# КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ТАРАСА ШЕВЧЕНКА



## ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**Кафедра прикладних інформаційних систем**

**Звіт до лабораторної роботи №11**

# з курсу

**«Безпека мереж і комп’ютерних систем»**

*Студента 2 курсу*

*групи ПП-21 спеціальності 122 «Комп'ютерні науки» ОП «Прикладне програмування»*

%username%

*Викладач:*

проф. Сайко В.Г.

## Київ – 202

**1.Назва роботи**

Блокові симетричні алгоритми шифрування.

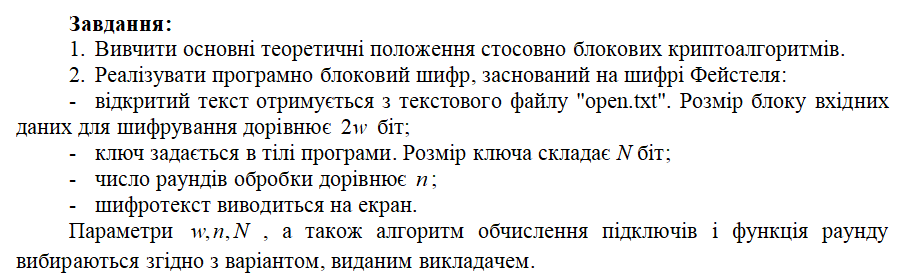
1. **Тема роботи**

Блокові симетричні алгоритми шифрування.

1. **Мета роботи**

Ознайомитися з основними поняттями, присвяченими принципам функціонування блокових алгоритмів. Дослідити структуру шифру Фейстеля.

1. **Умова завдання**



1. **Рішення**

Код на С#:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab11\_feistel\_

{

class Program

{

private static byte Encrypt(byte msg, Func<byte, byte, byte> FunctionF, byte[] keys)

{

byte step = msg;

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

{

step = FeistelStep(step, keys[i], FunctionF);

}

return step;

}

private static byte Decrypt(byte msg, Func<byte, byte, byte> FunctionF, byte[] keys)

{

byte step = msg;

step = InversionLR(step);

for (int i = keys.Length - 1; i >= 0; i--)

{

step = FeistelStep(step, keys[i], FunctionF);

}

step = InversionLR(step);

return step;

}

private static byte FunctionF(byte x, byte key)

{

return Xor(x, key);

}

private static byte FeistelStep(byte msg, byte key, Func<byte, byte, byte> FunctionF)

{

var R = GetR(msg);

var L = GetL(msg);

var funcResult = OperateR(R, key, FunctionF);

var xorResult = OperateL(L, funcResult, Xor);

var finalResult = InversionLR(xorResult, R);

return finalResult;

}

private static byte Xor(byte x, byte y)

{

return (byte)((int)x ^ (int)y);

}

private static byte GetR(byte x)

{

var temp = (byte)(((int)x) << 4);

return (byte)(((int)temp) >> 4);

}

private static byte GetL(byte x)

{

var temp = (byte)(((int)x) >> 4);

return (byte)(((int)temp) << 4);

}

private static byte InversionLR(byte l, byte r)

{

l = (byte)(((int)l) >> 4);

r = (byte)(((int)r) << 4);

return Xor(r, l);

}

private static byte InversionLR(byte msg)

{

var R = GetR(msg);

var L = GetL(msg);

return InversionLR(L, R);

}

private static byte OperateL(byte l, byte key, Func<byte, byte, byte> function)

{

key = (byte)(((int)key) << 4);

return function(l, key);

}

private static byte OperateR(byte r, byte key, Func<byte, byte, byte> function)

{

return function(r, key);

}

private static void PrintOutByte(string initialText, byte x)

{

var temp = (int)x;

int bits = 0;

int factor = 1;

for (int i = 7; i >= 0; i--)

{

bits += (temp % 2) \* factor;

factor \*= 10;

temp /= 2;

}

Console.WriteLine(initialText + "\t" + bits.ToString("00000000") + "\n");

}

private static byte EnsureKeyHas4Bits(byte key)

{

return (byte)((int)key % 16);

}

public static void EncryptFile(string fileInPath, string fileOutPath, Func<byte, byte, byte> FunctionF, byte[] keys)

{

byte[] file = File.ReadAllBytes(fileInPath);

byte[] encFile = new byte[file.Length];

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

{

encFile[i] = Encrypt(file[i], FunctionF, keys);

}

File.WriteAllBytes(fileOutPath, encFile);

}

public static void DecryptFile(string fileInPath, string fileOutPath, Func<byte, byte, byte> FunctionF, byte[] keys)

{

byte[] file = File.ReadAllBytes(fileInPath);

byte[] decFile = new byte[file.Length];

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

{

decFile[i] = Decrypt(file[i], FunctionF, keys);

}

File.WriteAllBytes(fileOutPath, decFile);

}

static void Main(string[] args)

{

byte msg = 150;

PrintOutByte("msg: ", msg);

byte[] keys = { 15, 1, 19, 3, 9 };

var encrypted = Encrypt(msg, FunctionF, keys);

PrintOutByte("enc: ", encrypted);

var decrypted = Decrypt(encrypted, FunctionF, keys);

PrintOutByte("dec: ", decrypted);

EncryptFile("plaintext.txt", "encoded.txt", FunctionF, keys);

DecryptFile("encoded.txt", "decoded.txt", FunctionF, keys);

/\*

if (msg == decrypted)

Console.WriteLine("u did it");

else

Console.WriteLine("loser!");

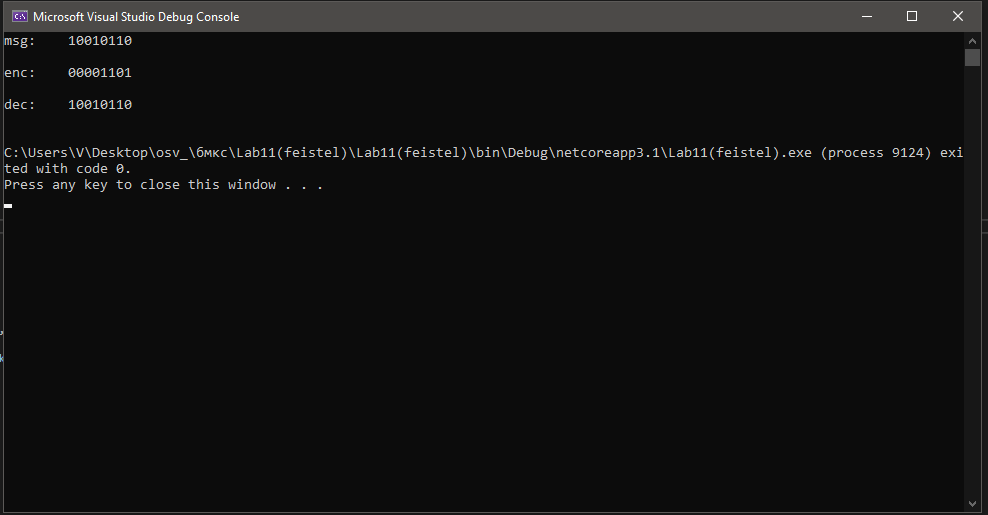
\*/

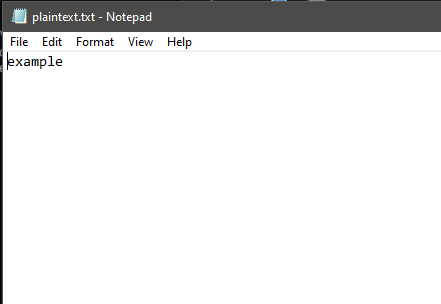
}

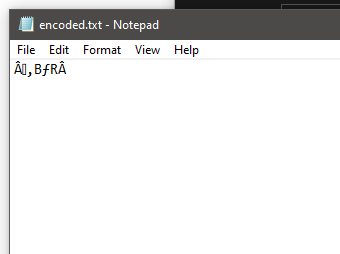
}

}

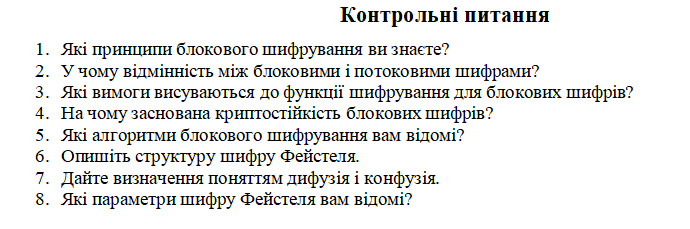
Результат роботи:







**Контрольні запитання:**



Обробка блоку декількох байт за одну ітерацію (як правило 8 або 16).

1. Потоковий шифрує потік байтів, блочний шифрує блоки з декількох байтів за ітерацію.
2. До функції блокового шифрування  висуваються наступні вимоги:

Функція  має бути зворотною.

Не повинно існувати інших методів отримання повідомлення  по відомому блоку , окрім як повним перебором ключів .

Не повинно існувати інших методів визначення, яким ключем було зашифроване повідомлення, окрім як повним перебором ключів.

1. Щоб надати необхідну стійкість шифру, шифрувальна функція працює циклічно: це називається циклічним кодуванням

1. Ітераційні блочні шифри, SP-сітки, шифр Фейстеля

6. На вхід алгоритму шифрування подається блок відкритого тексту завдовжки  бітів і ключ . Блок відкритого тексту розділяється на дві рівні частини?,  й , які послідовно проходять через  раундів обробки, а потім об'єднуються знову для отримання блоку шифрованого тексту відповідної довжини.

Для раунду  як вхідні дані виступають  и ,, отримані на виході поперднього раунду, і підключ , що обчислюється по загальному ключу . Усі підключі відрізняються як від загального ключа, так і один від одного.

Усі раунди обробки проходять за однією і тією ж схемою. Спочатку для лівої половини блоку даних виконується операція підстановки. Вона полягає в застосуванні до правої половини блоку даних деякої функції раунду  і в наступному додаванні отриманого результату з лівою половиною блоку даних за допомогою операції ХОR. Для усіх раундів функція  має одну і ту ж структуру, але залежить від параметра – підключа раунда . Після підстановки виконується перестановка, дві половини блоку даних міняються місцями. Уся ця структура в цілому є частинним випадком так званої підстановлювально-перестановочної схеми, запропонованої Шеноном.

Після закінчення останнього раунду виконується ще одна перестановка, яка, по суті, відміняє перестановку, виконану в останньому раунді. Тому, знехтувавши лише зовнішньою узгодженістю представлення, ці дві перестановки зі схеми можна видалити.

1. Суть дифузії полягає в розсіянні статистичних особливостей відкритого тексту по широкому діапазону статистичних характеристик шифрованого тексту. Це досягається тим, що значення кожного елементу відкритого тексту впливає на значення багатьох елементів шифрованого тексту, або будь-який з елементів шифрованого тексту залежить від безлічі елементів відкритого тексту.

Що стосується конфузії, то перед нею ставиться завдання в максимальній мірі ускладнити статистичний взаємозв'язок між шифрованим текстом і ключем з метою протистояння спробам визначити ключ. Це досягається використанням складних підстановочних алгоритмів: прості лінійні подстановочні функції збільшують складність алгоритму лише в незначній мірі.

1. Ключ, число раундів обробки, розмір вхідних даних та ключа.

**Висновки**

В результаті виконання даної лабораторної роботи я ознайомився з основними поняттями, присвяченими принципам функціонування блокових алгоритмів. Дослідив структуру шифру Фейстеля, створив програмний засіб, що реалізовує схожий шифр, використовуючи мову програмування С#. Створив клас з методами, викликав їх у функції main, отримав та оцінив результат.

Вважаю дану лабораторну роботу виконаною в повному обсязі.