**ЛАБОРАТОРНА РОБОТА № 2**

Багаторазовий шифрблокнот

***Мета роботи:*** Зламати шифротекст, отриманий за допомогою багаторазового шифрблокноту(шифру Вернама).

Шифротексти (в hex-форматі):

Шифротекст #1:

315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3

Шифротекст #2:

234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f

Шифротекст #3:

32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb

Шифротекст #4:

32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa

Шифротекст #5:

3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070

Шифротекст #6:

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4

Шифротекст #7:

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce

Шифротекст #8:

315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3

Шифротекст #9:

271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027

Шифротекст #10:

466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83

Цільовий шифротекст для дешифрування:

32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904

1. Виконаємо операцію XOR над шифротекстами та проXORимо результат з пробілом. В результаті в деяких випадках ми будемо мати велику літеру, що означатиме, що в одному тексті на данній позиції – пробіл, в іншому – та сама літера з нижнім регістром. Тобто можемо визначити значення ключа на цій позиції і розшифрувати символи в інших текстах:

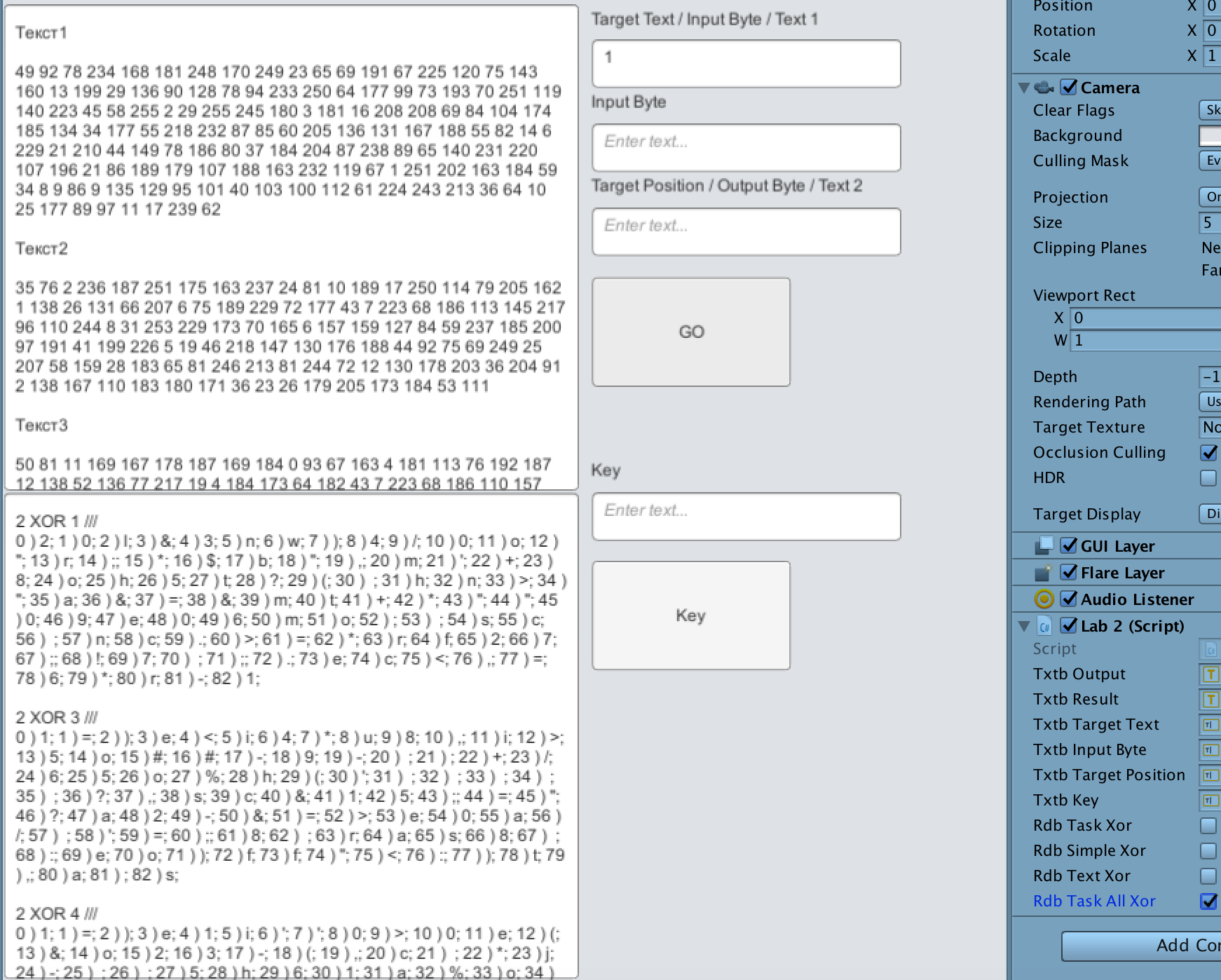


Рисунок 1 – Виконуємо операцію XOR над шифротекстами та пробілом.

1. Далі, основуючись на отриманих данних, посимвольно розшифровуємо тексти, роблячи припущення:

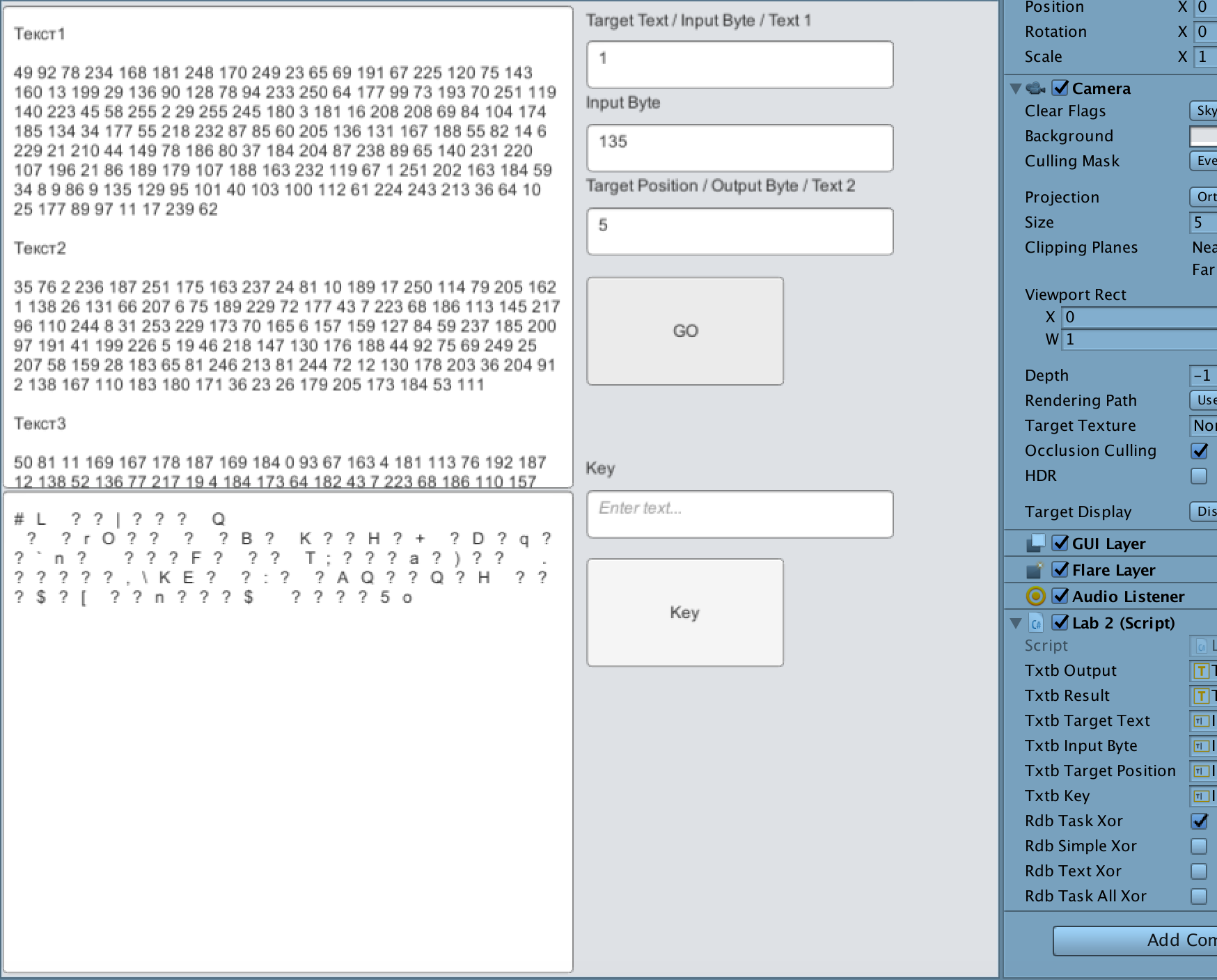


Рисунок 2 – Статистика байтів на кожній позиції в межах довжини ключа

1. В результаті отримуємо розшифрований фінальний текст та ключ:

Ключ : 102 57 110 137 201 219 216 204 152 116 53 42 205 99 149 16 46 175 206 120 170 127 237 40 160 127 107 201 141 41 197 11 105 176 51 154 25 248 170 64 26 156 109 112 143 128 192 102 199 99 254 240 18 49 72 205 216 232 2 208 91 169 135 119 51 93 174 252 236 213 156 67 58 107 38 139 96 191 78 240 60 154 97

Розшифрований фінальний текст:

The secret message is: When using a stream cipher, never use the key more than once

Рисунок 3 – Результат

Лістинг:

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using System.Text;  
using System;  
using UnityEngine.UI;  
  
public class Lab2 : MonoBehaviour {  
  
      
    List<string> InputTexts = new List<string>();  
    List<string[]> CipherTexts = new List<string[]>();  
    List<byte[]> ByteCipherTexts = new List<byte[]>();  
  
    int index = 0;  
    string[] temp;  
    byte[] tempb;  
  
    byte InputByte;  
    int InputPosition = 0;  
    int targetText = 10;  
  
    string finaltext = String.Empty;  
    int t = 0;  
  
    byte InputSimpleXorByte;  
    byte InputSimpleXorByte2;  
    byte OutputSimpleXorByte;  
  
    byte[] buffer;  
    List<byte[]> TaskXORByteCipherTexts = new List<byte[]>();   
    int InputTextXor1 = 0;  
    int InputTextXor2 = 0;  
    int count = 0;  
  
    string[] KeyString;  
    byte[] key;  
    byte[] finalTextByte;  
  
    public Text txtbOutput;  
    public Text txtbResult;  
    public InputField txtbTargetText;  
    public InputField txtbInputByte;  
    public InputField txtbTargetPosition;  
    public InputField txtbKey;  
  
    public bool rdbTaskXor;  
    public bool rdbSimpleXor;  
    public bool rdbTextXor;  
    public bool rdbTaskAllXor;  
  
    void Start()  
    {  
        InputTexts.Add(firstInputText);  
        InputTexts.Add(secondInputText);  
        InputTexts.Add(thirdInputText);  
        InputTexts.Add(fourthInputText);  
        InputTexts.Add(fifthInputText);  
        InputTexts.Add(sixthInputText);  
        InputTexts.Add(seventhInputText);  
        InputTexts.Add(eighthInputText);  
        InputTexts.Add(ninthInputText);  
        InputTexts.Add(tenthInputText);  
        InputTexts.Add(targetInputText);  
  
        for (int i = 0; i < InputTexts.Count; i++) {  
            CipherTexts.Add(temp);  
            CipherTexts[i] = new string[InputTexts[i].Length/2];  
        }  
  
        CreateCipherByte();  
        ConvertToByteFromString();  
        Output();  
    }  
  
    public void CreateCipherByte()  
    {  
        for (int i = 0; i < InputTexts.Count; i++) {  
            for (int j = 0; j < InputTexts[i].Length; j = j + 2) {  
                CipherTexts[i][index] = Convert.ToString(InputTexts[i][j]);  
                CipherTexts[i][index] += Convert.ToString(InputTexts[i][j+1]);  
                index++;  
            }  
            index = 0;  
        }  
  
    }  
    public void ConvertToByteFromString()  
    {  
        for (int i = 0; i < InputTexts.Count; i++)  
        {  
            ByteCipherTexts.Add(tempb);  
            ByteCipherTexts[i] = new byte[CipherTexts[i].Length];  
            for (int j = 0; j < CipherTexts[i].Length; j++)  
                ByteCipherTexts[i][j] = Convert.ToByte(CipherTexts[i][j],16);  
        }  
    }  
  
    public void Output()  
    {  
        for (int i = 0; i < ByteCipherTexts.Count; i++)  
        {  
            txtbOutput.text += Environment.NewLine;  
            txtbOutput.text += "Текст" +  (i + 1);  
            txtbOutput.text += Environment.NewLine;  
            txtbOutput.text += Environment.NewLine;  
            for (int j = 0; j < ByteCipherTexts[i].Length; j++)  
                txtbOutput.text += ByteCipherTexts[i][j] + " ";  
            txtbOutput.text += Environment.NewLine;  
        }  
    }  
  
    public void ConvertToStringFromByte(int inp\_text) {  
        finaltext = Encoding.ASCII.GetString(ByteCipherTexts[inp\_text]);  
        txtbResult.text += Environment.NewLine;  
  
        for (int i = 0; i < finaltext.Length; i++) {  
            txtbResult.text += finaltext[i];  
            txtbResult.text += "   ";  
        }  
    }  
  
    public void btnGoXOR\_Click()  
    {  
*//        txtbResult.text = "";*  
        buffer = null;  
        count = 0;  
  
        if (rdbTaskXor)  
        {  
            InputByte = Convert.ToByte(txtbInputByte.text);  
            InputPosition = Convert.ToInt32(txtbTargetPosition.text);  
            targetText = Convert.ToInt32(txtbTargetText.text);  
  
            for (int i = 0; i < ByteCipherTexts[targetText].Length; i++)  
                if (i == InputPosition)  
                    ByteCipherTexts[targetText][i] = Convert.ToByte(Convert.ToInt32(ByteCipherTexts[targetText][i]) ^ Convert.ToInt32(InputByte));  
            ConvertToStringFromByte(targetText);  
        }  
        else if (rdbSimpleXor)  
        {  
            InputSimpleXorByte = Convert.ToByte(txtbTargetText.text);  
            InputSimpleXorByte2 = Convert.ToByte(txtbInputByte.text);  
            OutputSimpleXorByte = Convert.ToByte(Convert.ToInt32(InputSimpleXorByte) ^ Convert.ToInt32(InputSimpleXorByte2));  
            txtbTargetPosition.text = OutputSimpleXorByte + "";  
        }  
  
        else if (rdbTextXor)  
        {  
            InputTextXor1 = Convert.ToInt32(txtbTargetText.text);  
            InputTextXor2 = Convert.ToInt32(txtbTargetPosition.text);  
  
            if (ByteCipherTexts[InputTextXor1].Length > ByteCipherTexts[InputTextXor2].Length)  
                buffer = new byte[ByteCipherTexts[InputTextXor2].Length];  
            else  
                buffer = new byte[ByteCipherTexts[InputTextXor1].Length];  
  
            for (int i = 0; i < ByteCipherTexts.Count; i++) {  
                for (int j = 0; j < buffer.Length; j++)  
                {  
                    buffer[j] = Convert.ToByte(Convert.ToInt32(ByteCipherTexts[InputTextXor1][j]) ^ Convert.ToInt32(ByteCipherTexts[InputTextXor2][j]) ^ 32);  
                }  
            }  
  
            finaltext = Encoding.ASCII.GetString(buffer);  
            txtbResult.text += Environment.NewLine;  
            txtbResult.text += (InputTextXor1 + 1) + "XOR" + (InputTextXor2 + 1) + " /// ";  
            for (int i = 0; i < finaltext.Length; i++)  
            {  
  
                txtbResult.text += i + " ) ";  
                txtbResult.text += finaltext[i];  
                txtbResult.text += ";";  
                txtbResult.text += " ";  
            }  
        }  
  
        else if (rdbTaskAllXor)  
        {  
            while (count != 11)  
            {  
  
                TextsXOR();  
                count++;  
            }  
        }  
    }  
  
    public void TextsXOR() {  
        buffer = null;  
        InputTextXor1 = Convert.ToInt32(txtbTargetText.text);  
        buffer = new byte[ByteCipherTexts[10].Length];  
  
        if (count == InputTextXor1 && count!=10)  
            count++;  
        for (int i = 0; i < ByteCipherTexts.Count; i++)  
        {  
            for (int j = 0; j < buffer.Length; j++)  
            {  
                buffer[j] = Convert.ToByte(Convert.ToInt32(ByteCipherTexts[InputTextXor1][j]) ^ Convert.ToInt32(ByteCipherTexts[count][j]) ^ 32);  
            }  
        }  
  
        finaltext = Encoding.ASCII.GetString(buffer);  
        txtbResult.text += Environment.NewLine;  
        txtbResult.text += (InputTextXor1 + 1) + " XOR " + (count + 1) + " /// ";  
        txtbResult.text += Environment.NewLine;  
        for (int i = 0; i < finaltext.Length; i++)  
        {  
  
            txtbResult.text += i + " ) ";  
            txtbResult.text += finaltext[i];  
            txtbResult.text += ";";  
            txtbResult.text += " ";  
        }  
        txtbResult.text += Environment.NewLine;  
    }  
  
    public void btnKey\_Click()  
    {  
        key = new byte[ByteCipherTexts[10].Length];  
        KeyString = new string[ByteCipherTexts[10].Length];  
        finalTextByte = new byte[ByteCipherTexts[10].Length];  
  
        KeyString = txtbKey.text.Split(' ');  
        for (int i = 0; i < key.Length; i++) {  
            key[i] = Convert.ToByte(KeyString[i]);  
        }  
  
        for (int i = 0; i < key.Length; i++) {  
            finalTextByte[i] = Convert.ToByte(Convert.ToInt32(key[i]) ^ Convert.ToInt32(ByteCipherTexts[10][i]));  
        }  
  
        finaltext = Encoding.ASCII.GetString(finalTextByte);  
        for (int i = 0; i < finaltext.Length; i++) {  
            txtbResult.text += finaltext[i];  
        }  
    }  
  
}