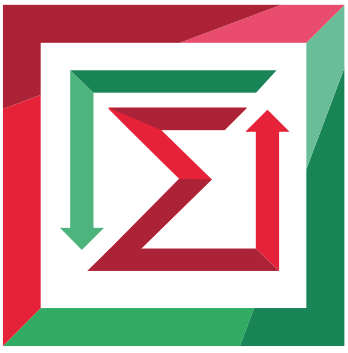
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

НГТУ
НЭТИ

Кафедра теоретической и прикладной информатики

Лабораторная работа № 3
по дисциплине «Программные Средства Защиты Информации»
Стандарт симметричного шифрования AES



ФАКУЛЬТЕТ:	ПМИ
ГРУППА:	ПМИМ-01
СТУДЕНТЫ:	Ершов П. К. Малышкина Е. Д. Слободчикова А. Э.
БРИГАДА:	2
ПРЕПОДАВАТЕЛЬ:	Авдеенко Т. В.

Новосибирск
2021

1. Цель работы

Ознакомиться с блочными составными шифрами, освоить криптографические преобразования подстановки и перестановки. Изучить и реализовать шифрование информации при помощи сети Фейстеля.

2. Задание

1. Реализовать приложение для шифрования, позволяющее выполнять следующие действия:
   1.1. Шифровать данные при помощи реализованного алгоритма AES:
      1) шифруемый текст должен храниться в файле;
      2) ключ шифрования должен задаваться случайным образом;
      3) зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
      4) в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.

2. Реализовать приложение для дешифрования, позволяющее выполнять следующие действия:

   2.1. Дешифровать данные при помощи реализованного алгоритма AES:
      5) шифруемый текст должен храниться в файле;
      6) ключ шифрования должен задаваться случайным образом;
      7) зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
      8) в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.

3. Исследования

3.1. Демонстрация работы на случайных данных

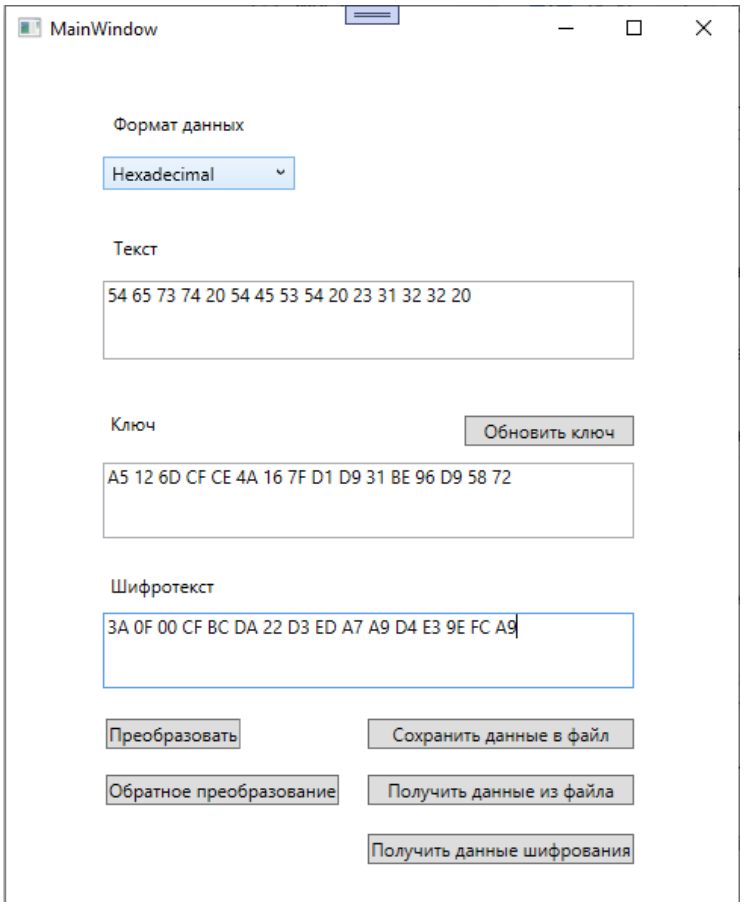


Рисунок 1. Пример работы разработанной программы

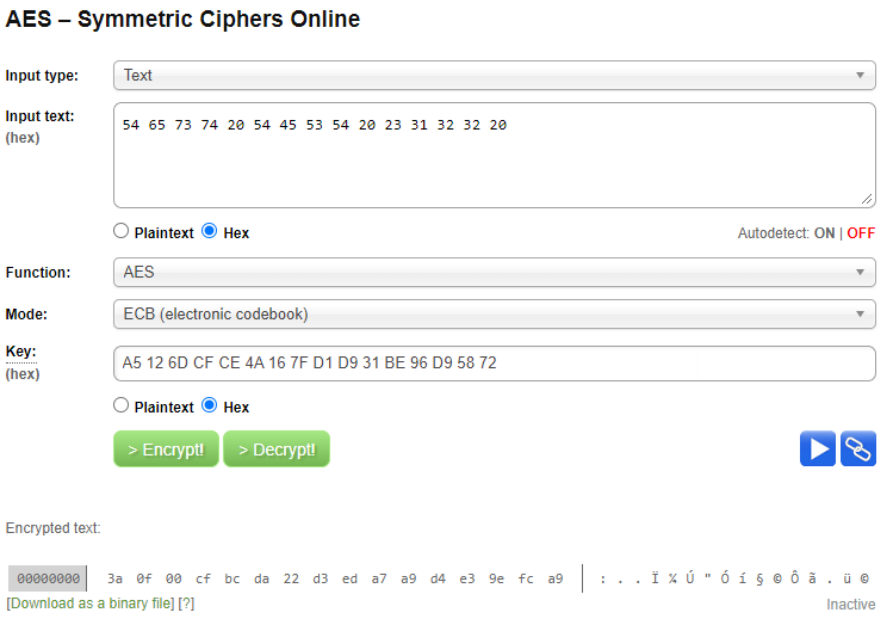


Рисунок 2. Пример работы онлайн шифровщика AES

3.2. Демонстрация работы на данных из варианта

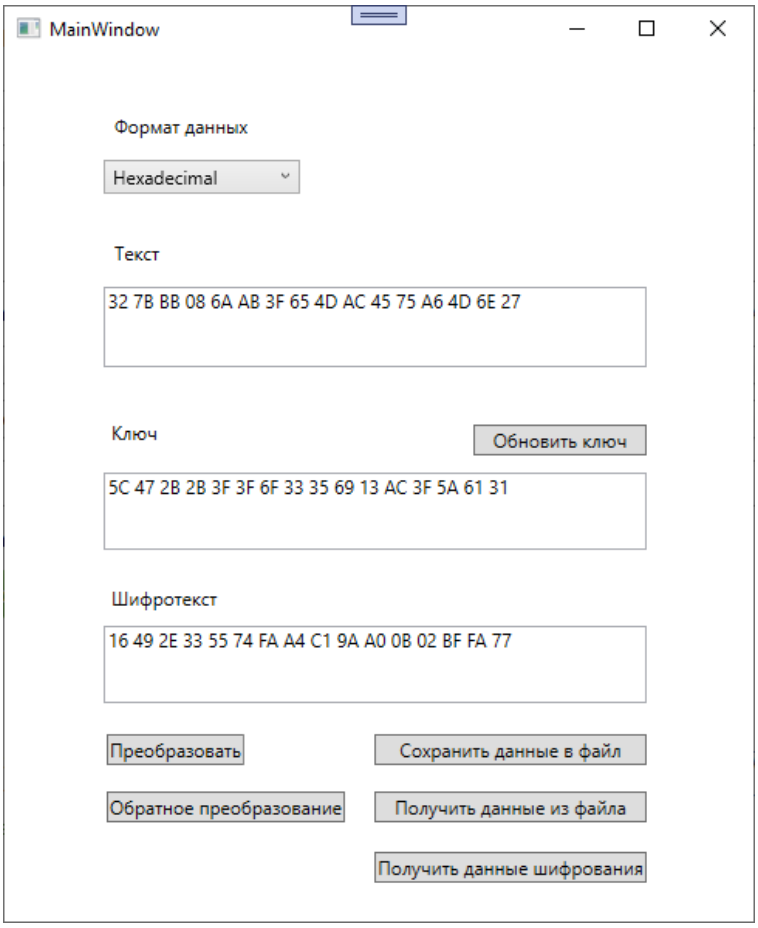


Рисунок 3. Пример шифрования из предложенного варианта

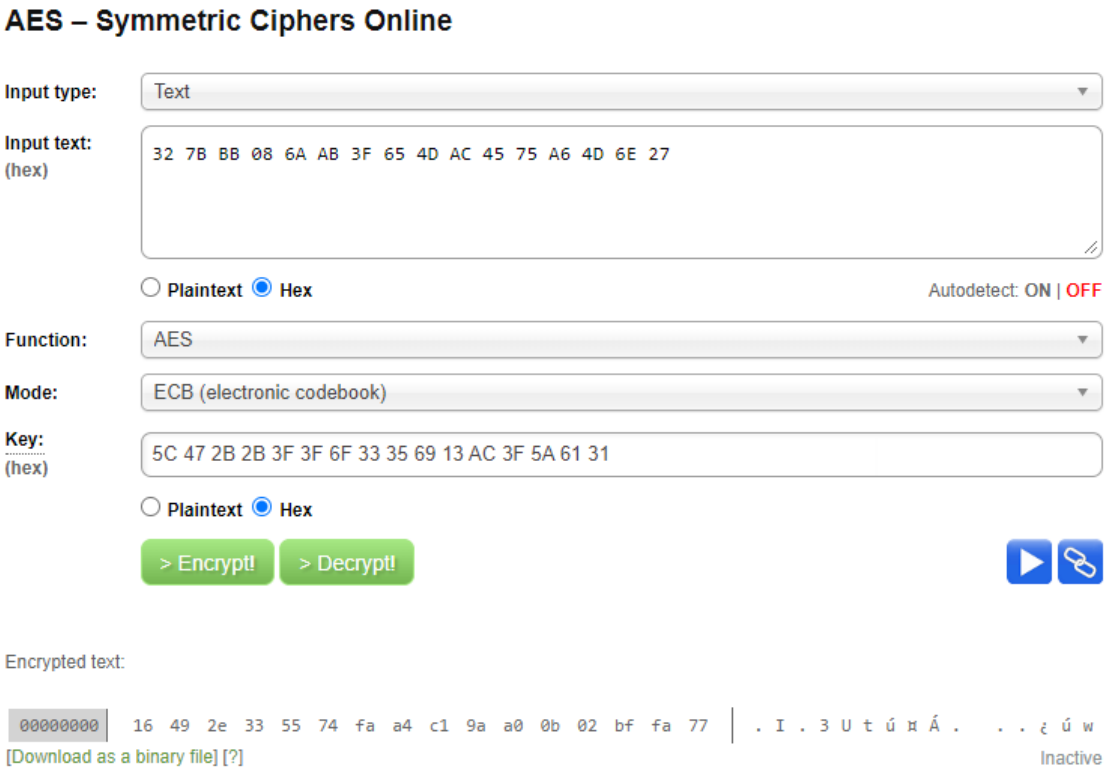


Рисунок 4. Пример шифрования онлайн приложения

4. Выводы

Разработанная программа выполняется корректно на всех наборах данных.

5. Код программы

AESClass.cs

```
public class AESClass
    {
        private static int Nb, Nk, Nr;
        private static byte[,] w;

        private static int KeyLenght = 16;

        private static int[] sbox = { 0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F,
            0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76, 0xCA, 0x82,
            0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C,
            0xA4, 0x72, 0xC0, 0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
            0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15, 0x04, 0xC7, 0x23,
            0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27,
            0xB2, 0x75, 0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52,
            0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84, 0x53, 0xD1, 0x00, 0xED,
            0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58,
            0xCF, 0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9,
            0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8, 0x51, 0xA3, 0x40, 0x8F, 0x92,
            0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
            0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E,
            0x3D, 0x64, 0x5D, 0x19, 0x73, 0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A,
            0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB, 0xE0,
            0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62,
            0x91, 0x95, 0xE4, 0x79, 0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E,
            0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08, 0xBA, 0x78,
            0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
            0xBD, 0x8B, 0x8A, 0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E,
            0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E, 0xE1, 0xF8, 0x98,
            0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55,
            0x28, 0xDF, 0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41,
            0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16 };


        private static int[] inv_sbox = { 0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5,
            0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB, 0x7C, 0xE3,
```

```
            0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4,
            0xDE, 0xE9, 0xCB, 0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D,
            0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E, 0x08, 0x2E, 0xA1,
            0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B,
            0xD1, 0x25, 0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4,
            0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92, 0x6C, 0x70, 0x48, 0x50,
            0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D,
            0x84, 0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4,
            0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06, 0xD0, 0x2C, 0x1E, 0x8F, 0xCA,
            0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
            0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF,
            0xCE, 0xF0, 0xB4, 0xE6, 0x73, 0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD,
            0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E, 0x47,
            0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E,
            0xAA, 0x18, 0xBE, 0x1B, 0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79,
            0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4, 0x1F, 0xDD,
            0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27,
            0x80, 0xEC, 0x5F, 0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D,
            0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF, 0xA0, 0xE0, 0x3B,
            0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53,
            0x99, 0x61, 0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1,
            0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D };


        private static int[] Rcon = { 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,
0x6c, 0xd8, 0xab, 0x4d, 0x9a,
            0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
            0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
            0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
            0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
            0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
            0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
            0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
            0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
            0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
            0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
            0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
            0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
            0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
            0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
```

```
0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb };

public static byte[] RandKey()
{
    Random rnd = new Random((int)DateTime.Now.Ticks);
    var bt = new Byte[KeyLenght];
    rnd.NextBytes(bt);
    return bt;
}

private static byte[] xor_func(byte[] a, byte[] b)
{
    byte[] outp = new byte[a.Length];
    for (int i = 0; i < a.Length; i++)
    {
        outp[i] = (byte)(a[i] ^ b[i]);
    }
    return outp;

}
private static byte[,] generateSubkeys(byte[] key)
{
    byte[,] tmp = new byte[Nb * (Nr + 1), 4];

    int i = 0;
    while (i < Nk)
    {

        tmp[i, 0] = key[i * 4];
        tmp[i, 1] = key[i * 4 + 1];
        tmp[i, 2] = key[i * 4 + 2];
        tmp[i, 3] = key[i * 4 + 3];
        i++;
    }
    i = Nk;
    while (i < Nb * (Nr + 1))
    {
        byte[] temp = new byte[4];
        for (int k = 0; k < 4; k++)
```

```
                temp[k] = tmp[i - 1, k];
            if (i % Nk == 0)
            {
                temp = SubWord(rotateWord(temp));
                temp[0] = (byte)(temp[0] ^ (Rcon[i / Nk] & 0xff));
            }
            else if (Nk > 6 && i % Nk == 4)
            {
                temp = SubWord(temp);
            }
            byte[] tmp2 = new byte[4] { tmp[i - Nk, 0], tmp[i - Nk, 1], tmp[i - Nk, 2], tmp[i - Nk, 3]
};
            byte[] result = new byte[4];
            result = xor_func(tmp2, temp);
            tmp[i, 0] = result[0];
            tmp[i, 1] = result[1];
            tmp[i, 2] = result[2];
            tmp[i, 3] = result[3];
            i++;
        }

        return tmp;
    }

    private static byte[] SubWord(byte[] inp)
    {
        byte[] tmp = new byte[inp.Length];

        for (int i = 0; i < tmp.Length; i++)
            tmp[i] = (byte)(sbox[inp[i] & 0x000000ff] & 0xff);

        return tmp;
    }

    private static byte[] rotateWord(byte[] input)
    {
        byte[] tmp = new byte[input.Length];
        tmp[0] = input[1];
        tmp[1] = input[2];
```

```
    tmp[2] = input[3];
    tmp[3] = input[0];

    return tmp;
}

private static byte[,] AddRoundKey(byte[,] state, byte[,] w, int round)
{
    byte[,] tmp = new byte[4, 4];
    for (int c = 0; c < Nb; c++)
    {
        for (int l = 0; l < 4; l++)
            tmp[l, c] = (byte)(state[l, c] ^ w[round * Nb + c, l]);
    }

    return tmp;
}

private static byte[,] SubBytes(byte[,] state)
{
    byte[,] tmp = new byte[4, 4];
    for (int row = 0; row < 4; row++)
        for (int col = 0; col < Nb; col++)
            tmp[row, col] = (byte)(sbox[(state[row, col] & 0x000000ff)] & 0xff);

    return tmp;
}

private static byte[,] InvSubBytes(byte[,] state)
{
    for (int row = 0; row < 4; row++)
        for (int col = 0; col < Nb; col++)
            state[row, col] = (byte)(inv_sbox[(state[row, col] & 0x000000ff)] & 0xff);

    return state;
}

private static byte[,] ShiftRows(byte[,] state)
{
```

```
    byte[] t = new byte[4];
    for (int r = 1; r < 4; r++)
    {
        for (int c = 0; c < Nb; c++)
            t[c] = state[r, (c + r) % Nb];
        for (int c = 0; c < Nb; c++)
            state[r, c] = t[c];
    }

    return state;
}

private static byte[,] InvShiftRows(byte[,] state)
{
    byte[] t = new byte[4];
    for (int r = 1; r < 4; r++)
    {
        for (int c = 0; c < Nb; c++)
            t[(c + r) % Nb] = state[r, c];
        for (int c = 0; c < Nb; c++)
            state[r, c] = t[c];
    }
    return state;
}

private static byte[,] MixColumns(byte[,] s)
{
    int[] sp = new int[4];
    byte b02 = (byte)0x02, b03 = (byte)0x03;
    for (int c = 0; c < 4; c++)
    {
        sp[0] = FFMul(b02, s[0, c]) ^ FFMul(b03, s[1, c]) ^ s[2, c] ^ s[3, c];
        sp[1] = s[0, c] ^ FFMul(b02, s[1, c]) ^ FFMul(b03, s[2, c]) ^ s[3, c];
        sp[2] = s[0, c] ^ s[1, c] ^ FFMul(b02, s[2, c]) ^ FFMul(b03, s[3, c]);
        sp[3] = FFMul(b03, s[0, c]) ^ s[1, c] ^ s[2, c] ^ FFMul(b02, s[3, c]);
        for (int i = 0; i < 4; i++)
            s[i, c] = (byte)(sp[i]);
    }
```

```
            return s;
        }

        private static byte[,] InvMixColumns(byte[,] s)
        {
            int[] sp = new int[4];
            byte b02 = (byte)0x0e, b03 = (byte)0x0b, b04 = (byte)0x0d, b05 = (byte)0x09;
            for (int c = 0; c < 4; c++)
            {
                sp[0] = FFMul(b02, s[0, c]) ^ FFMul(b03, s[1, c]) ^ FFMul(b04, s[2, c]) ^ FFMul(b05, s[3,
c]);
                sp[1] = FFMul(b05, s[0, c]) ^ FFMul(b02, s[1, c]) ^ FFMul(b03, s[2, c]) ^ FFMul(b04, s[3,
c]);
                sp[2] = FFMul(b04, s[0, c]) ^ FFMul(b05, s[1, c]) ^ FFMul(b02, s[2, c]) ^ FFMul(b03, s[3,
c]);
                sp[3] = FFMul(b03, s[0, c]) ^ FFMul(b04, s[1, c]) ^ FFMul(b05, s[2, c]) ^ FFMul(b02, s[3,
c]);
                for (int i = 0; i < 4; i++)
                    s[i, c] = (byte)(sp[i]);
            }

            return s;
        }



        public static byte FFMul(byte a, byte b)
        {
            byte aa = a, bb = b, r = 0, t;
            while (aa != 0)
            {
                if ((aa & 1) != 0)
                    r = (byte)(r ^ bb);
                t = (byte)(bb & 0x80);
                bb = (byte)(bb << 1);
                if (t != 0)
                    bb = (byte)(bb ^ 0x1b);
                aa = (byte)((aa & 0xff) >> 1);
            }
            return r;
        }
```

```
public static byte[] encryptBloc(byte[] inp)
{
    byte[] tmp = new byte[inp.Length];



    byte[,] state = new byte[4, Nb];

    for (int i = 0; i < inp.Length; i++)
        state[i / 4, i % 4] = inp[i % 4 * 4 + i / 4];

    state = AddRoundKey(state, w, 0);
    for (int round = 1; round < Nr; round++)
    {
        state = SubBytes(state);
        state = ShiftRows(state);
        state = MixColumns(state);
        state = AddRoundKey(state, w, round);
    }
    state = SubBytes(state);
    state = ShiftRows(state);
    state = AddRoundKey(state, w, Nr);

    for (int i = 0; i < tmp.Length; i++)
        tmp[i % 4 * 4 + i / 4] = state[i / 4, i % 4];

    return tmp;
}

public static byte[] decryptBloc(byte[] inp)
{
    byte[] tmp = new byte[inp.Length];

    byte[,] state = new byte[4, Nb];

    for (int i = 0; i < inp.Length; i++)
        state[i / 4, i % 4] = inp[i % 4 * 4 + i / 4];
```

```
            state = AddRoundKey(state, w, Nr);
            for (int round = Nr - 1; round >= 1; round--)
            {
                state = InvSubBytes(state);
                state = InvShiftRows(state);
                state = AddRoundKey(state, w, round);
                state = InvMixColumns(state);

            }
            state = InvSubBytes(state);
            state = InvShiftRows(state);
            state = AddRoundKey(state, w, 0);

            for (int i = 0; i < tmp.Length; i++)
                tmp[i % 4 * 4 + i / 4] = state[i / 4, i % 4];

            return tmp;
        }

        public static (byte[] s1, byte[] s2, byte[] s3, byte[] s4, byte[] s5) OneRound(byte[] inp, byte[]
key)
        {
            Nb = 4;
            Nk = key.Length / 4;
            Nr = Nk + 6;
            byte[,] state = new byte[4, Nb];

            byte[] buff1 = new byte[inp.Length];
            byte[] buff2 = new byte[inp.Length];
            byte[] buff3 = new byte[inp.Length];
            byte[] buff4 = new byte[inp.Length];
            byte[] buff5 = new byte[inp.Length];

            w = generateSubkeys(key);

            for (int i = 0; i < inp.Length; i++)
                state[i / 4, i % 4] = inp[i % 4 * 4 + i / 4];

            state = AddRoundKey(state, w, 0);
```

```
    for (int i = 0; i < inp.Length; i++)
        buff1[i % 4 * 4 + i / 4] = state[i / 4, i % 4];

    state = SubBytes(state);
    for (int i = 0; i < inp.Length; i++)
        buff2[i % 4 * 4 + i / 4] = state[i / 4, i % 4];

    state = ShiftRows(state);
    for (int i = 0; i < inp.Length; i++)
        buff3[i % 4 * 4 + i / 4] = state[i / 4, i % 4];

    state = MixColumns(state);
    for (int i = 0; i < inp.Length; i++)
        buff4[i % 4 * 4 + i / 4] = state[i / 4, i % 4];

    state = AddRoundKey(state, w, 0);

    for (int i = 0; i < inp.Length; i++)
        buff5[i % 4 * 4 + i / 4] = state[i / 4, i % 4];

    return (buff1, buff2, buff3, buff4, buff5);
}

public static byte[] encrypt(byte[] inp, byte[] key)
{
    Nb = 4;
    Nk = key.Length / 4;
    Nr = Nk + 6;

    int lenght = 0;
    byte[] padding = new byte[1];
    int i;
    lenght = 16 - inp.Length % 16;
    padding = new byte[lenght];
    padding[0] = (byte)0x00;

    for (i = 1; i < lenght; i++)
        padding[i] = 0;
```

```
    byte[] tmp = new byte[inp.Length + lenght];
    byte[] bloc = new byte[16];



    w = generateSubkeys(key);


    int count = 0;


    for (i = 0; i < tmp.Length; i++)
    {
        if (i > 0 && i % 16 == 0)
        {
            bloc = encryptBloc(bloc);
            Array.Copy(bloc, 0, tmp, i - 16, bloc.Length);
        }
        if (i < inp.Length)
            bloc[i % 16] = inp[i];
        else
        {
            bloc[i % 16] = padding[count % 16];
            count++;
        }
    }
    if (bloc.Length == 16)
    {
        bloc = encryptBloc(bloc);
        Array.Copy(bloc, 0, tmp, i - 16, bloc.Length);
    }


    return tmp;
}


public static byte[] decrypt(byte[] inp, byte[] key)
{
    int i;
    byte[] tmp = new byte[inp.Length];
    byte[] bloc = new byte[16];


    Nb = 4;
```

```
    Nk = key.Length / 4;
    Nr = Nk + 6;
    w = generateSubkeys(key);


    for (i = 0; i < inp.Length; i++)
    {
        if (i > 0 && i % 16 == 0)
        {
            bloc = decryptBloc(bloc);
            Array.Copy(bloc, 0, tmp, i - 16, bloc.Length);
        }
        if (i < inp.Length)
            bloc[i % 16] = inp[i];
    }
    bloc = decryptBloc(bloc);
    Array.Copy(bloc, 0, tmp, i - 16, bloc.Length);


    tmp = deletePadding(tmp);

    return tmp;
}

private static byte[] deletePadding(byte[] input)
{
    int count = 0;

    int i = input.Length - 1;
    while (input[i] == 0)
    {
        count++;
        i--;
    }

    byte[] tmp = new byte[input.Length - count - 1];
    Array.Copy(input, 0, tmp, 0, tmp.Length);
    return tmp;
}
```

```
        public static (string output, int code, byte[] chipout) Converter(byte[] Text, byte[] Key, string
flag = "Text", string type = "encrypt") //Универсальный преобразователь

        {

            string chiphrtext = "";

            int code = 0;


            byte[] tt = null;

            if (flag == "Text")

            {

                if (type == "encrypt")

                    tt = encrypt(Text, Key);

                else

                    tt = decrypt(Text, Key);

                chiphrtext = ConverteUtility.ConvertByteArrayToString(tt);


            }


            if (flag == "Binary")

            {

                if (type == "encrypt")

                    tt = encrypt(Text, Key);

                else

                    tt = decrypt(Text, Key);

                chiphrtext = ConverteUtility.ConvertByteArraToBinaryStr(tt);


            }

            if (flag == "Hexadecimal")

            {

                if (type == "encrypt")

                    tt = encrypt(Text, Key);

                else

                    tt = decrypt(Text, Key);

                chiphrtext = ConverteUtility.ByteArrayToHexString(tt);

            }

            return (chiphrtext, code, tt);

        }

    }
```

ConverteUtility.cs

```
public static class ConverteUtility
    {
        private const string AllowedCharHex = "0123456789ABCDEF";

        private const string AllowedCharBin = "01";

        private static Encoding enc = Encoding.GetEncoding(1251);



        public static bool CheckIncorrectFormat(string text, string format = "Bin")
        {
            bool flag = false;
            string sample = "";
            string buff = text.Replace(" ", "");
            switch (format)
            {
                case "Bin":
                    {
                        sample = AllowedCharBin;
                        break;
                    }
                case "Hex":
                    {
                        sample = AllowedCharHex;
                        break;
                    }
            }
            foreach (var i in buff)
            {
                flag = false;
                foreach (var j in sample)
                {
                    if (i == j)
                    {
                        flag = true;
                        break;
```

```
            }
        }
        if (!flag)
            break;
    }
    return flag;
}

public static bool CheckIncorrectLength(string text)
{
    var buff = text.Replace(" ", "");
    if (buff.Length % 2 != 0)
        return false;
    else
        return true;
}

public static string GenStartVal(int leng)
{
    var output = "";
    for (int i = 0; i < leng; i++)
        output += "0";
    return output;
}

public static string ByteArrayToHexString(byte[] Bytes)
{
    return BitConverter.ToString(Bytes).Replace("-", " ");
}

public static byte[] HexStringToByteArray(string HexStr)
{
    var Hex = HexStr.Replace(" ", "");
    byte[] Bytes = new byte[Hex.Length / 2];
    int[] HexValue = new int[] { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                 0x08, 0x09, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                                 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

    for (int x = 0, i = 0; i < Hex.Length; i += 2, x += 1)
```

```
    {
        Bytes[x] = (byte)(HexValue[Char.ToUpper(Hex[i + 0]) - '0'] << 4 |
                          HexValue[Char.ToUpper(Hex[i + 1]) - '0']);
    }

    return Bytes;
}
public static byte[] ConvertStringToByteArray(string text)
{
    Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
    return enc.GetBytes(text);
}

public static string ConvertByteArrayToString(byte[] text)
{
    Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
    return enc.GetString(text);
}

public static string ConvertStringToBinaryStr(string text)
{
    Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
    var s = new StringBuilder();
    foreach (bool bb in new BitArray(enc.GetBytes(text)))
        s.Append(bb ? '1' : '0');
    return s.ToString();
}

public static string ConvertByteArraToBinaryStr(byte[] text)
{
    Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
    var s = new StringBuilder();
    foreach (bool bb in new BitArray(text))
        s.Append(bb ? '1' : '0');
    return s.ToString();
}

public static string UniConvert(string text, string inflag = "Text", string outflag = "Text")
{
```

```
string output = "";
byte[] buff = new byte[text.Length];
if (inflag == outflag)
    return text;
switch (inflag)
{
    case "Text":
        {
            buff = ConvertStringToByteArray(text);
            break;
        }
    case "Binary":
        {
            buff = ConvertBinaryStrToByte(text);
            break;
        }
    case "Hexadecimal":
        {
            buff = HexStringToByteArray(text);
            break;
        }
}

switch (outflag)
{
    case "Text":
        {
            output = ConvertByteArrayToString(buff);
            break;
        }
    case "Binary":
        {
            output = ConvertByteArraToBinaryStr(buff);
            break;
        }
    case "Hexadecimal":
        {
            output = ByteArrayToHexString(buff);
            break;
```

```
            }
        }

        return output;
    }

    public static byte[] ConvertBinaryStrToByte(string binary)
    {
        Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
        int numOfBytes = binary.Length / 8;
        byte[] bytes = new byte[numOfBytes];
        var buf = new BitArray(bytes);
        for (int i = 0; i < binary.Length; i++)
        {
            if (binary[i] == '1')
                buf[i] = true;
            else
                buf[i] = false;
        }
        buf.CopyTo(bytes, 0);
        return bytes;
    }



    public static string PadToByte(string binary)
    {
        string output = "";
        if (binary.Length < 8)
        {
            for (int i = 0; i < 8 - binary.Length; i++)
                output += "0";
            output += binary;
            return output;
        }
        if (binary.Length % 8 != 0)
        {
            for (int i = 0; i < (binary.Length / 8 + 1) * 8 - binary.Length; i++)
                output += "0";
            output += binary;
```

```
            return output;
        }
        return binary;
    }

    public static string GetScramKey(uint[] scrambler)
    {
        var s = new StringBuilder();
        foreach (var i in scrambler)
        {
            var ttt = Convert.ToInt32(i);
            var b = BitConverter.GetBytes(Convert.ToInt16(ttt));
            byte[] b1 = new byte[1] {b[0]};
            var buf = new BitArray(b1);
            var t = buf.Length - 1;
            s.Append(buf.Get(t) ? '1' : '0');
        }
        return s.ToString();
    }
}
```

FileUtility.cs

```
public static class FileUtility

    {

        public static string RootDirectory = Directory.GetCurrentDirectory();




        public static DataModel DeserializeString(string filename) =>
JsonSerializer.Deserialize<DataModel>(filename);

        public static string Serialize(DataModel Data) => JsonSerializer.Serialize(Data);



        public static void JSONSave(string filename, string text) =>
File.WriteAllText($"{Directory.GetCurrentDirectory()}\\Resources\\{filename}", text);

        public static string JSONSrt(string filename) =>
File.ReadAllText($"{Directory.GetCurrentDirectory()}\\Resources\\{filename}");

    }
```

MainWindow.xaml.cs

```
using Lab1_Gamming_Srammbling.CryptoClass;
using Lab1_Gamming_Srammbling.Models;
using Lab1_Gamming_Srammbling.Utilitiets;
using System;
using System.Collections;
using System.Windows;

namespace Lab1_Gamming_Srammbling
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private string TextFormarFlag = "Text";
        private string OldTextFormarFlag = "Text";
        private string KeyFormarFlag = "Rand";
        private string ResourceFile = "Data.json";
        private string TextFile = "Text.json";
        private string KeyFile = "Key.json";
        private string ChiphrFile = "ChiphrText.json";
        private string ScramFile = "ScramStart.json";
        private string ScramblerKey = "";
        private byte[] TextArray = null;
        private byte[] KeyArray = null;
        private byte[] ChiphrArray = null;

        private void TextFormat_DropDownClosed(object sender, EventArgs e)
        {

            if (TextFormarFlag == "Text")
            {
                Text.Text = ConverteUtility.UniConvert(Text.Text, TextFormarFlag, TextFormat.Text);
                Key.Text = ConverteUtility.UniConvert(Key.Text, TextFormarFlag, TextFormat.Text);
                Chiphrtext.Text = ConverteUtility.UniConvert(Chiphrtext.Text, TextFormarFlag,
TextFormat.Text);
            }

            if (TextFormarFlag == "Binary")
                if (ConverteUtility.CheckIncorrectFormat(Text.Text, "Bin") &&
ConverteUtility.CheckIncorrectLength(Text.Text))
                {
                    Text.Text = ConverteUtility.UniConvert(Text.Text, TextFormarFlag, TextFormat.Text);
                    Key.Text = ConverteUtility.UniConvert(Key.Text, TextFormarFlag, TextFormat.Text);
                    Chiphrtext.Text = ConverteUtility.UniConvert(Chiphrtext.Text, TextFormarFlag,
TextFormat.Text);
                }
                else
                {
                    MessageBox.Show("Не корректный формат");
                    TextFormat.SelectedIndex = 1;
                }

            if (TextFormarFlag == "Hexadecimal")
                if (ConverteUtility.CheckIncorrectFormat(Text.Text, "Hex") &&
ConverteUtility.CheckIncorrectLength(Text.Text))
                {
                    Text.Text = ConverteUtility.UniConvert(Text.Text, TextFormarFlag, TextFormat.Text);
                    Key.Text = ConverteUtility.UniConvert(Key.Text, TextFormarFlag, TextFormat.Text);
                    Chiphrtext.Text = ConverteUtility.UniConvert(Chiphrtext.Text, TextFormarFlag,
TextFormat.Text);
                }
                else
```

```
        {
            MessageBox.Show("Не корректный формат");
            TextFormat.SelectedIndex = 2;
        }

        OldTextFormarFlag = TextFormat.Text;
}

private void TextFormat_DropDownOpened(object sender, EventArgs e)
{
    TextFormarFlag = TextFormat.Text;
    OldTextFormarFlag = TextFormat.Text;
}

private void CiphButton_Click(object sender, RoutedEventArgs e)
{
    var encryptResults = AESClass.Converter(TextArray, KeyArray, TextFormat.Text);
    ChiphrArray = encryptResults.chipout;
    if (encryptResults.code == 0)
        Chiphrtext.Text = encryptResults.output;
    if (encryptResults.code == 2)
        MessageBox.Show("Не корректная длина текста или ключа");
    if (encryptResults.code == 1)
        MessageBox.Show("Не корректный формат текста или ключа");
}

private void UpdateKey_Click(object sender, RoutedEventArgs e)
{
    KeyArray = AESClass.RandKey();
    if (KeyFormarFlag == "Rand")
    {
        if (TextFormat.Text == "Text")
        {
            Key.Text = ConverteUtility.ConvertByteArrayToString(AESClass.RandKey());
        }
        if (TextFormat.Text == "Binary")
        {
            Key.Text = ConverteUtility.ConvertByteArraToBinaryStr(AESClass.RandKey());
        }
        if (TextFormat.Text == "Hexadecimal")
        {
            Key.Text = ConverteUtility.ByteArrayToHexString(AESClass.RandKey());
        }
    }
}



private void SaveFile_Click(object sender, RoutedEventArgs e)
{
    var text = new TextModel();
    var chiphr = new ChiphrModel();
    var key = new KeyModel();

    text.Text = Text.Text;
    key.Key = Key.Text;
    chiphr.Chiphr = Chiphrtext.Text;

    if (text.Text != "")
        FileUtility.JSONSave(TextFile, FileUtility.Serialize(text));
    else
        MessageBox.Show("Добавте текст");

    if (key.Key != "")
        FileUtility.JSONSave(KeyFile, FileUtility.Serialize(key));
    else
        MessageBox.Show("Добавте ключ");
```

```
    if (chiphr.Chiphr != "")
        FileUtility.JSONSave(ChiphrFile, FileUtility.Serialize(chiphr));
    else
        MessageBox.Show("Добавте шифротекст");

}

private void FileLoad_Click(object sender, RoutedEventArgs e)
{
    Text.Clear();
    Key.Clear();
    Chiphrtext.Clear();
    var text = FileUtility.DeserializeString<TextModel>(FileUtility.JSONSrt(TextFile));
    var key = FileUtility.DeserializeString<KeyModel>(FileUtility.JSONSrt(KeyFile));
    Text.Text = text.Text;
    Key.Text = key.Key;
}


private void DeciphButton_Click(object sender, RoutedEventArgs e)
{
    Text.Text = Chiphrtext.Text;
    TextArray = ChiphrArray;
    Chiphrtext.Clear();

    var encryptResults = AESClass.Converter(TextArray, KeyArray, TextFormat.Text, "decryt");
    if (encryptResults.code == 0)
        Chiphrtext.Text = encryptResults.output;
    if (encryptResults.code == 2)
        MessageBox.Show("Не корректная длина текста или ключа");
    if (encryptResults.code == 1)
        MessageBox.Show("Не корректный формат текста или ключа");
}

private void LoadChiphFile_Click(object sender, RoutedEventArgs e)
{
    Text.Clear();
    Key.Clear();
    Chiphrtext.Clear();
    var chiphr = FileUtility.DeserializeString<ChiphrModel>(FileUtility.JSONSrt(ChiphrFile));
    var key = FileUtility.DeserializeString<KeyModel>(FileUtility.JSONSrt(KeyFile));
    Text.Text = chiphr.Chiphr;
    Key.Text = key.Key;
}

private void AESRUN_Click(object sender, RoutedEventArgs e)
{
    var text = ConverteUtility.HexStringToByteArray(AESText.Text);
    var key = ConverteUtility.HexStringToByteArray(AESKey.Text);

    var output = AESClass.OneRound(text, key);

    S1.Text = ConverteUtility.ByteArrayToHexString(output.s1);
    S2.Text = ConverteUtility.ByteArrayToHexString(output.s2);
    S3.Text = ConverteUtility.ByteArrayToHexString(output.s3);
    S4.Text = ConverteUtility.ByteArrayToHexString(output.s4);
    S5.Text = ConverteUtility.ByteArrayToHexString(output.s5);
}

private void Text_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)
{
    if (Text.Text != "" && TextFormat.Text == OldTextFormarFlag)
    {
        if (TextFormat.Text == "Text")
            TextArray = ConverteUtility.ConvertStringToByteArray(Text.Text);
        if (TextFormat.Text == "Binary")
```

```
                TextArray = ConverteUtility.ConvertBinaryStrToByte(Text.Text);
            if (TextFormat.Text == "Hexadecimal")
                TextArray = ConverteUtility.HexStringToByteArray(Text.Text);
        }
    }

    private void Key_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)
    {
        if (Key.Text != "" && TextFormat.Text == OldTextFormarFlag)
        {
            if (TextFormat.Text == "Text")
                KeyArray = ConverteUtility.ConvertStringToByteArray(Key.Text);
            if (TextFormat.Text == "Binary")
                KeyArray = ConverteUtility.ConvertBinaryStrToByte(Key.Text);
            if (TextFormat.Text == "Hexadecimal")
                KeyArray = ConverteUtility.HexStringToByteArray(Key.Text);
        }
    }
  }
}
```