МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Чувашский государственный университет имени И.Н. Ульянова»

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

Лабораторная работа №3

По дисциплине “Защита информации” на тему:

«Линейное шифрование данных (гаммирование)»

Вариант 6

Выполнил: ст. гр. ИВТ-11-21

Еремеев С.А.

Проверил: канд. техн. наук. Ковалев С.В.

Чебоксары 2024

**Теоретическая часть**

# Цель работы

Целью работы является знакомство с классическим криптографическим алгоритмом - алгоритмом линейного шифрования данных (шифрования гаммированием).

# Основные сведения

Если изученные в двух предыдущих лабораторных работах криптографические алгоритмы - подстановка и перестановка - основаны на блочном способе шифрования, то гаммирование, называемое также линейным шифрованием, основывается на поточном способе шифрования.

Шифрование гаммированием состоит в наложении на открытый текст X некоторой псевдослучайной последовательности K ("гаммы") символов того же алфавита. Для двоичного алфавита *V*={0,1} наложение K=*k1..km* на X=*v1..vm* состоит в замене *vi* на *vi* *ki* , где символом  обозначена операция сложения по модулю 2. Дешифрование сводится к наложению той же гаммы на шифртекст. Поскольку *vi* *ki* *ki* *vi*, результатом повторного выполнения той же операции будет действительно открытый текст. Для алфавита, включающего *n*>2 символов, где *n* - размер алфавита, при шифровании вместо операции  используется операция сложения по модулю *n*. При дешифровании происходит вычитание гаммы из шифртекста, а при получении отрицательной величины следует прибавить к ней величину *n*.

Чтобы обеспечить высокую криптостойкость шифртекста, последовательность K должна быть достаточно длинной, а именно - превосходить длину открытого текста X. Чтобы получить линейные последовательности элементов гаммы, длина которых превышает размер шифруемых данных, используются датчики псевдослучайных чисел (ПСЧ). Одним из таких датчиков, зарекомендовавшим свою эффективность, является так называемый линейный конгруэнтный датчик ПСЧ. Он вырабатывает последовательности псевдослучайных чисел *T(i)*, описываемые соотношением:

*T(i+1)=(A\*T(i)+C) mod B*,

где *A* и *C* - константы, *T(0)* - исходная величина, выбранная в качестве порождающего числа, а значение числа *B* устанавливается равным *2b*, где *b* - длина слова ЭВМ в битах. Такой датчик генерирует псевдослучайные числа с определенным периодом повторения, зависящим от выбранных значений *A* и *C*. Датчик имеет максимальный период *M* до того, как генерируемая последовательность чисел начнет повторяться. Для достижения максимальной криптостойкости, доступной данному методу шифрования, необходимо выбирать числа *A* и *C* таким образом, чтобы период *M* был максимальным. Как было показано Д.Кнутом, линейный конгруэнтный датчик ПСЧ имеет максимальную длину тогда и только тогда, когда *C* - нечетное и *A mod 4=1*.

Очевидно, что, поскольку гамма определяется датчиком ПСЧ, то совокупность значений *A*, *C*, *T(0)* и *B* фактически является тем секретным ключом, который и обеспечивает защиту передаваемой или хранимой информации.

Рассмотрим пример. Пусть задан алфавит *V* = {0,1, .., 9, <пробел>, A, B, .., Z, a, b, .., z, А, Б, .., Я, а, б, .., я}, состоящий из 129 элементов (10 цифр, пробел, по 26 строчных и прописных букв латинского алфавита, по 33 строчных и прописных букв русскаго алфавита). Вместо обычного ASCIIкода будем использовать в качестве кода символа его позицию в приведенном выше перечислении: цифры будут иметь коды от 0 до 9, пробел - 10, латинская “A” - 11, латинская “a” - 37, и т.д. до русской “я” с кодом 128. В качестве параметров датчика ПСЧ выберем нечетное *C*=13 и *A*=17 (такое, что *A mod 4 = 1*). Выберем произвольное порождающее число *T(0)*, например, 22. Выберем *B*=129; разумеется, получаемая последовательность будет иметь период повторения существенно меньший, чем у последовательности, получаемой при *B*=232, однако для удобства и в демонстрационных целях мы примем именно такое значение.

Допустим, что необходимо зашифровать открытый текст X = “Произвольный блок открытого текста”. Используя указанную выше кодировку, получим следующую последовательность:

79 113 111 105 104 98 111 108 125 110 124 106 10 97

108 111 107 10 111 115 107 113 124 115 111 99 111 10

115 101 107 114 115 96,

где код 79 соответствует символу “П”, код 113 - символу “р”, и т.д. Длина этого открытого текста составляет 34 символа.

Используя порождающее число 22 и указанные параметры датчика ПСЧ, получим 34 элемента гаммы:

22 0 13 105 121 6 115 33 58 96 97 114 16 27 85 39

31 24 34 75 127 108 43 99 19 78 49 72 76 15 10 54 28

102

Теперь наложим полученную гамму на открытый текст, используя операцию сложения по модулю 129 (так как в нашем алфавите 129 элементов). Другими словами, если сумма *i*-го символа открытого текста и *i*го элемента гаммы будет превосходить 128, то из полученного числа следует вычесть модуль - 129. В результате этой операции мы получим следующий шифртекст:

101 113 124 81 96 104 97 12 54 77 92 91 26 124 64

21 9 34 16 61 105 92 38 85 1 48 31 82 62 116 117 39 14

69, или: “ерыСазбBrНЬЫPыБK9XFyиЬbХ1lUТzуфcDЁ”.

Для дешифрации надо вычесть ту же гамму из шифртекста; при этом, если разность *i*-го символа шифртекста и *i*-го элемента гаммы даст отрицательное число, то к нему следует прибавить 129:

79 113 111 105 104 98 111 108 125 110 124 106 10 97

108 111 107 10 111 115 107 113 124 115 111 99 111 10

115 101 107 114 115 96,

что соответствует исходному тексту.

Слабым местом линейного шифрования является простота определения гаммы, а затем, если необходимо, то и закономерностей ее генерации, по открытому тексту и криптограмме: очевидно, что из Y=XK вытекает K=XY.

**Практическая часть**

# Индивидуальное задание

Алфавит задается при помощи расширенной таблицы ASCII (размер -

256 элементов, с кодами от 0 до 255), параметр *B*=256. Параметры датчика ПСЧ из нижеприведенной таблицы.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Задание |  |  |
| № п.п. | *A* | *C* | *T(0)* |
| 6 | 5 | 51 | 13 |

# Текст программы, реализующей индивидуальное задание

int A = 25, C = 37, T0 = 7, B = 256;

Console.Write("Введите строку открытого текста: ");

string strNach = Console.ReadLine();

string encryptedText = Encrypt(strNach, A, C, T0, B, out int[] gamma, out int[] kodascii, out int[] encryptedCodes);

Console.WriteLine("\nШифртекст: " + encryptedText + " ");

string decryptedText = Decrypt(encryptedText, A, C, T0, B, gamma, out int[] decryptedCodes);

Console.WriteLine("Дешифрованный текст: " + decryptedText + " ");

Console.WriteLine("\nТаблица шифрования/дешифрования:");

Console.WriteLine("---------------------------------------------------------------------------------");

Console.WriteLine("| Индекс | Символ | Код ASCII | Гамма | Код XOR | Символ (шифр.) |");

Console.WriteLine("-------------------------------------------------------------------------------");

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

{

    char encryptedChar = encryptedText[i];

    string encryptedCharDisplay = IsPrintableChar(encryptedCodes[i])

        ? encryptedChar.ToString()

        : $"0x{encryptedCodes[i]:X2}";

    Console.WriteLine($"| {i,6} | {strNach[i],6} | {kodascii[i],9} | {gamma[i],5} | {encryptedCodes[i],7} | {encryptedCharDisplay,15}|");

}

Console.WriteLine("-------------------------------------------------------------------");

static bool IsPrintableChar(int code)

{

    return code >= 32 && code <= 126;

}

static string Encrypt(string text, int A, int C, int T0, int B, out int[] gamma, out int[] kodascii, out int[] XOR)

{

    int numChar = text.Length;

    kodascii = new int[numChar];

    gamma = new int[numChar];

    XOR = new int[numChar];

    string encrypted = "";

    for (int i = 0; i < numChar; i++)

    {

        kodascii[i] = (int)text[i];

    }

    gamma[0] = T0;

    for (int i = 1; i < numChar; i++)

    {

        gamma[i] = (A \* gamma[i - 1] + C) % B;

    }

    for (int i = 0; i < numChar; i++)

    {

        XOR[i] = (kodascii[i] + gamma[i]) % 256;

    }

    for (int i = 0; i < numChar; i++)

    {

        encrypted += (char)XOR[i];

    }

    return encrypted;

}

static string Decrypt(string text, int A, int C, int T0, int B, int[] gamma, out int[] decryptedCodes)

{

    int numChar = text.Length;

    int[] kodascii = new int[numChar];

    decryptedCodes = new int[numChar];

    string decrypted = "";

    for (int i = 0; i < numChar; i++)

    {

        kodascii[i] = (int)text[i];

    }

    for (int i = 0; i < numChar; i++)

    {

        decryptedCodes[i] = (kodascii[i] - gamma[i] + 256) % 256;

    }

    for (int i = 0; i < numChar; i++)

    {

        decrypted += (char)decryptedCodes[i];

    }

    return decrypted;

}

# Пример открытого текста и соответствующей ему шифрограммы

**Открытый текст**: Hello, World!

**Шифртекст**: O9EÂúä=QþM4

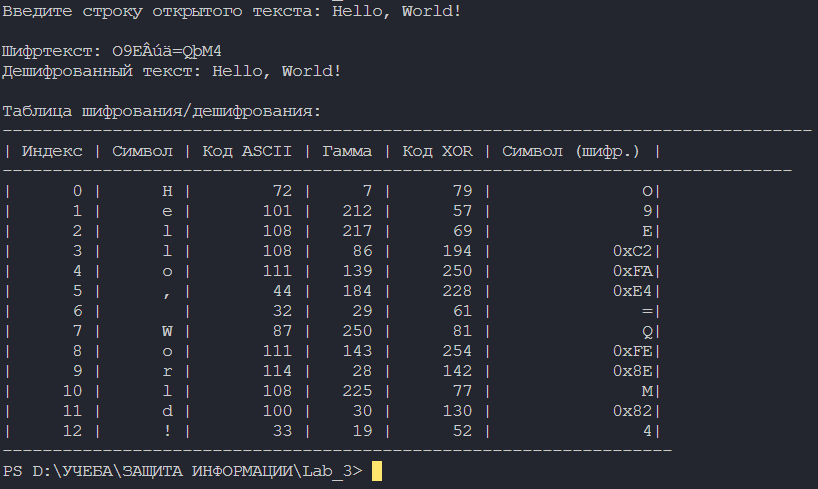
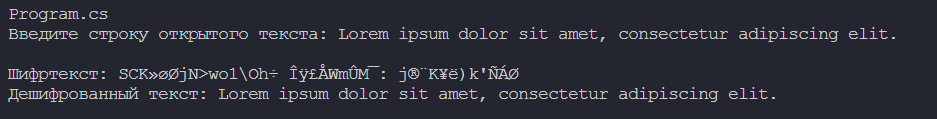


Рисунок 1 –Пример 1 работы программы



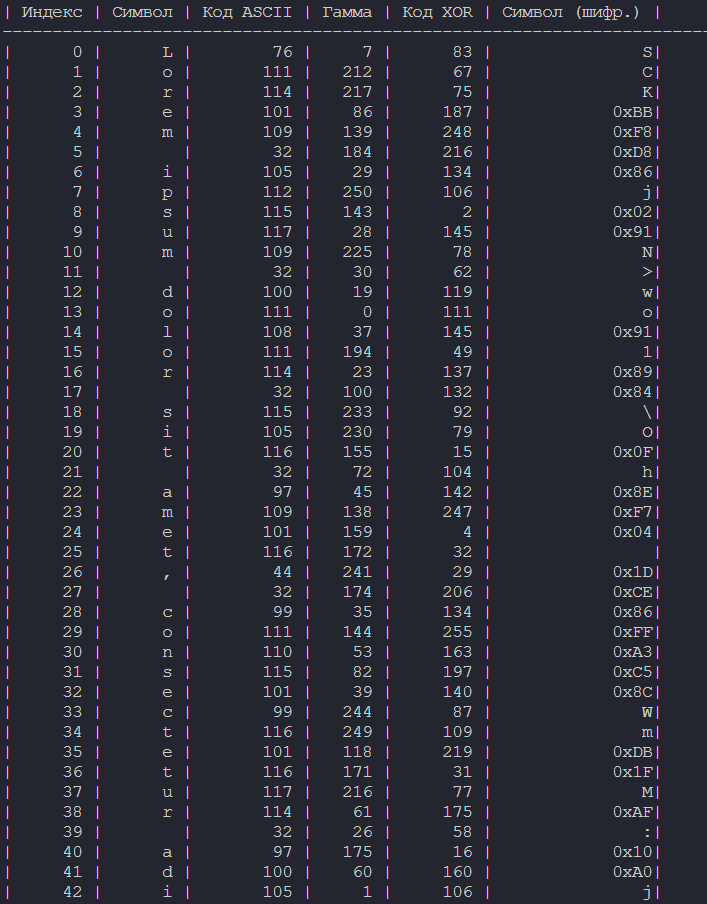


Рисунок 2 – Пример 2 работы программы

# Вывод

Ознакомился с классическим криптографическим алгоритмом – алгоритмом шифрования данных при помощи перестановки.