МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность Информационные системы и технологии

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6 НА ТЕМУ:

Исследование потоковых шифров

Выполнила студентка 3 курса 1 группы

Пригодич Вера Валерьевна

Минск 2023

**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:

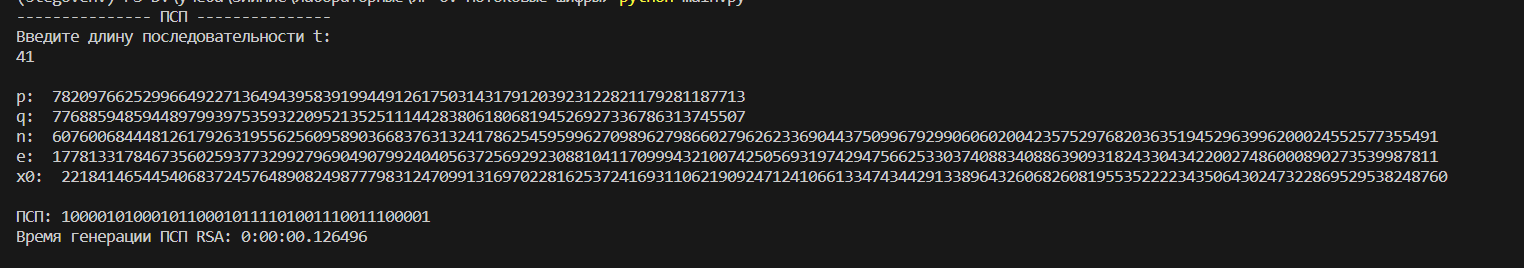


Рисунок 1 – Генерация ПСП на основе алгоритма RSA

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) |
| 8 | 6 | 20, 21, 22, 23, 60, 61 |

Секретный ключ состоит из 6 значений: 20, 21, 22, 23, 60, 61. 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 20, 21, 22, 23, 60, 61 получим зашифрованную последовательность байтов.

Результат работы приложения будет иметь следующий вид:

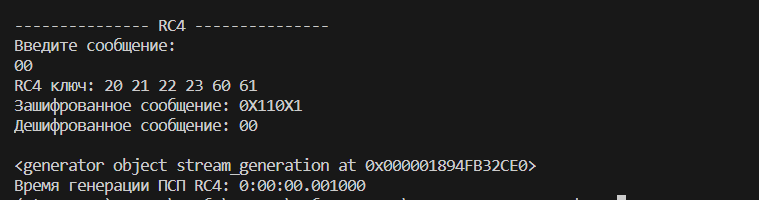


Рисунок 2 – Шифрование и дешифрование сообщения алгоритмом RC4

Время генерации ПСП в алгоритме RC4 значительно меньше времени генерации ПСП на основе алгоритма RSA.

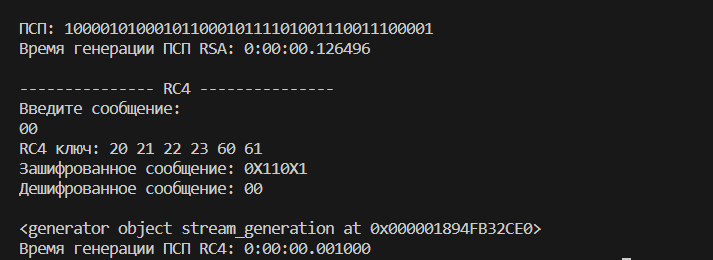


Рисунок 3 – Время генерации ПСП

**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.

**Контрольные вопросы**

1. В чем состоит особенность потоковых шифров?

Особенность потоковых шифров заключается в их способности шифровать данные в режиме реального времени. Это означает, что данные могут быть шифрованы и переданы по мере их поступления, а не ожидать, пока все данные будут собраны.

1. В чем состоят преимущества и недостатки синхронных и асинхронных потоковых шифров?

Преимущества синхронных потоковых шифров:

* Очень быстрое шифрование и дешифрование потоков данных
* Меньшее количество вычислительных операций, чем у асинхронных потоковых шифров
* Простота реализации и интеграции в устройствах
* Низкие требования к вычислительным ресурсам и памяти

Недостатки синхронных потоковых шифров:

* Невозможность изменения ключа в процессе передачи данных
* Невозможность обеспечить конфиденциальность при передаче данных на большие расстояния без потерь, так как синхронные потоковые шифры чувствительны к ошибкам передачи данных и сбоям синхронизации
* Риск повторения псевдослучайной последовательности, если генератор псевдослучайных чисел не был настроен правильно или если использован один и тот же ключ для генерации псевдослучайной последовательности более одного раза

Преимущества асинхронных потоковых шифров:

* Возможность изменения ключа в процессе передачи данных, что обеспечивает более высокую степень безопасности передачи данных
* Хорошая устойчивость к ошибкам передачи данных и сбоям синхронизации
* Более высокая степень безопасности, так как асинхронные потоковые шифры используют криптографически стойкие методы генерации псевдослучайных чисел

Недостатки асинхронных потоковых шифров:

* Большее количество вычислительных операций, чем у синхронных потоковых шифров
* Более высокие требования к вычислительным ресурсам и памяти, чем у синхронных потоковых шифров
* Могут возникнуть проблемы с использованием и реализацией, если устройства не имеют вычислительных мощностей или не имеют доступа к сети

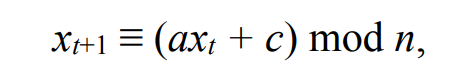
1. Какими свойствами должен обладать генератор псевдослучайных чисел для использования в криптографических целях?

Генератор псевдослучайных чисел (ГПСЧ) для использования в криптографических целях должен обладать следующими свойствами:

* Непредсказуемость: Сгенерированные числа должны быть непредсказуемыми для злоумышленника, даже если он знает все предыдущие числа.
* Статистическая равномерность: Сгенерированные числа должны быть статистически равномерными, то есть каждое возможное значение должно быть сгенерировано с приблизительно равной вероятностью.
* Длинный период: ГПСЧ должен иметь длинный период, то есть количество генерируемых чисел должно быть достаточно большим, чтобы не повторяться в пределах используемого срока службы.
* Непредсказуемость второго порядка: для защиты от атак второго порядка, ГПСЧ должен генерировать числа, которые не могут быть предсказаны, даже если злоумышленник знает часть изначальных чисел.
* Невоспроизводимость: нельзя должным образом вычислить предыдущие числа по последующим числам.
* Криптографическая стойкость: ГПСЧ должен устойчиво устоять перед широким спектром атак, включая атаки на основе статистических тестов, линейных и дифференциальных криптоанализов, а также других методов атак.
* Эффективность: ГПСЧ должен генерировать числа достаточно быстро, чтобы быть полезным в практических приложениях, и при этом не потреблять слишком много ресурсов.

1. Дать характеристику линейным конгруэнтным генераторам. Области их применения.

Часто используемый алгоритм генерирования (программно или аппаратно) ПСП реализуется на основе так называемого линейного конгруэнтного генератора, описываемого следующим рекуррентным соотношением:



где *xt* и *xt* + 1 – соответственно *t*-й (предыдущий) и (*t* + 1)-й (текущий, вычисляемый) члены числовой последовательности; *а*, *с* и *n* – константы. Период такого генератора (период ПСП) не превышает *n*.

Если параметры *a*, *b* и *c* выбраны правильно, то генератор будет порождать случайные числа с максимальным периодом, равным *c*. При программной реализации значение с обычно устанавливается равным 2*b* – 1 или 2*b* , где *b* – длина слова в битах.

Достоинством линейных конгруэнтных генераторов псевдослучайных чисел является их простота и высокая скорость получения псевдослучайных значений. Линейные конгруэнтные генераторы находят применение при решении задач моделирования и математической статистики, однако в криптографических целях их нельзя рекомендовать к использованию, так как специалисты по криптоанализу научились восстанавливать всю последовательность ПСП по нескольким ее значениям.

Области применения ЛКГ:

* Игровая индустрия: ЛКГ широко используются в компьютерных играх, чтобы генерировать случайные числа, такие как кости, карты, случайные события и т. д.
* Моделирование: ЛКГ могут использоваться в научных и инженерных расчетах, в которых требуется генерация случайных чисел для моделирования случайных процессов или стохастических систем.
* Тестирование ПО: ЛКГ могут использоваться для генерации тестовых данных при тестировании программного обеспечения.
* Другие приложения: ЛКГ также могут использоваться в других приложениях, где требуется быстрое и простое генерирование псевдослучайных чисел.

1. Значения x0, x1, x2, x3, полученные с помощью линейного конгруэнтного генератора, равны соответственно: 1, 12, 3, 6. Найти параметры а, с и n генератора ПСЧ, удовлетворяющие выражению

Для линейного конгруэнтного генератора, значение следующего числа генерируется по формуле:

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

где:

*Xn* - текущее значение

*a* - множитель

*c* - инкремент

*n* - модуль (*n* > 0)

Для того чтобы найти параметры *a*, *c* и *n* генератора ПСЧ, используем систему уравнений:

*X*1 = (*aX*0 + *c*) mod *n*

*X*2 = (*aX*2 + *c*) mod *n*

*X*3 = (*aX*3 + *c*) mod *n*

Подставляем значения *X*0, *X*1, *X*2, *X*3:

12 = (*a* + c) mod *n* => *c* = (12 - *a*) mod *n* =>

3 = (12*a* + 12 - *a*) mod *n* = (12 + 11*a*) mod *n*

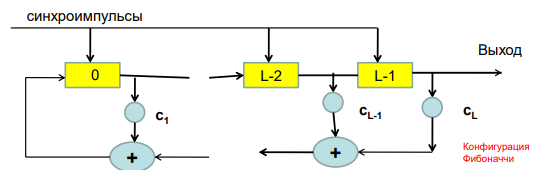
6 = (3*a* + c) mod *n =* (3*a* + 12 – *a*) = (2a + 12) mod *n*

Для решения этого уравнения можно перебирать значения *n* и находить такие значения *a*, которые удовлетворяют уравнению.

Для решения данной задачи, предположим, что n = 13, тогда:

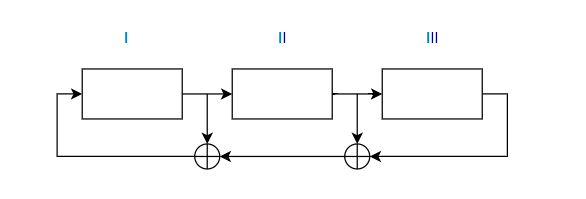
a = 7, c = 5 и n = 13.

1. Представить общую структурную схему генератора ПСП на основе регистров сдвига с линейной обратной связью. Пояснить особенности его функционирования.

****

Общая структурная схема генератора ПСП включает в себя регистры сдвига, которые состоят из последовательности битов, связанных друг с другом линейной обратной связью. Каждый бит регистра сдвига определяется как логическое ИЛИ некоторых (но не всех) битов регистра, включая некоторые биты, которые могут быть замаскированы.

1. Синтезировать структурную схему генератора ПСП на основе регистров сдвига с линейной обратной связью, формально обозначаемого следующим образом: а) 3210; б) 420; в) 5410; г) 520; д) 320. Составить таблицу состояний генератора и определить период ПСП.
2. 3210



Многочлен 3210 является приводимым, значит его период зависит от входной последовательности. Таблица состояний генератора может иметь следующий вид:

|  |  |  |  |
| --- | --- | --- | --- |
| ЦОС | I | II | III |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |

1. 420

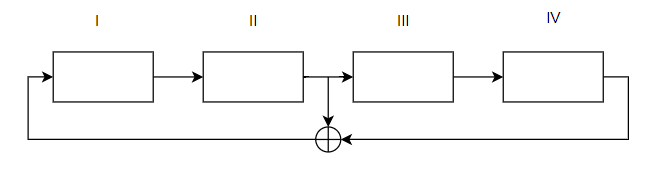
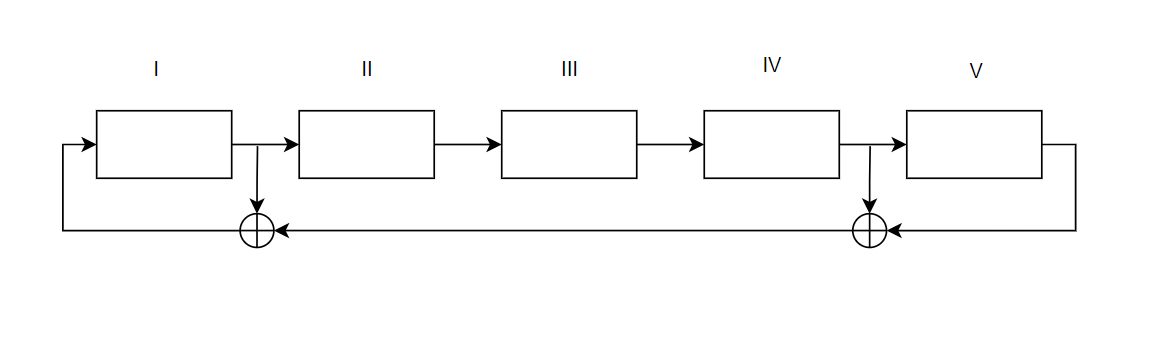


Таблица состояний генератора:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ЦОС | I | II | III | IV |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |

1. 5410



Многочлен 3210 является приводимым, значит его период зависит от входной последовательности. Таблица состояний генератора может иметь следующий вид:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ЦОС | I | II | III | IV | V |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |

1. 520

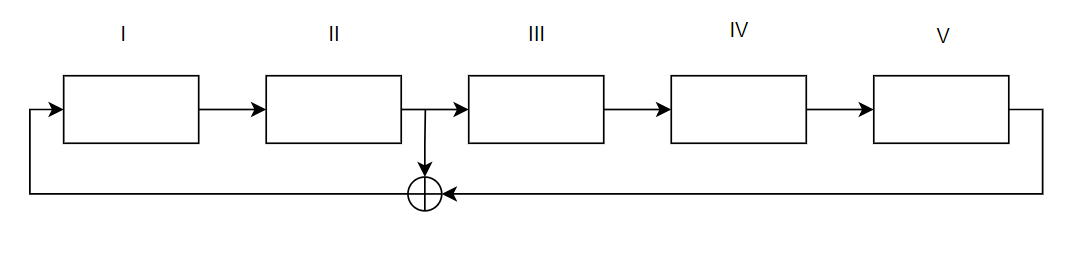


Таблица состояний генератора может иметь следующий вид:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ЦОС | I | II | III | IV | V |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |

1. 320

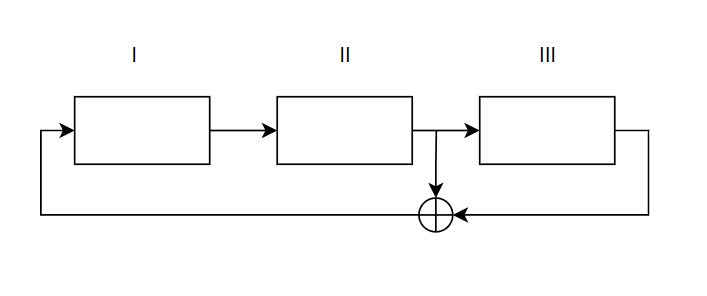


Таблица состояний генератора может иметь следующий вид:

|  |  |  |  |
| --- | --- | --- | --- |
| ЦОС | I | II | III |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |

1. Определить первые 12 бит ПСП, задаваемого формально в виде чисел 5410, если начальные состояния ячеек (слева направо) соответствуют последовательности 10101.

Биты будут повторятся следующим образом:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ЦОС | I | II | III | IV | V |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 1 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 | 1 |

1. Как устроен генератор ПСП на основе RSA? На чем основана криптостойкость реализуемого алгоритма?

Генератор ПСП (последовательности псевдослучайных чисел) на основе RSA (Rivest-Shamir-Adleman) использует принципы криптографии RSA для генерации псевдослучайных чисел.

Для генерации ПСП на основе RSA используется следующий алгоритм:

* Генерируются два простых числа *p* и *q*.
* Вычисляется значение n как произведение *p* и *q*.
* Вычисляется значение функции Эйлера от *n*, которое равно (*p* - 1)(*q* - 1).
* Выбирается целое число *e*, которое является взаимно простым со значением функции Эйлера и меньше её.
* Выбирается случайное начальное значение *x0*.

Последовательность генерируется с использованием соотношения:

xt ≡ (xt – 1)е mod n. (1)

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.

1. Вычислить *х*1, *х*5, *х*9, *х*11 по методу генерации псевдослучайных чисел BBS, если *p* = 11, *q* = 19, *х* = 3.

*n* = 11 \* 19 = 209

*х*0 = 3^2 mod 209 = 9

*х*1 = 9^2 mod 209 = 81

*х*2 = 81^2 mod 209 = 82

*х*3 = 81^2 mod 209 = 36

*х*4 = 36^2 mod 209 = 42

*х*5 = 42^2 mod 209 = 92

*х*6 = 92^2 mod 209 = 104

*х*7 = 104^2 mod 209 = 157

*х*8 = 157^2 mod 209 = 196

*х*9 = 196^2 mod 209 = 169

*х*10 = 169^2 mod 209 = 137

*х*11 = 137^2 mod 209 = 168

Ответ: *х*1 = 81, *х*5 = 92, *х*9 = 169, *х*11 = 168.

1. Пояснить базовый алгоритм, реализованный в шифре RC4.

См. выше.

1. Пояснить принципы формирования истинных случайных последовательностей, основанных на «естественных случайностях».

Истинные случайные последовательности могут быть сформированы на основе естественных случайностей, таких как шум термоэлектрического преобразования, флуктуации радиоактивных изотопов, шум электронных устройств и т.д. Принцип формирования таких последовательностей заключается в том, что они создаются с помощью естественных физических процессов, которые в принципе невозможно точно предсказать заранее, и могут быть использованы для генерации случайных чисел.

Принципы формирования истинных случайных последовательностей могут различаться в зависимости от используемых методов. Например, при использовании шума термоэлектрического преобразования, случайная последовательность может быть сформирована путем измерения температуры на резисторе, который находится в контакте с термоэлектрической парой. Резистор генерирует шум в результате случайных тепловых движений электронов, и этот шум может быть использован для формирования последовательности случайных чисел.

В общем случае, при формировании истинных случайных последовательностей на основе естественных случайностей, необходимо обеспечить достаточно высокую энтропию генерируемых значений, чтобы исключить возможность предсказания последующих значений на основе предыдущих. Также важно контролировать любые возможные искажения, которые могут возникнуть в процессе измерений или передачи данных.

1. Начало формы