Author: Duma Odette Alexandra
Group: 823

Laboratory 1: Classical Cryptography (private key)

Create a project with the following features:
(i) A graphical interface. The user will be given the possibility to choose the characters of the alphabet to be used out of the blank and the 26 letters of the English alphabet. The implicit alphabet will have all the 27 characters.
(ii) Given an encryption key and a plaintext, encrypts the plaintext. There will be a key validation and a plaintext validation.
(iii) Given an encryption key and a ciphertext, computes the decryption key and then decrypts the ciphertext.

Cipher text: Permutation Cipher

1) Details

In classical cryptography, a permutation cipher is a transposition cipher, in which the key is a permutation. To apply the cipher, a random permutation of size (order) e is generated (the larger the value of e the more secure the cipher). The plain text is broken into segments of size e and the letters within that segment are permuted according to the key.

Because the cipher operates on blocks of letters of size e, the plaintext and the ciphertext have to have a length which is multiple of e. As a result, the plaintext may be padded. If the padding is identifiable, then part of the key is revealed, thus the system is not very secure (there are even more weaknesses).

The operations of encryption and decryption are very similar. The only difference is that, while the encryption works with a permutation, the decryption operates with the inverse of the same permutation, and both the algorithms are almost identical from some points of view.

2) Code Samples

Permutations are done using the “Random” feature, as following:

```
public List<E> ShuffleList<E>(List<E> inputList)
        {
            List<E> randomList = new List<E>();

            Random r = new Random();
            int randomIndex = 0;
            while (inputList.Count > 0)
            {
                randomIndex = r.Next(0, inputList.Count); //Choose a random object in the list
                randomList.Add(inputList[randomIndex]);   //add it to the new, random list
                inputList.RemoveAt(randomIndex);          //remove to avoid duplicates
            }

            return randomList; //return the new random list
        }
```

For the permutation, we are using a **list of integers**.
The alphabet is loaded in a **list of chars** when the application starts, if the user will use the default alphabet (n=27, m=5),

```
alpha = new List<char>(n);
                alpha.Add(' ');
                for (int i = 1; i < n-1; i++)
                    alpha.Add((char)(96 + i));
```

Or when the “Load Alphabet” button is clicked (for user-defined alphabets).

```
string a = txtalpha.Text;
```

```
alpha = new List<char>(n);
            alpha.Add(' '); //add space by default
            for (int i = 0; i < n; i++)
                alpha.Add(a[i]);
```

The input text is transferred from the text box to a **string**, after some validation (all letters must belong to the given alphabet, for example). Further on, all the operations are done using this string.

When the "Encrypt" button is pressed, the encryption is done, using the given permutation (default or user-defined):

```
while (s.Length % m != 0)  //add spaces if the number of characters is not divisible by 5
                s = s + " ";         //in order to be able to break the text in 5-characters long
strings.

while (p < s.Length)
           {
               for (int j = 0; j < m; j++)
               {
                    if (valid(s[p + l2[j] - 1]))    //verifies if the character is in the alphabet
                        res = res + s[p + l2[j] - 1];
}}
```

The process for decryption is analogue. The inverse of the permutation is computed beforehand.

```
string s = txtIn.Text;
            while (s.Length % m != 0)
                s = s + " ";
string res = "";

            //compute the inverse of the permutation
            List<int> aux = new List<int>(m);
            for (int i = 0; i < m; i++)
            {
                int val = index(i + 1, l2);
                aux.Add(val + 1);
            }

            //basically, the whole crypting algorithm is reversed
            int p = 0;
            while (p < s.Length)
            {
                for (int j = 0; j < m; j++)
                {
                    if (valid(s[p + aux[j] - 1]))
                        res = res + s[p + aux[j] - 1];
}}
```

3) Testing

Alphabet: default
Input text: every rose has its thorn
Permutation: 42135
After encryption: rveeysr oesh a tistnohr
After decryption: every rose has its thorn

Alphabet: acehirst
Input text: this is a secret
Permutation: 2413
After encryption: hstii s easrtce
After decryption: this is a secret