МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Допущен к защите

И.о. заведующего кафедрой ПМИ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Разова

**Поточный шифр**

Курсовой проект

Выполнил студент группы ФИб-3302-51-00   / Д. А. Савин /

Руководитель доцент кафедры ПМИ   / Д. Ю. Ляпунов /

Работа защищена с оценкой     \_\_\_\_.\_\_\_\_.2020 г.

Члены комиссии:     /     /

    /     /

Киров 2020

Содержание

[Введение 3](#_Toc62119571)

[1 Анализ предметной области 4](#_Toc62119572)

[1.1 Основные понятия и определения 4](#_Toc62119573)

[1.2 Потоковые алгоритмы шифрования 5](#_Toc62119574)

[1.3 Выводы по разделу 11](#_Toc62119575)

[2 Программная реализация алгоритмов потокового шифрования 12](#_Toc62119576)

[2.1 Программная реализация алгоритма RC4 12](#_Toc62119577)

[2.2 Программная реализация алгоритма ISAAC 15](#_Toc62119578)

[2.3 Экспериментальное исследование реализованных алгоритмов на различных наборах данных 17](#_Toc62119579)

[2.3.1 Тестирование программы RC4 17](#_Toc62119580)

[2.3.2 Тестирование программы ISAAC 18](#_Toc62119581)

[2.4 Выводы по разделу 18](#_Toc62119582)

[Заключение 19](#_Toc62119583)

[Библиографический список 20](#_Toc62119584)

[Приложения 21](#_Toc62119585)

[Приложение А. Листинг программы RC4 21](#_Toc62119586)

[Приложение Б. Листинг программы ISAAC 24](#_Toc62119587)

# Введение

В мире существует множество информации: текст, изображения, видеофайлы и т. д. Обмен информацией может осуществлять по небезопасным каналам связи, что может приводить к риску утечки информации. Тогда можно защитить эту информацию таким методом, как шифрование.

Проблема исследования – недостаток или отсутствие сравнительных исследований алгоритмов шифрования.

Целью курсового проекта является сравнительное исследование алгоритмов шифрования.

Для достижения цели необходимо решить следующие задачи.

1. Выполнить обзор существующих алгоритмов шифрования.
2. Выбор алгоритмов для исследования.
3. Программная реализация выбранных алгоритмов.
4. Экспериментальное исследование реализованных алгоритмов.

Курсовой проект включает два раздела. В первом разделе знакомство с предметной областью и обзор алгоритмов. Во втором разделе программная реализация алгоритмов шифрования и сравнивание алгоритмов.

# Анализ предметной области

## Основные понятия и определения

*Шифрование* представляет собой сокрытие информации от неавторизованных лиц с предоставлением в это же время авторизованным пользователям доступа к ней. Пользователи называются авторизованными, если у них есть соответствующий ключ для дешифрования информации. Это очень простой принцип. Вся сложность заключается в том, как реализуется весь этот процесс.

Еще одной важной концепцией, о которой необходимо знать, является то, что целью любой системы шифрования является максимальное усложнение получения доступа к информации неавторизованными лицами, даже если у них есть зашифрованный текст и известен алгоритм, использованный для шифрования. Пока неавторизованный пользователь не обладает ключом, секретность и целостность информации не нарушается.

С помощью шифрования обеспечиваются три состояния безопасности информации.

1. Конфиденциальность. Шифрование используется для *сокрытия информации* от неавторизованных пользователей при передаче или при хранении.
2. Целостность. Шифрование используется для предотвращения изменения информации при передаче или хранении.
3. Идентифицируемость. Шифрование используется для аутентификации источника информации и предотвращения отказа отправителя информации от того факта, что данные были отправлены именно им.

Термины, связанные с шифрованием:

*Обычный текст*. Информация в исходном виде. Также называется открытым текстом.

*Шифрованный текст*. Информация, подвергнутая действию алгоритма шифрования.

*Алгоритм*. Метод, используемый для преобразования *открытого текста* в шифрованный текст.

*Ключ*. Входные данные, посредством которых с помощью алгоритма происходит преобразование *открытого текста* в шифрованный или обратно.

*Шифрование*. Процесс преобразования *открытого текста* в шифр.

*Дешифрование*. Процесс преобразования шифра в *открытый текст*.[1]

## Потоковые алгоритмы шифрования

Прежде чем преступить к детальной классификации шифров, необходимо четко понимать, что в любом шифре используется ключ для шифрования, а вот в зависимости от того какой ключ используется для дешифрования, они будут делиться на симметричные и асимметричные шифры, таким образом что:

– Симметричные шифры используют один и тот же ключ для шифрования и дешифрования;

– Асимметричные шифры используют разные ключи для шифрования и дешифрования.

Таким образом, мы будим классифицировать любой шифр по двум составляющим «блочный \ поточный», «симметричный \ асимметричный».

*Поточный шифр* – это симметричный шифр, в котором каждый символ открытого текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в потоке открытого текста. Поточный шифр реализует другой подход к симметричному шифрованию, нежели блочные шифры. Допустим, что в режиме гаммирования для поточных шифров при передаче по каналу связи произошло искажение одного знака шифротекста. Очевидно, что в этом случае все знаки, принятые без искажения, будут расшифрованы правильно. Произойдёт потеря лишь одного знака текста. А теперь представим, что один из знаков шифротекста при передаче по каналу связи был потерян. Это приведёт к неправильному расшифровыванию всего текста, следующего за потерянным знаком. Практически во всех каналах передачи данных для поточных систем шифрования присутствуют помехи. Поэтому для предотвращения потери информации решают проблему синхронизации шифрования и дешифрования текста.

Таким образом, по способу решения этой проблемы шифросистемы подразделяются на синхронные системы и системы с самосинхронизацией. Стоит заметить, что обе системы довольно часто применяются на практике, однако, по словам экспертов, алгоритмы с самосинхронизирующейся системой наиболее просты и надежны в применении. Классификация поточных шифров приведена на рисунке 1.2.1.

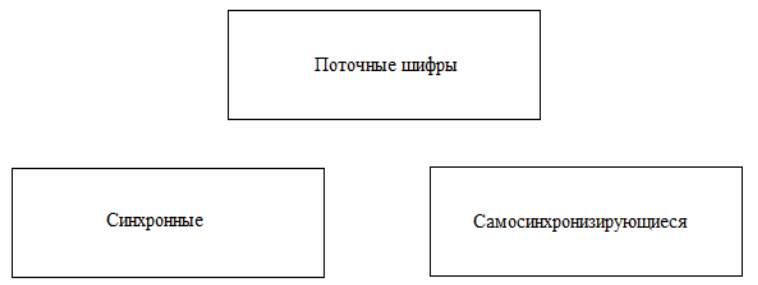


Рисунок (1.2.1) – Классификация поточных шифров.

Синхронные поточные шифры (СПШ) – шифры, в которых поток ключей генерируется независимо от открытого текста и шифротекста. При шифровании генератор потока ключей выдаёт биты потока ключей, которые идентичны битам потока ключей при дешифровании. Потеря знака шифротекста приведёт к нарушению синхронизации между этими двумя генераторами и невозможности расшифровывания оставшейся части сообщения. Очевидно, что в этой ситуации отправитель и получатель должны повторно синхронизоваться для продолжения работы. Обычно синхронизация производится вставкой в передаваемое сообщение специальных маркеров. В результате этого пропущенный при передаче знак приводит к неверному расшифровыванию лишь до тех пор, пока не будет принят один из маркеров. Необходимо заметить, что выполняться синхронизация должна так, чтобы ни одна часть потока ключей не была повторена.

Плюсы СПШ:

– отсутствие эффекта распространения ошибок (только искажённый бит будет расшифрован неверно);

– предохраняют от любых вставок и удалений шифротекста, так как они приведут к потере синхронизации и будут обнаружены.

Минусы СПШ:

– уязвимы к изменению отдельных бит шифрованного текста. Если злоумышленнику известен открытый текст, он может изменить эти биты так, чтобы они расшифровывались, как ему надо.

Самосинхронизирующиеся поточные шифры (асинхронные поточные шифры (АПШ)) – шифры, в которых поток ключей создаётся функцией ключа и фиксированного числа знаков шифротекста. Итак, внутреннее состояние генератора потока ключей является функцией предыдущих N битов шифротекста. Поэтому расшифровывающий генератор потока ключей, приняв N битов, автоматически синхронизируется с шифрующим генератором. Реализация этого режима происходит следующим образом: каждое сообщение 54 начинается случайным заголовком длиной N битов; заголовок шифруется, передаётся и расшифровывается; расшифровка является неправильной, зато после этих N бит оба генератора будут синхронизированы.

Плюсы АПШ:

– Размешивание статистики открытого текста. Так как, каждый знак открытого текста влияет на следующий шифротекст, то статистические свойства открытого текста распространяются на весь шифротекст. Следовательно, АПШ может быть более устойчивым к атакам на основе избыточности открытого текста, чем СПШ.

Минусы АПШ:

– распространение ошибки (каждому неправильному биту шифротекста соответствуют N ошибок в открытом тексте);

– чувствительны к вскрытию повторной передачей [2, c.52-54].

Алгоритм RC4.

Алгоритм RC4, предложенный  Рональдом Ривестом в 1994 году, относится к классу алгоритмов, разработанных специально для потоковых шифров. Генераторы псевдослучайных чисел, построенные с помощью таких алгоритмов, как правило, значительно быстрее генераторов, основанных на блоковых шифрах. Алгоритм RC4 работает с n-битовыми словами (обычно ). Все вычисления проводятся по модулю (остаток вычисляется очень быстро путем выделения младших бит в с помощью логической операции «и»). RC4 использует L-словный ключ и генерирует последовательность слов …, конкретный вид которой определяется ключом . Состояние генератора задается таблицей из слов и двух переменных и . В каждый момент времени таблица содержит все возможные n-битовые числа в перемешанном виде. Так как каждый элемент таблицы принимает значения в промежутке , то его можно трактовать двояко: либо как число, либо как номер другого элемента в таблице. Секретный ключ задает только начальное перемешивание чисел в таблице, которое формируется с помощью следующего алгоритма:

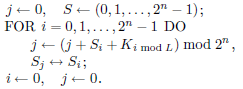


Рисунок (1.2.2)

После этого генератор готов к работе. Генерация очередного псевдослучайного слова осуществляется следующим образом: [3, c.184]

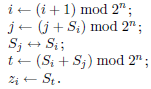


Рисунок (1.2.3)

Шифрование по этому алгоритму примерно в 10 раз быстрее, чем шифрование DES при программной реализации [4, c.44].

Компания RSA Data Security объявила, что шифр обладает иммунитетом к методам линейного и дифференциального криптоанализа, высоко не линеен и непохоже, чтобы у него были короткие циклы [5, c.262].

Алгоритм ISAAC.

ISAAC –­ это ориентированный на криптографические приложения генератор псевдослучайных чисел из семейства генераторов, разработанных в 1993-1996 годах молодым американским программистом Робертом Дженкинсом. Концептуально схема напоминает алгоритм RC4 компании RSA.

В отличие от большинства генераторов псевдослучайных чисел, в основе работы которых лежат потоковые шифры, ISAAC разработан без использования линейных регистров сдвига с обратной связью.

Среднее количество машинных инструкций, требуемых для получения 32-битного значения — 18,75. 64-битная версия ISAAC (ISAAC-64) требует 19 инструкций для получения одного 64-битного значения.

При проектировании ISAAC к нему был предъявлен следующий список требований:

* криптографическая стойкость;
* невозможность получения внутреннего состояния по имеющимся выходным результатам;
* отсутствие коротких циклов;
* отсутствие каких-либо тенденций в распределении бит на всем цикле;
* упорядоченные состояния должны быстро становиться хаотичными.

Алгоритм работы

Так же, как и в предыдущих алгоритмах, в ISAAC есть массив , определяющий внутреннее состояние системы, так же состоящий из случайно расположенных в массиве элементов от *0* до длины  бит, итератор i и три переменные ,  и , отвечающие за предыдущие состояния генератора, массив выходных данных  той же длины, что и . Однако помимо этих переменных здесь вводятся также переменные, которые определяют значение функции, зависящей от обоих итераторов:

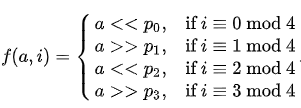


Рисунок (1.2.4)

## Выводы по разделу

Таким образом, существуют синхронные и асинхронные поточные шифры. Характерными особенностями поточного шифрования являются высокая производительность, отсутствие эффекта размножения ошибок, малая уязвимость, благодаря чему оно применяется при шифровании информации в каналах связи.

# Программная реализация алгоритмов потокового шифрования

## Программная реализация алгоритма RC4

Программную реализацию алгоритма было решено писать на языке C#, в Microsoft Visual Studio 2019.

Программа состоит из класса *RC4*, функций класса, а также функций: шифрования, дешифрования, генератора псевдослучайной последовательности.

Процесс шифрования можно описать таким образом: сначала идет инициализация S-блока: алгоритм использует ключ, подаваемый на вход пользователем, сохранённый в , и имеющий длину  байт. Инициализация начинается с заполнения массива , далее этот массив перемешивается путём перестановок, определяемых ключом. Так как только одно действие выполняется над , то должно выполняться утверждение, что  всегда содержит один набор значений, который был дан при первоначальной инициализации ( *i*).

public void init(byte[] key)

{

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

{

S[i] = (byte)i;

}

j = 0;

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

{

j = (j + S[i] + key[i % key.Length]) % 256;

swap(S, i, j);

}

i = j = 0;

}

Генератор ключевого потока RC4 переставляет значения, хранящиеся в S. В одном цикле RC4 определяется одно *n*-битное слово K из ключевого потока:

public byte kword()

{

i = (i + 1) % 256;

j = (j + S[i]) % 256;

swap(S, i, j);

byte K = S[(S[i] + S[j]) % 256];

return K;

}

В дальнейшем ключевое слово будет сложено по модулю два с исходным текстом, которое пользователь хочет зашифровать, и получен зашифрованный текст:

public byte[] code()

{

byte[] data = text.Take(text.Length).ToArray();

byte[] res = new byte[data.Length];

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

{

res[i] = (byte)(data[i] ^ kword());

}

return res;

}

Алгоритм шифрования.

1. Функция генерирует последовательность битов.
2. Затем последовательность битов посредством операции «суммирование по модулю два» (xor) объединяется с открытым текстом. В результате получается шифрограмма.

Алгоритм расшифровки.

1. Повторно создаётся (регенерируется) поток битов ключа (ключевой поток).
2. Поток битов ключа складывается с шифрограммой {\displaystyle c\_{i}}операцией «[xor](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2)». В силу свойств операции «[xor](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2)» на выходе получается исходный (незашифрованный) текст.

Программная реализация также состоит из функций считывания из файла и записи в него:

Чтение из файла:

public Byte[] ReadByteArrayFromFile(string fileName)

{

Byte[] buffer = null;

try

{

FileStream fs = new FileStream(fileName, FileMode.Open, FileAccess.Read);

BinaryReader br = new BinaryReader(fs);

long numBytes = new FileInfo(fileName).Length;

buffer = br.ReadBytes((int)numBytes);

br.Close();

fs.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return buffer;

}

}

Запись в файл:

public void WriteByteArrayToFile(Byte[] buffer, string fileName)

{

try

{

FileStream fs = new FileStream(fileName, FileMode.Create, FileAccess.ReadWrite);

BinaryWriter bw = new BinaryWriter(fs);

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

bw.Write(buffer[i]);

bw.Close();

fs.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

## Программная реализация алгоритма ISAAC

Программную реализацию алгоритма было решено писать на языке C#, в Microsoft Visual Studio 2019.

Программа состоит из функций шифрования/дешифрования, генератора псевдослучайной последовательности, а также вспомогательных процедур для ввода или вывода в файл.

Функция шифрования/дешифрования:

Инициализируем вспомогательные переменные n, l, а также массив, в котором будет содержаться зашифрованный текст, и операцией XOR шифруем исходную информацию, содержащуюся в переменной msg типа string.

static byte[] code(string msg)

{

int n, l;

byte[] v = new byte[msg.Length];

l = msg.Length;

// XOR message

for (n = 0; n < l; n++)

{

v[n] = (byte)(RandA() ^ (byte)msg[n]);

}

return v;

}

Генератор псевдослучайной последовательности:

static byte RandA()

{

return (byte)(Random() % 95 + 32);

}

static uint Random()

{

uint result = randrsl[randcnt];//заполнен случайными числами

randcnt++;

if (randcnt > 255)

{

isaac(); randcnt = 0;

}

return result;

}

Вспомогательная функция чтения из файла:

public static string ReadFromFile(string fileName)

{

string textFromFile="";

try

{

FileStream fstream = File.OpenRead(fileName);

// преобразуем строку в байты

byte[] array = new byte[fstream.Length];

// считываем данные

fstream.Read(array, 0, array.Length);

// декодируем байты в строку

textFromFile = System.Text.Encoding.Default.GetString(array);

fstream.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return textFromFile;

}

Процедура записи в файл:

public static void WriteByteArrayToFile(Byte[] buffer, string fileName)

{

try

{

FileStream fs = new FileStream(fileName, FileMode.Create, FileAccess.ReadWrite);

BinaryWriter bw = new BinaryWriter(fs);

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

{

bw.Write(buffer[i]);

}

bw.Close();

fs.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

## Экспериментальное исследование реализованных алгоритмов на различных наборах данных

Экспериментальное исследование алгоритмов RC4 и ISAAC проводилось на ПК с такими характеристиками:

Процессор: Intel Pentium CPU G4600 3.60GHz 2 ядра 4 потока, Кеш-память: 3MB

ОЗУ: 8,00 ГБ

Система: Windows 10 Pro x64

Данные тестирования занесены в таблицы 2.3.1.1, 2.3.2.1, которые состоят из:

1) Тип файла

2) Размер файла

3) Время на шифрование

4) Время на дешифрование

### Тестирование программы RC4

|  |  |  |  |
| --- | --- | --- | --- |
| Тип файла | Размер файла(MB) | Время на шифрование (с) | Время на дешифрование (с) |
| .txt | 492 | 13.88 | 13.43 |
| .png | 11.9 | 0.36 | 0.37 |
| .gif | 2.03 | 0.07 | 0.07 |
| .jpg | 2.97 | 0.10 | 0.10 |
| .pdf | 204 | 6.60 | 6.79 |

Таблица 2.3.1.1

### Тестирование программы ISAAC

|  |  |  |  |
| --- | --- | --- | --- |
| Тип файла | Размер файла(MB) | Время на шифрование (с) | Время на дешифрование (с) |
| .txt | 492 | 10.95 | 11.06 |
| .png | 11.9 | 0.30 | 0.25 |
| .gif | 2.03 | 0.05 | 0.04 |
| .jpg | 2.97 | 0.08 | 0.06 |
| .pdf | 204 | 5.46 | 5.21 |

Таблица 2.3.2.1

## Выводы по разделу

Как видно из таблиц, работа программы с алгоритмом ISAAC чуть быстрее, чем программа с алгоритмом RC4.

.txt – 10.95 секунд против 13.88 секунды на шифрование и 11.06 против 13.43 на дешифрование;

.pdf – 5.46 секунд против 6.60 секунды на шифрование и 5.21 против 6.79 на дешифрование;

Из этих двух алгоритмов логичнее использовать алгоритм ISAAC.

# Заключение

При разработке приложения стало ясно, что защита информации важна чтобы решить эту проблему были разобраны алгоритмы потокового шифрования.

Разработанные приложения имеют вид классического консольного приложения, поэтому файлы для сжатия должны находиться в одном месте с приложением. В дальнейшем планируется добавить возможность выбора места сохранения, а также выбор файла в независимости от его расположения.

При помощи данного курсового проекта были получены навыки работы с информацией, а также навыки работы с C#.

# Библиографический список

* 1. Мэйволд, Э. Безопасность сетей / Мэйволд Э. - Москва : Национальный Открытый Университет "ИНТУИТ", 2016. (Шаг за шагом) - ISBN 5-9570-0046-9.
  2. Гатченко Н.А., Исаев А.С., Яковлев А.Д. «Криптографическая защита информации» – СПб: НИУ ИТМО, 2012.
  3. Рябко Б. Я., Фионов А. Н. Р98 Криптографические методы защиты информации: учебное пособие. М.: Горячая линия Телеком, 2005. 229 с. ISBN 5-89176-233-1
  4. Криптографическая защита информации: учебное пособие / А.В.Яковлев, А.А. Безбогов, В.В. Родин, В.Н. Шамкин. – Тамбов : Изд-во Тамб. гос. техн. ун-та, 2006. – 140 с. – 100 экз. – ISBN 5-8265-0503-6.
  5. ПОТОЧНЫЕ ШИФРЫ Результаты зарубежной открытой криптологии Москва 1997.
  6. Р. Дж. ISAAC (англ.) // Lecture Notes in Computer Science. — Fast Software Encryption, 1996. — Vol. 1039. — P. 41—49
  7. Шнайер Б. RC4 // Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — С. 236. — 816 с. — ISBN 5-89392-055-4..
  8. Пол Г., Сабэмой М. Потоковый шифр RC4 // Потоковый шифр RC4 и его варианты = RC4 Stream Cipher and Its Variants. — Бока-Ратон: CRC Press, 2001. — С. 16—19. — 285 с.

# Приложения

## Приложение А. Листинг программы RC4

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Diagnostics;

using System.Threading;

namespace RC4

{

class Program

{

public class RC4

{

public byte[] text; //текст для шифрования/расшифрования

private byte[] S = new byte[256];

private int i = 0;

private int j = 0;

//для удобства обмена

private void swap(byte[] array, int ind1, int ind2)

{

byte temp = array[ind1];

array[ind1] = array[ind2];

array[ind2] = temp;

}

//инициализация, алгоритм ключевого расписания

public void init(byte[] key)

{

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

{

S[i] = (byte)i;

}

j = 0;

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

{

j = (j + S[i] + key[i % key.Length]) % 256;

swap(S, i, j);

}

i = j = 0;

}

//генератор псевдослучайной последовательности

public byte kword()

{

i = (i + 1) % 256;

j = (j + S[i]) % 256;

swap(S, i, j);

byte K = S[(S[i] + S[j]) % 256];

return K;

}

//функция шифрования/дешифрования

public byte[] code()

{

byte[] data = text.Take(text.Length).ToArray();

byte[] res = new byte[data.Length];

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

{

res[i] = (byte)(data[i] ^ kword());

}

return res;

}

//бинарная запись в файл

public void WriteByteArrayToFile(Byte[] buffer, string fileName)

{

try

{

FileStream fs = new FileStream(fileName, FileMode.Create, FileAccess.ReadWrite);

BinaryWriter bw = new BinaryWriter(fs);

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

bw.Write(buffer[i]);

bw.Close();

fs.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

//бинарное чтение из файла

public Byte[] ReadByteArrayFromFile(string fileName)

{

Byte[] buffer = null;

try

{

FileStream fs = new FileStream(fileName, FileMode.Open, FileAccess.Read);

BinaryReader br = new BinaryReader(fs);

long numBytes = new FileInfo(fileName).Length;

buffer = br.ReadBytes((int)numBytes);

br.Close();

fs.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return buffer;

}

}

static void Main(string[] args)

{

Stopwatch stopWatch = new Stopwatch();

string elapsedTime = "";

TimeSpan ts;

Console.WriteLine("Выберите действие:");

Console.WriteLine(" 1. Шифровать");

Console.WriteLine(" 2. Дешифровать");

Console.WriteLine(" 3. Выход");

Console.Write(" \n\n>>> ");

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

int key = 0;

// Console.Write("Введите ключ: ");

string query = "key";

string outfile = "";

int.TryParse(query, out key);

byte[] bytekey = BitConverter.GetBytes(key);

Console.Write("Входной файл: ");

string infile = Console.ReadLine();

if (menu == 1)

{

Console.WriteLine("Результирующий файл: " + infile + ".rc4");

outfile = infile + ".rc4";

}

else

{

if (menu==2)

{

string[] s1 = infile.ToCharArray().Select(c => c.ToString()).ToArray();

string[] s2= new string[s1.Length];

int count = 0;

s2[0] = s1[0];

for (int i=1; i<infile.Length; i++)

{

if (s1[i]==".")

{

if (count == 0)

{

count++;

}

else

{

break;

}

}

s2[i] = s1[i];

}

outfile = String.Concat(s2);

Console.WriteLine("Результирующий файл: " + outfile);

}

}

var ob = new RC4();

byte[] arr;

switch (menu)

{

case 1:

ob.text = ob.ReadByteArrayFromFile(infile);

ob.init(bytekey);

stopWatch.Start();

arr = ob.code();

stopWatch.Stop();

ob.WriteByteArrayToFile(arr, outfile);

ts = stopWatch.Elapsed;

Console.WriteLine("Сообщение зашифровано!");

elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}", ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);

Console.WriteLine("RunTime " + elapsedTime);

break;

case 2:

ob.text = ob.ReadByteArrayFromFile(infile);

ob.init(bytekey);

stopWatch.Start();

arr = ob.code();

stopWatch.Stop();

ob.WriteByteArrayToFile(arr, outfile);

ts = stopWatch.Elapsed;

Console.WriteLine("Сообщение расшифровано!");

elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("RunTime " + elapsedTime);

break;

case 3:

Environment.Exit(0);

break;

default:

Console.WriteLine("Такого действия нет!");

break;

}

}

}

}

## Приложение Б. Листинг программы ISAAC

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Diagnostics;

using System.Threading;

namespace ISAAC

{

class Program

{

static uint[] randrsl = new uint[256];

static uint randcnt;

static uint[] mm = new uint[256];

static uint aa = 0, bb = 0, cc = 0;

static void isaac()

{

uint i, x, y;

cc++;

bb += cc;

for (i = 0; i <= 255; i++)

{

x = mm[i];

switch (i & 3)

{

case 0:

aa = aa ^ (aa << 13);

break;

case 1:

aa = aa ^ (aa >> 6);

break;

case 2:

aa = aa ^ (aa << 2);

break;

case 3:

aa = aa ^ (aa >> 16);

break;

}

aa = mm[(i + 128) & 255] + aa;

y = mm[(x >> 2) & 255] + aa + bb;

mm[i] = y;

bb = mm[(y >> 10) & 255] + x;

randrsl[i] = bb;

}

}

// if (flag==TRUE), используем содержимое randrsl[] для инициализации mm[].

static void mix(ref uint a, ref uint b, ref uint c, ref uint d, ref uint e, ref uint f, ref uint g, ref uint h)

{

a = a ^ b << 11;

d += a; b += c;

b = b ^ c >> 2;

e += b; c += d;

c = c ^ d << 8;

f += c; d += e;

d = d ^ e >> 16;

g += d; e += f;

e = e ^ f << 10;

h += e; f += g;

f = f ^ g >> 4;

a += f; g += h;

g = g ^ h << 8;

b += g; h += a;

h = h ^ a >> 9;

c += h; a += b;

}

static void Init(bool flag)

{

short i; uint a, b, c, d, e, f, g, h;

aa = 0; bb = 0; cc = 0;

a = 0x9e3779b9; b = a; c = a; d = a;

e = a; f = a; g = a; h = a;

for (i = 0; i <= 3; i++)

mix(ref a, ref b, ref c, ref d, ref e, ref f, ref g, ref h);

i = 0;

do

{ // заполняем mm[]

if (flag)

{ // используем всю информацию в seed

a += randrsl[i]; b += randrsl[i + 1]; c += randrsl[i + 2]; d += randrsl[i + 3];

e += randrsl[i + 4]; f += randrsl[i + 5]; g += randrsl[i + 6]; h += randrsl[i + 7];

}

mix(ref a, ref b, ref c, ref d, ref e, ref f, ref g, ref h);

mm[i] = a;

mm[i + 1] = b;

mm[i + 2] = c;

mm[i + 3] = d;

mm[i + 4] = e;

mm[i + 5] = f;

mm[i + 6] = g;

mm[i + 7] = h;

i += 8;

}

while (i < 255);

if (flag)

{

// Делаем второй проход, чтобы все seed повлияло на все mm

i = 0;

do

{

a += mm[i]; b += mm[i + 1]; c += mm[i + 2]; d += mm[i + 3];

e += mm[i + 4]; f += mm[i + 5]; g += mm[i + 6]; h += mm[i + 7];

mix(ref a, ref b, ref c, ref d, ref e, ref f, ref g, ref h);

mm[i] = a; mm[i + 1] = b; mm[i + 2] = c; mm[i + 3] = d;

mm[i + 4] = e; mm[i + 5] = f; mm[i + 6] = g; mm[i + 7] = h;

i += 8;

}

while (i < 255);

}

isaac(); // заполняем первый набор результатов

randcnt = 0; // используем первый набор результатов

}

static void Seed(string seed, bool flag)

{

for (int i = 0; i < 256; i++) mm[i] = 0;

for (int i = 0; i < 256; i++) randrsl[i] = 0;

int m = seed.Length;

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

{

randrsl[i] = seed[i];

}

// Инициализация

Init(flag);

}

static uint Random()

{

uint result = randrsl[randcnt];

randcnt++;

if (randcnt > 255)

{

isaac(); randcnt = 0;

}

return result;

}

// Получаем случайный символ в диапазоне ASCII

static byte RandA()

{

return (byte)(Random() % 95 + 32);

}

// XOR шифрование. Вывод: ASCII-байтовый массив

static byte[] code(string msg)

{

int n, l;

byte[] v = new byte[msg.Length];

l = msg.Length;

// XOR message

for (n = 0; n < l; n++)

{

v[n] = (byte)(RandA() ^ (byte)msg[n]);

}

return v;

}

public static string ReadFromFile(string fileName)

{

string textFromFile="";

try

{

FileStream fstream = File.OpenRead(fileName);

// преобразуем строку в байты

byte[] array = new byte[fstream.Length];

// считываем данные

fstream.Read(array, 0, array.Length);

// декодируем байты в строку

textFromFile = System.Text.Encoding.Default.GetString(array);

fstream.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return textFromFile;

}

//бинарная запись в файл

public static void WriteByteArrayToFile(Byte[] buffer, string fileName)

{

try

{

FileStream fs = new FileStream(fileName, FileMode.Create, FileAccess.ReadWrite);

BinaryWriter bw = new BinaryWriter(fs);

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

{

bw.Write(buffer[i]);

}

bw.Close();

fs.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

public static void Main()

{

Stopwatch stopWatch = new Stopwatch();

string elapsedTime="";

TimeSpan ts;

string text, key;

byte[] encrypt, decrypt;

string fileNameOut="";

Console.WriteLine("Выберите действие:");

Console.WriteLine(" 1. Шифровать");

Console.WriteLine(" 2. Дешифровать");

Console.WriteLine(" 3. Выход");

Console.Write(" \n\n>>> ");

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

Console.Write("Входной файл: ");

string fileNameIn = Console.ReadLine();

if (menu == 1)

{

Console.WriteLine("Результирующий файл: " + fileNameIn + ".ic");

fileNameOut = fileNameIn + ".ic";

}

else

{

if (menu == 2)

{

string[] s1 = fileNameIn.ToCharArray().Select(c => c.ToString()).ToArray();

string[] s2 = new string[s1.Length];

int count = 0;

s2[0] = s1[0];

for (int i = 1; i < fileNameIn.Length; i++)

{

if (s1[i] == ".")

{

if (count == 0)

{

count++;

}

else

{

break;

}

}

s2[i] = s1[i];

}

fileNameOut = String.Concat(s2);

Console.WriteLine("Результирующий файл: " + fileNameOut);

}

}

switch (menu)

{

case 1:

text = ReadFromFile(fileNameIn);

key = "key";

stopWatch.Start();

Seed(key, true);

// encrypt

encrypt = code(text);

stopWatch.Stop();

ts = stopWatch.Elapsed;

WriteByteArrayToFile(encrypt, fileNameOut);

Console.WriteLine("Файл зашифрован! " + fileNameOut);

elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("RunTime " + elapsedTime);

break;

case 2:

text = ReadFromFile(fileNameIn);

key = "key";

// decrypt

stopWatch.Start();

Seed(key, true);

decrypt = code(text);

stopWatch.Stop();

ts = stopWatch.Elapsed;

WriteByteArrayToFile(decrypt, fileNameOut);

Console.WriteLine("Файл расшифрован! " + fileNameOut);

elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("RunTime " + elapsedTime);

break;

case 3:

Environment.Exit(0);

break;

default:

Console.WriteLine("Такого действия нет!");

break;

}

}

}

}