**ЛАБОРАТОРНА РОБОТА № 1**

Процес шифрування

1. Зсунути черговий символ відкритого тексту на величину, яку визначає черговий символ ключа і таблиця Віженера.

2. Якщо символи ключа вичерпані, то почати використання символів ключа спочатку.

Комп’ютеризований варіант шифру Віженера

1. І для ключа , і для тексту розглядати символи як байти.

2. Замість зсуву використати операцію XOR .

Переваги: 1. Ми більше не обмежені лише строковими літерами.

2. Зручно реалізовувати на комп’ютері.

Метод Касіски (Kasiski)

Етап 1. З’ясування довжини ключа.

Етап 2. Статистичний аналіз тексту з врахуваням періодичного застосування ключа. Визначення побайтово відкритого тексту.

Ідеї для визначення довжини ключа.

Нехай pi (0<= i <=255) частота використання

i-го байту у відкритому тексті. Тоді:

1. pi = 0 для і < 32 або і > 127.

2. p98 = частота використання “b”.

3. Розподіл частот далекий від рівномірного.

**ВИХІДНИЙ КОД**

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

namespace interactive\_viz

{

class Program

{

static string text

//private static string text = "";

static HashSet<int> filterBytes(List<string> bytes, int start, int offset)

{

HashSet<int> filteredBytesList = new HashSet<int>();

for (int w = start; w < bytes.Count; w += offset)

filteredBytesList.Add(int.Parse(bytes[w], NumberStyles.HexNumber));

return filteredBytesList;

}

static string openByKeyValue(List<string> bytes, int keyChangePosition, int keyLength, int keyCandidateValue)

{

for (int i = 0; i < bytes.Count - 1; i += keyLength)

{

var x = int.Parse(bytes[i + keyChangePosition], NumberStyles.HexNumber);

var y = keyCandidateValue;

var newByte = x ^ y;

bytes[i + keyChangePosition] = "" + (char)newByte;

}

return string.Join(string.Empty, bytes);

}

//static bool analyze(HashSet<char> symbolicQuery)

//{

//}

static int[] getKeysVariants(List<string> bytes, List<HashSet<char>> candidatesList, int keyLength)

{

List<List<char>> symbolic = new List<List<char>>();

int[] keysCandidates = new int[keyLength];

for (int l = 0; l < keyLength; l++)

{

for (int k = 0; k < candidatesList.ElementAt(l).Count; k++)

{

var set = filterBytes(bytes, l, keyLength);

symbolic.Add(new List<char>());

for (int i = 0; i < set.Count; i++)

{

var a = set.ElementAt(i);

var b = (int) candidatesList.ElementAt(l).ElementAt(k);

symbolic.ElementAt(k).Add((char) (a ^ b));

}

}

keysCandidates[l] = candidatesList[l].Count > 0 ? candidatesList[l].ElementAt(0) : 0; // MAKE ALGORITHM STRONGER

}

return keysCandidates;

}

static List<HashSet<char>> makeCandidatesList(List<string> bytes, int keyLength)

{

List<HashSet<char>> candidatesList = new List<HashSet<char>>();

for (int k = 0; k < keyLength; k++)

{

candidatesList.Add(new HashSet<char>());

var set = filterBytes(bytes, k, keyLength);

for (int i = 0; i < 256; i++)

{

if (set.All(x =>

{

var response = x ^ i;

if ((response == 32) || (response >= 35 && response < 60) || (response >= 65 && response < 90) || (response >= 97 && response < 122))

return true;

return false;

}))

{

candidatesList.ElementAt(k).Add((char)i);

}

}

}

return candidatesList;

}

static Dictionary<int, double> probateKeyLength()

{

Dictionary<int, double> stats = new Dictionary<int, double>();

for (int i = 2; i < 20; i++)

{

var query = (from Match m in Regex.Matches(text, @".{1," + (i \* 2) + "}")

select m.Value).ToList();

var group = query.Select(x => x.Substring(0, 2)).GroupBy(k => k);

var L = query.Count();

var S = group.Sum(g =>

{

var c = g.Count();

return Math.Pow((double)c / L, 2);

});

stats.Add(i, S);

}

return stats;

}

static void Main(string[] args)

{

var bytes = (from Match m in Regex.Matches(text, @".{1,2}")

select m.Value).ToList();

var order = probateKeyLength().OrderByDescending(o => o.Value);

foreach (var orderItem in order)

{

Console.WriteLine($"Key length = {orderItem.Key} S = {orderItem.Value}");

}

Console.WriteLine();

Console.Write("Probate keyLength = ");

int keyLength = int.Parse(Console.ReadLine());

var candidatesList = makeCandidatesList(bytes, keyLength);

var keyCandidate = getKeysVariants(bytes, candidatesList, keyLength);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine($"\n\n KEY BYTE DEC SNAPSHOT : {String.Join(" ", keyCandidate)}\nPress any key for step by step flow\n");

Console.ResetColor();

//var keyCandidate = new int[7] { 186, 31, 145, 178, 83, 205, 62 };

// var keyCandidate = new int[7] { 105, 104, 100, 109,105,116,114 };

for (int i = 0; i < keyCandidate.Length; i++)

{

Console.WriteLine(openByKeyValue(bytes, i, keyLength, keyCandidate[i]));

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Press Space To Start Encrypt Cycle");

Console.ReadKey();

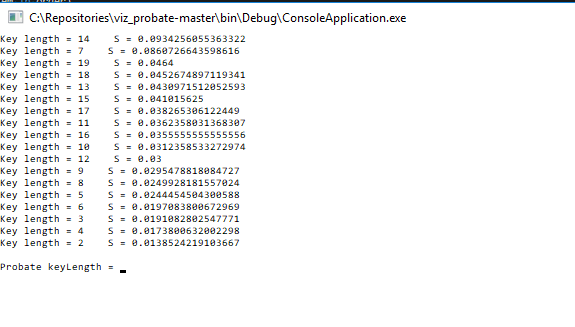
Console.WriteLine();

Console.WriteLine();

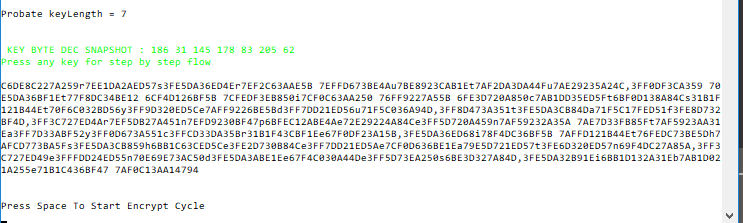
}

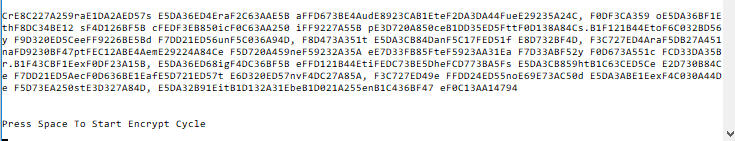
**РЕЗУЛЬТАТ ВИКОНАННЯ**

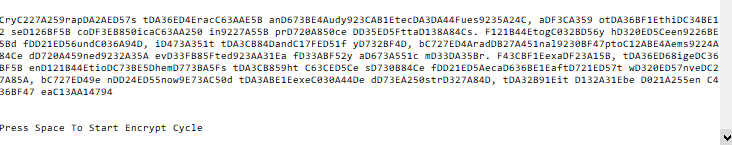
**Виявлення варіантів довжини ключа**

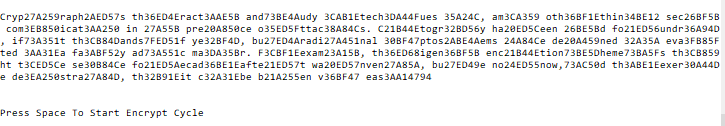


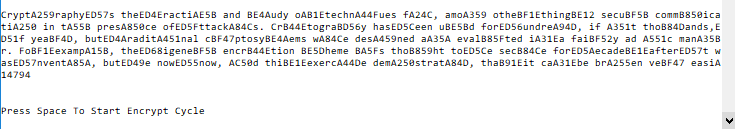
**Послідовне розшифрування в автоматичному режимі**

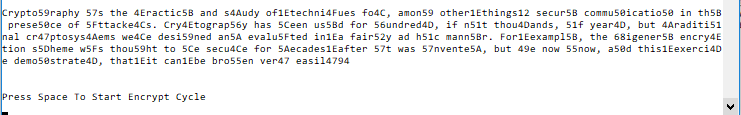












**РЕЗУЛЬТАТ**

