Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Дисциплина: «Защита информации»

Профиль: «Разработка программно-информационных систем»

Семестр 5

ОТЧЕТ

по лабораторной работе №8

Тема: «Методы гаммирования»

Выполнил: студент группы РИС-22-1б

Баяндин К. С. \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель кафедры ИТАС

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_

Пермь, 2024

**ЦЕЛЬ РАБОТЫ**

Получить практические навыки по шифрованию методом гаммирования.

**ЗАДАНИЕ**

Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 32 бита и используя в алгоритме шифрования операцию умножения по модулю 2N, где за N взять число блоков открытого текста. (7, 19)

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**Шифрование методом гаммирования**

Под гаммированием понимают процесс наложения по определенному закону гаммы шифра на открытые данные. Гамма шифра - это псевдослучайная последовательность, выработанная по заданному алгоритму для шифрования открытых данных и расшифрования зашифрованных данных.

Процесс шифрования заключается в генерации гаммы шифра и наложении полученной гаммы на исходный открытый текст обратимым образом, например, с использованием операции сложения по модулю 2.

Следует отметить, что перед шифрованием открытые данные разбивают на блоки  одинаковой длины, обычно по 64 бита. Гамма шифра вырабатывается в виде последовательности блоков  аналогичной длины.

Уравнение шифрования можно записать в виде



где  - *i*-й блок шифртекста; - *i*-й блок гаммы шифра; - *i*-й блок открытого текста; *М* - количество блоков открытого текста.

Процесс расшифрования сводится к повторной генерации гаммы шифра и наложению этой гаммы на зашифрованные данные. Уравнение расшифрования имеет вид обратный вид.

Получаемый этим методом шифртекст достаточно труден для раскрытия, поскольку теперь ключ является переменным. По сути дела, гамма шифра должна изменяться случайным образом для каждого шифруемого блока. Если период гаммы превышает длину всего шифруемого текста и злоумышленнику неизвестна никакая часть исходного текста, то такой шифр можно раскрыть только прямым перебором всех вариантов ключа. В этом случае криптостойкость шифра определяется длиной ключа.

**«Разоблачение» гаммирования.**

С точки зрения теории криптоанализа метод шифрования однократной случайной равновероятной гаммой той же длины, что и открытый текст, является невскрываемым (далее для краткости авторы будут употреблять термин «однократное гаммирование», держа в уме все вышесказанное). Обоснование, которое привел Шеннон, основываясь на введенном им же понятии информации, не дает возможности усомниться в этом - из-за равных априорных вероятностей криптоаналитик не может сказать о дешифровке, верна она или нет. Кроме того, даже раскрыв часть сообщения, дешифровщик не сможет хоть сколько-нибудь поправить положение - информация о вскрытом участке гаммы не дает информации об остальных ее частях.

Логично было бы предположить, что для организации канала конфиденциальной связи в открытых сетях следовало бы воспользоваться именно схемой шифрования однократного гаммирования. Ее преимущества вроде бы очевидны. Есть, правда, один весомый недостаток, который сразу бросается в глаза, - это необходимость иметь огромные объемы данных, которые можно было бы использовать в качестве гаммы. Для этих целей обычно пользуются датчиками настоящих случайных чисел (в западной литературе аналогичный термин носит название True Random Number Generator или TRNG). Это уже аппаратные устройства, которые по запросу выдают набор случайных чисел, генерируя их с помощью очень большого количества физических параметров окружающей среды. Статистические характеристики таких наборов весьма близки к характеристикам "белого шума", что означает равновероятное появление каждого следующего числа в наборе. А это, в свою очередь, означает для нас действительно равновероятную гамму.

К сожалению, для того чтобы организовать конфиденциальный канал передачи данных, потребуется записать довольно большое количество этих данных и обменяться ими по секретному каналу. Уже одно это условие делает однократное гаммирование во многих случаях неприемлемым. В самом деле, зачем передавать что-то по открытому незащищенному каналу, когда есть возможность передать все это по секретному защищенному? И хотя на простой вопрос, является ли метод использования однократной случайной равновероятной гаммы стойким к взлому, существует положительный ответ, его использование может оказаться попросту невозможным.

Да и к тому же метод однократного гаммирования криптостоек только в определенных, можно даже сказать, тепличных условиях. Что же касается общего случая, то все не так просто.

Показать слабости шифра однократного гаммирования можно, говоря наукообразно, с помощью примера или, что называется, "на пальцах". Представим следующую ситуацию.

Допустим, в тайной деловой переписке используется метод однократного наложения гаммы на открытый текст.

Чтобы дальнейшие рассуждения были как можно более понятны, рассмотрим следующее свойство шифротекста. Предположим, что мы знаем часть гаммы, которая была использована для шифрования текста «Приветствую, мой ненаглядный сосед!» в формате ASCII, кодировка WIN-1251.

П р и в е т с т в у ю , \_ м о й

CF F0 E8 E2 E5 F2 F1 F2 E2 F3 FE 2C 20 EC EE E9

\_ н е н а г л я д н ы й \_ с о с

20 ED E5 ED E0 E3 EB FF E4 ED FB E9 20 F1 EE F1

е д !

E5 E4 21

**ХОД РАБОТЫ**

Для работы был выбран язык C# и фреймворк Windows Forms.

Были созданы: окно для ввода сообщения, кнопка шифровать и вывод полученного результата.

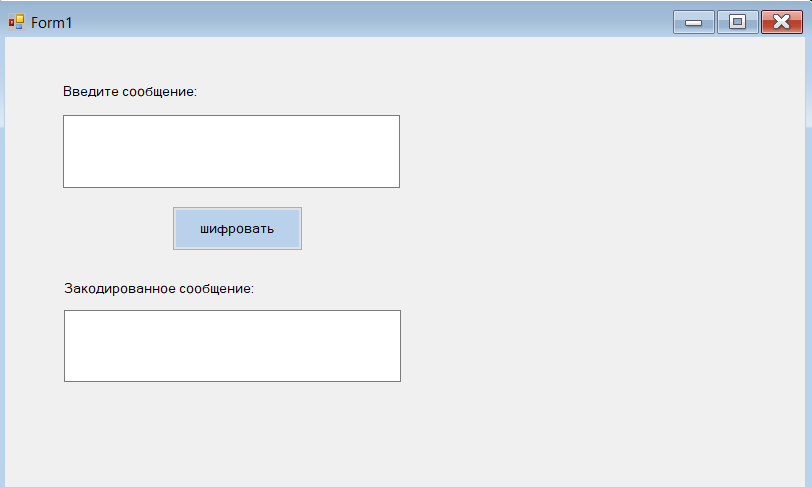


Рисунок 1 – Главная форма программы.

Результат работы представлен на рисунке 2.

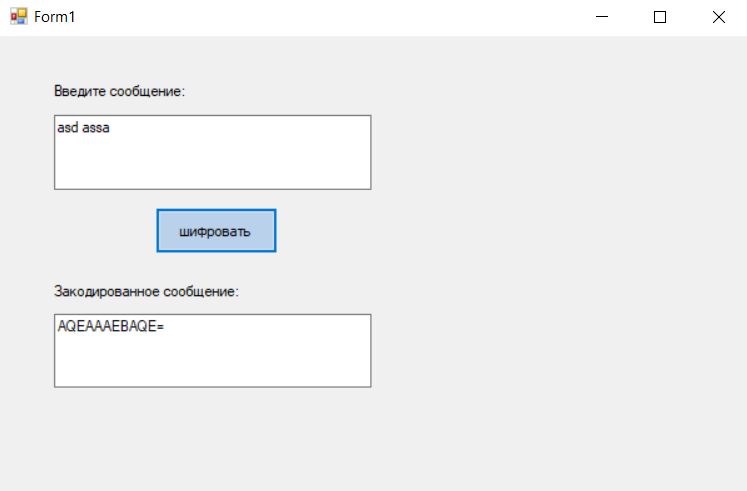


Рисунок 2 – Результат работы программы

Весь алгоритм представлен в классе *Gamming*. Задается размер блока в байтах – 4 (32 бита) и ключ, который должен быть не меньше вводимого сообщения (в случае если он все-таки меньше, то к ключу добавляются 0 в конец). Метод шифрования разбивает введенное сообщение на блоки и обращается к методу *MultiplyBlocksModulo2N*. В данном методе реализована операция умножения по модулю 2N, где за N взято число блоков открытого текста.

**ЛИСТИНГ ПРОГРАММЫ**

**Class Gamming.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace lab8

{

public class Gamming

{

private const int BlockSize = 4; // Размер блока в байтах (32 бита)

List<byte[]> messageBlocks; // сообщение, разделенное на блоки

public string Encrypt(string message, string key)

{

// Преобразование сообщения и ключа в байтовые массивы

byte[] messageBytes = Encoding.ASCII.GetBytes(message);

byte[] keyBytes = Encoding.ASCII.GetBytes(key);

byte[] paddedKeyBytes = messageBytes;

// Проверка, что длина ключа равна или больше длины сообщения

if (keyBytes.Length < messageBytes.Length)

{

//throw new ArgumentException("Ключ должен быть не короче сообщения.");

// Дополнение ключа нулями, если он короче сообщения

paddedKeyBytes = new byte[messageBytes.Length];

Array.Copy(keyBytes, paddedKeyBytes, keyBytes.Length);

}

// Разделение сообщения и ключа на блоки

messageBlocks = SplitToBlocks(messageBytes);

List<byte[]> keyBlocks = SplitToBlocks(paddedKeyBytes);

// Шифрование сообщения блоками

List<byte[]> encryptedBlocks = new List<byte[]>();

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

{

encryptedBlocks.Add(MultiplyBlocksModulo2N(messageBlocks[i], keyBlocks[i], messageBlocks.Count));

}

// Объединение зашифрованных блоков в единый массив

byte[] encryptedBytes = encryptedBlocks.SelectMany(b => b).ToArray();

// Преобразование зашифрованного сообщения в строку Base64

return Convert.ToBase64String(encryptedBytes);

}

public string Decrypt(string encryptedMessage, string key)

{

// Преобразование зашифрованного сообщения из Base64 в байтовый массив

byte[] encryptedBytes = Convert.FromBase64String(encryptedMessage);

byte[] keyBytes = Encoding.ASCII.GetBytes(key);

// Проверка, что длина ключа равна или больше длины сообщения

if (keyBytes.Length < encryptedBytes.Length)

{

throw new ArgumentException("Ключ должен быть не короче сообщения.");

}

// Дополнение ключа нулями, если он короче сообщения

byte[] paddedKeyBytes = new byte[encryptedBytes.Length];

Array.Copy(keyBytes, paddedKeyBytes, keyBytes.Length);

// Разделение зашифрованного сообщения и ключа на блоки

List<byte[]> encryptedBlocks = SplitToBlocks(encryptedBytes);

List<byte[]> keyBlocks = SplitToBlocks(paddedKeyBytes);

// Дешифрование сообщения блоками

List<byte[]> decryptedBlocks = new List<byte[]>();

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

{

decryptedBlocks.Add(MultiplyBlocksModulo2N(encryptedBlocks[i], keyBlocks[i], messageBlocks.Count));

}

// Объединение расшифрованных блоков в единый массив

byte[] decryptedBytes = decryptedBlocks.SelectMany(b => b).ToArray();

// Преобразование расшифрованного сообщения в строку

return Encoding.ASCII.GetString(decryptedBytes);

}

private List<byte[]> SplitToBlocks(byte[] bytes)

{

List<byte[]> blocks = new List<byte[]>();

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

{

blocks.Add(bytes.Skip(i).Take(BlockSize).ToArray());

}

return blocks;

}

private byte[] MultiplyBlocksModulo2N(byte[] block1, byte[] block2, int N)

{

if (block1.Length != block2.Length)

{

throw new ArgumentException("Блоки должны быть одинаковой длины.");

}

byte[] result = new byte[block1.Length];

for (int i = 0; i < block1.Length; i++)

{

result[i] = (byte)((block1[i] \* block2[i]) % (2 \* N)); // Умножение по модулю 2N

}

return result;

}

}

}