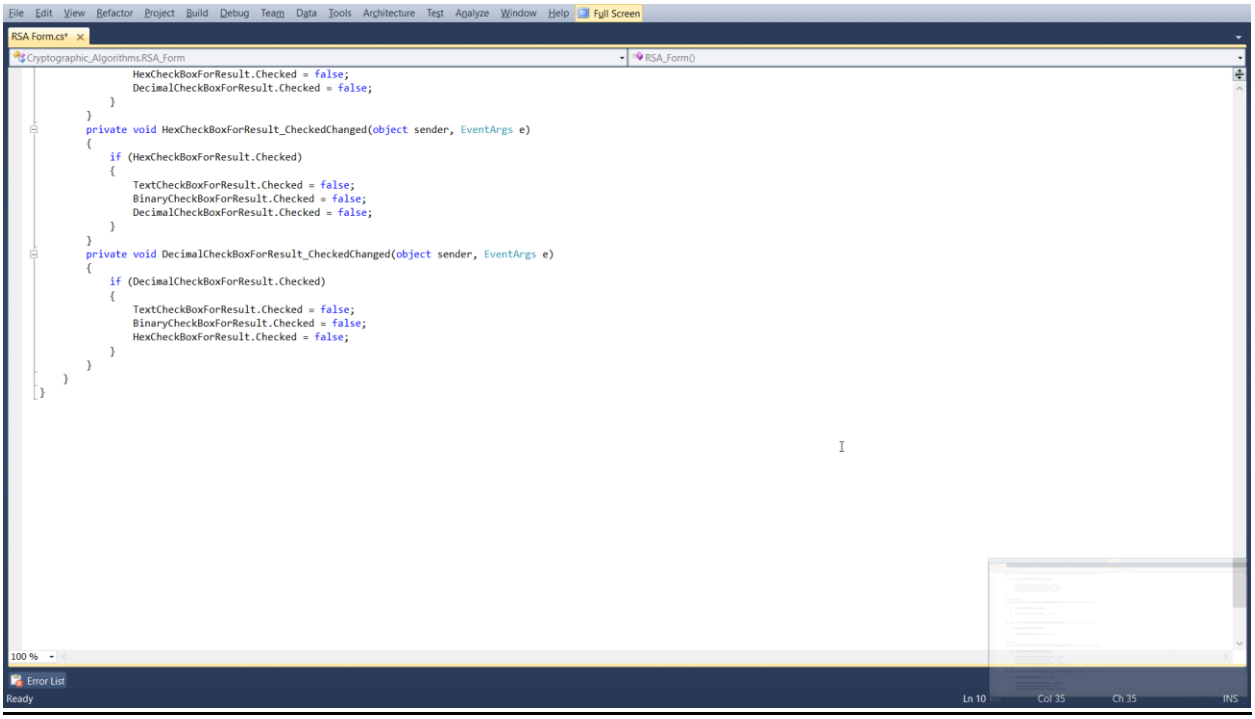


```
public ulong modInverse(ulong a, ulong m)
{
    a = a % m;
    for (ulong x = 1; x < m; x++)
        if ((a * x) % m == 1)
            return x;
    return 1;
}

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.ToUInt64(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.ToUInt64(data[i], 2);
```


File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
RSA Form.cs*
Cryptographic_Algorithms.RSA_Form
RSA_Form()
public ulong[] BinaryToDecimal(string[] data)
{
ulong[] Total = new ulong[data.Length];
for (int i = 0; i < data.Length; i++)
{
Total[i] = Convert.ToUInt64(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.ToUInt64(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*
Cryptographic_Algorithms.RSA_Form
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*
Cryptographic_Algorithms.RSA_Form
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)
{
DecrypytCheckBox.Checked = false;
}
}
private void DecrypytCheckBox_CheckedChanged(object sender, EventArgs e)
{
if (DecrypytCheckBox.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*
Cryptographic_Algorithms.RSA_Form
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 (XA) 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 (XA) then user A compute a public key using the primitive root (a), the Secret Key (XA), and the large prime integer (q), where public key (YA) $YA = a^{XA} \ mod \ q$. The chosen large prime integer and the Secret Key (XA) are shared between the two users.

Step 2: Encryption

User B select random integer k (1 <= k < q), then compute one-time key ($K = YA^k \ mod \ q$). Then compute the cipher text, where the cipher text is equal to C1 and C2, since $C1 = a^k \ mod \ q$ and $C2 = KM \ mod \ q$, where M is the plain text and cipher text is equal to (C1, C2).

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:

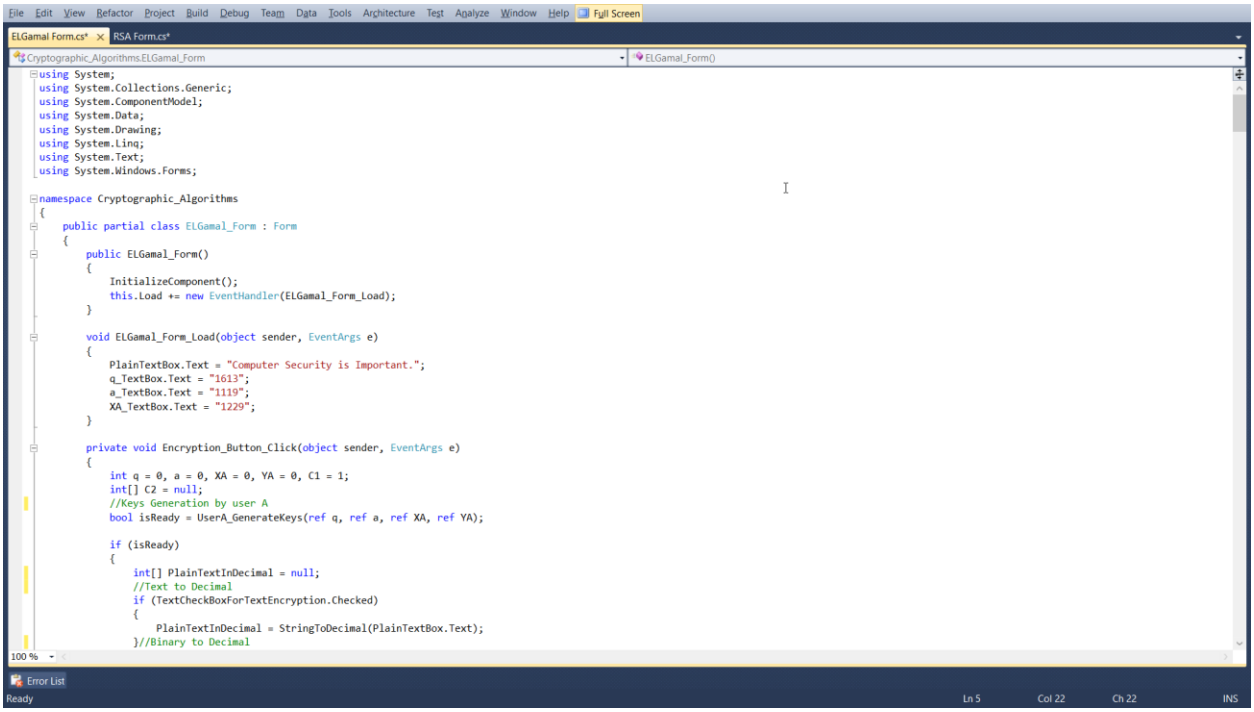
User A uses the same large prime integer (q) and the Secret Key of user A (XA).

Step 2: Decryption

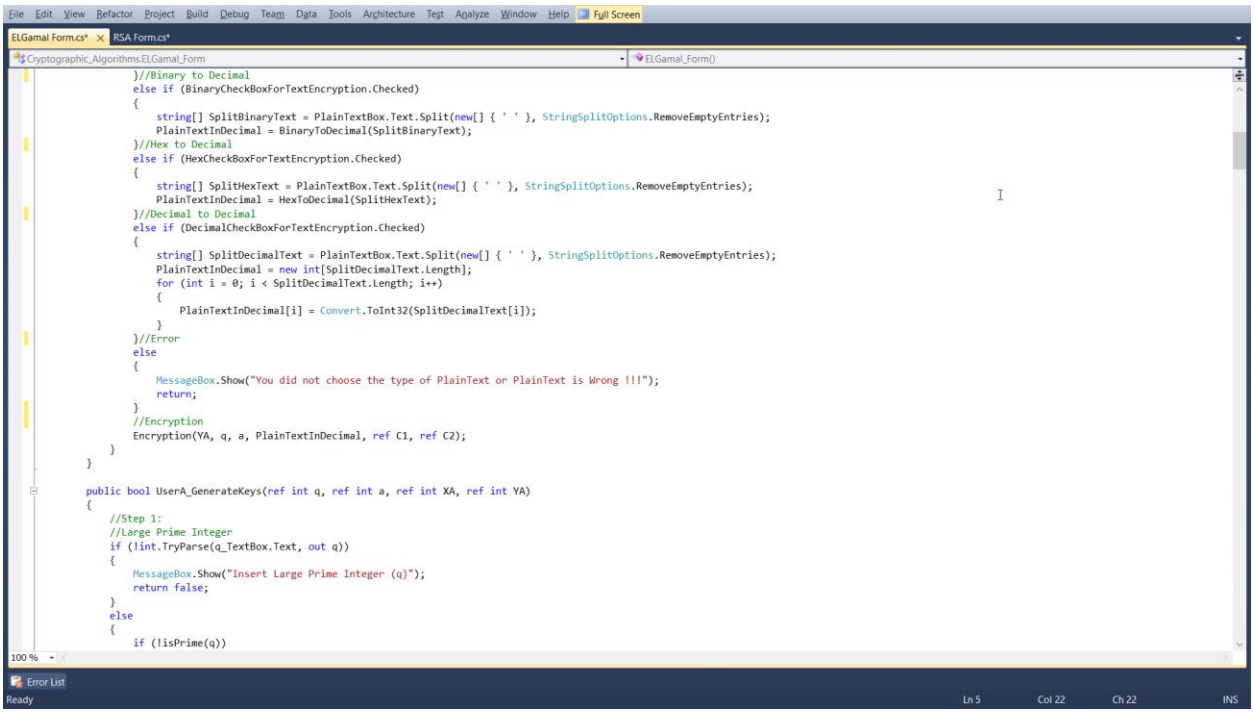
User A compute one-time key using C1 of the cipher text, his secret key and the large prime integer (q) ($K = C1^{XA} \ mod \ q$). Then compute the K inverse by using K and q inModules inverse function. Then compute the plain text using C2, K inverse, and q where ($P = M = C2 * K^{-1} \ mod \ q$).

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

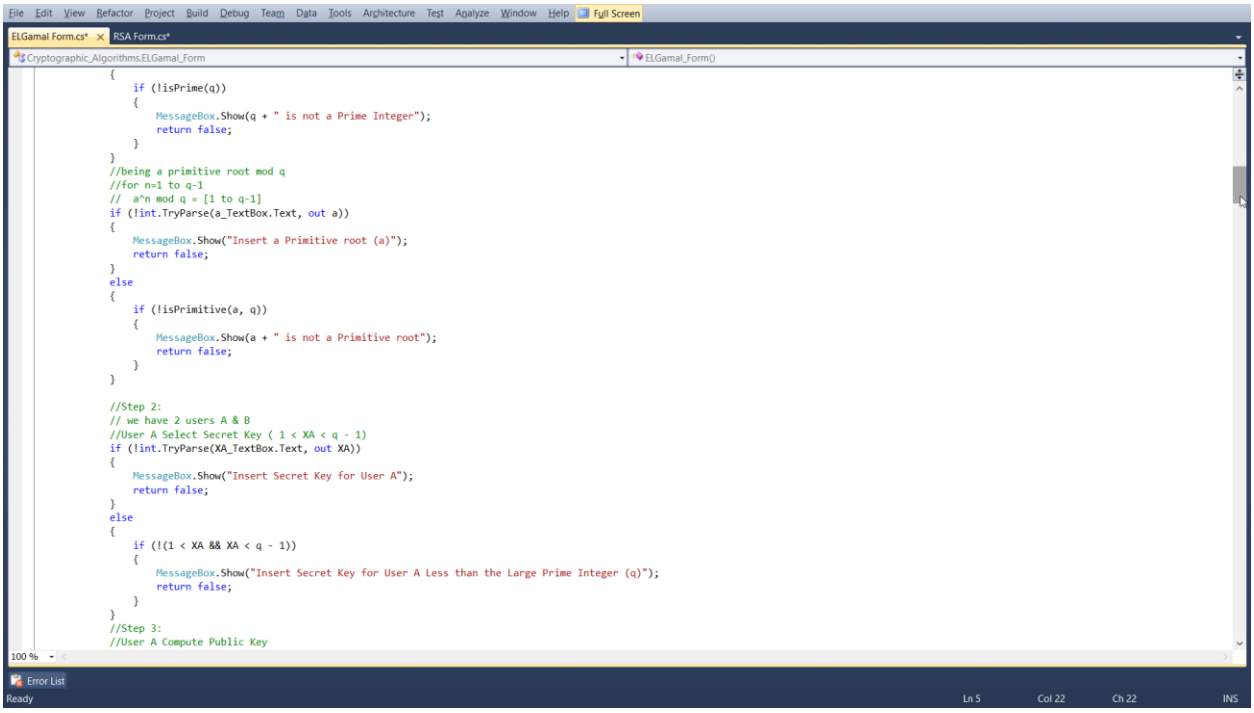
Screen Shots:



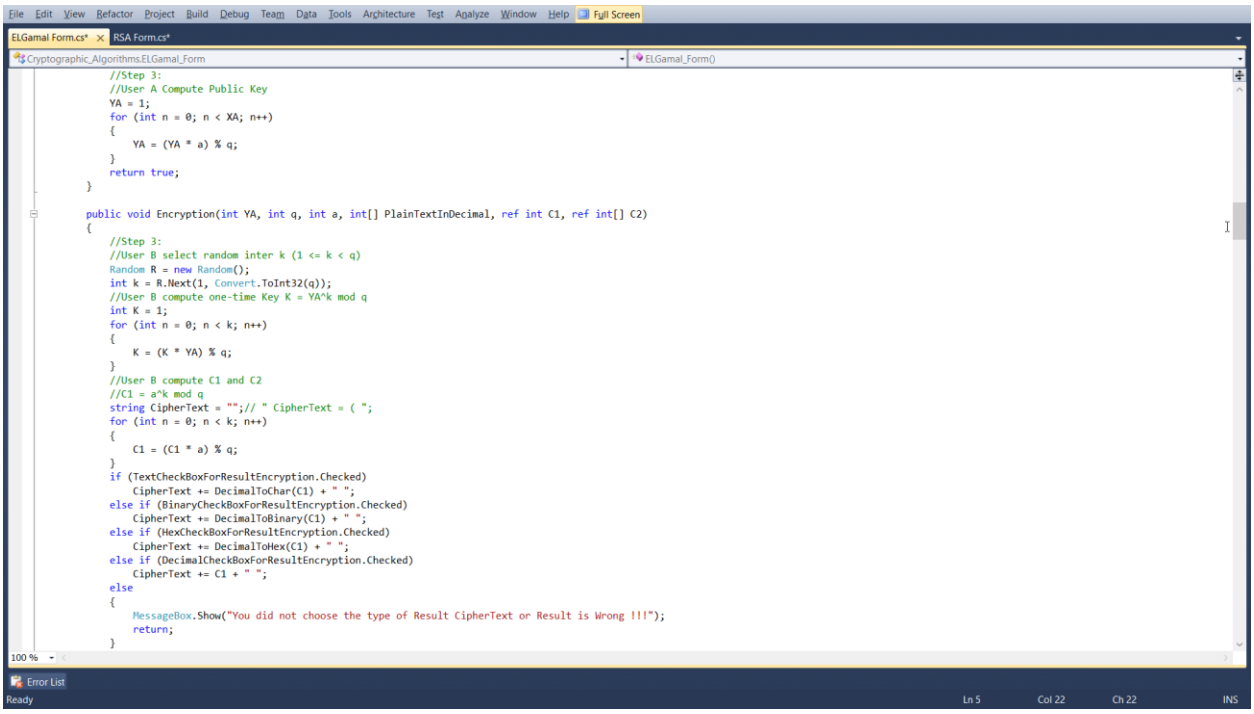
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

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

        void ELGamal_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
```



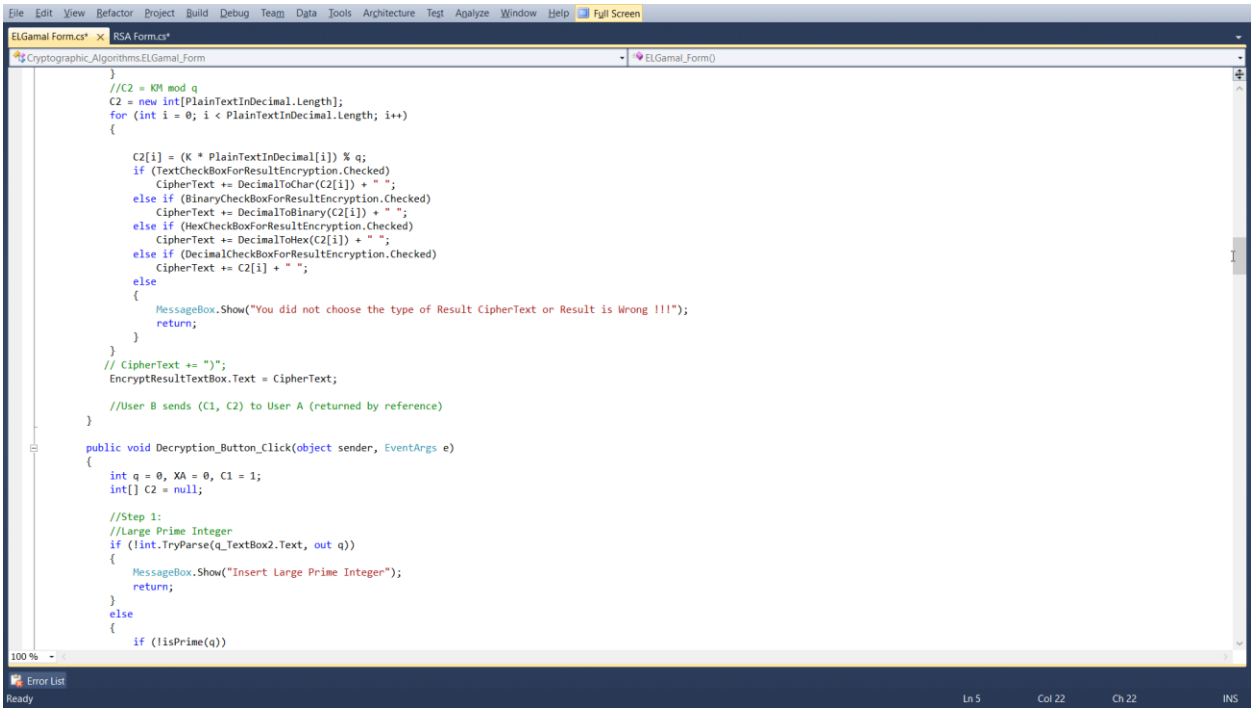
```
                }//Binary to Decimal
                else if (BinaryCheckBoxForTextEncryption.Checked)
                {
                    string[] SplitBinaryText = PlainTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                    PlainTextInDecimal = BinaryToDecimal(SplitBinaryText);
                }//Hex to Decimal
                else if (HexCheckBoxForTextEncryption.Checked)
                {
                    string[] SplitHexText = PlainTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
                    PlainTextInDecimal = HexToDecimal(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))
```



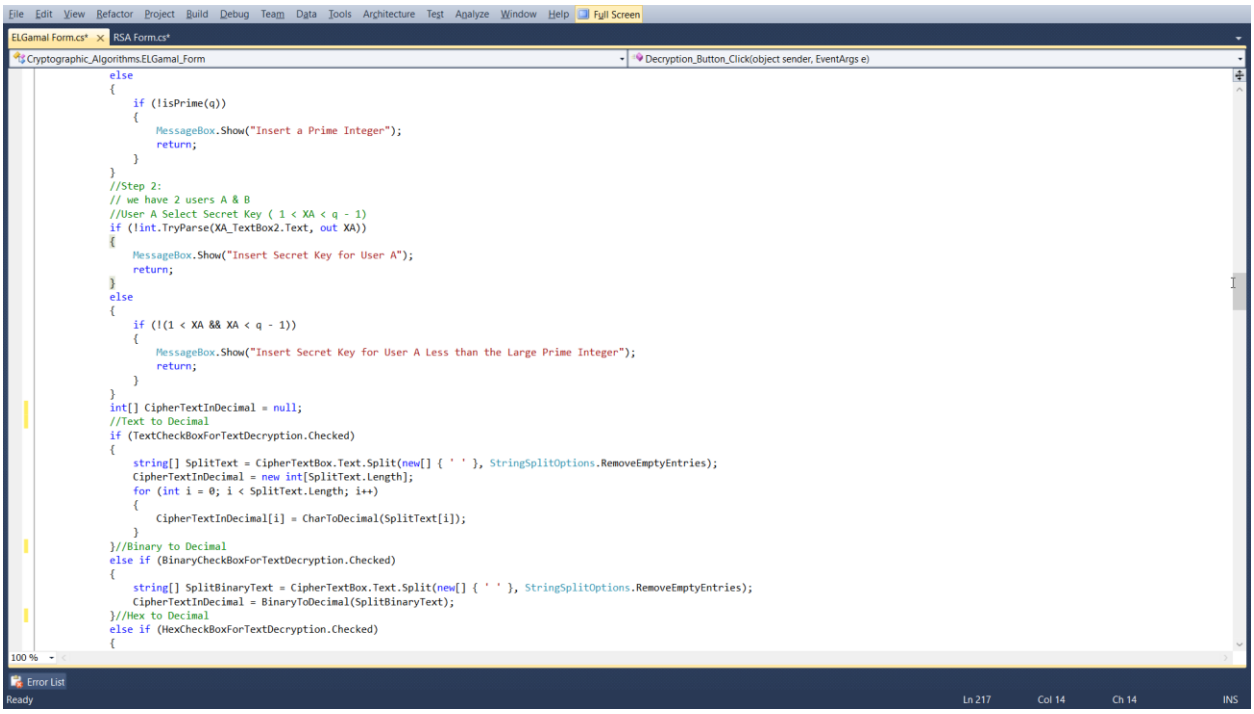
```
            {
                if (!isPrime(q))
                {
                    MessageBox.Show(q + " is not a Prime Integer");
                    return false;
                }
            }
            //being a primitive root mod q
            //for n=1 to q-1
            //  a^n mod q = [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
```



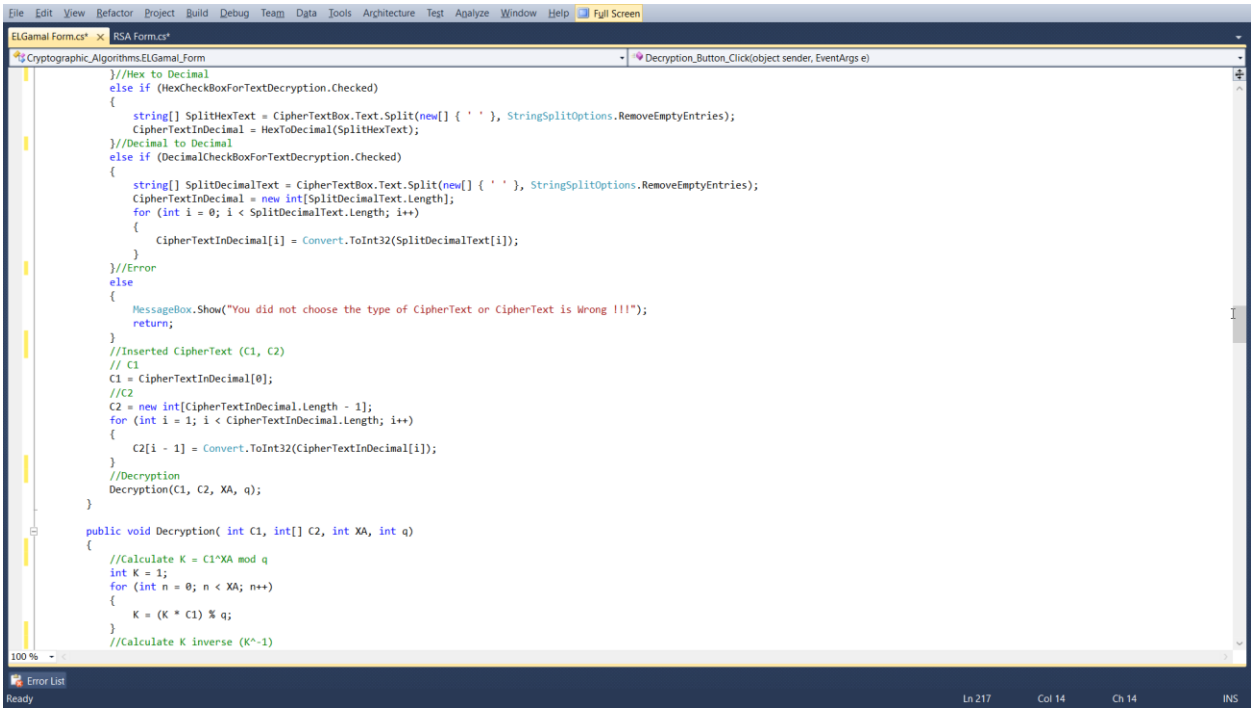
```
            //Step 3:
            //User A Compute Public Key
            YA = 1;
            for (int n = 0; n < XA; n++)
            {
                YA = (YA * a) % q;
            }
            return true;
        }

        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 + " ";
            else
            {
                MessageBox.Show("You did not choose the type of Result CipherText or Result is Wrong !!!");
                return;
            }
```



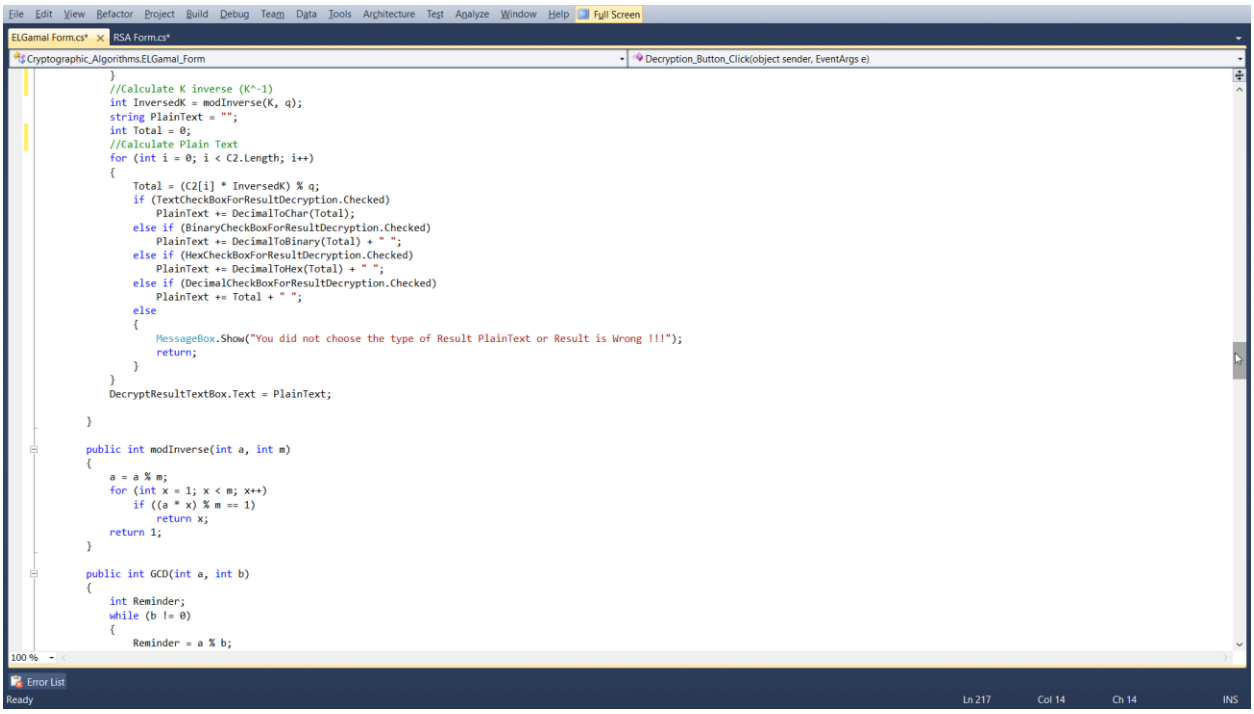
```
            }
            //C2 = KM mod q
            C2 = new int[PlainTextInDecimal.Length];
            for (int i = 0; i < PlainTextInDecimal.Length; i++)
            {

                C2[i] = (K * PlainTextInDecimal[i]) % q;
                if (TextCheckBoxForResultEncryption.Checked)
                    CipherText += DecimalToChar(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))
```



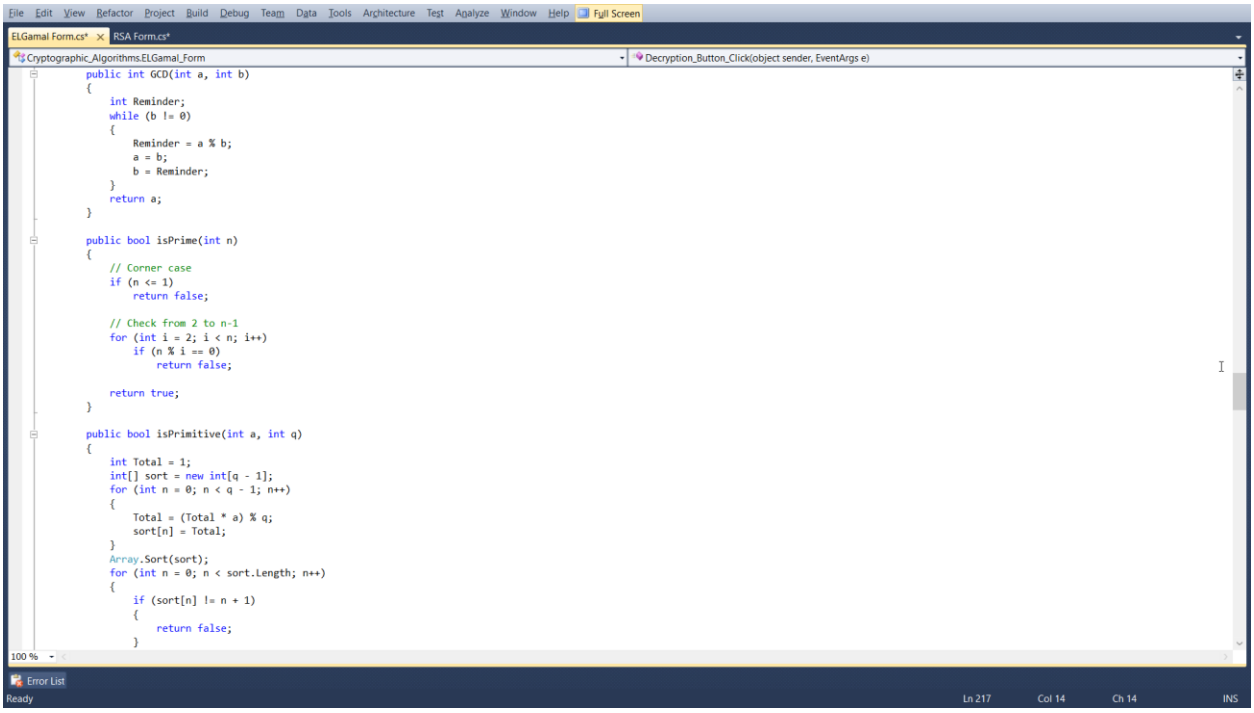
```
            else
            {
                if (!isPrime(q))
                {
                    MessageBox.Show("Insert a Prime Integer");
                    return;
                }
            }
            //Step 2:
            // we have 2 users A & B
            //User A Select Secret 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)
            {
```

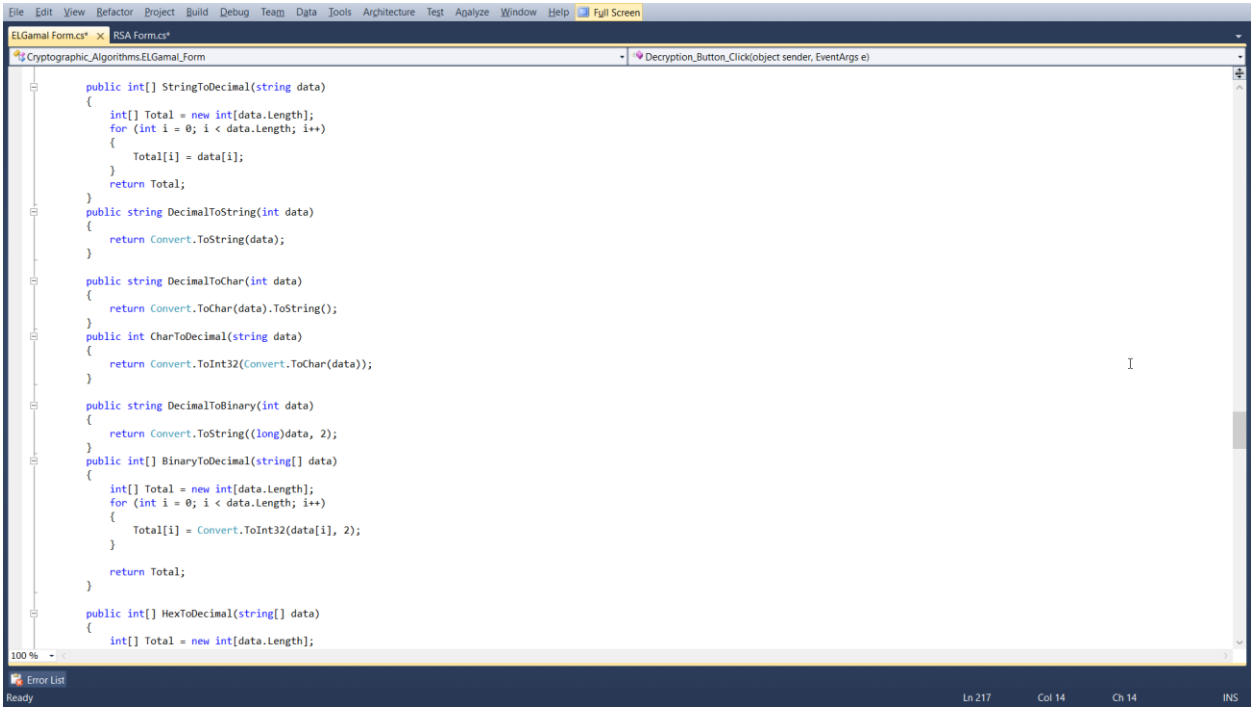


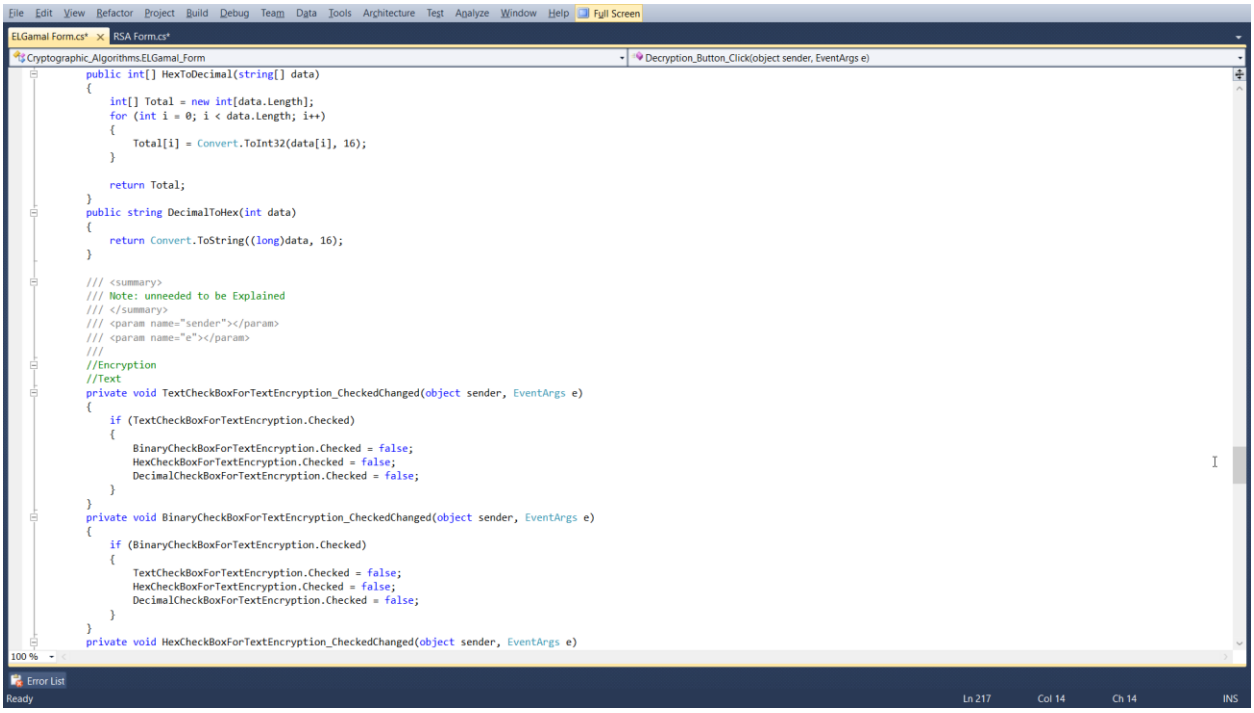
```
        }//Hex to Decimal
        else if (HexCheckBoxForTextDecryption.Checked)
        {
            string[] SplitHexText = CipherTextBox.Text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
            CipherTextInDecimal = HexToDecimal(SplitHexText);
        }//Decimal to Decimal
        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)
```

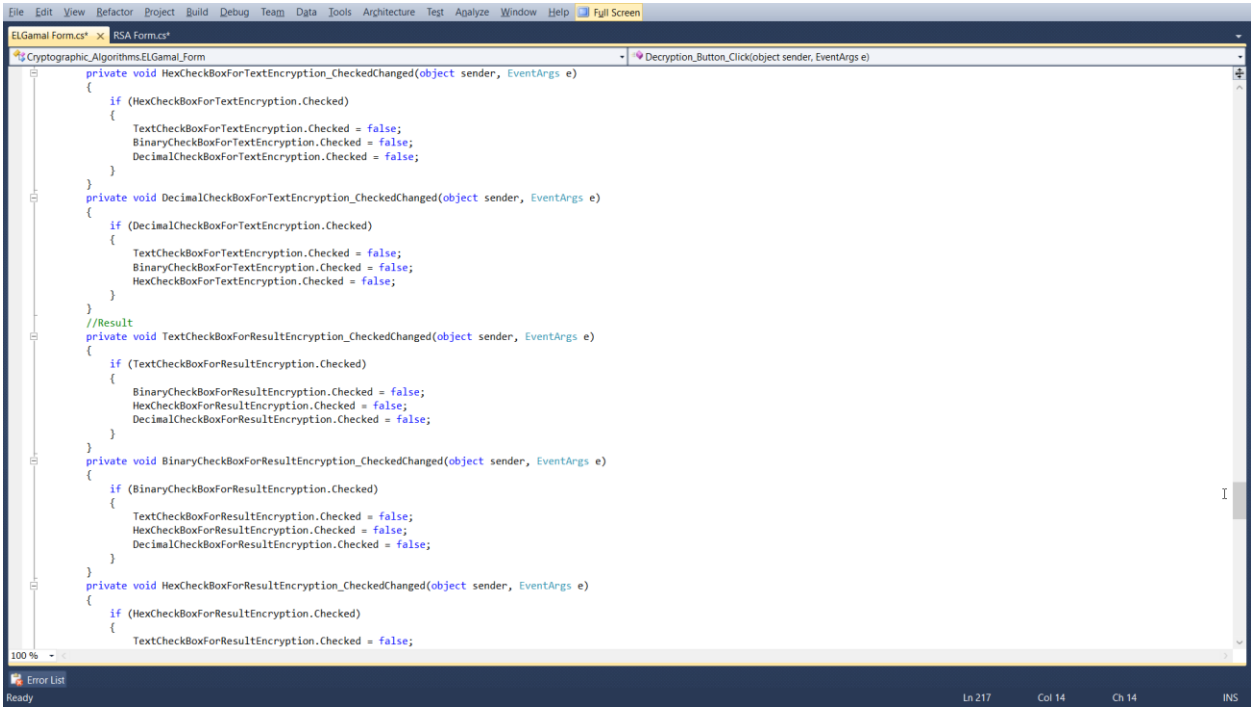


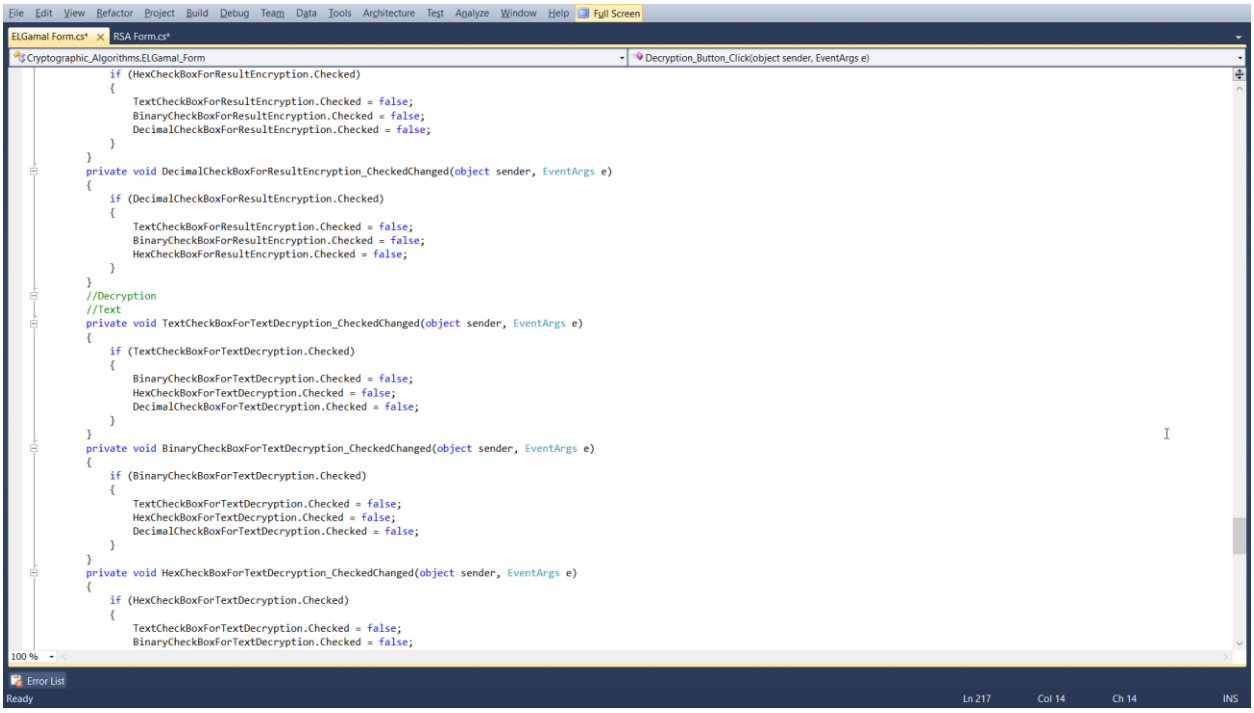
```
        }
        //Calculate K inverse (K^-1)
        int InversedK = modInverse(K, q);
        string PlainText = "";
        int Total = 0;
        //Calculate Plain Text
        for (int i = 0; i < C2.Length; i++)
        {
            Total = (C2[i] * InversedK) % q;
            if (TextCheckBoxForResultDecryption.Checked)
                PlainText += DecimalToChar(Total);
            else if (BinaryCheckBoxForResultDecryption.Checked)
                PlainText += DecimalToBinary(Total) + " ";
            else if (HexCheckBoxForResultDecryption.Checked)
                PlainText += DecimalToHex(Total) + " ";
            else if (DecimalCheckBoxForResultDecryption.Checked)
                PlainText += Total + " ";
            else
            {
                MessageBox.Show("You did not choose the type of Result PlainText or Result is Wrong !!!");
                return;
            }
        }
        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;
```

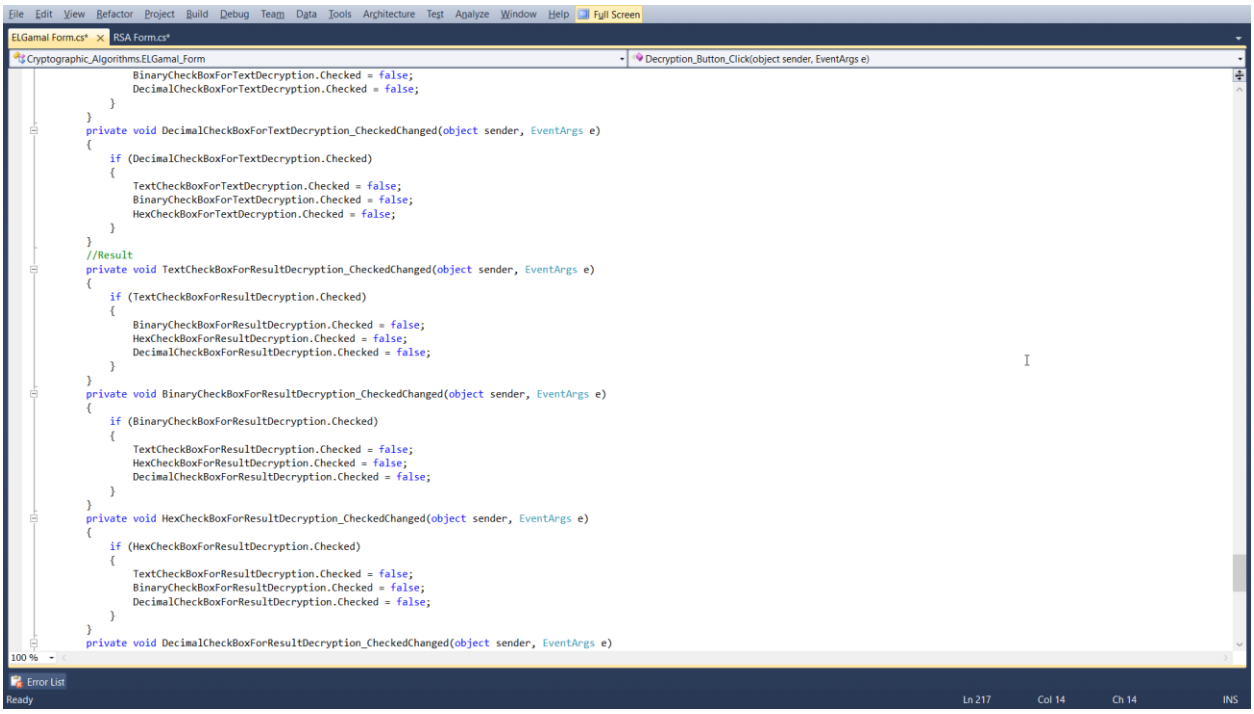

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGamal Form.cs* RSA Form.cs*
Cryptographic_Algorithms.ELGamal_Form
Decryption_Button_Click(object sender, EventArgs e)
public int GCD(int a, int b)
{
int Reminder;
while (b != 0)
{
Reminder = 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

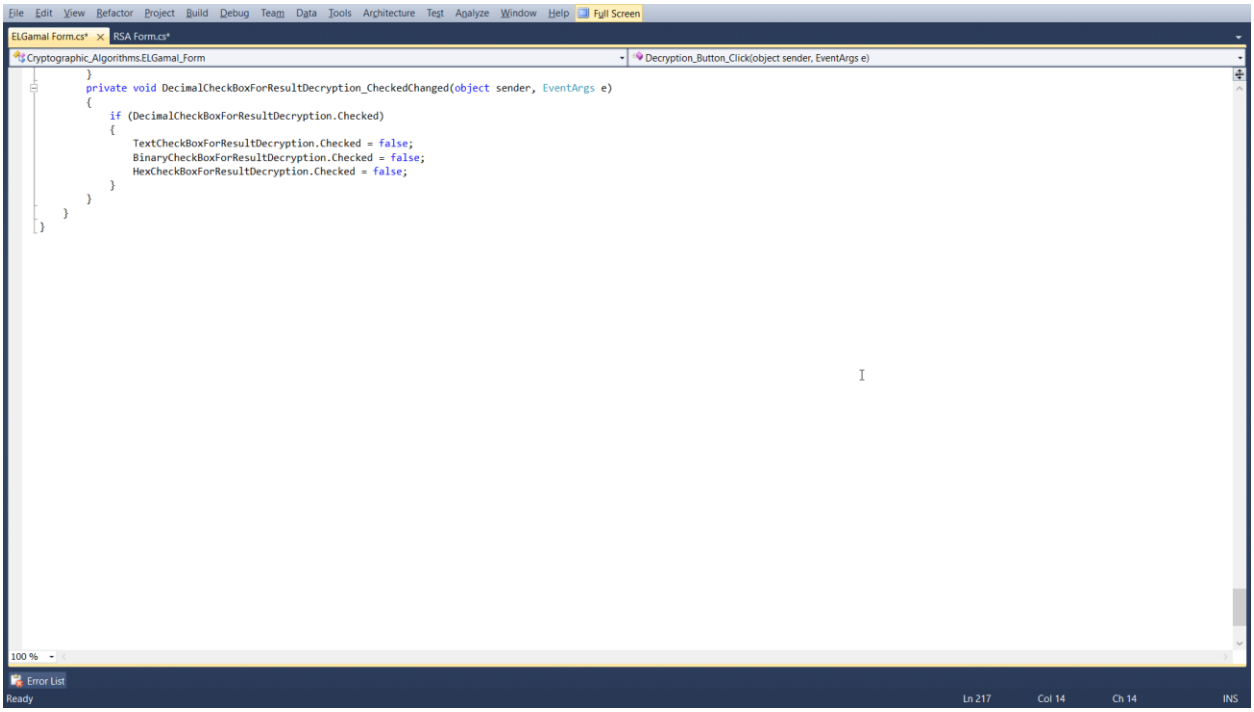

File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help Full Screen
ELGamal Form.cs* RSA Form.cs*
Cryptographic_Algorithms.ELGamal_Form
Decryption_Button_Click(object sender, EventArgs e)
public int[] StringToDecimal(string data)
{
int[] Total = new int[data.Length];
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
ELGamal Form.cs* RSA Form.cs*
Cryptographic_Algorithms.ELGamal_Form
Decryption_Button_Click(object sender, EventArgs e)
public int[] 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;
}
public 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
ELGamal Form.cs* RSA Form.cs*
Cryptographic_Algorithms.ELGamal_Form
Decryption_Button_Click(object sender, EventArgs e)
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
ELGamal Form.cs* RSA Form.cs*
Cryptographic_Algorithms.ELGamal_Form
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 %
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
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 %
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
Decryption_Button_Click(object sender, EventArgs e)
}
private void DecimalCheckBoxForResultDecryption_CheckedChanged(object sender, EventArgs e)
{
if (DecimalCheckBoxForResultDecryption.Checked)
{
TextCheckBoxForResultDecryption.Checked = false;
BinaryCheckBoxForResultDecryption.Checked = false;
HexCheckBoxForResultDecryption.Checked = false;
}
}
}
}
100 %
Error List
Ready
Ln 217
Col 14
Ch 14
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.

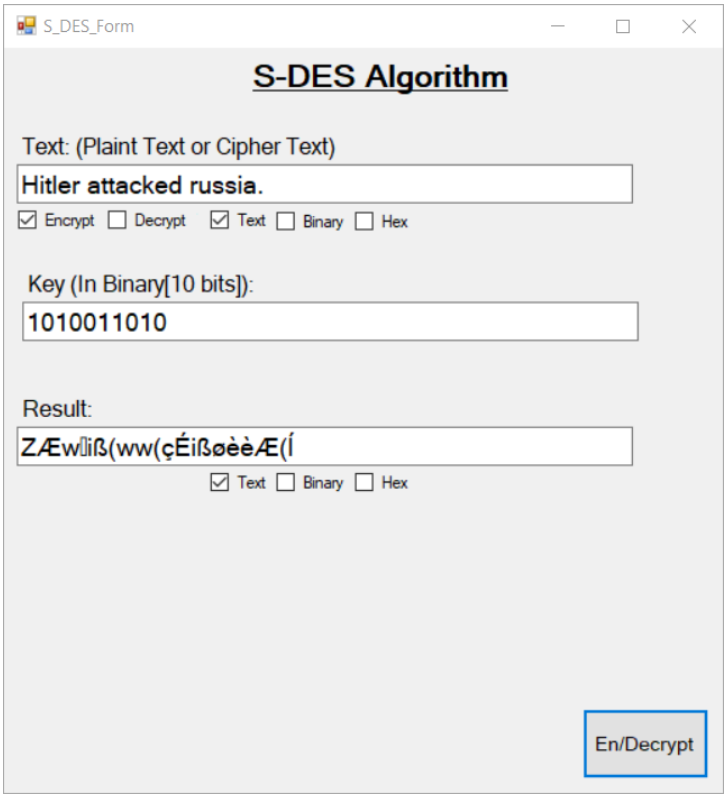


Figure 11: Encrypting Result

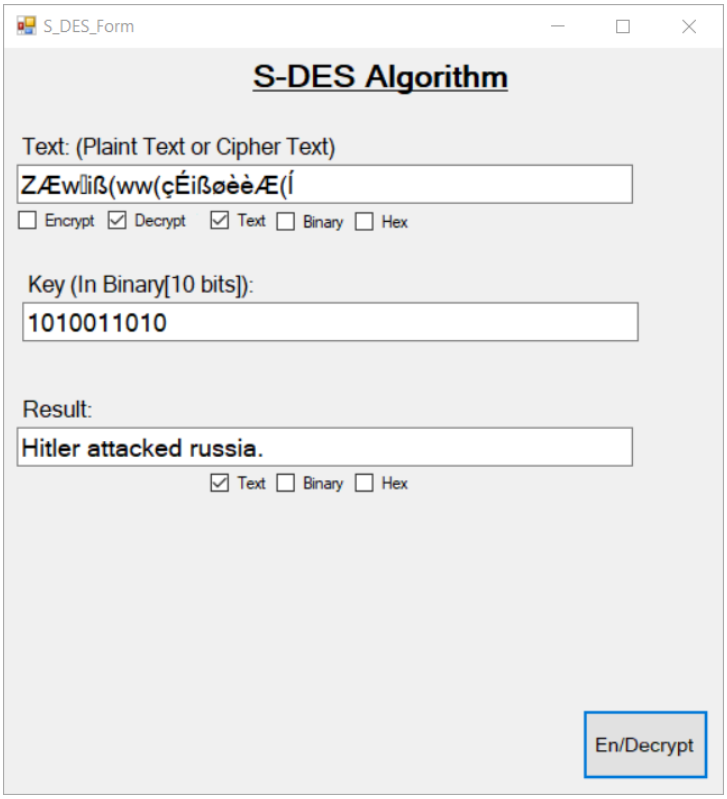


Figure 12: Decrypting Result

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.

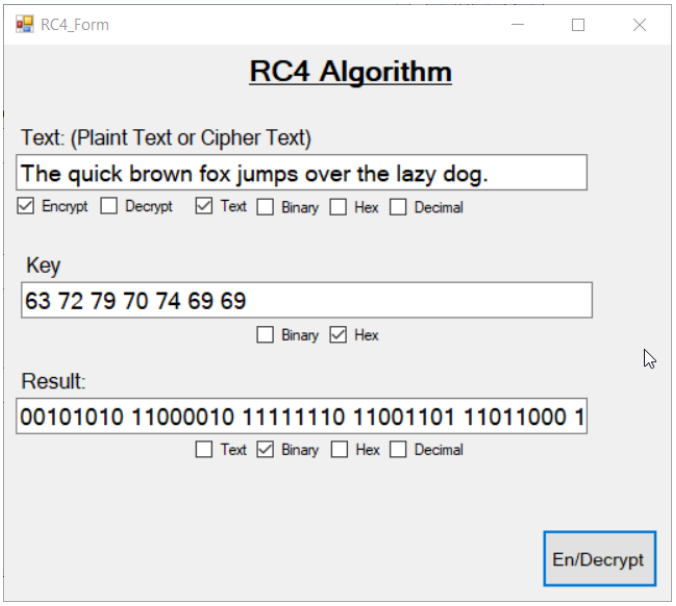


Figure 13: Encrypting Result

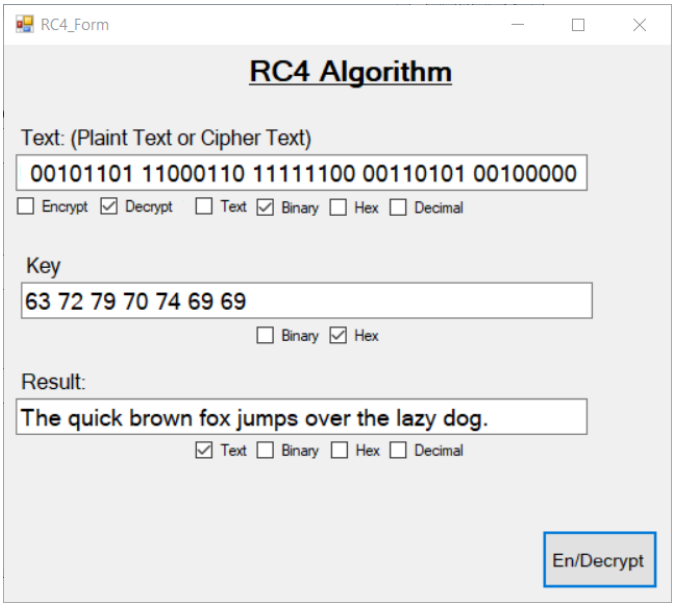


Figure 14: Decrypting Result

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.

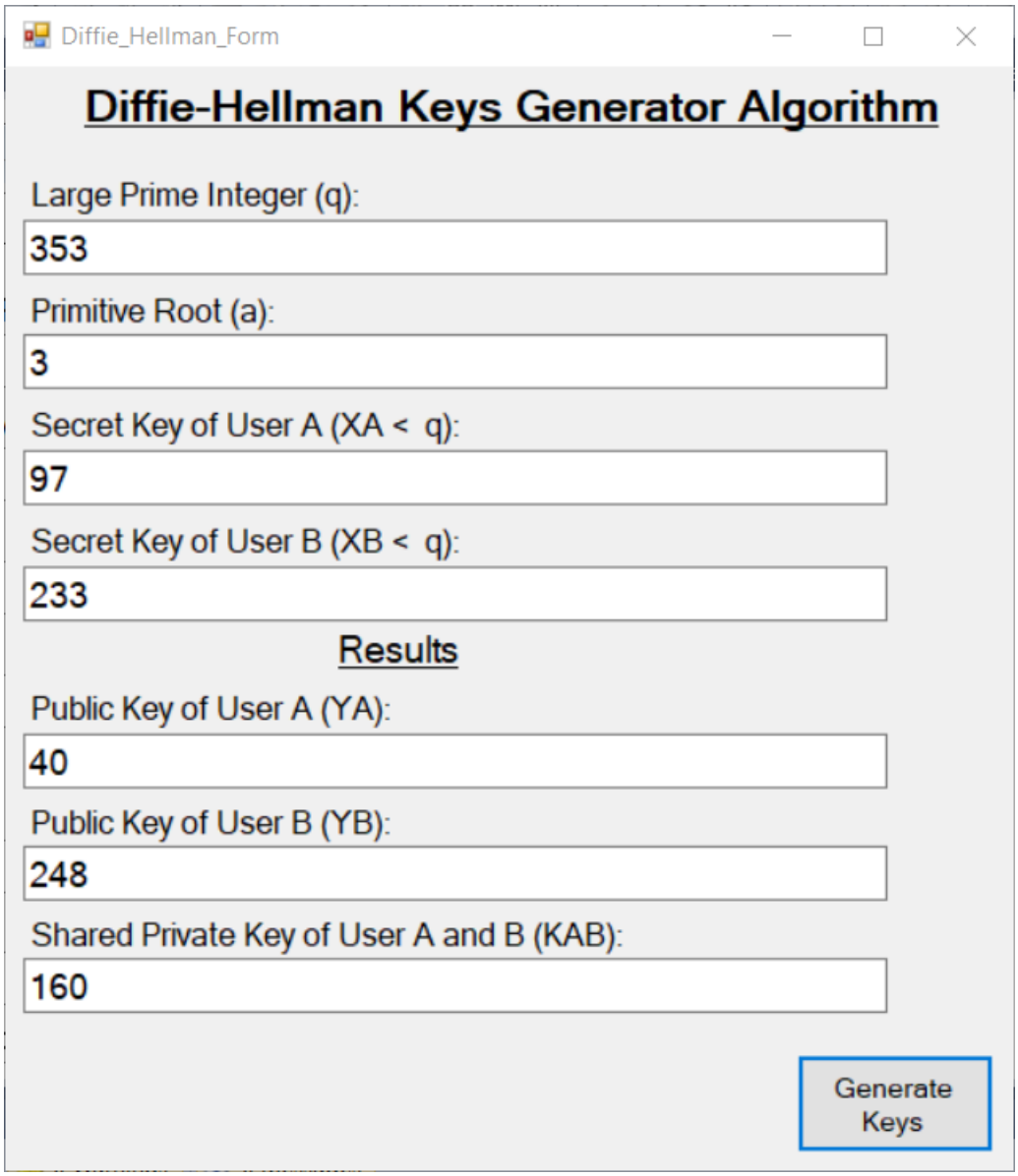


Figure 15: Results of Public Keys and Private Keys

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.

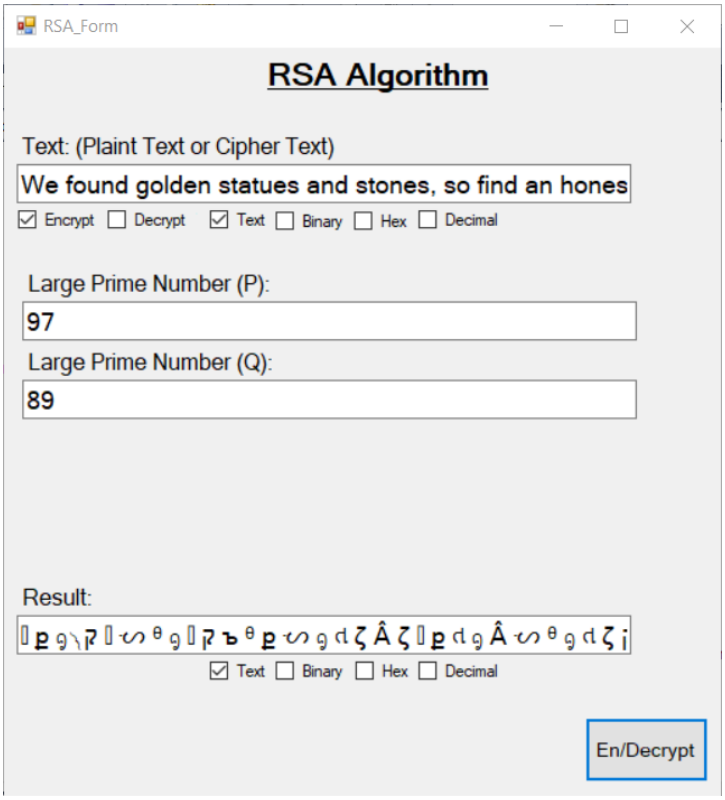


Figure 16: Encrypting Result

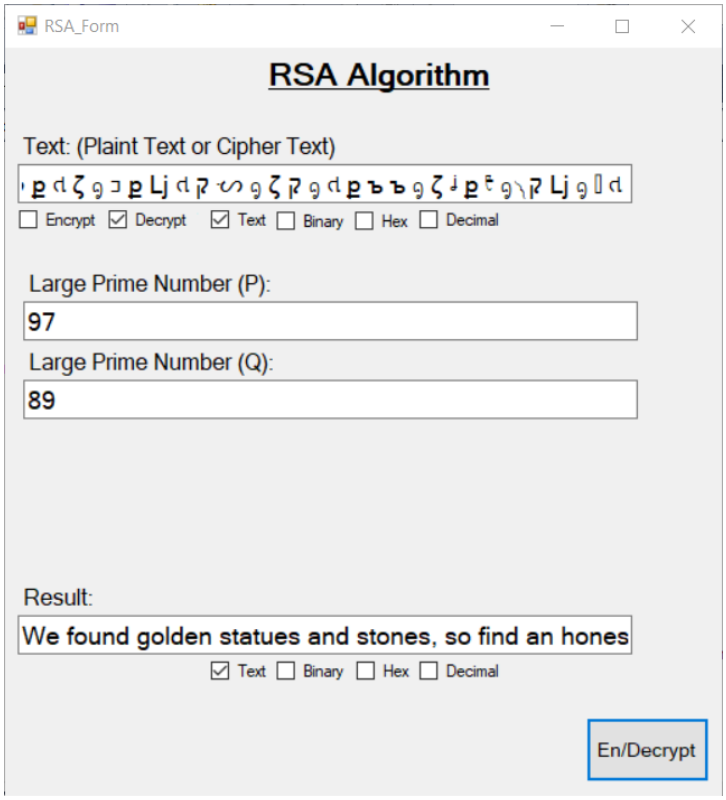


Figure 17: Decrypting Result

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.

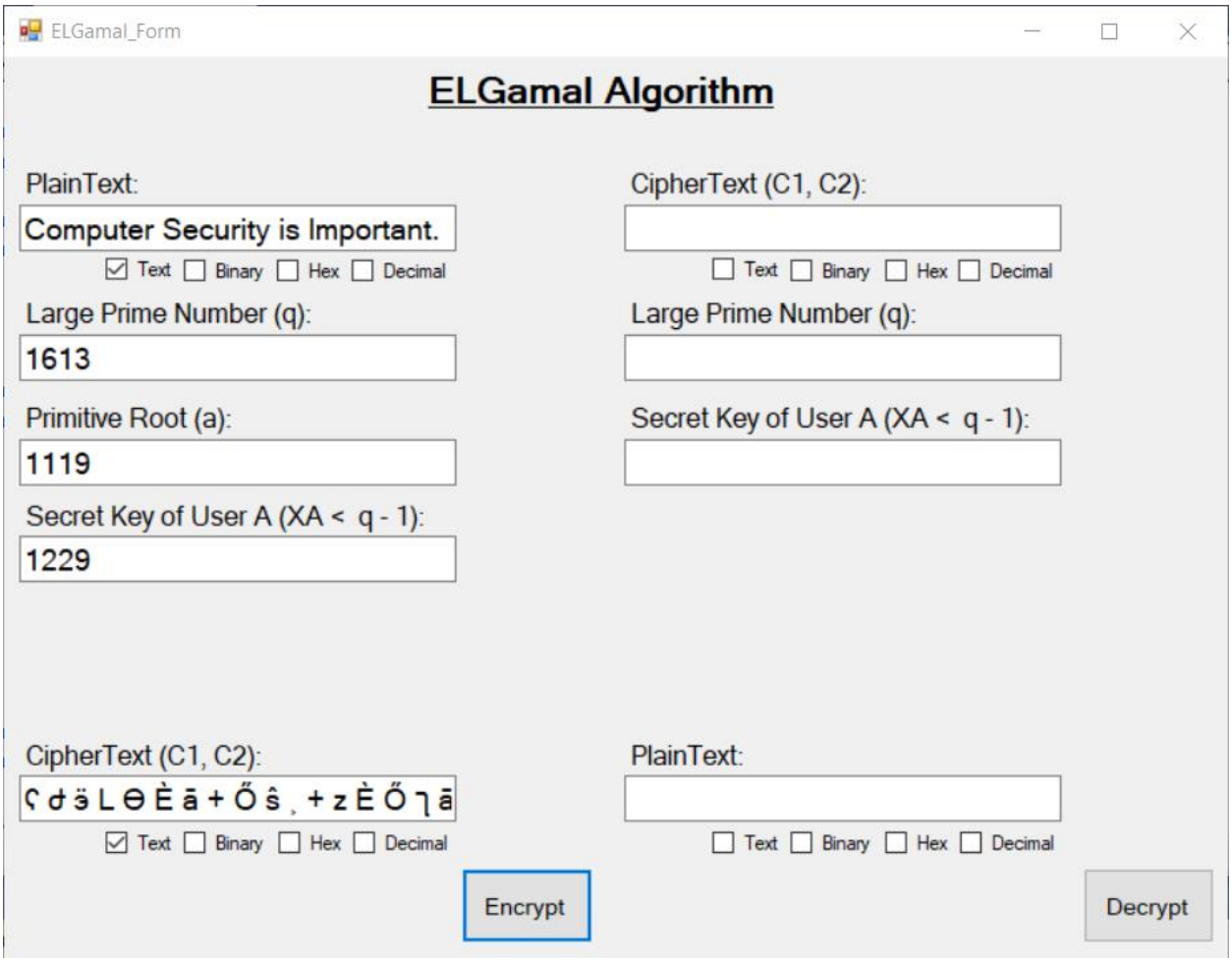


Figure 18: Encrypting Result (CipherText)

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.

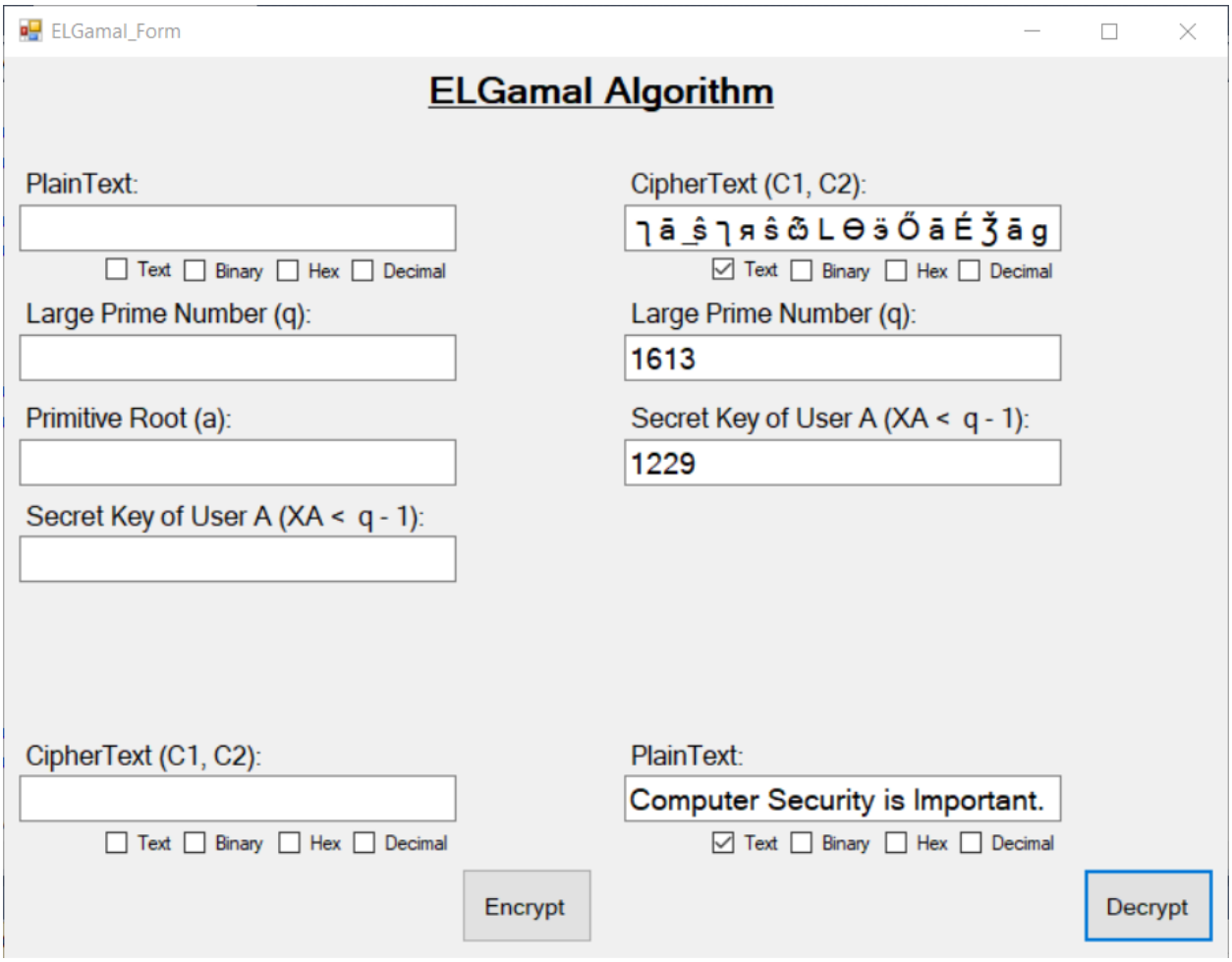


Figure 19: Decrypting Result (PlainText)