МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №8 НА ТЕМУ:

Исследование потоковых шифров

Выполнила студентка 3 курса 4 группы

Чёрной Я.Р.

Минск 2025

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

**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.

Все потоковые шифры делятся на 2 класса: синхронные и асинхронные (или самосинхронизирующиеся).

Синхронные потоковые шифры (СПШ) характеризуются тем, что поток ключей генерируется независимо от открытого текста и шифртекста. Главное свойство СПШ – нераспространение ошибок. Синхронные потоковые шифры уязвимы к атакам на основе изменения отдельных битов шифртекста.

В самосинхронизирующихся потоковых шифрах символы ключевой гаммы зависят от исходного секретного ключа шифра и от конечного числа последних знаков зашифрованного текста.

Ключевые последовательности (случайные последовательности (СП), либо псевдослучайные последовательности (ПСП)) вырабатываются специальными блоками систем потокового шифрования – генераторами.

Часто используемый алгоритм генерирования (программно или аппаратно) ПСП реализуется на основе линейного конгруэнтного генератора, достаточно распространенным является использование регистров сдвига (РС) в качестве генераторов ПСП в силу простоты реализации на основе цифровой логики, широкое распространение получил алгоритм генерации ПСП, называемый алгоритмом BBS

Генератор псевдослучайных чисел может быть на основе алгоритма RSA. Он устроен следующим образом:

Последовательность генерируется с использованием соотношения

xt ≡ (xt – 1)е mod n. (1)

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 *x0*. Выходом генератора на *t*-м шаге является младший бит числа *xt*. Безопасность генератора опирается на сложность взлома алгоритма RSA, т. е. на разложение числа *n* на простые сомножители.

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 битов. 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, а сама перестановка зависит от ключа переменной длины. Другими элементами внутреннего состояния являются 2 счетчика (каждый размером в одно слово; обозначим их *i* и *j*) с нулевыми начальными значениями. В основе вычислений лежит операция по mod 2*n*. Генератор ключевого потока RC4 переставляет значения, хранящиеся в *S*, и каждый раз выбирает различное значение из *S* в качестве результата. В одном цикле RC4 определяется одно *n*-битное слово *K* из ключевого потока, которое в последующем суммируется с исходным текстом для получения зашифрованного текста. Эта часть алгоритма называется генератором ПСП.

Байт *K* используется в операции XOR с открытым текстом для получения 8-битного шифртекста или для его расшифрования.

Так же достаточно проста и инициализация *S*-блока. Этот алгоритм использует ключ, который подается на вход пользователем. Сначала *S*-блок заполняется линейно: *S*0 = 0, *S*1 = 1, …, *S*255 = 255. Затем заполняется секретным ключом другой 256-байтный массив. Если необходимо, ключ повторяется многократно, чтобы заполнить весь массив: *K*0, *K*1, …, *K*255. Далее массив *S* перемешивается путем перестановок, определяемых ключом.

**Практическое задание**

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.

|  |  |  |
| --- | --- | --- |
| Вариант | Алгоритм генерации ПСП | Параметры |
| 2 | RSA | *p*, *q*, *e* – 256-разрядные числа |

В данном случае, для генерации ПСП на основе RSA, необходимо выбрать два простых числа *p* и *q*, каждое из которых будет иметь длину 256 бит (то есть каждое число будет состоять из 256 двоичных разрядов). Выбор 256-разрядных простых чисел p и q обеспечит высокую степень защиты от атак, основанных на факторизации чисел на простые множители. Такие числа невозможно эффективно факторизовать на текущий момент при использовании наиболее известных алгоритмов факторизации.

В приложении для генерации этих чисел используется библиотека *Crypto.Util.number* (листинг 1).

    # Генерация двух 256-разрядных простых чисел

    p = Crypto.Util.number.getPrime(256, randfunc = Crypto.Random.get\_random\_bytes)

    q = Crypto.Util.number.getPrime(256, randfunc = Crypto.Random.get\_random\_bytes)

Листинг 1 – Генерация *p*, *q*

Например, сгенерированные таким образом p и q могут принимать следующие значения:

*p* =

100148503798148397260882949024782135675088928894197678172480871597365270535129

*q* =

11198040523770520961764683848152092110827665974402165189171445670862344238922

Далее необходимо сгенерировать целое число *е*, взаимно простое с произведением (*р* – 1)(*q* – 1). Для этого используется расширенный алгоритм Евклида, заданный в программе рекурсивно (листинг 2).

phi = (p-1)\*(q-1)

e = generate\_coprime(phi)

def extended\_gcd(a, b):

    if b == 0:

        return a, 1, 0

    gcd, x1, y1 = extended\_gcd(b, a % b)

    x = y1

    y = x1 - (a // b) \* y1

    return gcd, x, y

def generate\_coprime(phi):

    while True:

        e = random.randrange(2, phi)

        gcd, x, y = extended\_gcd(e, phi)

        if gcd == 1:

            return e

Листинг 2 – Генерация *e*

В качестве *x*0 используется случайное число в диапазоне от 0 до (*n* – 1), *xt* задается по формуле. В выход генератора записывается младший бит числа *xt*.

def generate\_rsa\_prs(t):

    seq = ''

    # Генерация двух 256-разрядных простых чисел

    p = Crypto.Util.number.getPrime(256, randfunc = Crypto.Random.get\_random\_bytes)

    q = Crypto.Util.number.getPrime(256, randfunc = Crypto.Random.get\_random\_bytes)

    print ("\np: ", p)

    print ("q: ", q)

    # Вычисление произведения p и q

    n = p \* q

    print ("n: ", n)

    # Выбор открытого ключа e

    phi = (p-1)\*(q-1)

    e = generate\_coprime(phi)

    print ("e: ", e)

    x0 = random.randint(0, n-1)

    print ("x0: ", x0)

    x\_t = pow(x0,e,n)

    for i in range(t):

        x\_t = pow(x\_t,e,n)

        seq += str(x\_t % 2)

    return seq

Листинг 3 – Функция генерации ПСП на основе алгоритма RSA

Результат работы приложения будет иметь следующий вид:

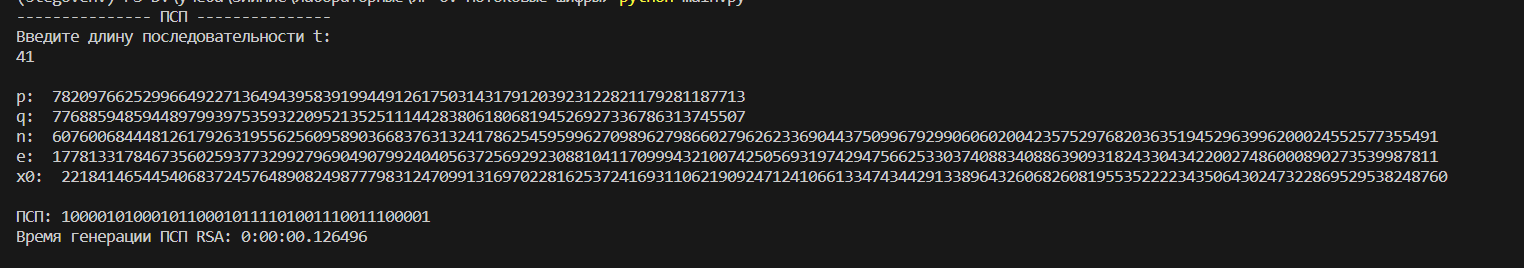


Рисунок 1 – Генерация ПСП на основе алгоритма RSA

**Вывод:** поточные шифры обычно используется в том случае, когда шифруемые символы открытого текста представляются одной буквой, битом или реже – байтом. Разработка и использование приложений для реализации поточных шифров требует глубоких знаний в области криптографии, математических основ криптографии и программирования.