МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

кафедра «Вычислительная техника».

**Лабораторная работа №3**

**тема: «Изучение работы блоковых шифров и способов их объединения»**

по дисциплине: «Информационная безопасность и защита информации»

Выполнил:

студент 4 курса, гр. ИВТВМбд-41

Захарычев Никита Алексеевич.

Проверил:

доцент кафедры ВТ,

кандидат технических наук

Мартынов Антон Иванович.

г. Ульяновск, 2018

**Задание**

1. Изучить блочные алгоритмы шифрования.

2. Изучить режимы использования блочных шифров (ECB, CBC, CFB и OFB).

3. Изучить способы объединения блочных шифров.

4. Реализовать систему в соответствии с вариантами указанными в Таблице 1 и заданием: Изучить принцип работы алгоритма, который указан в варианте.

**Вариант №5**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **D** |
| Шифр Виженера | CFB | 1 байт | **-** |

**Ход работы**

Основной идеей блочного шифрования является операции над группами бит фиксированной длины – блоками, характерный размер которых меняется в определенных пределах. Если исходный текст (или его остаток) меньше размера блока, перед шифрованием его дополняют. Блочный шифр является важной компонентой многих криптографических протоколов и широко используется для защиты данных, передаваемых по сети.

**Шифр Виженера**

Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Для зашифровывания может использоваться таблица алфавитов, называемая tabula recta или квадрат (таблица) Виженера. Применительно к латинскому алфавиту таблица Виженера составляется из строк по 26 символов, причём каждая следующая строка сдвигается на несколько позиций. Таким образом, в таблице получается 26 различных шифров Цезаря. На каждом этапе шифрования используются различные алфавиты, выбираемые в зависимости от символа ключевого слова. Например, предположим, что исходный текст имеет такой вид:

**ATTACKATDAWN**

Человек, посылающий сообщение, записывает ключевое слово («LEMON») циклически до тех пор, пока его длина не будет соответствовать длине исходного текста:

**LEMONLEMONLE**

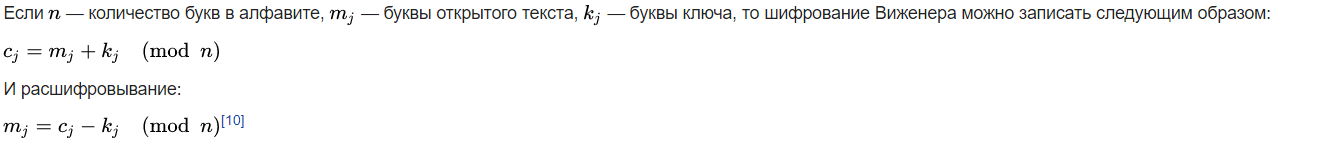
Первый символ исходного текста A зашифрован последовательностью L, которая является первым символом ключа. Первый символ L шифрованного текста находится на пересечении строки L и столбца A в таблице Виженера. Точно так же для второго символа исходного текста используется второй символ ключа; то есть второй символ шифрованного текста X получается на пересечении строки E и столбца T. Остальная часть исходного текста шифруется подобным способом.

Исходный текст: ATTACKATDAWN

Ключ: LEMONLEMONLE

Зашифрованный текст: LXFOPVEFRNHR

Расшифровывание производится следующим образом: находим в таблице Виженера строку, соответствующую первому символу ключевого слова; в данной строке находим первый символ зашифрованного текста. Столбец, в котором находится данный символ, соответствует первому символу исходного текста. Следующие символы зашифрованного текста расшифровываются подобным образом.

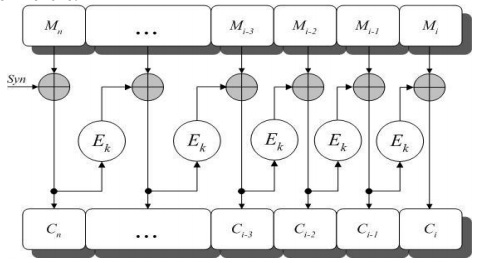


В компьютере такая операция соответствует сложению кодов ASCII символов сообщения и ключа по некоторому модулю. Кажется, что если таблица будет более сложной, чем циклическое смещение строк, то шифр станет надежнее. Это действительно так, если ее менять чаще, например, от слова к слову. Но составление таких таблиц, представляющих собой латинские квадраты, где любая буква встречается в строке или столбце один раз, трудоемко и его стоит делать лишь на ЭВМ. Для ручного же многоалфавитного шифра полагаются лишь на длину и сложность ключа, используя приведенную таблицу, которую можно не держать в тайне, а это упрощает шифрование и расшифровывание.

**Режим обратной связи по шифрованному тексту (CFB)**

В режиме обратной связи по шифрованному тексту (CFB) предыдущий блок шифрованного текста шифруется еще раз, и для получения очередного блока шифрованного текста результат складывается поразрядно по модулю 2 с блоком исходного текста. Для начала процесса шифрования также используется начальный вектор.

Стойкость режима CFB равна стойкости блочного шифра, лежащего в основе, и структура исходного текста скрывается за счет использования операции сложения по модулю 2. Манипулирование исходным текстом путем удаления блоков из начала или конца шифрованного текста становится невозможным. Однако в режиме CFB при шифровании двух идентичных блоков на следующем шаге шифрования также получатся идентичные зашифрованные блоки, что создает возможность утечки информации об исходном тексте.



**Реализация основных функций программы**

**Шифрование**

// Шифрование

public static char[] Encode(List<Byte> word, List<Byte> keys)

{

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

int keyword\_index = 0;

prev = (keys[0] + keys[keyword\_index]) % charact\_len;

foreach (var symbol in word)

{

int c, m = 0;

if (cfb == 0)

{

c = (symbol + keys[keyword\_index]) % charact\_len;

}

else

{

c = prev ^ symbol;

}

result.Add((char)c);

prev = c;

keyword\_index++;

if ((keyword\_index + 1) == keys.Count)

keyword\_index = 0;

}

return result.ToArray();

}

**Дешифрование**

// Дешифрование

public static char[] Decode(List<Byte> word, List<Byte> keys)

{

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

int keyword\_index = 0;

prev = (keys[0] + keys[keyword\_index]) % charact\_len;

foreach (var symbol in word)

{

int p, l = 0;

if (cfb == 0)

{

p = (symbol - keys[keyword\_index] + charact\_len) % charact\_len;

}

else

{

p = prev ^ symbol;

}

result.Add((char)p);

prev = symbol;

keyword\_index++;

if ((keyword\_index + 1) == keys.Count)

keyword\_index = 0;

}

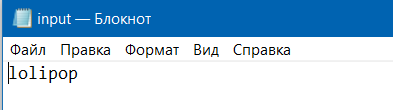
return result.ToArray();

}

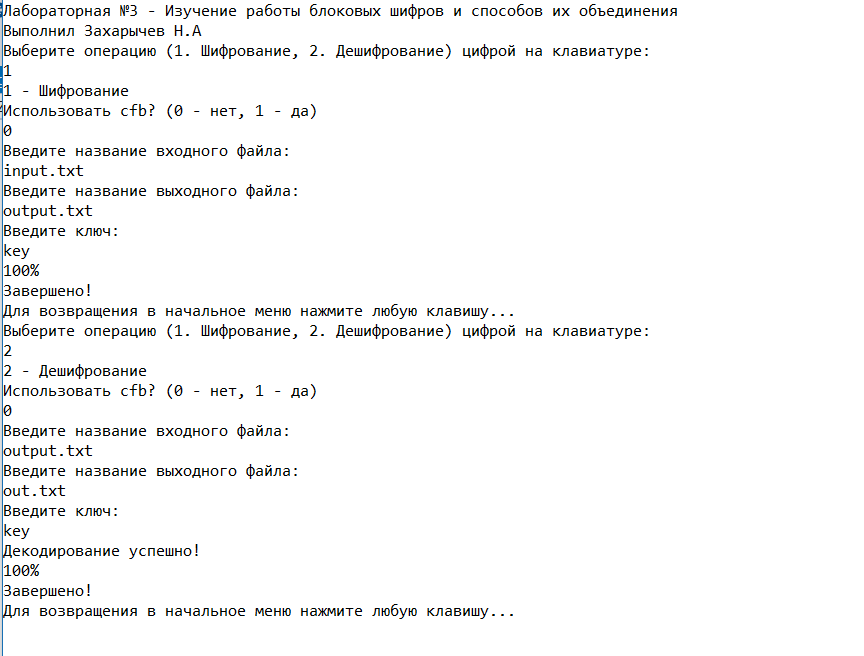
**Тестирование**

Было принято решение производить тестирование на файле с расширением .txt, для того, чтобы наглядно убедиться в корректности работы приложения.

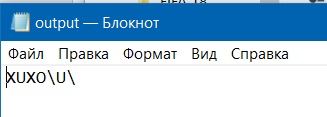
Исходный файл:



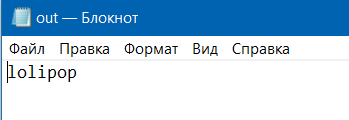
Операции над файлом:



Зашифрованный файл:



Расшифрованный файл:



В результате тестирования можно сделать вывод, что приложение работает корректно.

**Вывод**

В ходе выполнения данной лабораторной работы были получены навыки в шифровании/дешифровании произвольного файла с помощью блочных шифров с различными режимами шифрования.

**Исходный код**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace ZashitaInf\_3

{

class Program

{

public static int charact\_len = 127;

public static string flag = "";

public static int cfb = 0;

public static int prev = 0;

public static int controlTotal = 0;

// Шифрование

public static char[] Encode(List<Byte> word, List<Byte> keys)

{

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

int keyword\_index = 0;

prev = (keys[0] + keys[keyword\_index]) % charact\_len;

foreach (var symbol in word)

{

int c, m = 0;

if (cfb == 0)

{

c = (symbol + keys[keyword\_index]) % charact\_len;

}

else

{

c = prev ^ symbol;

}

result.Add((char)c);

prev = c;

keyword\_index++;

if ((keyword\_index + 1) == keys.Count)

keyword\_index = 0;

}

return result.ToArray();

}

// Дешифрование

public static char[] Decode(List<Byte> word, List<Byte> keys)

{

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

int keyword\_index = 0;

prev = (keys[0] + keys[keyword\_index]) % charact\_len;

foreach (var symbol in word)

{

int p, l = 0;

if (cfb == 0)

{

p = (symbol - keys[keyword\_index] + charact\_len) % charact\_len;

}

else

{

p = prev ^ symbol;

}

result.Add((char)p);

prev = symbol;

keyword\_index++;

if ((keyword\_index + 1) == keys.Count)

keyword\_index = 0;

}

return result.ToArray();

}

static void Main(string[] args)

{

Console.WriteLine("Лабораторная №3 - Изучение работы блоковых шифров и способов их объединения");

Console.WriteLine("Выполнил Захарычев Н.А");

string swit = "0";

while (swit == "0")

{

Console.WriteLine("Выберите операцию (1. Шифрование, 2. Дешифрование) цифрой на клавиатуре:");

swit = Console.ReadLine();

if (swit == "1")

{

Console.WriteLine("1 - Шифрование");

Console.WriteLine("Использовать cfb? (0 - нет, 1 - да)");

try

{

cfb = Int32.Parse(Console.ReadLine());

if (cfb != 1 && cfb != 0) throw new Exception();

}

catch (Exception e)

{

Console.WriteLine("Введите 0 или 1!");

Console.ReadKey();

return;

}

Console.WriteLine("Введите название входного файла:");

string inp\_file = Console.ReadLine();

Console.WriteLine("Введите название выходного файла:");

string out\_file = Console.ReadLine();

Console.WriteLine("Введите ключ:");

string key = Console.ReadLine();

List<byte> keys = Encoding.ASCII.GetBytes(key).ToList();

flag = key;

using (BinaryWriter sw = new BinaryWriter(new FileStream(out\_file, FileMode.Create)))

using (BinaryReader sr = new BinaryReader(new FileStream(inp\_file, FileMode.Open)))

{

List<Byte> word = new List<byte>();

while (sr.BaseStream.Position != sr.BaseStream.Length)

{

byte brd = sr.ReadByte();

word.Add(brd);

controlTotal = controlTotal + (int)brd;

}

var result = Encode(word, keys);

sw.Write(result);

Console.WriteLine(Math.Round((double)sr.BaseStream.Position / sr.BaseStream.Length, 2) \* 100 + "%");

}

Console.WriteLine("Завершено!");

Console.WriteLine("Для возвращения в начальное меню нажмите любую клавишу...");

Console.ReadKey();

swit = "0";

}

if (swit == "2")

{

Console.WriteLine("2 - Дешифрование");

Console.WriteLine("Использовать cfb? (0 - нет, 1 - да)");

try

{

cfb = Int32.Parse(Console.ReadLine());

if (cfb != 1 && cfb != 0) throw new Exception();

}

catch (Exception e)

{

Console.WriteLine("Введите 0 или 1!");

Console.ReadKey();

return;

}

Console.WriteLine("Введите название входного файла:");

string inp\_file = Console.ReadLine();

Console.WriteLine("Введите название выходного файла:");

string out\_file = Console.ReadLine();

Console.WriteLine("Введите ключ:");

string key = Console.ReadLine();

List<byte> keys = Encoding.ASCII.GetBytes(key).ToList();

using (BinaryWriter sw = new BinaryWriter(new FileStream(out\_file, FileMode.Create)))

using (BinaryReader sr = new BinaryReader(new FileStream(inp\_file, FileMode.Open)))

{

List<Byte> word = new List<byte>();

var controlTotalFile = 0;

while (sr.BaseStream.Position != sr.BaseStream.Length)

{

byte brd = sr.ReadByte();

word.Add(brd);

}

var result = Decode(word, keys);

sw.Write(result);

foreach (var d in result)

{

controlTotalFile += (byte)d;

}

if (controlTotalFile != controlTotal)

{

throw new Exception("Ошибка декодирования!");

} else

{

Console.WriteLine("Декодирование успешно!");

}

Console.WriteLine(Math.Round((double)sr.BaseStream.Position / sr.BaseStream.Length, 2) \* 100 + "%");

}

Console.WriteLine("Завершено!");

Console.WriteLine("Для возвращения в начальное меню нажмите любую клавишу...");

Console.ReadKey();

swit = "0";

}

}

}