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**

Выполнил студент 3 курса 6 группы

Подобед Владислав

Минск 2024

**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 |
| 3 | BBS | n=256,p,q |

В данном случае, для генерации ПСП на основе BBS, необходимо выбрать два простых числа *p* и *q*, при условии. 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.

Например, p и q могут принимать следующие значения:

Первым шагом является нахождение подходящих простых чисел p и q. Поскольку n = 256 = 16 \* 16, p и q должны быть близки к 16 и давать остаток 3 при делении на 4. Посмотрим на простые числа, близкие к 16:

p = 13 (3 \* 4 + 1)

q = 19 (4 \* 4 + 3)

Теперь проверим, удовлетворяют ли эти числа всем условиям:

Оба p и q являются простыми числами - да, 13 и 19 простые.

n = p \* q = 13 \* 19 = 247 — это квадрат простого числа.

p и q дают остаток 3 при делении на 4 - да, 13 % 4 = 1 и 19 % 4 = 3.

Таким образом, мы можем использовать p = 13 и q = 19 для алгоритма BBS с n = 256

Далее необходимо шифровать сообщение, код представлен в листинге 1.

|  |
| --- |
| def encrypt\_number(keys, plaintext):      n, \_, \_ = keys      seed = random.randint(2, n - 1)      pseudorandom\_sequence = bbs\_generator(n, seed, len(str(plaintext)))      ciphertext = int(plaintext) ^ int(''.join(map(str, pseudorandom\_sequence)))      ciphertext\_binary = bin(ciphertext)[2:]      return ciphertext\_binary, seed, n |

Листинг 1 –функция encrypt\_number для шифрования

Функцию decrypt\_number позволяет нам расшифровывать сообщение, которое вначале зашифровали.

|  |
| --- |
| def crypt\_number(keys, plaintext):      n, \_, \_ = keys      seed = random.randint(2, n - 1)      pseudorandom\_sequence = bbs\_generator(n, seed, len(str(plaintext)))      ciphertext = int(plaintext) ^ int(''.join(map(str, pseudorandom\_sequence)))      ciphertext\_binary = bin(ciphertext)[2:]      return ciphertext\_binary, seed, n |

Листинг 2 –функция decrypt\_number для расшифрования

В качестве *x* используется случайное число, *xt* задается по формуле. В выход генератора записывается младший бит числа *xt*.

|  |
| --- |
| def bbs\_generator(n, seed, length):      x = seed      result = []      for \_ in range(length):          x = (x \* x) % n          result.append(x % 2)      return result |

Листинг 3 – Функция генерации ПСП на основе алгоритма BBS

Результат работы приложения будет иметь следующий вид:

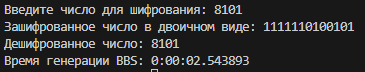


Рисунок 1 – Генерация ПСП на основе алгоритма RSA

Приложение 2 должно реализовывать алгоритм RC4 в соответствии с вариантом из табл. 6.8, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП. В качестве шифруемого сообщения может быть выбран произвольный текст.

|  |  |  |
| --- | --- | --- |
| Вариант | *n* | Ключ (в виде десятичных чисел) |
| 9 | 6 | 61, 60, 23, 22, 21, 20 |

Секретный ключ состоит из 6 значений: 61, 60, 23, 22, 21, 20. Далее необходимо инициализировать ключ. Создадим массив размера 64 байт и заполним его числами от 0 до 63. Затем перемешаем массив, используяключ, например, 20. Для этого выполним перестановку элементов массива в соответствии со следующим алгоритмом:

* Инициализируйте два указателя *i* и *j*, оба установлены в 0.
* Пройдите по массиву от 0 до 63 и выполните следующие действия:
  + Увеличьте указатель *i* на 1
  + Установите переменную *t* равной ключу *k*[*i*] по модулю 64
  + Обменяйте значения элементов массива *S*[*i*] и *S*[*t*]
  + Увеличьте указатель *j* на *t*

После этой инициализации массив *S* будет перемешан, используя ключ 20.

Необходимо сгенерировать псевдослучайную последовательность: с помощью массива S сгенерируем псевдослучайную последовательность байтов, используя следующий алгоритм:

* Инициализируйте два указателя *i* и *j*, оба установлены в 0.
* Для каждого байта сообщения выполните следующие действия:
  + Увеличьте указатель *i* на 1
  + Установите переменную t равной S[i]
  + Увеличьте указатель *j* на *t*
  + Обменяйте значения элементов массива *S*[*i*] и *S*[*j*]
  + Получите значение байта гаммы *g*, используя *S*[(*S*[*i*] + *S*[*j*]) % 64]
  + Выполните операцию XOR между байтом сообщения и байтом гаммы *g*

Шифрование сообщения: Результатом шифрования будет последовательность байтов, полученная в результате операции XOR из предыдущего шага. После генерации псевдослучайной последовательности с помощью RC4 алгоритма и ключей 61, 60, 23, 22, 21, 20 получим зашифрованную последовательность байтов.

Результат работы приложения будет иметь следующий вид:

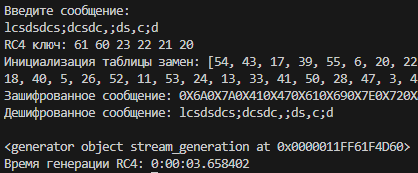


Рисунок 2 – Шифрование и дешифрование сообщения алгоритмом RC4

Время генерации ПСП в алгоритме RC4 немного больше времени генерации ПСП на основе алгоритма BBS.



Рисунок 3 – Время генерации ПСП

**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.