МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа № 8**

**ИССЛЕДОВАНИЕ ПОТОКОВЫХ ШИФРОВ**

Разработала: Некрасова А.П.

ФИТ 3 курс 5 группа

Преподаватель: Савельева М.Г.

Минск 2023

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации потоковых шифров.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости потоковых шифров.
2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для потокового зашифрования/расшифрования.
3. Выполнить анализ криптостойкости потоковых шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.
6. **Теоретические сведения**

Потоковый шифр (иногда говорят «поточный») – симметричный шифр, преобразующий каждый символ *mi* открытого текста в символ шифрованного, *ci*, зависящий от ключа и расположения символа в тексте. Термин «потоковый шифр» обычно используется в том случае, когда шифруемые символы открытого текста представляются одной буквой, битом или реже – байтом. Все потоковые шифры делятся на 2 класса: синхронные и асинхронные (или самосинхронизирующиеся).

Синхронные потоковые шифры (СПШ) — шифры, в которых поток ключей генерируется независимо от открытого текста и шифротекста.

Самосинхронизирующиеся потоковые шифры (асинхронные потоковые шифры (АПШ)) — шифры, в которых ключевой поток создаётся функцией ключа и фиксированного числа знаков шифротекста.

Основной задачей потоковых шифров является выработка некоторой последовательности (гаммы) для зашифрования, т.е. выходная гамма является ключевым потоком (ключом) для сообщения.

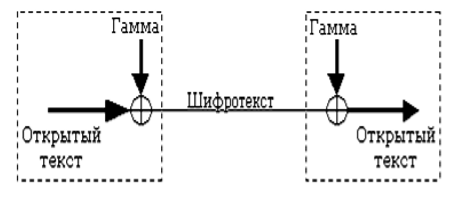


Рисунок 1.1 – Схема потокового шифра

Случайные числа (последовательности) – последовательность элементов, каждый из которых не может быть предсказан (вычислен) только на основе знания предшествующих ему элементов данной последовательности.

Псевдослучайные числа – последовательность элементов, полученная в результате выполнения некоторого алгоритма и используемая в конкретном случае вместо последовательности случайных чисел.

Алгоритм **линейного конгруэнтного генератора**, описывается следующим рекуррентным соотношением:

*xt*+1 = (*a\*xt* + *c*) mod *n* (1)

где а, с и n – константы. Период такого генератора не превышает *n*.

**Генератор ПСП на основе регистров сдвига.** РС с линейной обратной связью (РСЛОС) состоит из двух частей: собственно РС и функции обратной связи. На рис. 1.2 представлена общая схема РС с линейной обратной связью. Функция обратной связи реализуется с помощью сумматоров сложения по модулю два (элементы XOR; на рис. 1.2 обозначены в виде кружочков со знаком сложения).

РСЛОС строятся на основе примитивных порождающих полиномов (многочленов), которые мы подробно анализировали при изучении циклических помехоустойчивых кодов. Если многочлен является неприводимым, то период ПСП при ненулевом начальном условии (ненулевом состоянии) регистра будет максимально возможным: 2*L* – 1.

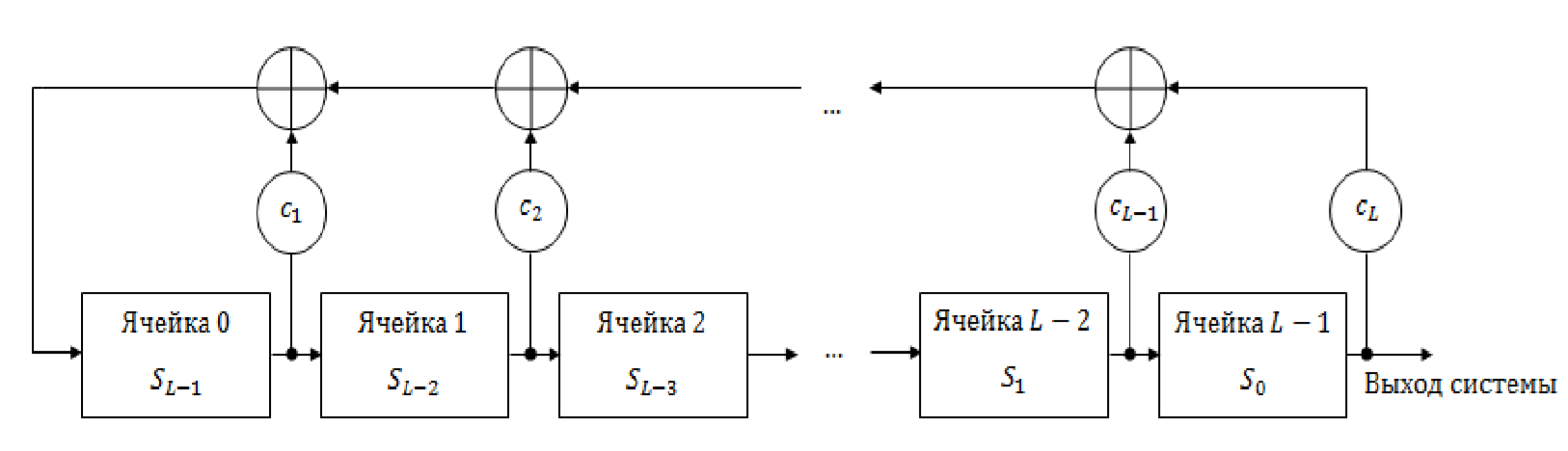


Рисунок 1.2 – Общая схема регистра сдвига с линейной обратной связью

Генератор псевдослучайных чисел на основе **алгоритма RSA** разработан для систем асимметричного зашифрования/расшифрования. Генератор ПСП на основе RSA устроен следующим образом. Последовательность генерируется с использованием соотношения:

*xt* = (*xt*-1)*е* mod *n* (2)

Начальными параметрами служат *n*, большие простые числа *p* и *q* (причем *n = p\*q*), целое число *е*, взаимно простое с произведением (*р – 1) \* ( q –1*), а также некоторое случайное начальное значение, *x0*. Выходом генератора является на *t*-м шаге является младший бит числа *xt*. Безопасность генератора опирается на сложности взлома алгоритма RSA, т. е. на разложении числа *n* на простые сомножители.

**Алгоритм *BBS*** или генератором на основе квадратичных вычетов. Начальное значение *x0* генератора вычисляется на основе соотношения:

*x0*= *x*2 mod *n* (3)

где *n*, как и в генераторе на основе RSA, является произведением простых чисел *p* и *q*, однако в нашем случае эти простые числа должны быть сравнимы с числом 3 по модулю 4, т. е. при делении *p* и *q* на 4 должен получаться одинаковый остаток: 3; число x должно быть взаимно простым с *n*; число *n* называют числом Блюма. Выходом генератора на *t*-м шаге является младший бит числа *xt*:

*xt* = (*xt*-1)2 mod *n* (4)

**Алгоритм RC4**, как и любой потоковый шифр, строится на основе генератора псевдослучайных битов (генератора ПСП). На вход генератора записывается ключ, а на выходе читаются псевдослучайные биты. Длина ключа может составлять от 40 до 2048 бит.

Ядро алгоритма состоит из функции генерации ключевого потока. Другая часть алгоритма – функция инициализации, которая использует ключ переменной длины *Ki* для создания начального состояния генератора ключевого потока.

В основе алгоритма – размер блока или слова, определяемый параметром *n*. Обычно *n* = 8, но можно использовать и другие значения. Внутренне состояние шифра определяется массивом слов (*S*-блоком) размером 2*n*. При *n* = 8 элементы блока представляют собой перестановку чисел от 0 до 255.

1. **Практическая часть**

В данной лабораторной работе было необходимо реализовывать генерацию ПСП. Для реализации генерации ПСП на основе алгоритма BBS была написана следующая функция, представленная в листинге 2.1.

|  |
| --- |
| class BBSGenerator  {  private BigInteger p;  private BigInteger q;  private BigInteger n;  private BigInteger seed;  public BBSGenerator(BigInteger p, BigInteger q, BigInteger seed)  {  this.p = p;  this.q = q;  this.n = p \* q;  this.seed = seed;  }  public int GetNext()  {  BigInteger x = (seed \* seed) % n;  seed = x;  string binaryNumber = Convert.ToString((int)(x % 2));  for (int i = 1; i < 8; i++)  {  x = (x \* x) % n;  binaryNumber += Convert.ToString((int)(x % 2));  }  int decimalNumber = Convert.ToInt32(binaryNumber);  int sequenceNumber = decimalNumber % 256;  return sequenceNumber;  }  } |

Листинг 2.1 – Функция генерации ПСП

Функция возвращает массив из чисел, сгенерированных с использованием соотношения (3) .

Начальными параметрами служат *n*, большие простые числа *p* и *q* (причем *n* = *pq*), целое число *е*, взаимно простое с произведением

(*р* – 1)(*q* – 1), а также некоторое случайное начальное значение *x*0.

Результат работы функции представлен на рисунке 2.1.

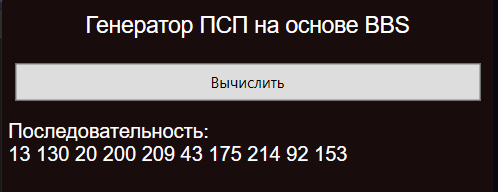


Рисунок 2.1 – Сгенерированная последовательность

Также в лабораторной работе нужно было реализовывать алгоритм RC4, и выполнять оценку скорости выполнения операций генерации ПСП.

Для начальной инициализация вектора-перестановки ключом, используется алгоритм ключевого расписания (рисунок 2.2).

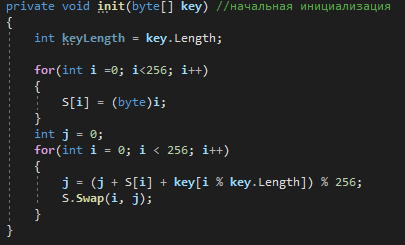


Рисунок 2.2 – Начальная инициализация вектора перестановки

Дальше нужно реализовать генератор псевдослучайной последовательности. Генератор ключевого потока RC4 переставляет значения, хранящиеся в *S*, и каждый раз выбирает различное значение из *S* в качестве результата.

Шифрование сообщения происходит следующим образом: для каждого байта массива/потока входных незашифрованных данных запрашиваем байт ключа и объединяем их при помощи операции XOR (рисунок 2.3).

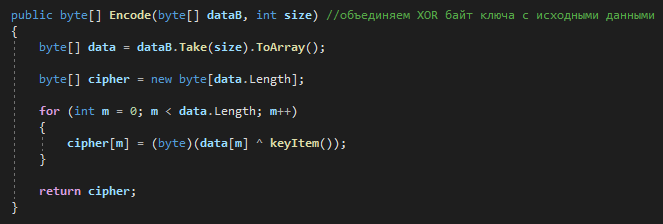


Рисунок 2.3 – Объединение байта ключа с исходными данными

Для расшифровки сообщения достаточно использовать этот же метод. Результат работы приложения представлен на рисунке 2.4.

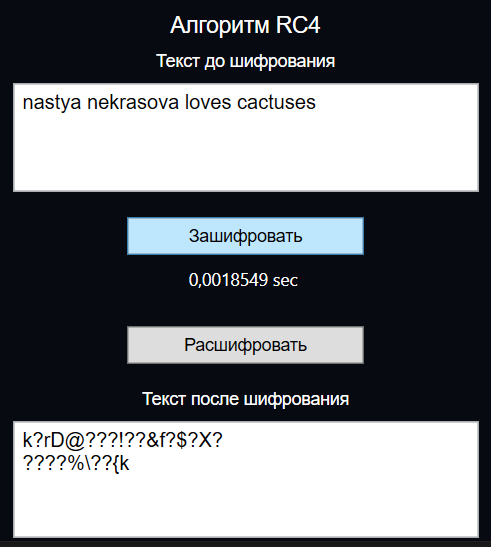


Рисунок 2.4 – Шифрование и расшифрование сообщения

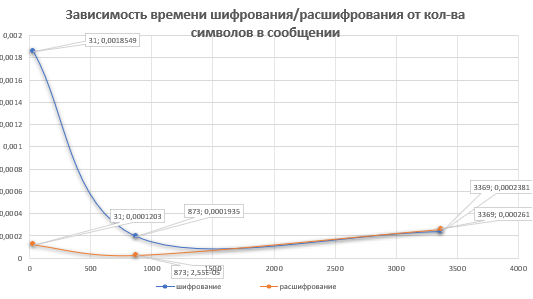


Рисунок 2.5 – Зависимость времени шифрования/расшифрования от количества символов в сообщении

**Вывод:** таким образом, в данной лабораторной работе я изучила и приобрела практические навыки разработки и использования приложений для реализации потоковых шифров, а именно генерации ПСП при помощи алгоритма BBS, шифрования/расшифрования сообщений при помощи алгоритма RC4.