Задания СРС:

**Цель: у**глубление теоретических знаний и развитие практических навыков по обеспечению кибербезопасности, формирование компетенций в области защиты информации, освоение принципов функционирования систем защиты, а также анализ современных киберугроз и методов противодействия им.

**Лекция 3: Поточные шифры и генераторы псевдослучайных чисел. Часть 2**

**Цель:**  
Закрепить знания о типах поточных шифров и ГПСЧ, проанализировать современные реализации и требования к их безопасности.

**Задание:**

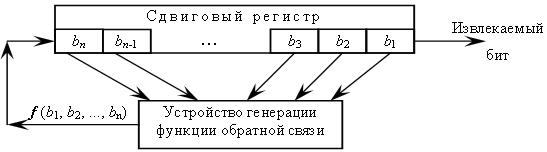
1. Повторить ключевые понятия из Лекции №2 (Часть 1):  
   – Синхронные и асинхронные поточные шифры  
   – Основные принципы генерации псевдослучайных последовательностей
2. Изучить современные поточные шифры, используемые в криптографии:  
   – A5/1 и A5/2 (использовались в GSM-связи)  
   – RC4 (применялся в WEP и TLS)  
   – Salsa20/ChaCha (современные шифры с открытым исходным кодом)
3. Выполнить аналитическое задание:  
   – Сравнить минимум два поточных шифра по следующим критериям:  
   • Скорость  
   • Стойкость к криптоанализу  
   • Применение  
   • Степень предсказуемости выходных битов
4. Подготовить:  
   – Презентацию (10–15 слайдов)  
   – Таблицу сравнения  
   – Краткий письменный вывод (0,5–1 страница)

**Цель лекции**: продолжить знакомство с *генераторами псевдослучайных чисел*, используемых для поточного шифрования информации, а также с некоторыми режимами блочных шифров для получения псевдослучайных чисел.

### Генераторы псевдослучайных чисел на основе сдвиговых регистров с обратной связью

В теории кодирования и криптографии широко применяются так называемые **сдвиговые регистры с обратной связью**. Они использовались в аппаратуре шифрования еще до начала массового использования ЭВМ и современных высокоскоростных программных *шифраторов*.

Сдвиговые регистры с обратной связью могут применяться для получения потока псевдослучайных *бит*. *Сдвиговый регистр* с обратной связью состоит из двух частей: собственно n-битного сдвигового регистра и устройства обратной связи ([рис. 8.1](https://www.intuit.ru/studies/courses/691/547/lecture/12385?page=1" \l "image.8.1)).



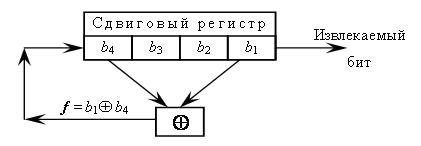
**Рис. 8.1.**Сдвиговый регистр с обратной связью

Извлекать биты из сдвигового регистра можно только по одному. Если необходимо извлечь следующий *бит*, все биты *регистра сдвигаются* вправо на 1 разряд. При этом на вход регистра слева поступает новый *бит*, который формируется устройством обратной связи и зависит от всех остальных битов сдвигового регистра. За счет этого биты регистра изменяются по определенному закону, который и определяет схему получения ПСЧ. Понятно, что через некоторое количество тактов работы регистра последовательность битов начнет повторяться. *Длина* получаемой последовательности до начала ее повторения называется *периодом* сдвигового регистра.

*Поточные шифры* с использованием сдвиговых регистров достаточно долго использовались на практике. Это связано с тем, что они очень хорошо реализуются с помощью цифровой аппаратуры.

Простейшим видом сдвигового регистра с обратной связью является *линейный сдвиговый регистр с обратной связью* (*linear* *feedback* *shift register* – *LFSR*). *Обратная связь* в этом устройстве реализуется просто как сумма по модулю 2 всех (или некоторых) битов регистра. Биты, которые участвуют в обратной связи, образуют *отводную последовательность*. Линейные сдвиговые регистры с обратной связью или их модификации часто применяется в криптографии.

Для того, чтобы стало понятнее, как работает *сдвиговый регистр* с обратной связью, рассмотрим 4-битовый *LFSR* с отводом от первого и четвертого разрядов, представленный на [рис. 8.2](https://www.intuit.ru/studies/courses/691/547/lecture/12385?page=1" \l "image.8.2).



**Рис. 8.2.**Пример 4-разрядного линейного сдвигового регистра

Запишем в изображенный на рисунке *регистр* начальное *значение* 1011. Вычислять последовательность внутренних состояний регистра удобно с помощью таблицы, представленной на [таблица 8.1](https://www.intuit.ru/studies/courses/691/547/lecture/12385?page=1" \l "table.8.1). В таблице отражены первые девять состояний регистра.

На каждом шаге все содержимое регистра сдвигается вправо на один разряд. При этом можно получить в качестве результата один *бит*. На освободившееся слева *место* поступает *бит*, равный результату вычисления функции обратной связи Описание: f = b_1 \oplus b_4. Выходную последовательность генератора псевдослучайных *бит* образует последний столбец таблицы (извлекаемый *бит*).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 8.1. Последовательность работы линейного сдвигового регистра | | | |
| **Номер состояния** | **Внутреннее состояние регистра b4, b3, b2, b1** | **Результат вычисления функции обратной связи**  **Описание: f = b_1 \oplus b_4** | **Извлекаемый бит ( b1 )** |
| 0 | 1 0 1 1 | 0 | 1 |
| 1 | 0 1 0 1 | 1 | 1 |
| 2 | 1 0 1 0 | 1 | 0 |
| 3 | 1 1 0 1 | 0 | 1 |
| 4 | 0 1 1 0 | 0 | 0 |
| 5 | 0 0 1 1 | 1 | 1 |
| 6 | 1 0 0 1 | 0 | 1 |
| 7 | 0 1 0 0 | 0 | 0 |
| 8 | 0 0 1 0 | 0 | 0 |

Линейный *сдвиговый регистр* размером n *бит* может находиться в одном из 2n-1 состояний (исключается состояние регистра из одних нулей - при появлении такого состояния далее будут порождаться лишь нули, и о псевдослучаности порождаемой последовательности говорить не приходится). Поэтому теоретически такой *регистр* может генерировать псевдослучайную последовательность с максимальным периодом 2n-1. Линейный *сдвиговый регистр* с обратной связью будет генерировать циклическую последовательность битов с максимальным периодом только при выборе в качестве отводной последовательности определенных *бит*. Разработана математическая теория, позволяющая выбрать подходящие номера разрядов для *бит* отводной последовательности.

Линейные сдвиговые регистры с обратной связью часто использовались и используются до сих пор при шифровании потоков данных. Для повышения криптостойкости в таких устройствах шифрования применяются комбинации нескольких сдвиговых регистров с обратной связью, а также вводятся дополнительные перемешивающие *операции*. Такие электронные схемы предлагались и выпускались еще до второй мировой войны. Аналогичные принципы заложены и в некоторые *поточные шифры*, созданные в конце XX века, например, в *алгоритм* А5, использовавшийся в Европе для шифрования сотовых цифровых каналов связи стандарта *GSM*. Несмотря на то, что некоторые криптоаналитики высказывают сомнения в надежности алгоритмов поточного шифрования с использованием линейных сдвиговых регистров с обратной связью, они положены в основу функционирования различных военных и гражданских устройств связи, используемых до настоящего времени.

Основным недостатком генераторов псевдослучайных чисел на базе линейных сдвиговых регистров является сложность программной реализации. Сдвиги и *битовые операции* легко и быстро выполняются в электронной аппаратуре, поэтому в разных странах выпускаются микросхемы и устройства для поточного шифрования на базе алгоритмов с использованием сдвиговых регистров с обратной связью.

### Использование режимов OFB и CTR блочных шифров для получения псевдослучайных чисел

Можно использовать любой блочный *алгоритм*, например *AES* или ГОСТ 28147-89, для поточного шифрования информации, используя *режимы OFB и CTR* блочных шифров.

Название режима **OFB** (*Output* *FeedBack*) переводится как "*обратная связь* по выходу".

Пусть минимальный *блок данных*, используемый для передачи, состоит из j *бит*; обычным значением является j=8 (то есть минимальной порцией передаваемых данных является 1 *байт*). В режиме *OFB* блочный *шифр* f на основе секретного ключа К и некоторого инициализирующего значения Y0 формирует псевдослучайную последовательность j-битовых чисел z1,z2,...,zk, которая затем может использоваться в качестве гаммы для шифрования сообщения. Результат зашифрования является входом процедуры шифрования следующего блока исходного сообщения. На каждом этапе шифрования из зашифрованного блока Yi выбирается j младших битов.

Таким образом, для получения псевдослучайной последовательности используется схема:

Yi=f(Yi-1,K),

zi=j младших бит Yi, 1<=i<=k

Если размер блока шифра равен N *бит*, то *параметр* j может принимать значения от 1 до N. *Значение* Y0 называют также *инициализирующим вектором*.

Последовательность чисел zi можно использовать в качестве гаммы для шифрования потока исходных данных, состоящего из символов хi:

Описание: y_i=x_i \oplus z_i

в результате чего получится *поток* зашифрованных символов yi.

Так как значения yi не зависят от открытого текста xi, то каждый раз, используя одни и те же параметры К и Y0, мы получим одну и ту же последовательность гаммы zi. Поэтому рекомендуется менять *значение* ключа К для передачи каждого нового сообщения.

Расшифрование сообщений для описанного режима может производиться только с начала последовательности, так как невозможно получить произвольный элемент последовательности zi, не вычислив все предыдущие.

Основное достоинство режима *OFB* заключается в том, что последовательность z может быть сформирована заранее для того, чтобы быстро шифровать или расшифровывать поточные сообщения в момент их поступления. Это может быть актуально для систем, обрабатывающих данные в реальном масштабе времени.

Еще одно важное достоинство режима *OFB* состоит в том, что если при передаче данных произошла ошибка, то она не распространяется на следующие зашифрованные блоки, и тем самым сохраняется возможность расшифрования последующих блоков. Например, если в результате передачи по зашумленному каналу связи появился ошибочный *бит* в блоке yi, то это приведет к невозможности расшифрования только этого блока и получения одного блока исходных данных xi. Дальнейшая последовательность блоков будет расшифрована корректно.

Название *режима CTR* происходит от слова "CounTeR" — "*счетчик*". Этот режим является модификацией режима *OFB*. Единственное отличие от *OFB* заключается в том, что в режиме *CTR* шифруется не предыдущий *выход* шифра, а *счетчик*, увеличиваемый на каждом шаге на 1. Первоначальное *значение* счетчика определяется некоторым инициализирующим значением Y0. Общая формула выглядит следующим образом:

Yi=f(Yi-1+1,K),

zi=j старших бит Yi

Преимущество режима *CTR* состоит в том, что любой элемент последовательности z может быть вычислен непосредственно. Этот факт связан с тем, что на каждом шаге Yi увеличивается на единицу, и, следовательно, если нам известен номер шага i, то *значение* Yi можно вычислить непосредственно, зная Y0 и i по формуле:

Yi=f(Y0+i,K),

Это дает возможность шифровать и дешифровать любые фрагменты сообщения независимо друг от друга.

### Алгоритм RC4

*Алгоритм* **RC4** разработан Р.Ривестом специально как *генератор* потока ключевой информации с ключом переменной длины. *Генераторы псевдослучайных чисел*, построенные с помощью таких алгоритмов, как *RC4*, как правило, значительно быстрее генераторов, основанных на блочных шифрах. *Алгоритм* *RC4* широко применяется в различных системах защиты информации, в компьютерных сетях (например, в протоколе *SSL*, для шифрования паролей в *Windows* NT, и др.). *Алгоритм* *RC4* довольно прост и мы полностью рассмотрим принцип его действия.

*RC4* — фактически *класс* алгоритмов, определяемых размером его блока или слова – параметром n. Обычно n = 8, но можно использовать и другие значения. Для упрощения анализа *алгоритма примем* n=4. Внутреннее состояние *RC4* состоит из массива размером 2n слов и двух счетчиков, каждый размером в одно *слово*. Два счетчика, оба при n=4 4-битовые, назовем i и j. Все вычисления проводятся по модулю 2n.

*Массив* используется как *таблица* замен, называемая *S-бокс*, и далее будет обозначаться как S. В каждый момент времени *таблица* S содержит все возможные n-битовые (в нашем случае 4-битовые) числа в перемешанном виде. Конкретная *перестановка* значений в таблице определяется ключом. Так как каждый элемент таблицы принимает значения в промежутке 0 до 15, то его можно трактовать двояко: либо как число, либо как номер другого элемента в таблице.

*Алгоритм* *RC4* состоит из двух этапов. На первом, подготовительном этапе производится *инициализация* таблицы замен S. На втором, основном этапе вычисляются псевдослучайные числа.

Посмотрим, как инициализируется *таблица* S. Вначале она заполняется последовательно числами от 0 до 15. *Ключ* представляется в виде последовательности 4-битовых слов, которыми заполняется другой *массив* K, такого же размера, как S. Если *ключ* оказался короче, чем надо, он повторяется нужное число раз. Затем выполняются следующие действия ( *алгоритм 1* ):

1. j = 0; i =0;

2. j = (j + Si + Ki) mod 16;

3. поменять местами Si и Sj;

4. i = i +1;

5. если i <16, то перейти на п.2

В результате выполнения этого алгоритма производится начальное заполнение таблицы замен S, причем это начальное перемешивание значений производится в зависимости от секретного ключа.

После того, как *таблица* S подготовлена, можно начинать генерацию случайных n-битовых слов. Для этого счетчикам i и j присваивается начальное *значение* 0. Затем для получения каждого нового значения zi выполняются следующие действия ( *алгоритм 2* ):

i = (i + 1) mod 16;

j = (j + Si) mod 16;

поменять местами Si и Sj;

a = (Si + Sj) mod 16;

zi = Sa.

Полученное 4-битовое *значение* zi может использоваться в качестве ключа для шифрования очередного 4-битового блока входного потока данных.

Например, пусть *секретный ключ* состоит из шести 4-битовых значений (приведем их в десятичном виде): 1, 2, 3, 4, 5, 6. Попробуем сгенерировать последовательность чисел по алгоритму *RC4*.

Заполним таблицу S последовательно числами от 0 до 15.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер элемента | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Значение | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Затем подготовим таблицу K, записав в нее *ключ* необходимое количество раз:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер элемента | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Значение | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 |

Затем перемешаем содержимое таблицы S. Для этого будем использовать *алгоритм* 1, описанный выше. Процесс выполнения представим в виде трассировочной таблицы ([таблица 8.2](https://www.intuit.ru/studies/courses/691/547/lecture/12385?page=2" \l "table.8.2)), в которой укажем все производимые действия. При выполнении вычислений необходимо помнить, что все *операции* сложения выполняются по модулю 16.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 8.2. Подготовительный этап (инициализация таблицы замен) алгоритма RC4 | | | |
| **Номер пункта алг.** | **Выполняемое действие (по mod 16)** | **Новое значение i** | **Новое значение j** |
| 1 | j = 0; i =0 | 0 |  |
| 2 | j = j + Si + Ki = 0 + 0 + 1 = 1 |  | 1 |
| 3 | Поменять местами Si и Sj, то есть S0 и S1 |  |  |
| 4 | i = i +1 | 1 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = j + Si + Ki = 1 + 0+ 2 = 3 |  | 3 |
| 3 | Поменять местами Si и Sj, то есть S1 и S3 |  |  |
| 4 | i = i +1 | 2 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (3 + 2 + 3) mod 16 = 8 |  | 8 |
| 3 | Поменять местами Si и Sj , то есть S2 и S8 |  |  |
| 4 | i = i +1 | 3 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (8 + 0 + 4) mod 16 = 12 |  | 12 |
| 3 | Поменять местами Si и Sj , то есть S3 и S12 |  |  |
| 4 | i = i +1 | 4 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (12 + 4 + 5) mod 16 = 5 |  | 5 |
| 3 | Поменять местами Si и Sj , то есть S4 и S5 |  |  |
| 4 | i = i +1 | 5 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (5 + 4 + 6) mod 16 = 15 |  | 15 |
| 3 | Поменять местами Si и Sj , то есть S5 и S15 |  |  |
| 4 | i = i +1 | 6 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (15 + 6 + 1) mod 16 = 6 |  | 6 |
| 3 | Поменять местами Si и Sj , то есть S6 и S6 |  |  |
| 4 | i = i +1 | 7 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (6 + 7 + 2) mod 16 = 15 |  | 15 |
| 3 | Поменять местами Si и Sj , то есть S7 и S15 |  |  |
| 4 | i = i +1 | 8 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (15 + 2 + 3) mod 16 = 4 |  | 4 |
| 3 | Поменять местами Si и Sj , то есть S8 и S4 |  |  |
| 4 | i = i +1 | 9 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (4 + 9 + 4) mod 16 = 1 |  | 1 |
| 3 | Поменять местами Si и Sj , то есть S9 и S1 |  |  |
| 4 | i = i +1 | 10 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (1 + 10 + 5) mod 16 = 0 |  | 0 |
| 3 | Поменять местами Si и Sj , то есть S10 и S0 |  |  |
| 4 | i = i +1 | 11 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (0 + 11 + 6) mod 16 = 1 |  | 1 |
| 3 | Поменять местами Si и Sj , то есть S11 и S1 |  |  |
| 4 | i = i +1 | 12 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (1 + 0 + 1) mod 16 = 2 |  | 2 |
| 3 | Поменять местами Si и Sj , то есть S12 и S2 |  |  |
| 4 | i = i +1 | 13 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (2 + 13 + 2) mod 16 = 1 |  | 1 |
| 3 | Поменять местами Si и Sj , то есть S13 и S1 |  |  |
| 4 | i = i +1 | 14 |  |
| 5 | i < 16, поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (1 + 14 + 3) mod 16 = 2 |  | 2 |
| 3 | Поменять местами Si и Sj , то есть S14 и S2 |  |  |
| 4 | i = i +1 | 15 |  |
| 5 | i < 16 , поэтому перейти на п.2 |  |  |
| 2 | j = (j + Si + Ki ) mod 16= (2 + 7 + 4) mod 16 = 13 |  | 13 |
| 3 | Поменять местами Si и Sj , то есть S15 и S13 |  |  |
| 4 | i = i +1 | 16 |  |
| 5 | i < 16 – неверно, поэтому закончить |  |  |

После выполнения алгоритма 1 получим инициализированную и подготовленную к основному этапу таблицу S:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер элемента | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Значение | 10 | 13 | 14 | 12 | 2 | 15 | 6 | 4 | 5 | 3 | 1 | 9 | 8 | 7 | 0 | 11 |

После того как *таблица* S подготовлена, можно начинать генерацию случайных 4-битовых слов. Вычислим первые 5 чисел псевдослучайной последовательности, используя *алгоритм* 2. Результаты вычисления последовательности значений также представим в виде таблицы ([таблица 8.3](https://www.intuit.ru/studies/courses/691/547/lecture/12385?page=3" \l "table.8.3))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 8.3. Основной этап (вычисление элементов псевдослучайной *последовательности) алгоритма* RC4 | | | | |
|  | **Выполняемое действие (по mod 16)** | **Новое знач. i** | **Новое знач. j** | **Новое знач. а** |
| Вычисление z1 | 1. i = (i + 1) =0+1=1 | 1 |  |  |
| 2. j = (j + Si) mod 16=(0+13) mod 16=13 |  | 13 |  |
| 3. Поменять местами S1 и S13 |  |  |  |
| 4. a = (Si + Sj )mod 16=(7+13)mod 16=4 |  |  | 4 |
| 5. z1 = S4 =2 |  |  |  |
| Вычисление z2 | 1. i = (i + 1) =1+1=2 | 2 |  |  |
| 2. j = (j + Si) mod 16=(13+14) mod 16=11 |  | 11 |  |
| 3. Поменять местами S2 и S11 |  |  |  |
| 4. a = (Si + Sj )mod 16=(9+14)mod 16=7 |  |  | 7 |
| 5. z2 = S7=4 |  |  |  |
| Вычисление z3 | 1. i = (i + 1) =2+1=3 | 3 |  |  |
| 2. j = (j + Si) mod 16=(11+12) mod 16=7 |  | 7 |  |
| 3. Поменять местами S3 и S7 |  |  |  |
| 4. a = (Si + Sj )mod 16=(4+12)mod 16=0 |  |  | 0 |
| 5. z3 = S0=10 |  |  |  |
| Вычисление z4 | 1. i = (i + 1) =3+1=4 | 4 |  |  |
| 2. j = (j + Si) mod 16=(7+2) mod 16=9 |  | 9 |  |
| 3. Поменять местами S4 и S9 |  |  |  |
| 4. a = (Si+ Sj )mod 16=(3+2)mod 16=5 |  |  | 5 |
| 5. z4= S5=15 |  |  |  |
| Вычисление z5 | 1. i = (i + 1) =4+1=5 | 5 |  |  |
| 2. j = (j + Si) mod 16=(9+15) mod 16=8 |  | 8 |  |
| 3. Поменять местами S5 и S8 |  |  |  |
| 4. a = (Si+ Sj)mod 16=(5+15)mod 16=4 |  |  | 4 |
| 5. z5 = S4 =3 |  |  |  |

В результате первые пять значений получились следующие: 2, 4, 10, 15, 3. При необходимости получения большего количества случайных чисел можно продолжить вычисления дальше. При n=4 генерируемые числа будут иметь размер 4 бита, то есть иметь значения от 0 до 15.

В рассмотренном примере размер n слова или блока алгоритма принимался равным четырем. Это *значение* можно брать и другим, например 8 или 16. В случае использования n=8 *таблица* замен S должна состоять из 28=256 значений, а элементами таблицы замен должны быть числа от 0 до 255. Размер счетчиков i и j должен также изменить до восьми *бит* (максимальное *значение* – 255 ). Кроме того, все вычисления в случае n=8 необходимо выполнять по модулю 256. Аналогичные изменения в алгоритме необходимо производить и при других значениях параметра n.

*Алгоритм* *RC4* тщательно изучался криптоаналитиками. В нем не обнаружено каких бы то ни было слабых мест. Помимо высокой устойчивости к криптоанализу, этот *алгоритм* очень быстр и может использоваться для генерации ключевой последовательности при *потоковом шифровании*.

### Генераторы настоящих случайных чисел в криптографии

Генераторы ПСЧ находят широкое применение в криптографии, например, при *потоковом шифровании*. Однако иногда бывает необходимо генерировать совершенно непредсказуемые или попросту абсолютно случайные числа. Такие генераторы называются *генераторами случайных чисел* (*random number generator*) или сокращённо ГСЧ (*RNG*). *Генератор* настоящих случайных чисел в зависимости от некоторого инициализирующего значения выдает последовательность, которая не может быть впоследствии повторена.

Одной из главных областей применения генераторов случайных чисел является формирование уникальных ключей для шифрования. В любой системе передачи секретных данных требуется множество ключей для всех пользователей системы. В принципе ключи шифрования можно получать с помощью *генератора псевдослучайных чисел*, используя, например, *алгоритм* *RC4* или блочный *шифр* в режиме *OFB*. Однако, если противник вдруг узнает *ключ*, использовавшийся для генерации псевдослучайных ключей, он сможет сгенерировать точно такие же ключи и вскрыть все передаваемые в системе сообщения. Следовательно, секретные ключи должны быть действительно случайными. Поэтому задача генерирования последовательностей настоящих случайных чисел представляет большой интерес для разработчиков криптосистем.

Наилучшие характеристики будут иметь генераторы случайных чисел, основанные на естественных случайностях реального мира. Например, можно создать ГСЧ, основанные на следующих данных:

* количество импульсов счетчика Гейгера за единицу времени, например, за одну секунду;
* числа, оказывающиеся на верхней грани игрального кубика при произвольном броске;
* количество самолетов, пролетающих над определенным районом в единицу времени, например, месяц.

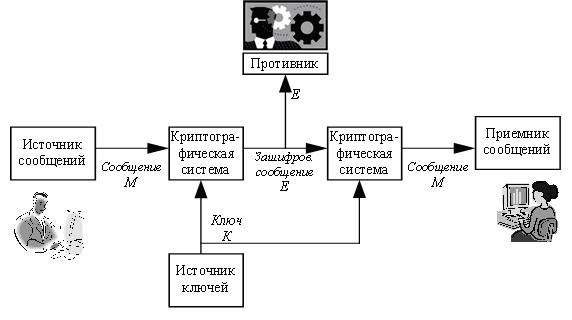
Кроме того, параметры других различных физических явлений могут быть положены в основу ГСЧ. К сожалению, многие методики получения настоящих случайных чисел не могут быть реализованы на практике, так как используемый в криптографических целях *генератор* должен быть компактным, быстрым (генерировать числа за доли секунды), независящим от внешних факторов и условий окружающей среды.

Тем не менее, разработчики интегральных схем конструируют и производят аппаратные ГСЧ, основанные на разных принципах. Например, разработан способ с использованием двух конденсаторов типа "металл - диэлектрик - полупроводник". Случайное *значение* является функцией разности зарядов этих конденсаторов. В другом устройстве обрабатывается и используется *значение* температурного шума полупроводникового диода.

Предлагаются и программно-аппаратные методы для получения случайных чисел. Известны методы, основанные на шуме звуковой карты персонального компьютера, значении счётчика тактов процессора, скорости вращения жесткого диска компьютера, значении *системного таймера*, скорости нажатия отдельных клавиш клавиатуры или движения мыши. Полученные каким-либо образом случайные данные обрабатываются криптостойким генератором ПСЧ и только после такой обработки используются.

### Управление секретными ключами

Рассмотрим еще раз общую структуру секретной системы, представленную в ["Простейшие методы шифрования с закрытым ключом"](https://www.intuit.ru/studies/courses/691/547/lecture/12373). Схема структуры такой системы изображена на [рис. 8.3](https://www.intuit.ru/studies/courses/691/547/lecture/12385?page=4" \l "image.8.3).



**Рис. 8.3.**Общая структура секретной системы, использующей симметричное шифрование

Отправитель, представляющий собой источник сообщений, и получатель (приемник зашифрованных сообщений) договариваются о выборе приемлемого шифра и ключа. Затем отправитель шифрует свое сообщение с использованием выбранного алгоритма шифрования и ключа и пересылает полученный шифротекст по (открытому) каналу связи. Получатель расшифровывает его, используя *шифр* и *ключ*.

Противник, скорее всего, может перехватить зашифрованное сообщение, так как предполагается, что оно передается по открытому каналу связи. В этом случае криптоаналитик противника может попытаться вскрыть шифротекст. Будем предполагать, что отправитель и *получатель сообщения* используют достаточно *надежный шифр*, и что *вероятность* его вскрытия невысока. В этом случае *безопасность* шифрования полностью зависит от безопасности ключа. Раскрытие ключа приведет к раскрытию передаваемых данных. Таким образом, *ключ* должен храниться в секрете до тех пор, пока он используется для закрытия данных. Поэтому для первоначального распределения ключей необходим надежный *канал связи*.

Таким образом, принципиальной является *надежность* канала передачи ключа участникам секретных переговоров. Самым надежным способом первоначального распределения ключей является обмен ключами при личной встрече абонентов *сети передачи данных*. Для доставки ключей можно также использовать специальных курьеров. Если в обмене секретными сообщениями планируется участие небольшого количества сторон, например, двух или трех, то оба указанных способа вполне допустимы. Если же количество взаимодействующих абонентов велико, то задача распределения ключей превращается в настоящую проблему.

При использовании секретных ключей существуют и другие трудности. Например, ключи должны время от времени меняться. Это связано с тем, что чем дольше используется *ключ*, тем больше *вероятность* его компрометации (раскрытия). Чем дольше используется *ключ*, тем больше потери от его компрометации, так как тем большее количество сообщений сможет раскрыть *злоумышленник* при получении ключа. Даже если *ключ* не будет раскрыт, проводить *криптоанализ* противнику удобнее, имея в своем распоряжении достаточное количество сообщений, зашифрованных одним и тем же ключом. Оптимальным считается использовать для каждого сеанса обмена зашифрованными сообщениями свой уникальный *ключ* – так называемый *сеансовый ключ*. Но где взять такое количество ключей для большой телекоммуникационной сети и как их распределять?

Таким образом, при большом числе взаимодействующих сторон требуется предварительная рассылка значительного количества ключей, а также последующее их хранение и при необходимости – смена. Предположим, в локальной сети имеется 100 пользователей. Пусть пользователи сети желают обмениваться секретными данными друг с другом по принципу "каждый с каждым". В этом случае для каждой пары пользователей необходим свой *секретный ключ* для шифрования сообщений. Из ста пользователей можно составить 100x99/2=4950 пар, следовательно, в системе передачи данных будут использоваться 4950 разных секретных ключей. Все эти ключи должны быть сгенерированы и распределены надежным образом. Кроме того, каждый из ста пользователей должен помнить 99 разных ключей, каждый для определенного абонента. Если же в обмене сообщениями участвует не сто, а тысяча человек, то задача управления ключами становится чрезвычайно сложной.

В связи с указанными трудностями на практике применяются специальные автоматизированные *системы управления ключами*. Такие системы позволяют генерировать ключи, хранить их и архивировать, восстанавливать утерянные ключи, заменять или изымать из обращения старые и ненужные ключи. Важнейшей частью системы управления ключами является *центр распределения ключей* (Key *Distribution* *Center* – *KDC*), функциями которого являются генерация, распределение и передача ключей.

Специалистами разработаны специальные процедуры (или протоколы), которые позволяют центру распределения ключей доставлять пользователям ключи для проведения отдельных сеансов связи ( *сеансовые ключи* ). К сожалению, все протоколы с использованием симметричного шифрования имеют те или иные недостатки. Рассмотрим один из возможных *протоколов обмена ключами*.

Предположим, при вступлении в сообщество пользователей сети обмена данными *центром распределения ключей* всем новым абонентам выдается индивидуальный *секретный ключ*. Вот как может выглядеть процедура распределения секретных ключей для проведения сеанса связи между двумя абонентами сети с использованием центра распределения ключей (для краткости будем называть его просто Центром):

1. Абонент А обращается в Центр и запрашивает сеансовый ключ для связи с абонентом Б.
2. В Центре создается случайный сеансовый ключ. Зашифровываются две копии этого сеансового ключа – одна с использованием секретного ключа абонента А, другая – с использованием секретного ключа абонента Б. Затем обе зашифрованные копии пересылаются из Центра абоненту А.
3. Абонент А расшифровывает свою копию сеансового ключа и пересылает вторую зашифрованную копию абоненту Б.
4. Абонент Б расшифровывает свою копию сеансового ключа.
5. Абоненты А и Б используют полученный сеансовый ключ для секретного обмена информацией.

Указанный протокол достаточно прост и может быть автоматизирован с помощью, например, программы передачи данных. Однако приведенная процедура распределения сеансовых ключей имеет несколько явных недостатков.

Первым недостатком данной системы является то, что Центр участвует во всех обменах. Сбои в работе Центра нарушат работу всей системы.

Вторым недостатком является то, *центр распределения ключей* должен хранить в каком-либо виде секретные ключи всех абонентов сети. Если *злоумышленник* найдет *доступ* к секретным ключам пользователей системы ("взломает" систему, подкупит администратора и т.д.), то он сможет читать и изменять все передаваемые сообщения.

И, наконец, остается проблема первоначального распределения секретных ключей при вступлении пользователя в *сеть*. Первоначальный *секретный ключ* должен быть доставлен по абсолютно надежному каналу связи, иначе весь протокол теряет всякий смысл. Хорошо, если первоначальный *ключ* может быть выдан лично новому пользователю, однако в некоторых случаях это невозможно, например, при территориальной распределенности *сети передачи данных*.

Эти и другие недостатки алгоритмов симметричного шифрования обнаружились разработчиками телекоммуникационных сетей при первых попытках построения защищенных систем передачи данных в 70-х годах XX века. Решением проблемы распределения ключей (а также некоторых других серьезных проблем) стало использование несимметричных алгоритмов шифрования, с которыми мы начнем знакомиться уже в следующей лекции.

### Ключевые термины

**CTR** – режим работы блочного шифра, который позволяет генерировать ключи при поточном шифрования информации.

**LFSR** (*linear* *feedback* *shift register* ) – линейный *сдвиговый регистр* с обратной связью.

**OFB** – режим работы блочного шифра, который позволяет генерировать ключи при поточном шифрования информации.

**Алгоритм RC4** – *алгоритм* генерации псевдослучайных чисел. Может использоваться для генерации ключей при поточном шифровании.

**Линейный сдвиговый регистр с обратной связью** (*linear* *feedback* *shift register* – *LFSR*) – вариант сдвигового регистра с обратной связью. *Обратная связь* в таком регистре реализуется просто как сумма по модулю 2 всех (или некоторых) битов регистра.

**Сдвиговый регистр с обратной связью** состоит n-битного сдвигового регистра и устройства обратной связи. Когда нужно извлечь *бит*, все биты *регистра сдвигаются* вправо на одну позицию. Новый крайний слева *бит* определяется функцией обратной связи от остальных битов. Сдвиговые регистры с обратной связью могут применяться для получения потока псевдослучайных *бит*.

### Краткие итоги

Для генерации ключевого потока при поточном шифровании могут использоваться сдвиговые регистры с обратной связью. *Сдвиговый регистр* с обратной связью состоит n-битного сдвигового регистра и устройства обратной связи. Когда нужно извлечь *бит*, все биты *регистра сдвигаются* вправо на одну позицию. Новый крайний слева *бит* определяется функцией обратной связи от остальных битов. Сдвиговые регистры с обратной связью могут применяться для получения потока псевдослучайных *бит*.

Можно использовать любой блочный *алгоритм*, например *AES* или ГОСТ 28147-89, для поточного шифрования информации, используя режимы *OFB* и *CTR* блочных шифров. Эти режимы позволяют на основе блочного алгоритма шифрования с использованием секретного ключа генерировать *поток* ключевой информации, который затем может использоваться в качестве гаммы при шифровании.

Существуют также алгоритмы генерации псевдослучайных чисел, созданные специально для криптографии. Одним из наиболее известных алгоритмов такого рода является *алгоритм* *RC4*. Это *алгоритм* с ключом переменной длины. На основе ключа *алгоритм* вырабатывает псевдослучайные числа, используя *операции* сложения (по некоторому модулю), перестановки и замены элементов внутреннего массива. *Генераторы псевдослучайных чисел*, построенные с помощью таких алгоритмов, как *RC4*, как правило, значительно быстрее генераторов, основанных на блочных шифрах.

Во всех системах, использующих *симметричное шифрование*, принципиально важной является *надежность* канала передачи ключа участникам секретных переговоров. Основными проблемами, возникающими в системах управления ключами, являются следующие: большое количество и территориальная распределенность взаимодействующих абонентов, необходимость смены ключей от сеанса к сеансу. Решением проблемы распределения ключей стало использование несимметричных алгоритмов шифрования.

### Набор для практики

#### Вопросы для самопроверки

1. Перечислите основные характеристики, достоинства и недостатки каждого из рассмотренных в лекции генераторов псевдослучайных чисел.
2. Каким образом могут использоваться для получения псевдослучайных чисел сдвиговые регистры с обратной связью? Объясните их принцип работы.
3. Каким образом блочный шифр в режиме *OFB* может быть использован для поточного шифрования данных?
4. Как организуется режим *CTR* блочного шифра?
5. В чем разница между генераторами случайных и псевдослучайных чисел?
6. Можно ли использовать генератор настоящих случайных чисел для получения гаммы при *потоковом шифровании*? Почему?
7. Для каких криптографических целей могут быть использованы генераторы настоящих случайных чисел?
8. Какие проблемы возникают при управлении секретными ключами в системах обмена шифрованными сообщениями?