МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образование «Белорусский государственный технологический университет»

Кафедра информационных систем и технологий

**«Исследование потоковых шифров»**

Студент:

Септилко Анастасия Антоновна

Вариант 8

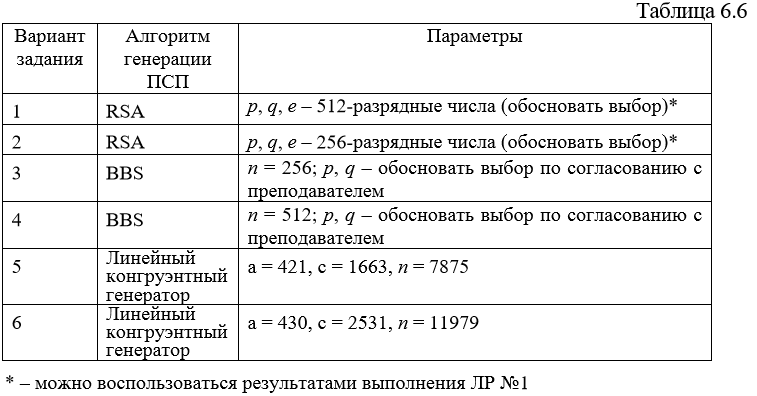
Преподаватель:

Блинова Евгения Александровна

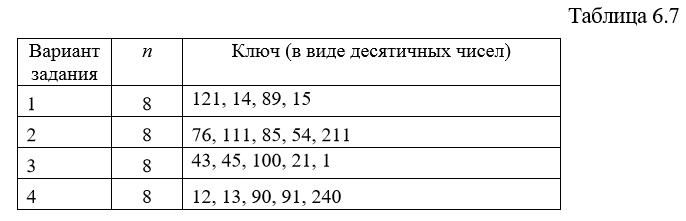
Минск 2020

**ЗАДАНИЕ:**

1. Разработать авторские многооконные приложения в соответствии с целью лабораторной работы. При этом можно воспользоваться готовыми библиотеками либо программными кодами, реализующими заданные алгоритмы. Приложение 1 должно реализовывать генерацию ПСП в соответствии с вариантом из табл. 6.6.



Приложение 2 должно реализовывать алгоритм RC4 в соответствии с вариантом из табл. 6.7, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП.



Собственно, алгоритм RSA разработан для систем асимметричного зашифрования/расшифрования и будет более детально рассмотрен с практической точки зрения ниже. Генератор же ПСП на основе RSA устроен следующим образом. Последовательность генерируется с использованием соотношения

xt = (xt-1)е mod n.

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.

Реализация метода Encrypt продемонстрирована на рисунке 1.1. Передаем в метод параметры p и q (введенные из консоли), подсчитанные e, d, n. Переменная е высчитывается с помощью метода Calculate\_e (m) – подсчет функции Эйлера. Переменная d высчитывается с помощью метода Calculate\_d(e, m). Считываем файл с текстом, и передаем его вместе с e и n в функцию RSA\_Encode (рисунок 1.2).

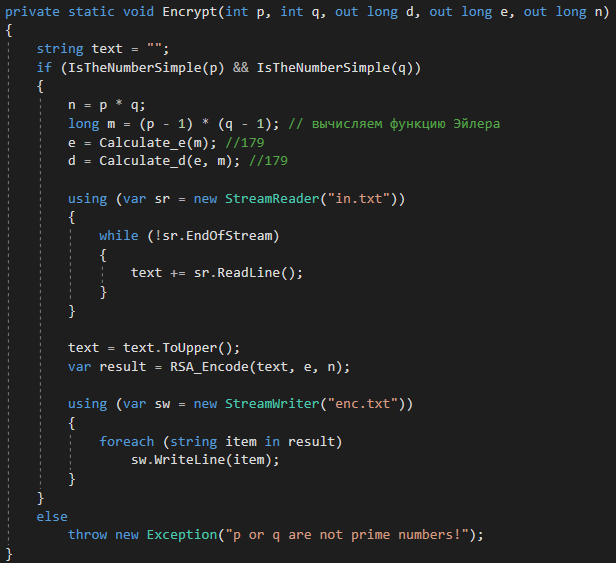


Рисунок 1.1 – Функция Encrypt()

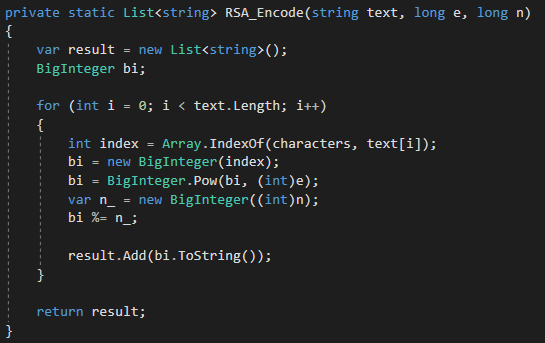


Рисунок 1.2 – Функция RSA\_Encode()

После функции RSA\_Encode у нас формируется последовательность, которая записывается в файл "enc.txt". Его пример можно посмотреть на рисунке 1.3. P=11, Q=19 в этом случае.

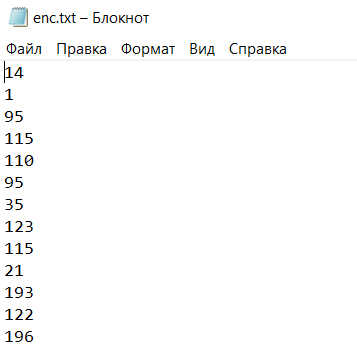


Рисунок 1.3 – Содержимое файла enc.txt

Процесс расшифровки происходит так же как и за шифровка но в обратном порядке. Функции для расшифрования текста представлены на рисунках 1.4 и 1.5.

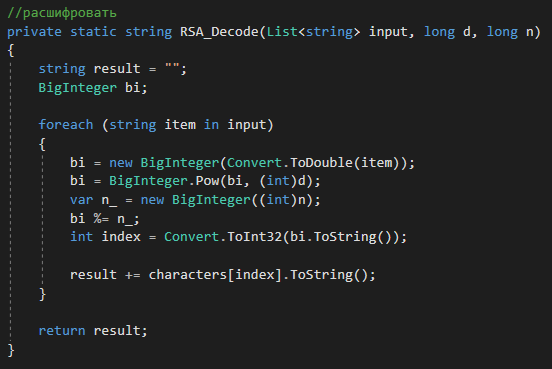


Рисунок 1.4 – Функции для расшифрования текста RSA\_Decode()

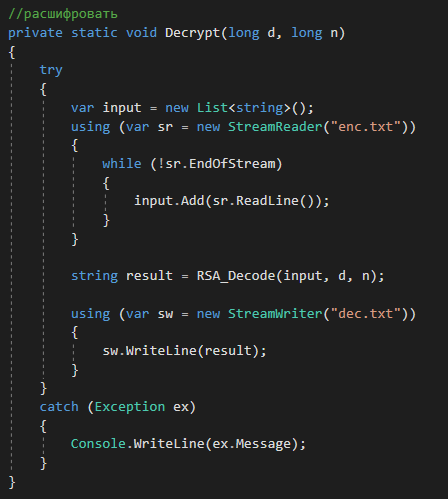


Рисунок 1.5 – Функции для расшифрования текста

После функции RSA\_Decode у нас формируется текст, который записывается в файл "dec.txt". Его пример можно посмотреть на рисунке 1.6. P=11, Q=19 в этом случае.

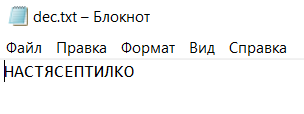
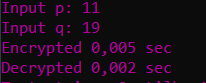


Рисунок 1.6 – Файл dec.txt

Время работы программы представлен ниже:



Для выполнения алгоритма RC4 нужно знать, что :

i = (i + 1) mod 256;

j = (j + Si) mod 256;

поменять местами Si и Sj;

a = (Si + Sj) mod 256;

K = Sa

1. j = 0; i =0;

2. j = (j + Si + Ki) mod 256;

3. поменять местами Si и Sj;

4. i = i +1;

5. если i <256, то перейти на п.2

Итак, входными данными у нас будет выступать массив байт. Ключ тоже выступает в качестве входных данных. Для алгоритма RC4 он может быть от 8 до 2048 бит, но обычно используется диапазон 40 — 256 бит.

Но данные для шифрования у нас — массив байт, а ключ почему-то в битах. Дело в том, что существует такое понятие как размер блока n. Тут используется n = 8, т. е за один шаг шифруется 1 байт. При n = 8 элементы блока представляют собой перестановку чисел от 0 до 255, а сама перестановка зависит от ключа переменной длины.

Для начальной инициализация вектора-перестановки ключом, используется алгоритм ключевого расписания (Key-Scheduling Algorithm), показано на рисунке 1.7.

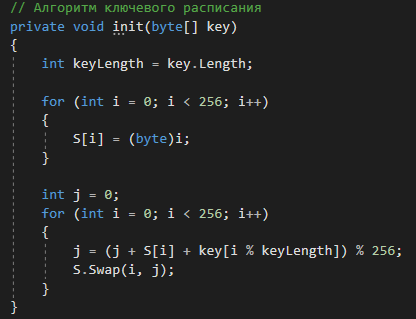


Рисунок 1.7 – Алгоритм Key-Scheduling Algorithm

Сначала заполняем 256-байтный массив ключом. Если необходимо, ключ повторяется многократно, чтобы заполнить весь массив. Для перестановки используется метод Swap, который меняет два элемента местами и расширяет стандартный список методов класса Array. Реализация метода представлена ниже на рисунке 1.8.

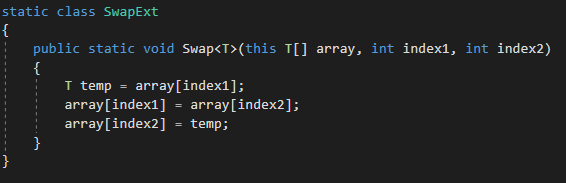


Рисунок 1.8 – Класс SwapExt

Метод init нужно вызвать перед шифровкой/расшифровкой, когда известен ключ. Можно сделать это в конструкторе (рисунок 1.9).

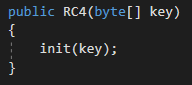


Рисунок 1.9 – Конструктор RC4()

Дальше нужно реализовать генератор псевдослучайной последовательности. При каждом вызове метод будет выдавать последующий байт ключевого потока, который мы и будем объединять xor'ом c байтом исходных данных. Генератор ключевого потока RC4 переставляет значения, хранящиеся в S, и каждый раз выбирает различное значение из S в качестве результата (рисунок 1.10).

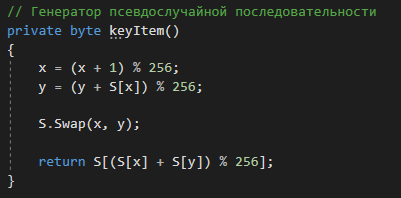


Рисунок 1.10 – Генератор псевдослучайной последовательности

Для каждого байта массива/потока входных незашифрованных данных запрашиваем байт ключа и объединяем их при помощи xor (^) показано на рисунке 1.11.

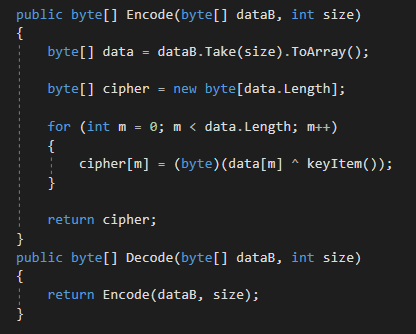


Рисунок 1.11 – Функция Encode()

Результат работы представлен ниже.

