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.

**Лекции №1: Простейшие методы шифрования с закрытым ключом**

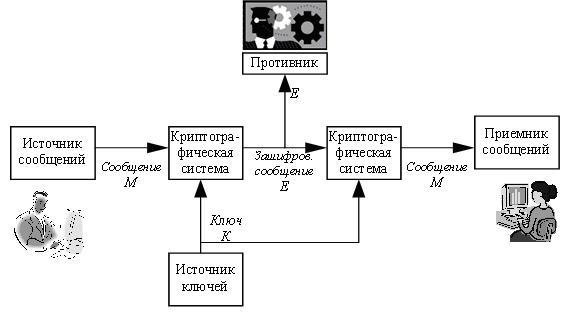
### Задание СРС :

1. Изучить основные понятия и принципы простейших методов симметричного (с закрытым ключом) шифрования:  
   – Шифр Цезаря  
   – Моноалфавитный шифр  
   – Многоалфавитный шифр (например, шифр Виженера)  
   – Побитовые логические операции (например, XOR-шифр)
2. Проанализировать достоинства и недостатки каждого метода.
3. Создать презентацию (10–15 слайдов) с визуальными примерами работы указанных шифров.
4. Кратко изложить материал в виде конспекта (1–2 страницы).
5. Подготовить устное объяснение выбранного метода шифрования на примере.

**Цель лекции**: познакомить студента с простейшими методами симметричного шифрования и конкретными шифрами, известными человечеству уже не одно столетие.

### Общая схема симметричного шифрования

Классическая, или одноключевая *криптография* опирается на использование **симметричных алгоритмов шифрования**, в которых *шифрование* и *расшифрование* отличаются только порядком выполнения и направлением некоторых шагов. Эти алгоритмы используют один и тот же секретный элемент (*ключ*), и второе действие (*расшифрование*) является простым обращением первого (шифрования). Поэтому обычно каждый из участников обмена может как зашифровать, так и расшифровать сообщение. Схематичная структура такой системы представлена на [рис. 2.1](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=1" \l "image.2.1).



**Рис. 2.1.**Общая структура секретной системы, использующей симметричное шифрование

На передающей стороне имеются источник сообщений и источник ключей. Источник ключей выбирает конкретный *ключ* К среди всех возможных ключей данной системы. Этот *ключ* К передается некоторым способом принимающей стороне, причем предполагается, что его нельзя перехватить, например, *ключ* передается специальным курьером (поэтому *симметричное шифрование* называется также шифрованием с *закрытым ключом*). Источник сообщений формирует некоторое сообщение М, которое затем шифруется с использованием выбранного ключа. В результате процедуры шифрования получается зашифрованное сообщение Е (называемое также криптограммой). Далее криптограмма Е передается *по* каналу связи. Так как *канал связи* является открытым, незащищенным, например, *радиоканал* или компьютерная *сеть*, то передаваемое сообщение может быть перехвачено противником. На принимающей стороне криптограмму Е с помощью ключа расшифровывают и получают исходное сообщение М.

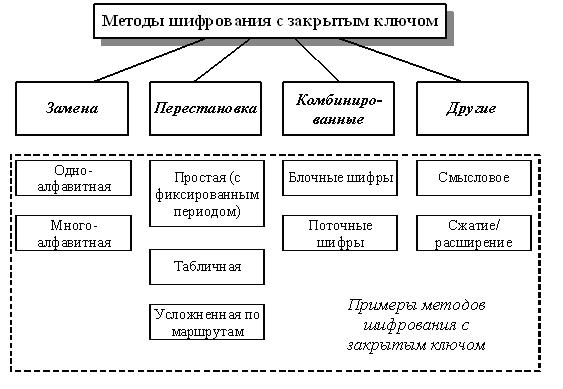
Если М – сообщение, К – *ключ*, а Е – зашифрованное сообщение, то можно записать

E=f(M,K)

то есть зашифрованное сообщение Е является некоторой функцией от исходного сообщения М и ключа К. Используемый в криптографической системе метод или *алгоритм* шифрования и определяет функцию f в приведенной выше формуле.

*По* причине большой избыточности естественных языков непосредственно в зашифрованное сообщение чрезвычайно трудно внести осмысленное изменение, поэтому классическая *криптография* обеспечивает также защиту от навязывания ложных данных. Если же естественной избыточности оказывается недостаточно для надежной защиты сообщения от модификации, *избыточность* может быть искусственно увеличена путем добавления к сообщению специальной контрольной комбинации, называемой *имитовставкой*.

Известны разные методы шифрования с закрытым ключом [рис. 2.2](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=1" \l "image.2.2). На практике часто используются алгоритмы перестановки, подстановки, а также комбинированные методы.



**Рис. 2.2.**Методы шифрования с закрытым ключом

В методах перестановки символы исходного текста меняются местами друг с другом *по* определенному правилу. В методах замены (или подстановки) символы открытого текста заменяются некоторыми эквивалентами шифрованного текста. С целью повышения надежности шифрования текст, зашифрованный с помощью одного метода, может быть еще раз зашифрован с помощью другого метода. В этом случае получается комбинированный или композиционный *шифр*. Применяемые на практике в настоящее время блочные или поточные симметричные шифры также относятся к комбинированным, так как в них используется несколько операций для зашифрования сообщения.

Основное отличие современной криптографии от криптографии "докомпьютерной" заключается в том, что раньше криптографические алгоритмы оперировали символами естественных языков, например, буквами английского или русского алфавитов. Эти буквы переставлялись или заменялись другими *по* определенному правилу. В современных криптографических алгоритмах используются *операции* над двоичными знаками, то есть над нулями и единицами. В настоящее время основными операциями при шифровании также являются *перестановка* или *подстановка*, причем для повышения надежности шифрования эти *операции* применяются вместе (комбинируются) и помногу раз циклически повторяются.

Принципы построения современных блочных шифров сформулированы в ["Принципы построения блочных шифров с закрытым ключом"](https://www.intuit.ru/studies/courses/691/547/lecture/12375), ["Алгоритмы шифрования DES и AES"](https://www.intuit.ru/studies/courses/691/547/lecture/12377), ["Алгоритм криптографического преобразования данных ГОСТ 28147-89"](https://www.intuit.ru/studies/courses/691/547/lecture/12379), а в этой лекции рассматриваются шифры подстановки и перестановки, применяемые человеком с древнейших времен. Мы должны познакомиться с этими шифрами, так как процедуры подстановки и перестановки используются в качестве составных операций и в современных блочных шифрах.

### Методы замены

Методы шифрования заменой (подстановкой) основаны на том, что символы исходного текста, обычно разделенные на блоки и записанные в одном алфавите, заменяются одним или несколькими символами другого алфавита в соответствии с принятым правилом преобразования.

#### Одноалфавитная замена

Одним из важных подклассов методов замены являются *одноалфавитные* (или моноалфавитные) подстановки, в которых устанавливается однозначное соответствие между каждым знаком ai исходного алфавита сообщений A и соответствующим знаком ei зашифрованного текста E. Одноалфавитная подстановка иногда называется также простой заменой, так как является самым простым шифром замены.

Примером одноалфавитной замены является шифр Цезаря, рассмотренный ранее. В рассмотренном в ["Основные понятия криптографии"](https://www.intuit.ru/studies/courses/691/547/lecture/12371)примере первая строка является исходным алфавитом, вторая (с циклическим сдвигом на k влево) – вектором замен.

В общем случае при одноалфавитной подстановке происходит однозначная замена исходных символов их эквивалентами из вектора замен (или таблицы замен). При таком методе шифрования ключом является используемая таблица замен.

Подстановка может быть задана с помощью таблицы, например, как показано на [рис. 2.3](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "image.2.3).



**Рис. 2.3.**Пример таблицы замен для двух шифров

В таблице на [рис. 2.3](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "image.2.3) на самом деле объединены сразу две таблицы. Одна (шифр 1) определяет замену русских букв исходного текста на другие русские буквы, а вторая (шифр 2) – замену букв на специальные символы. Исходным алфавитом для обоих шифров будут заглавные русские буквы (за исключением букв "Ё" и "Й"), пробел и точка.

Зашифрованное сообщение с использованием любого шифра моноалфавитной подстановки получается следующим образом. Берется очередной знак из исходного сообщения. Определяется его позиция в столбце "Откр. текст" таблицы замен. В зашифрованное сообщение вставляется шифрованный символ из этой же строки таблицы замен.

Попробуем зашифровать сообщение "ВЫШЛИТЕ ПОДКРЕПЛЕНИЕ" c использованием этих двух шифров ([рис. 2.4](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "image.2.4)). Для этого берем первую букву исходного сообщения "В". В таблице на [рис. 2.3](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "image.2.3) в столбце "Шифр 1" находим для буквы "В" заменяемый символ. Это будет буква "О". Записываем букву "О" под буквой "В". Затем рассматриваем второй символ исходного сообщения – букву "Ы". Находим эту букву в столбце "Откр. текст" и из столбца "Шифр 1" берем букву, стоящую на той же строке, что и буква "Ы". Таким образом получаем второй символ зашифрованного сообщения – букву "Н". Продолжая действовать аналогично, зашифровываем все исходное сообщение ([рис. 2.4](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "image.2.4)).



**Рис. 2.4.**Пример шифрования методом прямой замены

Полученный таким образом текст имеет сравнительно низкий уровень защиты, так как исходный и зашифрованный тексты имеют одинаковые статистические закономерности. При этом не имеет значения, какие символы использованы для замены – перемешанные символы исходного алфавита или таинственно выглядящие знаки.

Зашифрованное сообщение может быть вскрыто путем так называемого *частотного криптоанализа*. Для этого могут быть использованы некоторые статистические данные языка, на котором написано сообщение.

Известно, что в текстах на русском языке наиболее часто встречаются символы О, И. Немного реже встречаются буквы Е, А. Из согласных самые частые символы Т, Н, Р, С. В распоряжении криптоаналитиков имеются специальные таблицы частот встречаемости символов для текстов разных типов – научных, художественных и т.д.

Криптоаналитик внимательно изучает полученную криптограмму, подсчитывая при этом, какие символы сколько раз встретились. Вначале наиболее часто встречаемые знаки зашифрованного сообщения заменяются, например, буквами О. Далее производится попытка определить места для букв И, Е, А. Затем подставляются наиболее часто встречаемые согласные. На каждом этапе оценивается возможность "сочетания" тех или иных букв. Например, в русских словах трудно найти четыре подряд гласные буквы, слова в русском языке не начинаются с буквы Ы и т.д. На самом деле для каждого естественного языка (русского, английского и т.д.) существует множество закономерностей, которые помогают раскрыть специалисту зашифрованные противником сообщения.

Возможность однозначного криптоанализа напрямую зависит от длины перехваченного сообщения. Посмотрим, с чем это связано. Пусть, например, в руки криптоаналитиков попало зашифрованное с помощью некоторого шифра одноалфавитной замены сообщение:

ТНФЖ.ИПЩЪРЪ

Это сообщение состоит из 11 символов. Пусть известно, что эти символы составляют целое сообщение, а не фрагмент более крупного текста. В этом случае наше зашифрованное сообщение состоит из одного или нескольких целых слов. В зашифрованном сообщении символ Ъ встречается 2 раза. Предположим, что в открытом тексте на месте зашифрованного знака Ъ стоит гласная О, А, И или Е. Подставим на место Ъ эти буквы и оценим возможность дальнейшего криптоанализа [таблица 2.1](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "table.2.1)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 2.1. Варианты первого этапа криптоанализа | | | | | | | | | | |
| Зашифрованное сообщение | | | | | | | | | | |
| Т | Н | Ф | Ж | . | И | П | Щ | Ъ | Р | Ъ |
| После замены Ъ на О | | | | | | | | | | |
|  |  |  |  |  |  |  |  | О |  | О |
| После замены Ъ на А | | | | | | | | | | |
|  |  |  |  |  |  |  |  | А |  | А |
| После замены Ъ на И | | | | | | | | | | |
|  |  |  |  |  |  |  |  | И |  | И |
| После замены Ъ на Е | | | | | | | | | | |
|  |  |  |  |  |  |  |  | Е |  | Е |

Все приведенные варианты замены могут встретиться на практике. Попробуем подобрать какие-нибудь варианты сообщений, учитывая, что в криптограмме остальные символы встречаются по одному разу ([таблица 2.2](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "table.2.2)).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 2.2. Варианты второго этапа криптоанализа | | | | | | | | | | |
| Зашифрованное сообщение | | | | | | | | | | |
| Т | Н | Ф | Ж | . | И | П | Щ | Ъ | Р | Ъ |
| Варианты подобранных дешифрованных сообщений | | | | | | | | | | |
| Ж | Д | И |  | С | У | М | Р | А | К | А |
| Д | Ж | О | Н | А |  | У | Б | И | Л | И |
| В | С | Е | Х |  | П | О | Б | И | Л | И |
| М | Ы |  | П | О | Б | Е | Д | И | Л | И |

Кроме представленных на [таблица 2.2](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "table.2.2) сообщений можно подобрать еще большое количество подходящих фраз. Таким образом, если нам ничего не известно заранее о содержании перехваченного сообщения малой длины, дешифровать его однозначно не получится.

Если же в руки криптоаналитиков попадает достаточно длинное сообщение, зашифрованное методом простой замены, его обычно удается успешно дешифровать. На помощь специалистам по вскрытию криптограмм приходят статистические закономерности языка. Чем длиннее зашифрованное сообщение, тем больше вероятность его однозначного дешифрования.

В ["Алгоритм криптографического преобразования данных ГОСТ 28147-89"](https://www.intuit.ru/studies/courses/691/547/lecture/12379)будут более подробно рассмотрены вопросы теоретической стойкости криптосистем, а также принципы построения идеальных криптосистем.

Интересно, что если попытаться замаскировать статистические характеристики открытого текста, то задача вскрытия шифра простой замены значительно усложнится. Например, с этой целью можно перед шифрованием "сжимать" открытый текст с использованием компьютерных программ-архиваторов.

С усложнением правил замены увеличивается надежность шифрования. Можно заменять не отдельные символы, а, например, двухбуквенные сочетания – биграммы. Таблица замен для такого шифра может выглядеть, как на [таблица 2.3](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "table.2.3).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 2.3. Пример таблицы замен для двухбуквенных сочетаний | | | |
| **Откр. текст** | **Зашифр. текст** | **Откр. текст** | **Зашифр. текст** |
| аа | кх | бб | пш |
| аб | пу | бв | вь |
| ав | жа | ... | ... |
| ... | ... | яэ | сы |
| ая | ис | яю | ек |
| ба | цу | яя | рт |

Оценим размер такой таблицы замен. Если исходный алфавит содержит N символов, то вектор замен для биграммного шифра должен содержать N2 пар "откр. текст – зашифр. текст". Таблицу замен для такого шифра можно также записать и в другом виде: заголовки столбцов соответствуют первой букве биграммы, а заголовки строк – второй, причем ячейки таблицы заполнены заменяющими символами. В такой таблице будет N строк и N столбцов ([таблица 2.4](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=2" \l "table.2.4)).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 2.4. Другой вариант задания таблицы замен для биграммного шифра | | | | |
|  | **а** | **б** | **...** | **я** |
| а | кх | цу | ... | ... |
| б | пу | пш | ... | ... |
| в | жа | вь | ... | ... |
| ... | ... | ... | ... | ... |
| ю | ... | ... | ... | ек |
| я | ис | ... | ... | рт |

Возможны варианты использования триграммного или вообще n-граммного шифра. Такие шифры обладают более высокой криптостойкостью, но они сложнее для реализации и требуют гораздо большего количества ключевой информации (большой объем таблицы замен). В целом, все n-граммные шифры могут быть вскрыты с помощью частотного криптоанализа, только используется статистика встречаемости не отдельных символов, а сочетаний из n символов.

#### Пропорциональные шифры

К одноалфавитным методам подстановки относятся **пропорциональные** или **монофонические шифры**, в которых уравнивается частота появления зашифрованных знаков для защиты от раскрытия с помощью частотного анализа. Для знаков, встречающихся часто, используется относительно большое число возможных эквивалентов. Для менее используемых исходных знаков может оказаться достаточным одного или двух эквивалентов. При шифровании замена для символа открытого текста выбирается либо случайным, либо определенным образом (например, по порядку).

При использовании пропорционального шифра в качестве замены символам обычно выбираются числа. Например, поставим в соответствие буквам русского языка трехзначные числа, как указано на [таблица 2.5](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=3" \l "table.2.5).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 2.5. Таблица замен для пропорционального шифра | | | | | | | | | | | |
| Символ | Варианты замены | | | | | Символ | Варианты замены | | | | |
| А | 760 | 128 | 350 | 201 |  | С | 800 | 767 | 105 |  |  | |  |
| Б | 101 |  |  |  |  | Т | 759 | 135 | 214 |  |  | |  |
| В | 210 | 106 |  |  |  | У | 544 |  |  |  |  | |  |
| Г | 351 |  |  |  |  | Ф | 560 |  |  |  |  | |  |
| Д | 129 |  |  |  |  | Х | 768 |  |  |  |  | |  |
| Е | 761 | 130 | 802 | 352 |  | Ц | 545 |  |  |  |  | |  |
| Ж | 102 |  |  |  |  | Ч | 215 |  |  |  |  | |  |
| З | 753 |  |  |  |  | Ш | 103 |  |  |  |  | |  |
| И | 762 | 211 | 131 |  |  | Щ | 752 |  |  |  |  | |  |
| К | 754 | 764 |  |  |  | Ъ | 561 |  |  |  |  | |  |
| Л | 132 | 354 |  |  |  | Ы | 136 |  |  |  |  | |  |
| М | 755 | 742 |  |  |  | Ь | 562 |  |  |  |  | |  |
| Н | 763 | 756 | 212 |  |  | Э | 750 |  |  |  |  | |  |
| О | 757 | 213 | 765 | 133 | 353 | Ю | 570 |  |  |  |  | |  |
| П | 743 | 766 |  |  |  | Я | 216 | 104 |  |  |  | |  |
| Р | 134 | 532 |  |  |  | Пробел | 751 | 769 | 758 | 801 | 849 | | 035… |

В этом случае сообщение

БОЛЬШОЙ СЕКРЕТ

может быть зашифровано следующим образом:

101757132562103213762751800761754134130759

В данном примере варианты замен для повторяющихся букв (например, "О") выбирались по порядку.

Интересно, что шифры, в которых производится замена букв несколькими символами, пропорционально встречаемости в открытом тексте, описывали итальянские ученые еще в XIV-XV веках.

Пропорциональные шифры более сложны для вскрытия, чем шифры простой одноалфавитной замены. Однако, если имеется хотя бы одна пара "открытый текст – шифротекст", вскрытие производится тривиально. Если же в наличии имеются только шифротексты, то вскрытие ключа, то есть нахождение таблицы замен, становится более трудоемким, но тоже вполне осуществимым.

#### Многоалфавитные подстановки

В целях маскирования естественной частотной статистики исходного языка применяется многоалфавитная подстановка, которая также бывает нескольких видов. В **многоалфавитных подстановках** для замены символов исходного текста используется не один, а несколько алфавитов. Обычно алфавиты для замены образованы из символов исходного алфавита, записанных в другом порядке.

Примером многоалфавитной подстановки может служить схема, основанная на использовании таблицы Вижинера. Этот метод, известный уже в XVI веке, был описан французом Блезом Вижинером в "Трактате о шифрах", вышедшем в 1585 году.

В этом методе для шифрования используется таблица, представляющая собой квадратную матрицу с числом элементов NxN, где N — количество символов в алфавите ([таблица 2.6](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=3" \l "table.2.6)). В первой строке матрицы записывают буквы в порядке очередности их в исходном алфавите, во второй — ту же последовательность букв, но с циклическим сдвигом влево на одну позицию, в третьей — со сдвигом на две позиции и т. д.

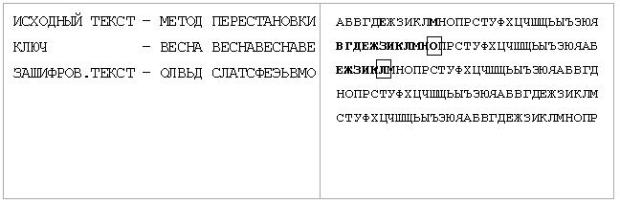
|  |  |
| --- | --- |
| Таблица 2.6. Подготовка таблицы шифрования | |
| АБВГДЕ........ | ..........ЭЮЯ |
| БВГДЕЖ........ | ..........ЮЯА |
| ВГДЕЖЗ........ | ..........ЯАБ |
| ГДЕЖЗИ......... | ..........АБВ |
| ДЕЖЭИК......... | ..........БВГ |
| ЕЖЗИКЛ......... | ..........ВГД |
| ........... | .......... |
| ЯАБВГД......... | ..........ЬЭЮ |

Для шифрования текста выбирают ключ, представляющий собой некоторое слово или набор символов исходного алфавита. Далее из полной матрицы выписывают подматрицу шифрования, включающую первую строку и строки матрицы, начальными буквами которых являются последовательно буквы ключа (например, если выбрать ключ "весна", то таблица шифрования будет такой, как на [таблица 2.7](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=3" \l "table.2.7)).

|  |
| --- |
| Таблица 2.7. Первый этап шифрования – составление подматрицы шифрования |
| АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ |
| ВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБ |
| ЕЖЗИКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГД |
| НОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИКЛМ |
| СТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИКЛМНОПР |

В процессе шифрования ([рис. 2.5](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=3" \l "image.2.5)) под каждой буквой шифруемого текста записывают буквы ключа, повторяющие ключ требуемое число раз, затем шифруемый текст по таблице шифрования ([таблица 2.7](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=3" \l "table.2.7)) заменяют буквами, расположенными на пересечениях линий, соединяющих буквы текста первой строки таблицы и буквы ключа, находящейся под ней.

Например, под первой буквой исходного текста "М" записана буква "В" ключа. В таблице кодирования находим столбец, начинающийся с "М" и строку, начинающуюся с "В". На их пересечении располагается буква "О". Она и будет первым символом зашифрованного сообщения (на [рис. 2.5](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=3" \l "image.2.5) эта буква выделена прямоугольной рамочкой). Следующая буква исходного сообщения – "Е", символ ключа – тоже "Е". Находим пересечение строки, начинающейся с "Е", и столбца, начинающегося с "Е". Это будет буква "Л" – второй символ зашифрованного сообщения.



**Рис. 2.5.**Механизм шифрования многоалфавитной заменой

Рассмотрим на примере процесс расшифрования сообщения по методу Вижинера. Пусть имеется зашифрованное с помощью ключа ВЕСНА сообщение КЕКХТВОЭЦОТССВИЛ (пробелы при шифровании пропущены). Расшифровка текста выполняется в следующей последовательности ([таблица 2.8](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=3" \l "table.2.8)):

* над буквами шифрованного текста сверху последовательно записывают буквы ключа, повторяя ключ требуемое число раз;
* в строке подматрицы таблицы Вижинера для каждой буквы ключа отыскивается буква, соответствующая знаку шифрованного текста. Находящаяся над ней буква первой строки и будет знаком расшифрованного текста;
* полученный текст группируется в слова по смыслу.

|  |  |
| --- | --- |
| Таблица 2.8. Механизм расшифрования | |
| КЛЮЧ | ВЕСНАВЕСНАВЕСНАВ |
| ЗАШИФРОВАННЫЙ ТЕКСТ | КЕКХТВОЭЦОТССВИЛ |
| РАСШИФРОВАННЫЙ ТЕКСТ | ЗАЩИТАИНФОРМАЦИИ |
| ИСХОДНЫЙ ТЕКСТ | ЗАЩИТА ИНФОРМАЦИИ |

Раскрыть шифр Вижинера, тем же способом, что и шифр одноалфавитной замены, невозможно, так как одни и те же символы открытого текста могут быть заменены различными символами зашифрованного текста. С другой стороны, различные буквы открытого текста могут быть заменены одинаковыми знаками зашифрованного текста.

Особенность данного метода многоалфавитной подстановки заключается в том, что каждый из символов ключа используется для шифрования одного символа исходного сообщения. После использования всех символов ключа, они повторяются в том же порядке. Если используется ключ из десяти букв, то каждая десятая буква сообщения шифруется одним и тем же символом ключа. Этот параметр называется *периодом* шифра. Если ключ шифрования состоит из одного символа, то при шифровании будет использоваться одна строка таблицы Вижинера, следовательно, в этом случае мы получим моноалфавитную подстановку, а именно шифр Цезаря.

С целью повышения надежности шифрования текста можно использовать подряд два или более зашифрования по методу Вижинера с разными ключами (составной шифр Вижинера).

На практике кроме метода Вижинера использовались также различные модификации этого метода. Например, шифр Вижинера с перемешанным один раз алфавитом. В этом случае для расшифрования сообщения получателю необходимо кроме ключа знать порядок следования символов в таблице шифрования.

Еще одним примером метода многоалфавитной подстановки является *шифр с бегущим ключом* или *книжный шифр*. В этом методе один текст используется в качестве ключа для шифрования другого текста. В эпоху "докомпьютерной" криптографии в качестве ключа для шифра с бегущим ключом выбирали какую-нибудь достаточно толстую книгу; от этого и произошло второе название этого шифра. Периодом в таком методе шифрования будет длина выбранного в качестве ключа произведения.

Методы многоалфавитной подстановки, в том числе и метод Вижинера, значительно труднее поддаются "ручному" криптоанализу. Для вскрытия методов многоалфавитной замены разработаны специальные, достаточно сложные алгоритмы. С использованием компьютера вскрытие метода многоалфавитной подстановки возможно достаточно быстро благодаря высокой скорости проводимых операций и расчетов.

В первой половине ХХ века для автоматизации процесса выполнения многоалфавитных подстановок стали широко применять *роторные шифровальные машины*. Главными элементами в таких устройствах являлись роторы – механические колеса, используемые для выполнения подстановки. Роторная шифровальная машина содержала обычно клавиатуру и набор роторов. Каждый ротор содержал набор символов (по количеству в алфавите), размещенных в произвольном порядке, и выполнял простую одноалфавитную подстановку. После выполнения первой замены символы сообщения обрабатывались вторым ротором и так далее. Роторы могли смещаться, что и задавало ключ шифрования. Некоторые роторные машины выполняли также и перестановку символов в процессе шифрования. Самым известным устройством подобного класса являлась немецкая шифровальная роторная машина Энигма (лат. Enigma — загадка), использовавшаяся во время второй мировой войны. Выпускалось несколько моделей Энигмы с разным числом роторов. В шифрмашине Энигма с тремя роторами можно было использовать 16900 разных алфавитов, и все они представляли собой различные перестановки символов.

#### Методы гаммирования

Еще одним частным случаем многоалфавитной подстановки является **гаммирование**. В этом способе шифрование выполняется путем сложения символов исходного текста и ключа по модулю, равному числу букв в алфавите. Если в исходном алфавите, например, 33 символа, то сложение производится по модулю 33. Такой процесс сложения исходного текста и ключа называется в криптографии *наложением гаммы*.

Пусть символам исходного алфавита соответствуют числа от 0 (А) до 32 (Я). Если обозначить число, соответствующее исходному символу, x, а символу ключа – k, то можно записать правило гаммирования следующим образом:

z = x + k (mod N),

где z – закодированный символ, N - количество символов в алфавите, а сложение по модулю N - операция, аналогичная обычному сложению, с тем отличием, что если обычное суммирование дает результат, больший или равный N, то значением суммы считается остаток от деления его на N. Например, пусть сложим по модулю 33 символы Г (3) и Ю (31):

3 + 31 (mod 33) = 1,

то есть в результате получаем символ Б, соответствующий числу 1.

Наиболее часто на практике встречается двоичное гаммирование. При этом используется *двоичный алфавит*, а сложение производится по модулю два. Операция сложения по модулю 2 часто обозначается Описание: \oplus, то есть можно записать:

Описание: z = x + k (mod 2) = x \oplus k.

Операция сложения по модулю два в *алгебре логики* называется также "исключающее ИЛИ" или по-английски XOR.

Рассмотрим пример. Предположим, нам необходимо зашифровать десятичное число 14 методом гаммирования с использованием ключа 12. Для этого вначале необходимо преобразовать исходное число и ключ (гамму) в двоичную форму: 14(10)=1110(2), 12(10)=1100(2). Затем надо записать полученные двоичные числа друг под другом и каждую пару символов сложить по модулю два. При сложении двух двоичных знаков получается 0, если исходные двоичные цифры одинаковы, и 1, если цифры разные:

Описание: 0 \oplus 0 = 0\\
0 \oplus 1 = 1\\
1 \oplus 0 = 1\\
1 \oplus 1 = 0\\


Сложим по модулю два двоичные числа 1110 и 1100:

Исходное число 1 1 1 0

Гамма 1 1 0 0

Результат 0 0 1 0

В результате сложения получили двоичное число 0010. Если перевести его в десятичную форму, получим 2. Таким образом, в результате применения к числу 14 операции гаммирования с ключом 12 получаем в результате число 2.

Каким же образом выполняется расшифрование? Зашифрованное число 2 представляется в двоичном виде и снова производится сложение по модулю 2 с ключом:

Зашифрованное число 0 0 1 0

Гамма 1 1 0 0

Результат 1 1 1 0

Переведем полученное двоичное значение 1110 в десятичный вид и получим 14, то есть исходное число.

Таким образом, при гаммировании по модулю 2 нужно использовать одну и ту же операцию как для зашифрования, так и для расшифрования. Это позволяет использовать один и тот же алгоритм, а соответственно и одну и ту же программу при программной реализации, как для шифрования, так и для расшифрования.

Операция сложения по модулю два очень быстро выполняется на компьютере (в отличие от многих других арифметических операций), поэтому наложение гаммы даже на очень большой открытый текст выполняется практически мгновенно.

Благодаря указанным достоинствам метод гаммирования широко применяется в современных технических системах сам по себе, а также как элемент комбинированных алгоритмов шифрования.

Сформулируем, как производится гаммирование по модулю 2 в общем случае:

* символы исходного текста и гамма представляются в двоичном коде и располагаются один под другим, при этом ключ (гамма) записывается столько раз, сколько потребуется;
* каждая пара двоичных знаков складывается по модулю два;
* полученная последовательность двоичных знаков кодируется символами алфавита в соответствии с выбранным кодом.

На [рис. 2.6](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=4" \l "image.2.6) показано, как применяется гаммирование к тексту с русскими символами. Символы кодируются в соответствии с принятой кодировкой, а затем производится сложение по модулю 2.

При использовании метода гаммирования ключом является последовательность, с которой производится сложение – гамма. Если гамма короче, чем сообщение, предназначенное для зашифрования, гамма повторяется требуемое число раз. Так в примере на [рис. 2.6](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=4" \l "image.2.6) длина исходного сообщения равна двенадцати байтам, а длина ключа – пяти байтам. Следовательно, для зашифрования гамма должна быть повторена 2 раза полностью и еще один раз частично.



**Рис. 2.6.**Механизм гаммирования

Чем длиннее ключ, тем надежнее шифрование методом гаммирования. Связь длины ключа с вероятностью вскрытия сообщения, а также некоторые принципы дешифрования сообщений, *закрытых методом* гаммирования, обсуждаются в ["Поточные шифры и генераторы псевдослучайных чисел. Часть 2"](https://www.intuit.ru/studies/courses/691/547/lecture/12385)и ["Шифрование, помехоустойчивое кодирование и сжатие информации"](https://www.intuit.ru/studies/courses/691/547/lecture/12397). На практике длина ключа ограничена возможностями аппаратуры обмена данными и вычислительной техники, а именно выделяемыми объемами памяти под ключ, временем обработки сообщения, а также возможностями аппаратуры подготовки и записи последовательностей ключей. Кроме того, для использования ключа вначале необходимо каким-либо надежным способом доставить его обеим сторонам, обменивающимся сообщениями. Это приводит к возникновению проблемы распределения ключей, сложность решения которой возрастает с увеличением длины ключа и количества абонентов в сети передачи сообщений.

### Методы перестановки

При использовании шифров **перестановки** *входной* *поток* исходного текста делится на блоки, в каждом из которых выполняется *перестановка* символов. Перестановки в классической "докомпьютерной" криптографии получались в результате записи исходного текста и чтения шифрованного текста *по* разным путям геометрической фигуры.

Простейшим примером перестановки является *перестановка с фиксированным периодом d*. В этом методе сообщение делится на блоки *по* *d* символов и в каждом блоке производится одна и та же *перестановка*. Правило, *по* которому производится *перестановка*, является ключом и может быть задано некоторой перестановкой первых *d* натуральных чисел. В результате сами буквы сообщения не изменяются, но передаются в другом порядке.

Например, для d=6 в качестве ключа перестановки можно взять 436215. Это означает, что в каждом блоке из 6 символов четвертый символ становится на первое *место*, третий – на второе, шестой – на третье и т.д. Пусть необходимо зашифровать такой текст:

ЭТО\_ТЕКСТ\_ДЛЯ\_ШИФРОВАНИЯ

Количество символов в исходном сообщении равно 24, следовательно, сообщение необходимо разбить на 4 блока. Результатом шифрования с помощью перестановки 436215 будет сообщение

\_ОЕТЭТ\_ТЛСКДИШР\_ЯФНАЯВОИ

Теоретически, если блок состоит из *d* символов, то число возможных перестановок d!=1\*2\*...\*(d-1)\*d. В последнем примере d=6, следовательно, число перестановок равно 6!=1\*2\*3\*4\*5\*6=720. Таким образом, если противник перехватил зашифрованное сообщение из рассмотренного примера, ему понадобится не более 720 попыток для раскрытия исходного сообщения (при условии, что размер блока известен противнику).

Для повышения криптостойкости можно последовательно применить к шифруемому сообщению две или более перестановки с разными периодами.

Другим примером методов перестановки является *перестановка по таблице*. В этом методе производится *запись* исходного текста *по* строкам некоторой таблицы и чтение его *по* столбцам этой же таблицы. Последовательность заполнения строк и чтения столбцов может быть любой и задается ключом.

Рассмотрим пример. Пусть в таблице кодирования будет 4 столбца и 3 строки (размер блока равен 3\*4=12 символов). Зашифруем такой текст:

ЭТО ТЕКСТ ДЛЯ ШИФРОВАНИЯ

Количество символов в исходном сообщении равно 24, следовательно, сообщение необходимо разбить на 2 блока. Запишем каждый блок в свою таблицу *по* строчкам ([таблица 2.9](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=5" \l "table.2.9)).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 2.9. Шифрование методом перестановки по таблице | | | |
| **1 блок** | | | |
| Э | Т | О |  |
| Т | Е | К | С |
| Т |  | Д | Л |
| **2 блок** | | | |
| Я |  | Ш | И |
| Ф | Р | О | В |
| А | Н | И | Я |

Затем будем считывать из таблицы каждый блок последовательно *по* столбцам:

ЭТТТЕ ОКД СЛЯФА РНШОИИВЯ

Можно считывать столбцы не последовательно, а, например, так: третий, второй, первый, четвертый:

ОКДТЕ ЭТТ СЛШОИ РНЯФАИВЯ

В этом случае порядок считывания столбцов и будет ключом.

В случае, если *размер сообщения* не кратен размеру блока, можно дополнить сообщение какими-либо символами, не влияющими на смысл, например, пробелами. Однако это делать не рекомендуется, так как это дает противнику в случае перехвата криптограммы информацию о размере используемой таблицы перестановок (длине блока). После определения длины блока противник может найти длину ключа (количество столбцов таблицы) среди делителей длины блока.

Посмотрим, как зашифровать и расшифровать сообщение, имеющее длину, не кратной размеру таблицы перестановки. Зашифруем *слово*

ПЕРЕМЕНКА

Количество символов в исходном сообщении равно 9. Запишем сообщение в таблицу *по* строкам ([таблица 2.10](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=5" \l "table.2.10)), а последние три ячейки оставим пустыми.

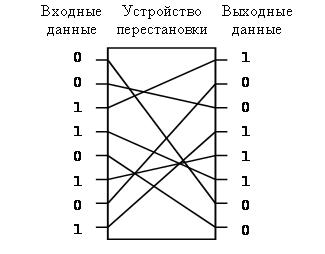
|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 2.10. Шифрование неполного блока методом перестановки по таблице | | | |
| П | Е | Р | Е |
| М | Е | Н | К |
| А |  |  |  |

Затем будем считывать из таблицы последовательно *по* столбцам:

ПМАЕЕРНЕК

Для расшифрования вначале определяют число полных столбцов, то есть количество символов в последней строке. Для этого делят *размер сообщения* (в нашем примере – 9) на количество столбцов или размер ключа (в примере – 4). *Остаток* от деления будет числом полных столбцов: 9 mod 4 = 1. Следовательно, в нашем примере был 1 полный столбец и три коротких. Теперь можно поставить буквы сообщения на свои места и расшифровать сообщение. Так как ключом при шифровании было число 1234 (столбцы считывались последовательно), то при расшифровании первые три символа (ПМА) записываются в первый столбец таблицы перестановки, следующие два (ЕЕ) – во второй столбец, следующие два (РН) – в третий, и последние два (ЕК) – в четвертый. После заполнения таблицы считываем строки и получаем исходное сообщение ПЕРЕМЕНКА.

Существуют и другие способы перестановки, которые можно реализовать программным и аппаратным путем. Например, при передаче данных, записанных в двоичном виде, удобно использовать аппаратный блок, который перемешивает определенным образом с помощью соответствующего электрического монтажа биты исходного n-разрядного сообщения. Так, если принять размер блока равным восьми битам, можно, к примеру, использовать такой блок перестановки, как на [рис. 2.7](https://www.intuit.ru/studies/courses/691/547/lecture/12373?page=5" \l "image.2.7).



**Рис. 2.7.**Аппаратный блок перестановки

Для расшифрования на приемной стороне устанавливается другой блок, восстанавливающий порядок цепей.

Аппаратно реализуемая *перестановка* широко используется на практике как составная часть некоторых современных шифров.

При перестановке любого вида в зашифрованное сообщение будут входить те же символы, что и в *открытый текст*, но в другом порядке. Следовательно, статистические закономерности языка останутся без изменения. Это дает криптоаналитику возможность использовать различные методы для восстановления правильного порядка символов.

Если у противника есть возможность пропускать через систему шифрования методом перестановки специально подобранные сообщения, то он сможет организовать атаку *по* выбранному тексту. Так, если *длина блока* в исходном тексте равна N символам, то для раскрытия ключа достаточно пропустить через шифровальную систему N-1 блоков исходного текста, в которых все символы, кроме одного, одинаковы. Другой вариант атаки *по* выбранному тексту возможен в случае, если *длина блока* N меньше количества символов в алфавите. В этом случае можно сформировать одно специальное сообщение из разных букв алфавита, расположив их, например, *по* порядку следования в алфавите. Пропустив подготовленное таким образом сообщение через шифровальную систему, специалисту *по* криптоанализу останется только посмотреть, на каких позициях очутились символы алфавита после шифрования, и составить схему перестановки.

Мы рассмотрели общую схему симметричного шифрования и классификацию простейших методов шифрования с закрытым ключом. В следующей лекции мы познакомимся с принципами построения современных блочных алгоритмов

### Ключевые термины

**Гаммирование** – метод шифрования, основанный на "наложении" гамма-последовательности на *открытый текст*. Обычно это суммирование в каком-либо конечном *поле* (суммирование *по* модулю). Например, в *поле* GF(2) такое суммирование принимает вид обычного "исключающего ИЛИ". При расшифровке операция проводится повторно, в результате получается *открытый текст*.

**Пропорциональные** или **монофонические шифры** – методы замены, в которых уравнивается частота появления зашифрованных знаков.

**Шифры замены (подстановки)** основаны на том, что символы исходного текста, обычно разделенные на блоки и записанные в одном алфавите, заменяются одним или несколькими символами другого алфавита в соответствии с принятым правилом преобразования.

**Шифр многоалфавитной замены (или подстановки)** – *группа* методов шифрования подстановкой, в которых для замены символов исходного текста используется не один, а несколько алфавитов *по* определенному правилу.

**Шифры перестановки** основаны на том, что *входной* *поток* исходного текста делится на блоки, в каждом из которых выполняется *перестановка* символов. Ключом такого шифра является используемая при шифровании перестановочная *матрица* или *вектор*, указывающий правило перестановки.

**Шифр простой (или одноалфавитной) замены, простой подстановочный шифр, моноалфавитный шифр**— *группа* методов шифрования, которые сводятся к созданию *по* определённому алгоритму таблицы шифрования, в которой для каждой буквы открытого текста существует единственная сопоставленная ей буква *шифртекста*. Само *шифрование* заключается в замене букв согласно таблице. Для расшифровки достаточно иметь ту же таблицу, либо знать *алгоритм*, *по* которой она генерируется.

**Симметричное шифрование (шифрование с закрытым ключом)** – методы обратимого преобразования данных, в которых используется один и тот же *ключ*, который обе стороны информационного обмена должны хранить в секрете от противника. Все известные из истории шифры, например, *шифр* Цезаря – это шифры с закрытым ключом.

### Краткие итоги

Симметричные шифры – способ шифрования, в котором для шифрования и расшифровывания применяется один и тот же криптографический *ключ*. *Ключ* шифрования должен сохраняться в секрете обеими сторонами.

Известны разные методы шифрования с закрытым ключом. На практике часто используются алгоритмы перестановки, подстановки, а также комбинированные методы.

В методах перестановки символы исходного текста меняются местами друг с другом *по* определенному правилу.

В методах замены (или подстановки) символы открытого текста заменяются некоторыми эквивалентами шифрованного текста. *Шифр* простой (или одноалфавитной) замены – *группа* методов шифрования, которые сводится к созданию *по* определённому алгоритму таблицы шифрования, в которой для каждой буквы открытого текста существует единственная сопоставленная ей буква *шифртекста*. Само *шифрование* заключается в замене букв согласно таблице. Для расшифровки достаточно иметь ту же таблицу, либо знать *алгоритм*, *по* которой она генерируется.

*Шифр* многоалфавитной замены – *группа* методов шифрования подстановкой, в которых для замены символов исходного текста используется не один, а несколько алфавитов *по* определенному правилу. Таким образом, при шифровании получаётся достаточно сложная последовательность, которую уже не так просто вскрыть, как один одноалфавитный *шифр*.

Частным случаем многоалфавитной подстановки является гаммирование – метод шифрования, основанный на "наложении" гамма-последовательности на *открытый текст*. Обычно это суммирование в каком-либо конечном *поле* (суммирование *по* модулю длины алфавита).

Самым важным эффектом, достигаемым при использовании многоалфавитного шифра, является маскировка частот появления тех или иных букв в тексте, на основании которой обычно очень легко вскрываются одноалфавитные шифры.

### Набор для практики

#### Вопросы для самопроверки

1. Поясните общую схему симметричного шифрования.
2. Что общего имеют все методы шифрования с закрытым ключом?
3. Назовите основные группы методов шифрования с закрытым ключом.
4. Приведите примеры шифров перестановки.
5. Сформулируйте общие принципы для методов шифрования подстановкой.
6. В чем заключаются многоалфавитные подстановки?
7. Приведите пример шифра одноалфавитной замены.
8. Опишите алгоритм любого метода шифрования перестановкой. Приведите пример шифрования некоторого сообщения этим методом. Каков алгоритм расшифрования в этом методе?
9. К какой группе методов шифрования с закрытым ключом относится метод с использованием таблицы Вижинера? Каковы алгоритмы шифрования и расшифрования в этом методе? Приведите пример шифрования некоторого сообщения этим методом.
10. Каким образом можно зашифровать и расшифровать сообщение методом табличной перестановки, если размер шифруемого сообщения не кратен размеру блока?
11. Что такое монофонические шифры?

### Лекция ә: Поточные шифры и генераторы псевдослучайных чисел. Часть 1

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

**Задание:**

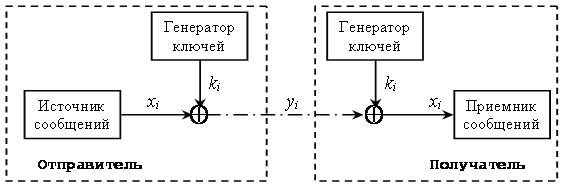
1. Повторить ключевые понятия из Лекции №2 (Часть 1):  
   – Синхронные и асинхронные поточные шифры  
   – Основные принципы генерации псевдослучайных последовательностей
2. Изучить современные поточные шифры, используемые в криптографии:  
   – A5/1 и A5/2 (использовались в GSM-связи)  
   – RC4 (применялся 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: познакомиться с понятием "*поточный шифр*", а также с принципами использования генераторов псевдослучайных ключей при *потоковом шифровании*

### Поточные шифры

Блочный *алгоритм* предназначен для шифрования блоков определенной длины. Однако может возникнуть необходимость шифрования данных не блоками, а, например, по символам. **Поточный шифр (stream cipher)** выполняет преобразование входного сообщения по одному биту (или байту) за операцию. Поточный *алгоритм* шифрования устраняет необходимость разбивать сообщение на *целое число* блоков достаточно большой длины, следовательно, он может работать в реальном времени. Таким образом, если передается *поток* символов, каждый символ может шифроваться и передаваться сразу.

Работа типичного поточного шифра представлена на [рис. 7.1](https://www.intuit.ru/studies/courses/691/547/lecture/12383?page=1" \l "image.7.1).



**Рис. 7.1.**Принцип работы поточного шифра

*Генератор* ключей выдает *поток* битов ki, которые будут использоваться в качестве гаммы. Источник сообщений генерирует биты открытого текста хi, которые складываются по модулю 2 с гаммой, в результате чего получаются биты зашифрованного сообщения уi:

Описание: y_i = x_i \oplus k_i, i=1,2,…,n

Чтобы из шифротекста y1, y2,..., yn восстановить сообщение x1, x2,..., xn, необходимо сгенерировать точно такую же ключевую последовательность k1, yk,..., kn, что и при шифровании, и использовать для расшифрования формулу

Описание: x_i = y_i \oplus k_i,\  i=1,2,…,n


Обычно исходное сообщение и ключевая последовательность представляют собой независимые потоки *бит*. Таким образом, так как шифрующее (и расшифрующее) преобразование для всех поточных шифров одно и то же, они должны различаться только способом построения генераторов ключей. Получается, что *безопасность* системы полностью зависит от свойств генератора потока ключей. Если *генератор* потока ключей выдает последовательность, состоящую только из одних нулей (или из одних единиц), то зашифрованное сообщение будет в точности таким же, как и исходный *поток* битов (в случае единичных ключей зашифрованное сообщение будет инверсией исходного). Если в качестве гаммы используется один символ, представленный, например, восемью битами, то хотя зашифрованное сообщение и будет внешне отличаться от исходного, *безопасность* системы будет очень низкой. В этом случае при многократном повторении кода ключа по всей длине текста существует опасность его раскрытия статистическим методом. Поясним это на простом примере цифрового текста, закрытого коротким цифровым кодом ключа методом гаммирования.

**Пример**. Пусть известно, что исходное сообщение представляло собой двоично-десятичное число, то есть число, каждая тетрада (четыре бита) которого получена при переводе десятичной цифры 0...9 в двоичный вид. Перехвачено 24 бита зашифрованного сообщения Y, то есть шесть тетрад Y1, Y2, Y3, Y4, Y5, Y6, а именно *значение* 1100 1101 1110 1111 0000 0001. Известно, что *ключ шифрования* состоял из четырех *бит*, которые тоже представляют собой однозначное десятичное число, то есть одно и то же *значение* 0≤K≤9 использовалось для шифрования каждых четырех *бит* исходного сообщения. Таким образом, *шифрование* числа X1, X2, X3, X4, X5, X6 ключом К можно представить в виде системы уравнений:

Описание: \begin {multiple}X_1 \oplus K=1100\ \ \ X_2 \oplus K=1101\ \ \ X_3 \oplus K=1110\\
X_4 \oplus K=1111\ \ \ X_5 \oplus K=0000\ \ \ X_6 \oplus K=0001
\end{multiple}

Исходя из условия, что Хi принимает десятичные значения от 0 до 9, для поиска неизвестного К определим все возможные значения X1’; и К, сумма которых по модулю 2 приводит к результату 1100:

K = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001

Y1 = 1100 1100 1100 1100 1100 1100 1100 1100 1100 1100

---------------------

X1’ = 1100 1101 1110 1111 1000 1001 1010 1011 0100 0101

Так как исходное *значение* состояло из цифр от 0 до 9, то можно исключить из рассмотрения значения ключа 0000, 0001, 0010, 0011, 0110, 0111, так при сложении с ними получаются значения большие 9 в десятичном эквиваленте. Такие значения не могли присутствовать в открытом тексте. Таким образом, первый этап анализа уже позволил сократить количество возможных ключей с десяти до четырех.

Для дальнейшего поиска неизвестного К определим все возможные значения X2’; и оставшихся вариантов ключа, сумма которых по модулю 2 приводит к результату Y2 = 1101:

K = 0100 0101 1000 1001

Y2 = 1101 1101 1101 1101

---------------------

X2’ = 1001 1000 0101 0100

Видно, что этот этап не позволил отбросить ни одного из оставшихся вариантов ключа. Попытаемся это сделать, используя Y3=1110:

K = 0100 0101 1000 1001

Y3 = 1110 1110 1110 1110

---------------------

X2’ = 1010 1011 0110 0111

После проведения этого этапа становится ясно, что ключом не могли быть значения 0100 и 0101. Остается два возможных значения ключа: 1000(2)=8(10) и 1001(2)=9(10).

Дальнейший *анализ* по данной методике в данном случае, к сожалению, не позволит однозначно указать, какой же из двух полученных вариантов ключа использовался при шифровании. Однако можно считать успехом уже то, что *пространство* возможных ключей снизилось с десяти до двух. Остается попробовать каждый из двух найденных ключей для дешифровки сообщений и проанализировать смысл полученных вскрытых текстов.

В реальных случаях, когда исходное сообщение составлено не только из одних цифр, но и из других символов, использование статистического анализа позволяет быстро и точно восстановить *ключ* и исходные сообщения при короткой длине ключа, закрывающего *поток* секретных данных.

### Принципы использования генераторов псевдослучайных чисел при потоковом шифровании

Современная *информатика* широко использует псевдослучайные числа в самых разных приложениях — от методов математической статистики и имитационного моделирования до криптографии. При этом от качества используемых **генераторов псевдослучайных чисел** (ГПСЧ) напрямую зависит качество получаемых результатов.

ГПСЧ могут использоваться в качестве генераторов ключей в поточных шифрах. Целью использования *генераторов псевдослучайных чисел* является получение "бесконечного" ключевого слова, располагая относительно малой длиной самого ключа. *Генератор псевдослучайных чисел* создает последовательность битов, похожую на случайную. На самом деле, конечно же, такие последовательности вычисляются по определенным правилам и не являются случайными, поэтому они могут быть абсолютно точно воспроизведены как на передающей, так и на принимающей стороне. Последовательность ключевых символов, использующаяся при шифровании, должна быть не только достаточно длинной. Если *генератор* ключей при каждом включении создает одну и ту же последовательность битов, то взломать такую систему также будет возможно. Следовательно, *выход* генератора потока ключей должен быть функцией ключа. В этом случае расшифровать и прочитать сообщения можно будет только с использованием того же ключа, который использовался при шифровании.

Для использования в криптографических целях *генератор псевдослучайных чисел* должен обладать следующими свойствами:

1. период последовательности должен быть очень большой;
2. порождаемая последовательность должна быть "почти" неотличима от действительно случайной;
3. вероятности порождения различных значений должны быть в точности равны;
4. для того, чтобы только законный получатель мог расшифровать сообщение, следует при получении потока ключевых битов ki использовать и учитывать некоторый секретный ключ, причем вычисление числа ki+1 по известным предыдущим элементам последовательности ki без знания ключа должно быть трудной задачей.

При наличии указанных свойств последовательности псевдослучайных чисел могут быть использованы в поточных шифрах.

### Линейный конгруэнтный генератор псевдослучайных чисел

*Генераторы псевдослучайных чисел* могут работать по разным алгоритмам. Одним из простейших генераторов является так называемый **линейный конгруэнтный генератор**, который для вычисления очередного числа ki использует формулу

ki=(a\*ki-1+b)mod c,

где а, b, с — некоторые *константы*, a ki-1 — предыдущее *псевдослучайное число*. Для получения k1 задается начальное *значение* k0. Возьмем в качестве примера a=5,b=3,c=11 и пусть k0= 1. В этом случае мы сможем по приведенной выше формуле получать значения от 0 до 10 (так как с = 11). Вычислим несколько элементов последовательности:

k1 = (5 \* 1 + 3) mod 11 = 8;

k2 = (5 \* 8 + 3) mod 11 = 10;

k3 = (5 \* 10 + 3) mod 11 = 9;

k4 = (5 \* 9 + 3) mod 11 = 4;

k5 = (5 \* 4 + 3) mod 11 = 1.

Полученные значения (8, 10, 9, 4, 1) выглядят похожими на случайные числа. Однако следующее *значение* k6 будет снова равно 8:

k6 = (5 \* 1 + 3) mod 11 = 8,

а значения k7 и k8 будут равны 10 и 9 соответственно:

k7 = (5 \* 8 + 3) mod 11 = 10;

k8= (5 \* 10 + 3) mod 11 = 9.

Выходит, наш *генератор псевдослучайных чисел* повторяется, порождая периодически числа 8, 10, 9, 4, 1. К сожалению, это свойство характерно для всех линейных конгруэнтных генераторов. Изменяя значения основных параметров a, b и c, можно влиять на длину периода и на сами порождаемые значения ki. Так, например, увеличение числа с в общем случае ведет к увеличению периода. 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 2b-1 или 2b, где b — *длина* слова ЭВМ 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 ПСЧ по нескольким значениям. Например, предположим, что противник может определить значения k0, k1, k2, k3. Тогда:

k1=(a\*k0+b) mod c

k2=(a\*k1+b) mod c

k3=(a\*k2+b) mod c

Решив систему из этих трех уравнений, можно найти a, b и c.

Для получения псевдослучайных чисел предлагалось использовать также квадратичные и кубические генераторы:

ki=(a12\*ki-1+a2\*ki-1+b) mod c

ki=(a13\*ki-1+a22\*ki-1+a3\*ki-1+b) mod c

Однако такие генераторы тоже оказались непригодными для целей криптографии по той же самой причине "предсказуемости".

### Метод Фибоначчи с запаздыванием

Известны и другие схемы получения псевдослучайных чисел.

**Метод Фибоначчи с запаздываниями** (*Lagged Fibonacci Generator*) — один из методов генерации псевдослучайных чисел. Он позволяет получить более высокое "качество" псевдослучайных чисел.

Наибольшую популярность фибоначчиевы датчики получили в связи с тем, что скорость выполнения арифметических операций с вещественными числами сравнялась со скоростью целочисленной арифметики, а фибоначчиевы датчики естественно реализуются в *вещественной арифметике*.

Известны разные схемы использования метода Фибоначчи с запаздыванием. Один из широко распространённых фибоначчиевых датчиков основан на следующей рекуррентной формуле:

Описание: 7_3

где ki — вещественные числа из диапазона [0,1], a, b — целые положительные числа, параметры генератора. Для работы фибоначчиеву датчику требуется знать max{a,b} предыдущих сгенерированных случайных чисел. При программной реализации для хранения сгенерированных случайных чисел необходим некоторый объем памяти, зависящих от параметров a и b.

**Пример**. Вычислим последовательность из первых десяти чисел, генерируемую методом Фибоначчи с запаздыванием начиная с k5 при следующих исходных данных: a = 4, b = 1, k0=0.1; k1=0.7; k2=0.3; k3=0.9; k4=0.5:

k5 = k1 - k4 = 0.7 - 0.5 = 0.2;

k6 = k2 - k5= 0.3 - 0.2 = 0.1;

k7 = k3 - k6 = 0.9 - 0.1 = 0.8;

k8 = k4 - k7 + 1 =0.5 - 0.8 + 1 = 0.7;

k9 = k5- k8 + 1 =0.2 - 0.7 + 1 = 0.5;

k10 = k6 - k9 + 1 =0.1 - 0.5 + 1 = 0.6;

k11 = k7 - k10 = 0.8 - 0.6 = 0.2;

k12 = k8 - k11 = 0.7 - 0.2 = 0.5;

k13 = k9 - k12 + 1 =0.5 - 0.5 + 1 = 1;

k14 = k10 - k13 + 1 =0.6 - 1 + 1 = 0.6.

Видим, что генерируемая последовательность чисел внешне похожа на случайную. И действительно, исследования подтверждают, что получаемые случайные числа обладают хорошими статистическими свойствами.

Для генераторов, построенных по *методу Фибоначчи* с запаздыванием, существуют рекомендуемые параметры a и b, так сказать, протестированные на качество. Например, исследователи предлагают следующие значения: (a,b) = (55, 24), (17, 5) или (97,33). Качество получаемых случайных чисел зависит от значения *константы* a: чем оно больше, тем выше *размерность* пространства, в котором сохраняется равномерность случайных векторов, образованных из полученных случайных чисел. В то же время с увеличением величины *константы* a увеличивается *объём* используемой алгоритмом памяти.

В результате значения (a,b) = (17,5) рекомендуются для простых приложений. Значения (a,b) = (55,24) позволяют получать числа, удовлетворительные для большинства криптографических алгоритмов, требовательных к качеству случайных чисел. Значения (a,b) = (97,33) позволяют получать очень качественные случайные числа и используются в алгоритмах, работающих со случайными векторами высокой размерности.

Генераторы ПСЧ, основанные на методе Фибоначчи с запаздыванием, использовались для целей криптографии. Кроме того, они применяются в математических и статистических расчетах, а также при моделировании случайных процессов. *Генератор* ПСЧ, построенный на основе метода Фибоначчи с запаздыванием, использовался в широко известной системе Matlab.

### Генератор псевдослучайных чисел на основе алгоритма BBS

Широкое распространение получил *алгоритм* генерации псевдослучайных чисел, называемый **алгоритмом BBS** (от фамилий авторов — L. Blum, M. Blum, M. Shub) или *генератором с квадратичным остатком*. Для целей криптографии этот метод предложен в 1986 году.

Он заключается в следующем. Вначале выбираются два больших простых1 числа p и q. Числа p и q должны быть оба *сравнимы* с 3 по модулю 4, то есть при делении p и q на 4 должен получаться одинаковый *остаток* 3. Далее вычисляется число M = p\* q, называемое целым числом Блюма. Затем выбирается другое случайное *целое число* х, взаимно простое (то есть не имеющее общих делителей, кроме единицы) с М. Вычисляем х0= х2mod M. х0 называется стартовым числом генератора.

На каждом n-м шаге работы генератора вычисляется хn+1= хn2 mod M. Результатом n-го шага является один (обычно младший) *бит* числа хn+1. Иногда в качестве результата принимают *бит* чётности, то есть количество единиц в двоичном представлении элемента. Если количество единиц в записи числа четное – *бит* четности принимается равным 0, нечетное – *бит* четности принимается равным 1.

**Например**, пусть p = 11, q = 19 (убеждаемся, что 11 mod 4 = 3, 19 mod 4 = 3). Тогда M = p\* q = 11\*19=209. Выберем х, взаимно простое с М: пусть х = 3. Вычислим стартовое число генератора х0:

х0 = х2 mod M = 32 mod 209 = 9 mod 209 = 9.

Вычислим первые десять чисел хi по алгоритму *BBS*. В качестве случайных *бит* будем брать младший *бит* в двоичной записи числа хi:

|  |  |  |
| --- | --- | --- |
| х1=92 mod 209= 81 mod 209= 81 | младший бит: | 1 |
| х2=812 mod 209= 6561 mod 209= 82 | младший бит: | 0 |
| х3=822 mod 209= 6724 mod 209= 36 | младший бит: | 0 |
| х4=362 mod 209= 1296 mod 209= 42 | младший бит: | 0 |
| х5=422 mod 209= 1764 mod 209= 92 | младший бит: | 0 |
| х6=922 mod 209= 8464 mod 209= 104 | младший бит: | 0 |
| х7=1042 mod 209= 10816 mod 209= 157 | младший бит: | 1 |
| х8=1572 mod 209= 24649 mod 209= 196 | младший бит: | 0 |
| х9=1962 mod 209= 38416 mod 209= 169 | младший бит: | 1 |
| х10=1692 mod 209= 28561 mod 209= 137 | младший бит: | 1 |

Самым интересным для практических целей свойством этого метода является то, что для получения n-го числа последовательности не нужно вычислять все предыдущие n чисел хi. Оказывается хn можно сразу получить по формуле

Описание: x_n=x_0^{2^n mod ((p-1)(q-1)} mod \: M

Например, вычислим х10 сразу из х0:

Описание: x_{10}=x_0^{2^{10} mod ((11-1)(19-1)} mod \ 209 = x_0^{1024 mod \ 180} mod \ 209 =\\
9^{124} \ mod \ 209=(9^4)^{31} \ mod \ 209=81^{31} \ mod \ 209 =\\
(81^{15} \ mod \ 209)(81^{16} \ mod \ 209)=((81^3)^5) \ mod \ 209)((81^4)^4) \ mod \ 209)=\\
(26^5 \ mod \ 209)(42^4 \ mod \ 209)=(144*104) \ mod \ 209=14976 \ mod \ 209=137

В результате действительно получили такое же *значение*, как и при последовательном вычислении, – 137. Вычисления кажутся достаточно сложными, однако на самом деле их легко оформить в виде небольшой процедуры или программы и использовать при необходимости.

Возможность "прямого" получения хn позволяет использовать *алгоритм* *BBS* при потоковой шифрации, например, для файлов с произвольным доступом или фрагментов файлов с записями *базы данных*.

*Безопасность* алгоритма *BBS* основана на сложности разложения большого числа М на множители. Утверждается, что если М достаточно велико, его можно даже не держать в секрете; до тех пор, пока М не разложено на множители, никто не сможет предсказать *выход* генератора ПСЧ. Это связано с тем, что задача разложения чисел вида n = pq (р и q — простые числа) на множители является вычислительно очень трудной, если мы знаем только n, а р и q — большие числа, состоящие из нескольких десятков или сотен *бит* (это так называемая *задача факторизации*).

Кроме того, можно доказать, что *злоумышленник*, зная некоторую последовательность, сгенерированную генератором *BBS*, не сможет определить ни предыдущие до нее биты, ни следующие. *Генератор* *BBS* *непредсказуем в левом направлении* и *в правом направлении*. Это свойство очень полезно для целей криптографии и оно также связано с особенностями разложения числа М на множители.

Самым существенным недостатком алгоритма является то, что он недостаточно быстр, что не позволяет использовать его во многих областях, например, при вычислениях в реальном времени, а также, к сожалению, и при *потоковом шифровании*.

Зато этот *алгоритм* выдает действительно хорошую последовательность псевдослучайных чисел с большим периодом (при соответствующем выборе исходных параметров), что позволяет использовать его для криптографических целей при генерации ключей для шифрования.

### Ключевые термины

**Stream cipher** – *поточный шифр*.

**Алгоритм BBS** – один из методов генерации псевдослучайных чисел. Название алгоритма происходит от фамилий авторов - L. Blum, M. Blum, M. Shub. *Алгоритм* может использоваться в криптографии. Для вычислений очередного числа xn+1 по алгоритму *BBS* используется формула хn+1= хn2 mod M, где M = pq является произведением двух больших простых p и q.

**Генератор псевдослучайных чисел (ГПСЧ)** – некоторый *алгоритм* или устройство, которые создают последовательность битов, внешне похожую на случайную.

**Линейный конгруэнтный генератор** псевдослучайных чисел – один из простейших ГПСЧ, который для вычисления очередного числа ki использует формулу ki=(a\*ki-1+b)mod c, где а, b, с — некоторые *константы*, a ki-1 — предыдущее *псевдослучайное число*.

**Метод Фибоначчи с запаздываниями** – один из методов генерации псевдослучайных чисел. Может использоваться в криптографии.

**Поточный шифр** – *шифр*, который выполняет *шифрование* входного сообщения по одному биту (или байту) за операцию. Поточный *алгоритм* шифрования устраняет необходимость разбивать сообщение на *целое число* блоков. *Поточные шифры* используются для шифрования данных в реальном времени.

### Краткие итоги

*Поточный шифр* – это *шифр*, который выполняет *шифрование* входного сообщения по одному биту (или байту) за операцию. Поточный *алгоритм* шифрования устраняет необходимость разбивать сообщение на *целое число* блоков. Таким образом, если передается *поток* символов, каждый символ может шифроваться и передаваться сразу. *Поточные шифры* используются для шифрования данных в режиме реального времени.

В качестве генераторов ключей в поточных шифрах могут использоваться *генераторы псевдослучайных чисел* (ГПСЧ). Целью использования ГПСЧ является получение "бесконечного" ключевого слова, располагая относительно малой длиной самого ключа. Для использования в криптографических целях *генератор псевдослучайных чисел* должен обладать некоторыми свойствами, например, период последовательности, порождаемой генератором, должен быть очень большой.

Простейшими генераторами псевдослучайных чисел являются: линейный конгруэнтный *генератор*, *генератор* по *методу Фибоначчи* с запаздываниями, *генератор* на основе алгоритма Блюм – Блюма – Шуба (*BBS*).

### Набор для практики

#### Вопросы для самопроверки

1. Чем *поточный шифр* отличается от блочного?
2. Каким образом организуется шифрование потока данных переменной длины?
3. Какие числа называют "псевдослучайными"?
4. Какими свойствами должен обладать *генератор псевдослучайных чисел* для использования в криптографических целях?
5. Какие *генераторы псевдослучайных чисел* Вы можете назвать?
6. Перечислите основные характеристики, достоинства и недостатки каждого из рассмотренных в данной лекции *генераторов псевдослучайных чисел*.

**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:**

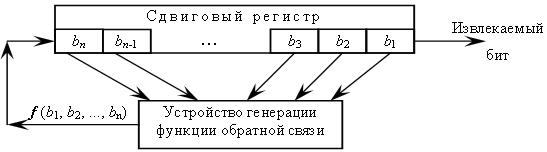
1. Повторить ключевые понятия из Лекции №2 (Часть 1):  
   – Синхронные и асинхронные поточные шифры  
   – Основные принципы генерации псевдослучайных последовательностей
2. Изучить современные поточные шифры, используемые в криптографии:  
   – A5/1 и A5/2 (использовались в GSM-связи)  
   – RC4 (применялся 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)).



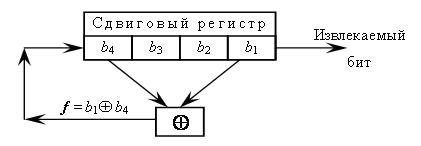
**Рис. 8.1.**Сдвиговый регистр с обратной связью

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.

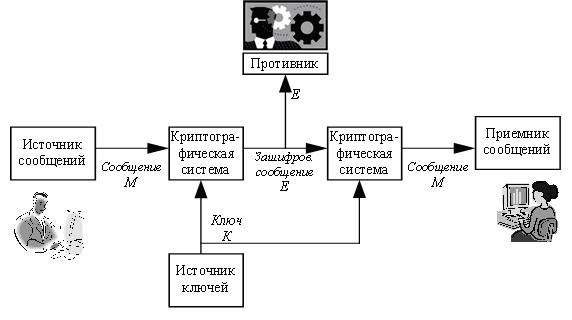
Кроме того, параметры других различных физических явлений могут быть положены в основу ГСЧ. К сожалению, многие методики получения настоящих случайных чисел не могут быть реализованы на практике, так как используемый в криптографических целях *генератор* должен быть компактным, быстрым (генерировать 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?

**Лекция 3: Введение в криптографию с открытым ключом**

**Цель:**  
Познакомиться с основными принципами криптографии с открытым ключом, понять её отличия от симметричной криптографии и области применения.

**Задание:**

1. Изучить понятие криптографии с открытым ключом:  
   – Что такое открытый и закрытый ключ  
   – Принцип асимметричного шифрования  
   – Роль доверия и центров сертификации
2. Рассмотреть основные алгоритмы:  
   – RSA  
   – Диффи — Хеллман (обмен ключами)  
   – Эллиптические кривые (ECC)  
   – ElGamal
3. Выполнить практическую часть:  
   – Построить простую схему работы RSA (на небольших числах вручную или в Excel/Python)  
   – Привести пример, как происходит шифрование и дешифрование
4. Подготовить:  
   – Презентацию (8–10 слайдов) с примерами применения (например, HTTPS, цифровые подписи)  
   – Краткий письменный доклад (1 страница):  
   • Преимущества и недостатки криптографии с открытым ключом  
   • Где используется в реальной жизни

**Цель лекции**: первоначальное знакомство с принципами шифрования с открытым ключом.

### Предпосылки создания методов шифрования с открытым ключом и основные определения

При использовании шифрования с закрытым ключом возникают две достаточно серьезные проблемы. Первая проблема заключается в изготовлении секретных ключей и доставке их участникам информационного обмена. При большом количестве и территориальной распределенности участников информационного обмена, использующих каналы связи общего назначения, например, обычную или электронную почту, часто бывает сложно гарантировать *безопасность* доставки такого ключа и его подлинность. В ["Поточные шифры и генераторы псевдослучайных чисел. Часть 2"](https://www.intuit.ru/studies/courses/691/547/lecture/12385)учебного пособия проблема распределения ключей для симметричного шифрования была подробно рассмотрена.

Второй проблемой является обеспечение подлинности партнеров при электронном общении. Развитие деловой переписки и электронной коммерции требует наличия методов, при использовании которых невозможно было бы подменить кого-либо из участников обмена. Получатель корреспонденции должен иметь возможность удостовериться в подлинности документа, а создатель электронного послания должен быть в состоянии доказать свое авторство получателю или третьей стороне. Следовательно, электронные документы должны иметь аналог обычной подписи.

Многие криптографы работали над решением этих проблем, в результате чего во второй половине семидесятых годов ХХ века были разработаны принципиально новые подходы, позволяющие решить перечисленные выше (и некоторые другие) задачи. Основой послужило открытие так называемых **асимметричных криптоалгоритмов**, или методов, в которых процедуры прямого и обратного криптопреобразования выполняются на различных ключах и не имеют между собой очевидных и легко прослеживаемых связей, которые позволили бы по одному ключу определить другой. *Асимметричные алгоритмы* гораздо больше основаны на свойствах математических функций, чем алгоритмы симметричного шифрования, использующие в основном только *операции* перестановки и замены. Большой вклад в эти исследования внесли американские ученые У. Диффи (W. Diffie), Э. Хеллман (M. Hellman), Р. Меркль (R. Merkle). Они первыми предложили способы решения обеих задач, которые радикально отличаются от всех предыдущих подходов к шифрованию.

*Асимметричные алгоритмы* шифрования называются также **алгоритмами с открытым ключом**. В отличие от *алгоритмов симметричного шифрования* (алгоритмов шифрования с закрытым ключом), в которых для шифрования и расшифрования используется один и тот же *ключ*, в ассиметричных алгоритмах один *ключ* используется для шифрования, а другой, отличный от первого, – для расшифрования. Алгоритмы называются асимметричными, так как ключи шифрования и расшифрования разные, следовательно, отсутствует *симметрия* основных криптографических процессов. Один из двух ключей является **открытым** (*public key*) и может быть объявлен всем, а второй – **закрытым** (*private key*) и должен держаться в секрете. Какой из ключей, открытый или закрытый, используется для шифрования, а какой для расшифрования, определяется назначением криптографической системы.

В настоящее время *асимметричные алгоритмы* широко применяются на практике, например, для обеспечения информационной безопасности телекоммуникационных сетей, в том числе сетей, имеющих сложную топологию; для обеспечения информационной безопасности в глобальной сети *Internet*; в различных банковских и платежных системах (в том числе использующих *интеллектуальные карты*).

*Алгоритмы шифрования* с открытым ключом можно использовать для решения, как *минимум*, трех задач:

1. Для шифрования передаваемых и хранимых данных в целях их защиты от несанкционированного доступа.
2. Для формирования цифровой подписи под электронными документами.
3. Для распределения секретных ключей, используемых потом при шифровании документов симметричными методами.

### Односторонние функции

Все *алгоритмы шифрования* с открытым ключом основаны на использовании так называемых *односторонних функций*. **Односторонней функцией** (*one-way* *function*) называется *математическая функция*, которую относительно легко вычислить, но трудно найти по значению функции соответствующее *значение* аргумента. То есть, зная х легко вычислить f(x), но по известному f(x) трудно найти подходящее *значение* x. Под словом "трудно вычислить" понимают, что для этого потребуется не один год расчетов с использованием ЭВМ. *Односторонние функции* применяются в криптографии также в качестве хеш-функций (см. ["Криптографические хеш-функции"](https://www.intuit.ru/studies/courses/691/547/lecture/12381)). Использовать *односторонние функции* для шифрования сообщений с целью их защиты не имеет смысла, так как обратно расшифровать зашифрованное сообщение уже не получится. Для целей шифрования используются специальные *односторонние функции* – **односторонние функции с люком** (или с секретом) – это особый вид *односторонних функций*, имеющих некоторый секрет (*люк*), позволяющий относительно быстро вычислить обратное *значение* функции.

Для *односторонней функции* с люком f справедливы следующие утверждения:

1. зная х, легко вычислить f(x),
2. по известному значению f(x) трудно найти x,
3. зная дополнительно некоторую секретную информацию, можно легко вычислить x.

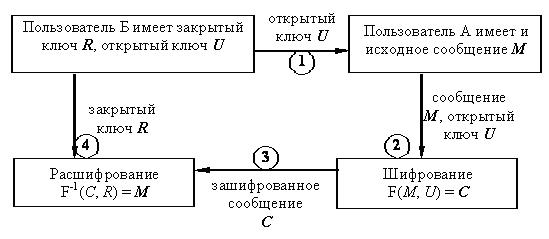
### Использование асимметричных алгоритмов для шифрования

В 70-х годах ХХ века Диффи и Хеллман предложили принцип шифрования, основанный на использовании двух разных ключей, хотя и связанных между собой, но устроенных так, что вычислить по одному из них (открытому) другой (закрытый) практически невозможно. Этот принцип может быть использован для решения проблемы снабжения пользователей ключами шифрования/расшифрования, а точнее – для устранения этой проблемы. Согласно Диффи и Хеллману предварительно распределяемые закрытые ключи вообще не должны использоваться для шифрования данных (так как секрет, который известен более чем одному человеку, – уже не секрет). *Закрытый ключ* должен быть известен только одному лицу – его владельцу. Такой принцип использования *асимметричных алгоритмов* получил название *открытого шифрования* или *шифрованием с открытым ключом*.

Согласно этому принципу, любой желающий может зашифровать сообщение открытым ключом. Расшифровать сообщение сможет только владелец закрытого ключа. Пусть, например, пользователи А и Б, имеющие возможность обмениваться электронными сообщениями, используют схему открытого шифрования. Предположим, *пользователь* А должен передать секретное сообщение пользователю Б так, чтобы никто другой не смог его прочитать. Для этого необходимо выполнить следующие действия:

1. Пользователь Б посылает пользователю А свой открытый ключ U по любому каналу связи, например, по электронной почте.
2. Пользователь А шифрует свое сообщение М полученным открытым ключом U и получает зашифрованное сообщение С.
3. Зашифрованное сообщение С пересылается пользователю Б.
4. Пользователь Б расшифровывает полученное сообщение С своим закрытым ключом R.

Если операцию шифрования обозначить как F, а операцию расшифрования как F-1, то схему протокола обмена информацией между пользователями можно изобразить, как на [рис. 9.1](https://www.intuit.ru/studies/courses/691/547/lecture/12387?page=1" \l "image.9.1).



**Рис. 9.1.**Схема открытого шифрования

Использование открытого шифрования снимает проблему распределения ключей. Раньше пользователи перед обменом зашифрованными данными должны были каким-либо образом по закрытому каналу связи согласовывать используемый *секретный ключ*. Для этого они могли встретиться лично или использовать курьера. Если один из пользователей считал нужный изменить *ключ*, он должен был передать новый *ключ* своему абоненту. *Криптография* с открытыми ключами все упрощает. Теперь абоненты не должны заботиться о возможности компрометации секретного ключа. Пользователи системы связи могут совершенно свободно обмениваться открытыми ключами и зашифрованными ими сообщениями. Если *пользователь* надежно хранит свой *закрытый ключ*, никто не сможет прочитать передаваемые сообщения.

Для упрощения процедуры обмена в сети передачи сообщений обычно используется *база данных* (подробнее об этом см. в ["Криптографические алгоритмы с открытым ключом и их использование"](https://www.intuit.ru/studies/courses/691/547/lecture/12391)), в которой хранятся открытые ключи всех пользователей. При необходимости любой *пользователь* системы может запросить из базы *открытый ключ* другого человека и использовать полученный *ключ* для шифрования сообщений.

### Цифровая подпись на основе алгоритмов с открытым ключом

Как и все люди, абоненты *сети передачи данных* могут не доверять друг другу или вести себя нечестно. Они могут подделывать чужие сообщения, отрицать свое авторство или выдавать себя за другое лицо. Особенно актуальными становятся эти проблемы в связи с развитием электронной коммерции и возможностью оплаты услуг через *Интернет*. Поэтому во многих системах связи получатель корреспонденции должен иметь возможность удостовериться в подлинности документа, а создатель электронного послания должен быть в состоянии доказать свое авторство получателю или третьей стороне. Следовательно, электронные документы должны иметь аналог обычной физической подписи. При этом подпись должна обладать следующими свойствами:

1. подпись воспроизводится только одним лицом, а подлинность ее может быть удостоверена многими;
2. подпись неразрывно связывается с данным сообщением и не может быть перенесена на другой документ;
3. после того, как документ подписан, его невозможно изменить;
4. от поставленной подписи невозможно отказаться, то есть лицо, подписавшее документ, не сможет потом утверждать, что не ставило подпись.

*Асимметричные алгоритмы* шифрования могут быть использованы для формирования **цифровой (электронной) подписи** (*digital signature*) – уникального числового дополнения к передаваемой информации, позволяющего проверить ее авторство. *Электронная (цифровая) подпись* (*ЭЦП*) представляет собой последовательность *бит* фиксированной длины, которая вычисляется определенным образом с помощью содержимого подписываемой информации и секретного ключа.

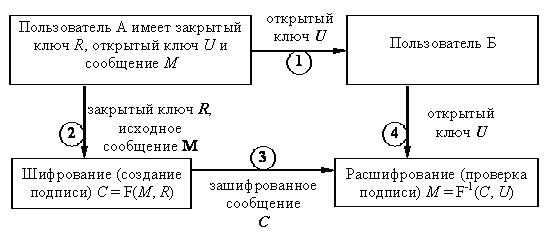
При формировании цифровой подписи специальным образом шифруется или все сообщение целиком, или результат вычисления хеш-функции от сообщения. Последний способ обычно оказывается предпочтительнее, так как подписываемое сообщение может иметь разный размер, иногда довольно большой, а хеш-код всегда имеет постоянную не очень большую длину. Рассмотрим подробнее оба варианта формирования *ЭЦП*.

Самый простой способ основывается, так же как и при открытом шифровании, на использовании пары связанных между собой ключей (открытого и закрытого). Однако роли закрытого и открытого ключей меняются – *ключ* подписывания становится секретным, а *ключ* проверки – открытым. Если при этом сохраняется свойство, что по открытому ключу нельзя практически найти *закрытый ключ*, то в качестве подписи может выступать само сообщение, зашифрованное секретным ключом. Таким образом подписать сообщение может только владелец закрытого ключа, но каждый, кто имеет его *открытый ключ*, может проверить подпись.

Пусть, например, *пользователь* А хочет отправить пользователю Б подписанное сообщение. Процедура создания и проверки подписи состоит из следующих шагов:

1. Пользователь А посылает пользователю Б свой открытый ключ U по любому каналу связи, например, по электронной почте.
2. Пользователь А шифрует сообщение М своим закрытым ключом R и получает зашифрованное сообщение С.
3. Зашифрованное сообщение пересылается пользователю Б.
4. Пользователь Б расшифровывает полученное сообщение С, используя открытый ключ пользователя А. Если сообщение расшифровалось, значит, оно подписано пользователем А.

Этот протокол можно изобразить в виде схемы, как на [рис. 9.2](https://www.intuit.ru/studies/courses/691/547/lecture/12387?page=2" \l "image.9.2).



**Рис. 9.2.**Первый вариант схемы создания и проверки цифровой подписи

До тех пор, пока *пользователь* А надежно хранит свой *закрытый ключ*, его подписи достоверны. Кроме того, невозможно изменить сообщение, не имея доступа к закрытому ключу абонента А; тем самым обеспечивается *аутентичность* и *целостность* данных.

Физическое *представление* пары ключей зависит от конкретной системы, поддерживающей использование *ЭЦП*. Чаще всего *ключ* записывается в *файл*, который, в *дополнение* к самому ключу, может содержать, например, информацию о пользователе - владельце ключа, о сроке действия ключа, а также некий набор данных, необходимых для работы конкретной системы (подробнее об этом см. ["Электронная цифровая подпись"](https://www.intuit.ru/studies/courses/691/547/lecture/12393)). Данные о владельце ключа позволяют реализовать другую важную функцию *ЭЦП* - установление авторства, поскольку при проверке подписи сразу же становится ясно, кто подписал то или иное сообщение. Обычно программные продукты, осуществляющие *проверку ЭЦП*, настраиваются так, чтобы результат исполнения появлялся на экране в удобном для восприятия виде с указанием поставившего подпись пользователя, например, так:

"Подпись файла приказ.doc верна (Автор: Соколов А.И.)"

На [рис. 9.2](https://www.intuit.ru/studies/courses/691/547/lecture/12387?page=2" \l "image.9.2)представлена схема формирования так называемой **цифровой подписи с восстановлением документа**. Цифровые подписи с восстановлением документа как бы содержат в себе подписываемый документ: в процессе проверки подписи автоматически вычисляется и *тело документа*. Если при расшифровывании сообщение восстановилось правильно, значит, подпись была верной. *Цифровая подпись* с восстановлением документа может быть реализована, например, с помощью одного из самых популярных алгоритмов формирования *ЭЦП* – *RSA*.

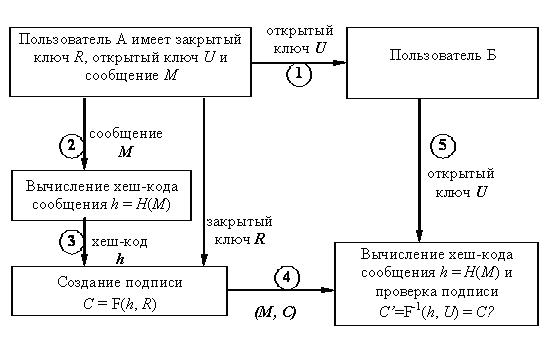
В случае использования цифровой подписи с восстановлением документа все сообщение целиком подписывается, то есть шифруется. В настоящее время на практике так обычно не делается. *Алгоритмы шифрования* с открытым ключом достаточно медленные, кроме того, для подтверждения целостности сообщения требуется много памяти. К тому же практически все применяемые алгоритмы вычисления *ЭЦП* используют для расчета сообщения заранее заданной стандартной длины. Например, в российском алгоритме формирования цифровой подписи ГОСТ Р34.10-94 этот размер определен равным 32 байтам. Поэтому для экономии времени и вычислительных ресурсов, а также для удобства работы асимметричный *алгоритм* обычно используется вместе с какой-нибудь однонаправленной хеш-функцией. В этом случае вначале с помощью хеш-функции из сообщения произвольной длины вычисляется хеш-код нужного размера, а затем для вычисления *ЭЦП* производится *шифрование* полученного на предыдущем этапе хеш-кода от сообщения.

*ЭЦП*, вычисленные по хеш-коду документа, называют **присоединяемыми цифровыми подписями**. Такие цифровые подписи представляют собой некоторый числовой код, который необходимо пристыковывать к подписываемому документу. Само сообщение при этом не шифруется и передается в открытом виде вместе с цифровой подписью отправителя.

Если *пользователь* А хочет отправить пользователю Б сообщение М, дополненное присоединенной цифровой подписью, то процедура создания и проверки подписи должна состоять из следующих шагов:

1. Пользователь А посылает пользователю Б свой открытый ключ U по любому каналу связи, например, по электронной почте.
2. Пользователь А с помощью некоторой надежной хеш-функции Н вычисляет хеш-код своего сообщения h = H(M).
3. Затем пользователь А шифрует хеш-код сообщения h своим закрытым ключом R и получает цифровую подпись С.
4. Исходное сообщение М вместе с цифровой подписью С пересылаются пользователю Б.
5. Пользователь Б вычисляет хеш-код h полученного сообщения М, а затем проверяет цифровую подпись С, используя открытый ключ пользователя А.

Этот протокол можно изобразить в виде схемы, как на [рис. 9.3](https://www.intuit.ru/studies/courses/691/547/lecture/12387?page=2" \l "image.9.3).



**Рис. 9.3.**Второй вариант схемы создания и проверки цифровой подписи

Хеш-*функция* не являются частью алгоритма *ЭЦП*, поэтому в схеме может быть использована любая надёжная хеш-*функция*.

Описанный процесс создания подписи не обеспечивает *конфиденциальность*. То есть сообщение, посланное таким способом, невозможно изменить, но можно прочитать. Даже если не использовать хеш-функцию, а шифровать все сообщение целиком, *конфиденциальность* не обеспечивается, так как любой может расшифровать сообщение, используя *открытый ключ* отправителя.

Во многих ситуациях приведенной схемы создания и использования цифровой подписи оказывается вполне достаточно. Однако бывают случаи, когда *пользователь* Б может смошенничать. Предположим, что пересылаемым документом был чек от пользователя А, например, за оказанные услуги. *Пользователь* Б удостоверился, что *цифровая подпись* на нем верная и использовал его для получения денег. Никто не может помешать пользователю Б снять одну или несколько копий с подписанного документа (тем более, если документ электронный) и периодически с небольшим интервалом предъявлять их в банк для получения денег.

Для устранения возможности такого жульничества в цифровые подписи часто включают *метки времени*. Дата и время подписания документа добавляются к сообщению и подписываются вместе со всем документом. При оплате чека *метка* времени может быть зафиксирована банком и занесена в базу данных. При попытке повторного предъявления чека банк это обнаружит и примет соответствующие меры.

Разновидностью цифровой подписи является *неотрицаемая цифровая подпись*. Как и обычная *цифровая подпись*, неотрицаемая *цифровая подпись* зависит от подписываемого документа и закрытого ключа автора. В отличие от обычной *ЭЦП* неотрицаемая подпись не может быть проверена без разрешения подписавшего. Таким образом, получатель корреспонденции не сможет показать подпись (или не сможет доказать правильность подписи) без согласия лица, подписавшего сообщение.

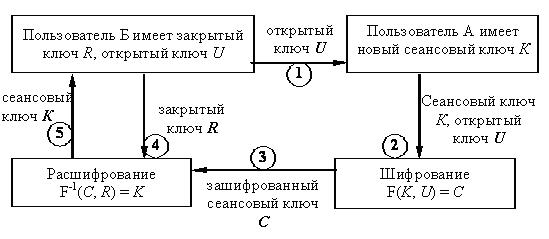
### Формирование секретных ключей с использованием асимметричных алгоритмов

На практике алгоритмы с открытым ключом редко используются для непосредственного шифрования сообщений. Этому препятствует относительная невысокая скорость *асимметричных алгоритмов* при шифровании и расшифровании больших объемов данных. Данный фактор связан с тем, что основной операцией в системах с открытым ключом является возведение в степень по большому модулю 500-1000 битовых чисел, что при программной реализации производится намного медленнее, чем *шифрование* того же объема данных классическими симметричными способами. Однако при обработке коротких блоков данных, например, ключей определенной длины, *алгоритмы шифрования* с открытым ключом могут использоваться достаточно эффективно. Поэтому часто используют следующую комбинированную схему: асимметричный *алгоритм* применяется для согласования *ключа сессии*, а затем этот *ключ* выступает в роли секретного ключа для шифрования сообщений *симметричным алгоритмом*.

Простейший протокол формирования секретного *ключа сессии* может выглядеть следующим образом (если пользователи некоторой системы связи имеют *доступ* к базе данных открытых ключей абонентов системы, предоставляемой центром распределения ключей, то они могут получать из нее открытые ключи друг друга):

1. Пользователь А получает открытый ключ пользователя Б из *центра распределения ключей* или непосредственно от пользователя Б.
2. Пользователь А генерирует случайный сеансовый ключ и зашифровывает его полученным открытым ключом.
3. Зашифрованный сеансовый ключ пересылается пользователю Б.
4. Пользователь Б расшифровывает полученный пакет своим закрытым ключом.
5. Пользователи А и Б используют согласованный сеансовый ключ для обмена шифрованными сообщениями.

Схему формирования парой пользователей А и В общего секретного ключа К для шифрования - расшифрования можно изобразить следующим образом ([рис. 9.4](https://www.intuit.ru/studies/courses/691/547/lecture/12387?page=3" \l "image.9.4)).



**Рис. 9.4.**Схема формирования общего секретного ключа

Эта схема напоминает [рис. 9.1](https://www.intuit.ru/studies/courses/691/547/lecture/12387?page=1" \l "image.9.1), и это неудивительно, ведь в ней используется тот же самый режим шифрования открытым ключом. Отличие заключается в том, *что* шифруется. В схеме на [рис. 9.4](https://www.intuit.ru/studies/courses/691/547/lecture/12387?page=3" \l "image.9.4) производится *шифрование* небольшого по размеру сеансового ключа, который в дальнейшем будет использован в качестве секретного ключа при шифровании *симметричным алгоритмом*. *Шифрование* небольшого по размеру блока данных выполняется достаточно быстро и не замедляет телекоммуникационные процессы даже в системе с многими тысячами пользователей.

Существуют и более сложные протоколы распределения ключей, обеспечивающие взаимное подтверждение подлинности участников сеанса связи, подтверждение достоверности сеанса *механизмом запроса*-ответа или другие требования.

### Требования к алгоритмам шифрования с открытым ключом

Рассмотрев основные способы применения алгоритмов шифрования с открытым ключом, изучим требования, которым должен, по мнению основоположников теории шифрования с открытым ключом Диффи и Хеллмана, удовлетворять *алгоритм* шифрования с открытым ключом. Эти требования следующие:

1. Вычислительно легко создавать пару (открытый ключ, закрытый ключ).
2. Вычислительно легко зашифровать сообщение открытым ключом.
3. Вычислительно легко расшифровать сообщение, используя закрытый ключ.
4. Вычислительно невозможно, зная открытый ключ, определить соответствующий закрытый ключ.
5. Вычислительно невозможно, зная только открытый ключ и зашифрованное сообщение, восстановить исходное сообщение.

Из этих общих требований видно, что реализация конкретного алгоритма с открытым ключом зависит от соответствующей *односторонней функции*.

Математиками и криптографами предложено большое количество алгоритмов шифрования с открытым ключом, основанных на различных односторонних функциях. Некоторые алгоритмы можно задействовать тремя, рассмотренными ранее в данной лекции способами, в то время как другие могут использоваться только одним или двумя способами. Мы рассмотрим четыре алгоритма с открытым ключом, три из которых достаточно давно применяются на практике, а четвертый вид алгоритмов совсем недавно начал применяться в системах защиты информации. Эти алгоритмы используются обычно для различных целей, что отражено в следующей таблице.

|  |  |  |  |
| --- | --- | --- | --- |
| **Название алгоритма** | **Возможность использования** | | |
| Шифрование / расшифрование данных | Цифровая подпись | Согласование или формирование ключа |
| RSA | Да | Да | Да |
| *Алгоритм Диффи-Хеллмана* | Нет | Нет | Да |
| Алгоритм Эль-Гамаля | Да | Да | Да |
| Алгоритмы с использованием эллиптических кривых | Да | Да | Да |

Перед изучением наиболее известных алгоритмов шифрования с открытым ключом, необходимо напомнить, что все *асимметричные алгоритмы* основаны на свойствах тех или иных математических функций. Доказательства правильности работы рассматриваемых алгоритмов могут быть достаточно сложными, поэтому мы ограничимся изучением основных принципов их работы. Многие криптографические алгоритмы базируются на результатах классической теории чисел. Основные факты и положения этой теории, необходимые для понимания алгоритмов и выполнения упражнений, сформулированы в ["Основные положения теории чисел, используемые в криптографии с открытым ключом"](https://www.intuit.ru/studies/courses/691/547/lecture/12389).

### Ключевые термины

**Алгоритм шифрования с открытым ключом** (или **асимметричные криптоалгоритмы** ) – криптографический *алгоритм*, в котором для шифрования и расшифрования используются разные ключи.

**Закрытый ключ** – *ключ*, используемый в асимметричных криптографических алгоритмах, который должен храниться в секрете.

**Односторонняя функция** – *математическая функция*, которую относительно легко вычислить, но трудно найти по значению функции соответствующее *значение* аргумента. То есть, зная х легко вычислить f(x), но по известному f(x) трудно найти подходящее *значение* x.

**Односторонняя функция с люком** (или **с секретом** ) – это особый вид *односторонних функций*, имеющих некоторый секрет (*люк*), позволяющий относительно быстро вычислить обратное *значение* функции.

**Открытый ключ** – *ключ*, используемый в асимметричных криптографических алгоритмах, который может не храниться в секрете.

**Присоединяемые цифровые подписи** – подписи, вычисленные по хеш-коду документа. Такие цифровые подписи представляют собой некоторый числовой код, который необходимо пристыковывать к подписываемому документу. Само сообщение при этом не шифруется и передается в открытом виде вместе с цифровой подписью отправителя.

**Цифровая (электронная) подпись** (*digital signature*) – уникальное числовое *дополнение* к передаваемой информации, позволяющее проверить ее авторство. *Электронная (цифровая) подпись* (*ЭЦП*) представляет собой последовательность *бит* фиксированной длины, которая вычисляется определенным образом с помощью содержимого подписываемой информации и секретного ключа.

**Цифровые подписи с восстановлением документа** – подписи, которые как бы содержат в себе подписываемый документ: в процессе проверки подписи автоматически вычисляется и *тело документа*. Если при расшифровывании сообщение восстановилось правильно, значит, подпись была верной.

### Краткие итоги

*Асимметричные алгоритмы* шифрования (или алгоритмы с открытым ключом) – криптографические алгоритмы, в которых один *ключ* используется для шифрования, а другой, отличный от первого, – для расшифрования. Алгоритмы называются асимметричными, так как ключи шифрования и расшифрования разные, следовательно, отсутствует *симметрия* основных криптографических процессов. Один из двух ключей является открытым (*public key*) и может быть объявлен всем, а второй – закрытым (*private key*) и должен держаться в секрете. Какой из ключей, открытый или закрытый, используется для шифрования, а какой для расшифрования, определяется назначением криптографической системы.

Все *алгоритмы шифрования* с открытым ключом основаны на использовании *односторонних функций*. Односторонней функцией называется *математическая функция*, которую относительно легко вычислить, но трудно найти по значению функции соответствующее *значение* аргумента. Использовать *односторонние функции* для шифрования сообщений с целью их защиты не имеет смысла, так как обратно расшифровать зашифрованное сообщение уже не получится. Для целей шифрования используются *односторонние функции с люком* (или с секретом) – особый вид *односторонних функций*, имеющих некоторый секрет (*люк*), позволяющий относительно быстро вычислить обратное *значение* функции.

*Алгоритмы шифрования* с открытым ключом можно использовать для решения следующих задач:

1. Для шифрования передаваемых и хранимых данных в целях их защиты от несанкционированного доступа.
2. Для формирования цифровой подписи под электронными документами.
3. Для распределения секретных ключей, используемых потом при шифровании документов симметричными методами.

Цифровая (электронная) подпись – уникальное числовое *дополнение* к передаваемой информации, позволяющее проверить ее авторство. *Электронная (цифровая) подпись* (*ЭЦП*) представляет собой последовательность *бит* фиксированной длины, которая вычисляется определенным образом с помощью содержимого подписываемой информации и секретного ключа. Различают присоединяемые цифровые подписи и цифровые подписи с восстановлением документа. Присоединяемые цифровые подписи – подписи, вычисленные по хеш-коду документа. Такие цифровые подписи представляют собой некоторый числовой код, который необходимо пристыковывать к подписываемому документу. Само сообщение при этом не шифруется и передается в открытом виде вместе с цифровой подписью отправителя. Цифровые подписи с восстановлением документа – подписи, которые как бы содержат в себе подписываемый документ: в процессе проверки подписи автоматически вычисляется и *тело документа*. Если при расшифровывании сообщение восстановилось правильно, значит, подпись была верной.

#### Вопросы для самопроверки

1. Чем *асимметричные алгоритмы* шифрования отличаются от симметричных?
2. Для решения каких задач могут на практике применяться алгоритмы шифрования с открытым ключом?
3. Какие математические функции называются односторонними? Для чего они могут применяться в криптографии?
4. Что такое цифровая подпись?
5. Каков алгоритм формирования цифровой подписи при использовании алгоритмов шифрования с открытым ключом?
6. Каким образом алгоритмы шифрования с открытым ключом могут использоваться для формирования общего секретного ключа у группы пользователей?
7. Какие требования предъявляются к асимметричным алгоритмам?

**Лекция 4: Основные положения теории чисел, используемые в криптографии с открытым ключом**

### ****Цель:****

Освоить ключевые математические основы, лежащие в основе алгоритмов асимметричной криптографии, включая операции с простыми числами, остатками и модульной арифметикой.

### ****Задание:****

#### ****1. Теоретическая часть: изучить и законспектировать****

* Понятие делимости, простых чисел и их свойства
* Евклидов алгоритм и расширенный алгоритм Евклида
* Модульная арифметика: операции по модулю
* Обратные элементы по модулю
* Теорема Эйлера и функция Эйлера φ(n)
* Маленькая теорема Ферма
* Остаточные классы и китайская теорема об остатках (обзорно)

#### ****2. Практическая часть:****

* Вычислить:
  + НОД двух заданных чисел с помощью алгоритма Евклида
  + Обратный элемент по модулю (например, 7^(-1) mod 26)
  + φ(n) для нескольких значений n
* Привести пример использования этих операций в RSA (например, как выбирается e и d)

#### ****3. Подготовить:****

* Презентацию (8–10 слайдов) с краткими объяснениями каждого математического инструмента
* Мини-доклад (1 страница) о роли теории чисел в криптографии и примерах её применения

**Цель лекции:** напомнить читателю основные положения теории чисел, используемые в криптографии с открытым ключом

### Простые и составные числа

Каждое *натуральное число*, большее единицы, делится по крайней мере на два числа: на 1 и на само себя. Если число не имеет делителей, кроме самого себя и единицы, то оно называется **простым**, а если у числа есть еще делители, то **составным**. *Единица* же не считается ни простым числом, ни составным. Например, числа 7, 29 — простые; числа 9, 15 — составные ( 9 делится на 3, 15 делится на 3 и на 5 ).

Интересный факт: если два простых числа отличаются на 2, то их называют числами-"близнецами". Чисел-"близнецов" не очень много. Например, "близнецами" являются 5 и 7, 29 и 31, 149 и 151, а также 242 206 083\*238 880±1 (наибольшая найденная на момент написания учебного пособия пара "близнецов").

Не о всяком числе можно сразу сказать, простое оно или составное. Если число меньше ста, то, скорее всего мы сразу сможем ответить на этот вопрос. Однако с большими числами дело сложнее. Возьмем, например, число 2009. Простое оно или составное? Попробуем найти возможные делители этого числа среди первых простых чисел. 2009 определенно не делится на 2 (так как оно нечетное), на 3 (так как сумма его цифр 2+9=11 не делится на 3 ), на 5. А вот, попробовав разделить 2009 на 7, мы увидим, что в результате получается *целый* результат – 287. Таким образом, получен ответ: число 2009 – составное. В данном случае ответ получен достаточно быстро. Бывает, что проверка на простоту производится гораздо дольше, а для работы с большими целыми числами требуются даже специальные компьютерные программы.

*Поиск* больших простых чисел имеет важное *значение* для математики и не только. Например, в криптографии большие простые числа используются в алгоритмах шифрования с открытым ключом. Для обеспечения надежности шифрования там используются простые числа длиной до 1024 *бит*.

Перемножить два числа сравнительно нетрудно, особенно если у нас есть калькулятор, а числа не слишком велики. Существует и обратная задача – *задача факторизации* – нахождение двух или более чисел, дающих при перемножении заданное число. Эта задача гораздо труднее, чем перемножение чисел, и любому, кто пытался ее решить, об этом известно. Например, если от нас требуется умножить 67 на 113, то результат, 7571, будет получен, наверно, меньше чем за минуту. Если же от нас требуется найти два числа, *произведение* которых равно 7571, то, скорее всего, это займет у нас гораздо больше времени.

*Поиск* сомножителей числа n может вестись, например, перебором всех простых чисел до Описание: \sqrt n, как в рассмотренном выше примере с числом 2009. Однако, если множители – большие простые числа, то на их *поиск* уйдет достаточно много времени.

Таким образом, факторизация большого числа требует значительных затрат времени даже в том случае, когда известно, что оно является произведением двух больших простых чисел.

Сложность задачи факторизации используется в некоторых криптографических алгоритмах, например, в системе шифрования *RSA*.

### Основная теорема арифметики

Любое составное число можно составить из некоторого количества простых с помощью умножения. Например, составное число 2009 можно получить так:

2009 = 7 \* 7 \* 41

В математике рассматривается так называемая **основная теорема арифметики**, которая утверждает, что любое *натуральное число* ( n>1 ) либо само является простым, либо может быть разложено на *произведение* простых делителей, причем единственным способом (если не обращать внимания на порядок следования сомножителей).

Воспользовавшись обозначением степени, разложение числа 2009 на простые множители можно записать так:

2009 = 72 \* 41

Разложение на множители называется **каноническим**, если все множители являются простыми и записаны в порядке возрастания.

Например, запишем *каноническое разложение* числа 150 на множители:

150 = 2 \* 3 \* 52

### Взаимно простые числа и функция Эйлера

Два числа называются взаимно простыми, если они не имеют ни одного общего делителя кроме единицы.

Например, числа 11 и 12 взаимно просты (у них нет общих делителей кроме единицы), числа 30 и 35 — нет (у них есть общий делитель 5 ).

Исследованием закономерностей, связанных с целыми числами, долго занимался швейцарский математик Леонард Эйлер (Leonard Euler). Одним из вопросов, которым он интересовался, был следующий: сколько существует натуральных чисел, не превосходящих n и взаимно простых с n? Ответ на этот вопрос был получен Эйлером в 1763 году и этот ответ связан с каноническим разложением числа n на простые множители. Так, если

Описание: n=p_1^{a_1}\times p_2^{a_2}\times \dots \times p_n^{a_n}

где p1, p2, ..., pn – разные простые множители, то число Описание: \phi натуральных чисел, не превосходящих n и взаимно простых с n можно точно определить по формуле

Описание: \varphi (n) = n \times \left ( 1- \frac{1}{p_1} \right ) \times \left ( 1- \frac{1}{p_2} \right ) \times \dots \times \left ( 1- \frac{1}{p_n} \right )

Число натуральных чисел, не превосходящих n и, взаимно простых с n, называется **функцией Эйлера** и обозначается Описание: \phi(n).

Например, найдем количество натуральных чисел, не превосходящих 12 и взаимно простых с 12. Из ряда натуральных чисел

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

взаимно простыми (не имеющими общих делителей) с 12 будут только числа 1, 5, 7, 11. Их количество равно четырем. Таким образом Описание: \phi(12) = 4.

Теперь попробуем подсчитать

Описание: \phi(12)

по формуле, предложенной Эйлером. Для этого вначале запишем *каноническое разложение* числа 12:

12 = 22 \* 3.

Теперь подсчитаем функцию Эйлера Описание: \phi(12):

Описание: \varphi (12) = 12 \times \left ( 1- \frac{1}{2} \right ) \times \left ( 1- \frac{1}{3} \right ) = 4 \times 3 \times \frac{1}{2} \frac{2}{3} = 4

Значения, вычисленные путем простого перебора взаимно простых чисел и по формуле Эйлера, совпали. Это неудивительно, так как формула для вычисления функции Эйлера может быть доказана строго математически.

Формулу Эйлера удобно использовать для больших n, если известно разложение числа n на простые множители. Для криптографии формула Эйлера важна тем, что она позволяет легко получить число

Описание: \phi(n)

для простых и некоторых других чисел. В криптографии используются два следующих следствия формулы Эйлера.

**Следствие 1**. Если p – *простое число*, то Описание: \phi(p) = p - 1.

Действительно, если p – *простое число*, то его *каноническое разложение* состоит только из него самого. Тогда

Описание: \varphi (p) = p \times \left ( 1- \frac{1}{p} \right ) = \frac{p(p-1)}{p} = p-1

**Следствие 2**. Пусть р и q — два различных простых ( Описание: p\ne q). Тогда

Описание: \phi (p*q) = (p-1)(q-1)

Эта формула объясняется следующим образом. Пусть р \* q = N, где р и q — два различных простых ( Описание: p\ne q ). Тогда

Описание: \varphi (p \times q) = \varphi (N)=N \times \left ( 1- \frac{1}{p} \right ) \times \left ( 1- \frac{1}{q} \right )= \frac{p \times q \times (p-1) \times (q-1)}{p \times q} = (p-1) \times (q-1)

Рассмотрим несколько примеров использования следствий формулы Эйлера.

**Пример 1**. Найдем Описание: \phi(13). 13 – *простое число*, значит, используя следствие 1 Описание: \phi(13) = 13 – 1 =12. Мы можем проверить себя (и Эйлера), выписав все числа, меньшие 13:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

и подсчитав все взаимно простые с ним. Их действительно 12.

**Пример 2**. Найдем Описание: \phi(35). 35 – составное число, значит, первое следствие нам не подходит. Однако 35 является произведением двух простых чисел: 35 = 5 \* 7. Используя следствие 2, вычисляем Описание: \phi(35):

Описание: \phi (35) = (5 – 1) * (7 - 1) = 4 * 6 = 24.

Проверяем, выписывая все числа, меньшие 35 и не имеющие с ним общих делителей:

1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 26, 27, 29, 31, 32, 33, 34.

Их действительно оказалось 24. На последнем примере видно, что использовать формулу Эйлера гораздо удобнее, чем рассматривать все числа из довольно большого диапазона и проверять на взаимную простоту.

### Арифметика остатков и теория сравнений

Немецкий математик Карл Фридрих Гаусс предложил *запись*

Описание: a \equiv b (mod\ m),

для двух чисел a и b, если они имеют одинаковые остатки от деления на m (читается a сравнимо с b по модулю m ). Например,

Описание: 1997 \equiv 1(mod 4),\\
7k +1 \equiv 1(mod 7),\ где\ k - любое\ целое\ число.

Сравнения обнаруживают полезные для математиков и криптографов свойства, во многом похожие на свойства равенств. Эти свойства позволяют существенно упрощать арифметические вычисления, если нас интересует только *остаток от деления* на некоторое число m. Так, например, свойства сравнений полезны при расчетах в алгоритмах шифрования с открытым ключом.

Простейшими свойствами сравнений являются следующие.

**Свойство 1**. Если a-b делится на m, то Описание: a \equiv b (mod\ m).

Например, Описание: 15 \equiv 1 (mod\  7), так как 15 -1 =14, а 14 кратно 7.

**Свойство 2**. Если

Описание: a \equiv b (mod\ m)

и

Описание: c \equiv d (mod\ m)

то

Описание: a+c \equiv b+d (mod\ m)\\
 ac \equiv bd (mod\ m)

Например, так как Описание: 13\equiv5(mod\ 8) и Описание: 11\equiv3(mod\ 8), то Описание: 13+11\equiv5+3\equiv0(mod\ m), а также Описание: 13*11\equiv5*3\equiv7(mod\ 8).

**Свойство 3**. Если

Описание: a \equiv b (mod\ m),

то

Описание: то\ a^k \equiv b^k (mod\ m), k \subset  N.

Например, так как

Описание: 25\equiv4(mod\ 7),

то

Описание: 25^{40} \equiv 4^{40} \equiv (4^2)^{20} \equiv 2^{20} \equiv (2^4)^5 \equiv 2^5 \equiv 4 (mod\ m)

**Свойство 4**. Если

Описание: ac \equiv bc (mod\ m)

и c взаимно просто с m, то

Описание: a \equiv b (mod\ m)

Например, известно, что

Описание: 1200 \equiv 45 (mod\ 7),

а так как

Описание: 1200=15*80\ и\ 45 =15*3, то\\
80 \equiv 3 (mod\ 7)

**Свойство 5**. Если

Описание: ac \equiv bc (mod\ mc),

то

Описание: a \equiv b (mod\ m)

Например, если

Описание: 44 \equiv 4 (mod\ 4(10),

то

Описание: 11 \equiv 1 (mod\ 10).

### Малая теорема Ферма

В основе алгоритма шифрования по системе *RSA* лежит теорема, сформулированная в начале семнадцатого столетия без доказательства французским математиком Пьером Ферма (Pierre Fermat). Её часто называют "Малой *теоремой Ферма*", и её не следует путать с известной "Великой *теоремой Ферма*" - её он также сформулировал без доказательства, а доказана она была только в 1993-94 годах. Леонард Эйлер в 1760 году опубликовал *доказательство* Малой теоремы Ферма и получил ее *обобщение*, известное под названием теоремы Ферма-Эйлера. Именно эта теорема используется в алгоритме зашифрования/расшифрования *RSA*.

**Малая теорема Ферма** формулируется следующим образом. Если p - *простое число*, а m - любое число, которое не делится на p, то

Описание: m^{p-1} \equiv 1(mod\ p),

то есть число mp-1 при делении на p дает *остаток* 1.

Например, пусть р=11, m = 3. Проверим, будет ли 310 mod 11 равно одному:

310 mod 11=32( ((32)2)2mod 11)) = 9(42 mod 11 )= 144 mod 11=1

*Обобщение*, сформулированное и доказанное Эйлером, справедливо для любого модуля, но в системе *RSA* используется частный случай, когда *модуль* является произведением только двух различных простых чисел. Поэтому рассмотрим формулировку теоремы для этого случая.

**Теорема Ферма-Эйлера** (для случая системы *RSA*). Если p и q - два различных простых числа, а m - любое число, которое не делится на p и q, то

Описание: m^{(p-1)(q-1)} \equiv 1(mod\ pq).

Например, пусть р=11, q = 5 (pq = 55), m = 3. Проверим, будет ли

Описание: 3^{40} \equiv 1(mod\ 55)

равно одному:

340 mod 55 = (35) 4mod 55 = 234 mod 55 = 279841 mod 55 = 1.

### Наибольший общий делитель

Пусть а и b — два целых положительных числа. Наибольший общий делитель чисел а и b есть наибольшее число с, которое делит и а, и b:

с = НОД(a, b).

Например, НОД(25,35) =5.

Для нахождения наибольшего общего делителя можно использовать следующий *алгоритм*, известный как **алгоритм Евклида**.

Алгоритм NOD (целые a, b, c);

Начало

1. Пока a<>b выполнять:

1.1.Если a>b то a:=a-b, иначе b:= b-a;

2. c:=a;

Конец.

После выполнения алгоритма результат будет содержаться в переменной с.

Поcмотрим, как с помощью алгоритма Евклида вычисляется НОД(18,9):

a: 18 9

b: 9 9

c: 9 9

Здесь каждый столбец представляет собой очередную итерацию алгоритма. Процесс продолжается до тех пор, пока b не станет равным a. Тогда в переменную с записывается ответ, в данном случае 9. Это и будет *значение* НОД(18,9).

### Обобщенный алгоритм Евклида

Для многих криптографических систем, рассмотренных в данной книге, актуален так называемый *обобщенный алгоритм Евклида*, с которым связана следующая теорема.

**Теорема**. Пусть а и b — два целых положительных числа. Тогда существуют целые (не обязательно положительные) числа х и у, такие, что

ах + by = НОД(a,b).

Обобщенный *алгоритм* Евклида служит для отыскания НОД(a, b) и х, у, удовлетворяющих записанному выше уравнению. Введем три строки U = (u1,u2,u3), V = (v1,v2,v3) и T = (t1,t2,t3).

*Алгоритм* записывается следующим образом (во входных параметрах должно соблюдаться условие a>=b ).

Алгоритм OAE (целые a, b);

Начало

1. U = (a,1,0), V = (b,0,1).

2. Пока v1 <> 0 выполнять:

2.1. q= u1 div v1;

2.2. T=(u1 mod v1, u2 – qv2, u3 – qv3);

2.3. U=V, V=T.

3. U=(НОД(a,b),x,y)).

Конец.

После окончания алгоритма результат будет содержаться в строке U. Операция div в алгоритме — это операция целочисленного деления.

**Пример**. Пусть а = 18, b = 9. Найдем числа х и у, удовлетворяющие уравнению

18х +9y = НОД(18,9).

Выполним *алгоритм* по шагам:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| q | U | | | V | | | T | | |
| u1 | u2 | u3 | v1 | v2 | v3 | t1 | t2 | t3 |
| 18 div 9 = 2 | 18 | 1 | 0 | 9 | 0 | 1 | 18 mod 9 = 0 | 1 - 2\*0 = 1 | 0 - 2\*1 = -2 |
|  | 9 | 0 | 1 | 0 | 1 | -2 |  |  |  |

В результате получили U=(НОД(a,b),x,y))= (9,0,1).

Выполним проверку: 18\*0 +9\*1 = 9 = НОД(18,9).

### Инверсия по модулю m

Во многих задачах криптографии для заданных чисел с, m требуется находить такое число d < m, что

cd mod m = 1

Такое d существует тогда и только тогда, когда числа с и m взаимно простые. Число d, удовлетворяющее равенству cd mod m = 1, называется **инверсией** с **по модулю** m и часто обозначается с-1 mod m. Данное обозначение для инверсии связано с тем, что *равенство* cd mod m = 1 можно переписать в виде

cс-1 mod m = 1.

Таким образом, *умножение* на с-1 соответствует делению на с при вычислениях по модулю m.

Инверсию по модулю m также можно вычислять с помощью обобщенного алгоритма Евклида.

Покажем, как это делается. *Равенство*, приведенное ниже означает, что для некоторого целого k имеет *место* *равенство* cd – km = 1. Учитывая, что с и d взаимно просты, можно преобразовать это *равенство* следующим образом:

m(-k) + cd = НОД(m,c).

Значит, мы можем вычислить с-1 mod m (или найти число d ) с помощью обобщенного алгоритма Евклида. При этом *значение* переменной k нас не интересует. Если число d получается отрицательным, то нужно прибавить к нему m, так как по определению число a mod m берется из *множества* {0,1,..., m - 1}.

Рассмотрим **пример**. Пусть m=9, c=5. Найдем 5-1 mod 9. Будем выполнять вычисления по обобщенному *алгоритму Евклида*, записывая все вычисления по шагам.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| q | U | | | V | | | T | | |
| u1 | u2 | u3 | v1 | v2 | v3 | t1 | t2 | t3 |
| 9 div 5 = 1 | 9 | 1 | 0 | 5 | 0 | 1 | 9 mod 5 = 4 | 1 - 1\*0 = 1 | 0-1\*1 = -1 |
| 5 div 4 =1 | 5 | 0 | 1 | 4 | 1 | -1 | 5 mod 4=1 | 0-1\*1=-1 | 1-1\*(-1)=2 |
| 4 div 1 =4 | 4 | 1 | -1 | 1 | -1 | 2 | 4 mod 1=0 | 1-4\*(-1)=5 | -1-4\*2=-9 |
|  | 1 | -1 | 2 | 0 | 1 | 7 |  |  |  |

Таким образом, получили, что 5-1 mod 9 = 2. Проверим: 5\*2 mod 9 =10 mod 9 = 1.

### Ключевые термины

**Алгоритм Евклида** – математический *алгоритм*, который может использоваться для нахождения наибольшего общего делителя двух чисел.

**Взаимно простые числа** – числа, не имеющие общих делителей (кроме единицы).

**Задача факторизации** – нахождение двух или более натуральных чисел, дающих при перемножении заданное число.

**Инверсия по модулю** – такое *натуральное число*, которое при умножении по модулю на данное число дает в результате единицу.

**Каноническое разложение на множители** – такое разложение на множители, при котором все множители являются простыми и записаны в порядке возрастания.

**Малая теорема Ферма** – известная теорема, сформулированная П. Ферма, лежащая в основе алгоритма шифрования по системе *RSA*

**Наибольший общий делитель** чисел а и b – наибольшее число с, которое делит и а и b: с = НОД(a, b).

**Основная теорема арифметики** – теорема утверждающая, что любое *натуральное число* большее единицы либо само является простым, либо может быть разложено на *произведение* простых делителей, причем единственным способом (если не обращать внимания на порядок следования сомножителей).

**Простое число** – *натуральное число*, которое не имеет делителей, кроме самого себя и единицы.

**Составное число** – *натуральное число*, которое делится, помимо самого себя и единицы, еще хотя бы на одно число.

**Функция Эйлера** позволяет подсчитать число натуральных чисел, не превосходящих n и, взаимно простых с n. Обозначается

Описание: \phi(n).

### Краткие итоги

Криптографические методы с открытым ключом, в большинстве своем, основаны на операциях с большими целыми числами.

Каждое *натуральное число*, большее единицы, делится по крайней мере на два числа: на 1 и на само себя. Если число не имеет делителей, кроме самого себя и единицы, то оно называется простым, а если у числа есть еще делители, то составным. *Единица* же не считается ни простым числом, ни составным. Для больших чисел задача проверки на простоту может быть достаточно сложной. Также вычислительно сложной является задача факторизации – нахождение двух или более натуральных чисел, дающих при перемножении заданное число.

Два числа называются взаимно простыми, если они не имеют ни одного общего делителя кроме единицы.

Наибольший общий делитель чисел а и b – наибольшее число с, которое делит и а и b. Для нахождения наибольшего общего делителя можно использовать *алгоритм* Евклида.

*Инверсия* по модулю – такое *натуральное число*, которое при умножении по модулю на данное число дает в результате единицу. Инверсию по модулю m можно вычислять с помощью так называемого обобщенного алгоритма Евклида.

#### Вопросы для самопроверки

1. Дайте определение простого и составного числа. Приведите по три примера простых и составных чисел.
2. Дать определение понятия "взаимно простые числа". Привести примеры взаимно простых чисел и чисел, не являющихся взаимно простыми.
3. Сформулируйте основную теорему арифметики.
4. В чем заключается задача факторизации?
5. Дайте определение наибольшего общего делителя.
6. Сформулируйте алгоритм Евклида для нахождения наибольшего общего делителя двух чисел.
7. Сформулируйте малую теорему Ферма.
8. Сформулируйте *теорема Ферма*-Эйлера (для случая системы RSA).
9. Сформулируйте обобщенный (расширенный) алгоритм Евклида.
10. Сформулировать принципы выполнения "операции взятия по модулю". Приведите примеры выполнения этой операции и поясните их.
11. Что такое инверсия по модулю n?

**Лекция №5: Криптографические алгоритмы с открытым ключом и их использование**

### ****Цель:**** Познакомиться с основными алгоритмами криптографии с открытым ключом и понять их принципы, применение и область использования.

### ****Задание:****

#### ****1. Теоретическая часть: изучить и изложить в конспекте****

* Принципы криптографии с открытым ключом
* Отличие от симметричной криптографии
* Структура ключей: открытый и закрытый ключ
* Основные алгоритмы:
  + **RSA**: принцип работы, ключевая генерация, шифрование/дешифрование
  + **ElGamal**: идея алгоритма, плюсы/минусы
  + **ECC (эллиптические кривые)**: особенности и преимущества
  + Использование алгоритмов для **цифровой подписи**

#### ****2. Практическая часть:****

* Составить таблицу сравнения RSA, ElGamal и ECC по критериям:
  + Безопасность
  + Скорость
  + Размер ключей
  + Область применения
* Рассчитать простой пример шифрования и расшифровки с помощью RSA (на небольших числах)
* Привести примеры реального использования (например, HTTPS, PGP, цифровая подпись)

#### ****3. Подготовить:****

* Презентацию (10–12 слайдов) с кратким описанием алгоритмов и их применения
* Краткий письменный отчёт (1–2 страницы) по выполненным задачам

**Цель лекции**: изучить принципы работы некоторых криптографических алгоритмов с открытым ключом.

### Алгоритм RSA

#### Основные сведения

Алгоритм шифрования с открытым ключом **RSA** был предложен одним из первых в конце 70-х годов ХХ века. Его название составлено из первых букв фамилий авторов: Р.Райвеста (R.Rivest), А.Шамира (A.Shamir) и Л.Адлемана (L.Adleman). Алгоритм RSA является, наверно, наиболее популярным и широко применяемым *асимметричным алгоритмом* в криптографических системах.

Алгоритм основан на использовании того факта, что задача разложения большого числа на простые сомножители является трудной. Криптографическая система RSA базируется на следующих двух фактах из теории чисел:

1. задача проверки числа на простоту является сравнительно легкой;
2. задача разложения чисел вида n = pq ( р и q — простые числа); на множители является очень трудной, если мы знаем только n, а р и q — большие числа (это так называемая задача факторизации, подробнее о ней см. ["Основные положения теории чисел, используемые в криптографии с открытым ключом"](https://www.intuit.ru/studies/courses/691/547/lecture/12389)).

Алгоритм RSA представляет собой блочный алгоритм шифрования, где зашифрованные и незашифрованные данные должны быть представлены в виде целых чисел между 0 и n -1 для некоторого n.

#### Шифрование

Итак, рассмотрим сам алгоритм. Пусть абонент А хочет передать зашифрованное сообщение абоненту Б. В этом случае абонент Б должен подготовить пару (открытый ключ; закрытый ключ) и отправить свой открытый ключ пользователю А.

Первым этапом является генерация открытого и закрытого ключей. Для этого вначале выбираются два больших простых числа Р и Q. Затем вычисляется произведение N:

N = PQ.

После этого определяется вспомогательное число f:

f = (Р - l)(Q - 1).

Затем случайным образом выбирается число d < f и взаимно простое с f.

Далее необходимо найти число е, такое, что

еd mod f = 1.

Числа d и N будут открытым ключом пользователя, а значение е – закрытым ключом.

Таким образом, на этом этапе у пользователя должна быть информация, указанная в следующей таблице:

|  |  |  |
| --- | --- | --- |
|  | Открытый ключ | Закрытый ключ |
| Пользователь системы | N, d | e |

Так как пользователь Б хочет получить зашифрованное сообщение от пользователя А, значит пользователь Б должен отправить свой открытый ключ (d, N) пользователю А. Числа Р и Q больше не нужны, однако их нельзя никому сообщать; лучше всего их вообще забыть.

На этом этап подготовки ключей закончен и можно использовать основной протокол RSA для шифрования данных.

**Второй этап – шифрование данных**. Если абонент А хочет передать некоторые данные абоненту Б, он должен представить свое сообщение в цифровом виде и разбить его на блоки m1, m2, m3, ... , где mi < N. Зашифрованное сообщение будет состоять из блоков сi.

Абонент А шифрует каждый блок своего сообщения по формуле

ci = mid mod N

используя *открытые параметры* пользователя Б, и пересылает зашифрованное сообщение С=(с1, с2, с3, ...) по открытой линии.

Абонент Б, получивший зашифрованное сообщение, расшифровывает все блоки полученного сообщения по формуле

mi = ce mod N

Все расшифрованные блоки будут точно такими же, как и исходящие от пользователя А.

Злоумышленник, перехватывающий все сообщения и знающий всю открытую информацию, не сможет найти исходное сообщение при больших значениях Р и Q.

#### Пример вычислений по алгоритму

Пусть пользователь А хочет передать пользователю Б сообщение. В этом случае вначале пользователь Б должен подготовить открытый и закрытый ключи. Пусть им выбраны, например, следующие параметры:

Р = 3, Q = 11, N = 3x11 = 33.

Тогда f = (Р - l)(Q - 1) = (3-1)(11-1) = 20.

Затем пользователь Б выбирает любое число d, не имеющее общих делителей с f (это необходимо для того, чтобы зашифрованное сообщение можно было потом однозначно восстановить). Пусть d = 13. Это число будет одним из компонентов открытого ключа.

Далее необходимо найти число е, которое можно будет использовать в качестве закрытого ключа для расшифрования сообщения. Значение е должно удовлетворять соотношению

еd mod f = 1.

Для малых значений f число е можно найти подбором. В общем случае для поиска е можно использовать обобщенный алгоритм Евклида, приведенный в ["Основные положения теории чисел, используемые в криптографии с открытым ключом"](https://www.intuit.ru/studies/courses/691/547/lecture/12389). В нашем случае подходит е=17. (Проверяем: 13\*17 mod 20 = 221 mod 20 = 1.)

Теперь пользователь Б должен запомнить свой закрытый ключ 17, отправить открытый ключ (13, 33) пользователю А и уничтожить числа Р = 3 и Q = 11.

Пользователь А, получивший открытый ключ (13, 33), увидев, что N=33, разбивает исходное сообщение на три блока, причем значение каждого меньше N. Например, пусть имеется три блока m1=8, m2=27, m3,=5. Затем пользователь А шифрует каждый блок:

c1=813 mod 33 = 17

c2 = 2713 mod 33 = 15

c3 = 513 mod 33 = 26

Зашифрованное сообщение, состоящее из трех блоков (17, 15, 26), передается пользователю Б, который, используя свой закрытый ключ е = 17 и N=33, расшифровывает сообщение:

m1 = 1717 mod 33 = 8

m2 = 1517 mod 33 = 27

m3 = 2617 mod 33 = 5

Таким образом, абонент Б расшифровал сообщение от абонента А.

#### Вопросы практического использования алгоритма RSA

На протяжении многих лет алгоритм RSA активно используется как в виде самостоятельных криптографических продуктов, так и в качестве встроенных средств в популярных приложениях. Открытое шифрование на базе алгоритма RSA применяется в популярном пакете шифрования PGP, операционной системе Windows, различных Интернет-браузерах, банковских компьютерных системах. Кроме того, различные международные стандарты шифрования с открытым ключом и формирования цифровой подписи используют RSA в качестве основного алгоритма.

Для обеспечения высокой надежности шифрования необходимо, чтобы выступающее в качестве модуля число N было очень большим – несколько сотен или тысяч бит. Только в этом случае будет практически невозможно по *открытым параметрам* определить закрытый ключ. Так, известно, что в конце 1995 года удалось практически реализовать раскрытие шифра RSA для 500-значного модуля. Для этого с помощью сети Интернет было задействовано более тысячи компьютеров.

Сами авторы RSA рекомендовали использовать следующие размеры модуля N: 768 бит - для частных лиц; 1024 бит - для *коммерческой информации*; 2048 бит - для особо секретной информации. С момента получения их рекомендаций прошло какое-то время, поэтому современные пользователи должны делать поправки в сторону увеличения размера ключей. Однако, чем больше размер ключей, тем медленнее работает система. Поэтому увеличивать размер ключа без необходимости не имеет смысла.

С размером ключей связан и другой аспект реализации RSA - *вычислительный*. При использовании алгоритма вычисления необходимы как при создании ключей, так и при шифровании/расшифровании, при этом, чем больше размер ключей, тем труднее производить расчеты. Для работы с громадными числами приходится использовать аппарат *длинной арифметики*. Числа, состоящие из многих сотен бит, не умещаются в регистры большинства микропроцессоров и их приходится обрабатывать по частям. При этом как шифрование, так и расшифрование включают возведение большого целого числа в целую степень по модулю N. При прямых расчетах промежуточные значения были бы невообразимыми. Чтобы упростить процесс вычислений используют специальные алгоритмы для работы с большими числами, основанные на свойствах модульной арифметики, а также оптимизацию при возведении в степень.

Алгоритм RSA реализуется как программным, так и аппаратным путем. Многие мировые фирмы выпускают специализированные микросхемы, производящие шифрование алгоритмом RSA. Программные реализации значительные медленнее, чем аппаратные. К достоинствам программного шифрования RSA относится возможность гибкой настройки параметров, возможность интеграции в различные программные пакеты. В целом, и программная, и аппаратная реализации RSA требуют для выполнения примерно в тысячи раз большего времени по сравнению с *симметричными алгоритмами*, например ГОСТ 28147-89.

Алгоритм RSA может использоваться для формирования электронной цифровой подписи, а также и для обмена ключами. Возможность применения алгоритма RSA для получения электронной подписи связана с тем, что секретный и открытый ключи в этой системе равноправны. Каждый из ключей, d или e, могут использоваться как для шифрования, так и для расшифрования. Это свойство выполняется не во всех криптосистемах с открытым ключом. Использование алгоритма RSA для формирования ЭЦП рассматривается в ["Электронная цифровая подпись"](https://www.intuit.ru/studies/courses/691/547/lecture/12393).

### Алгоритм Диффи-Хеллмана

#### Основные сведения

Первая публикация данного алгоритма появилась в 70-х годах ХХ века в статье Диффи и Хеллмана, в которой вводились основные понятия криптографии с открытым ключом. *Алгоритм Диффи-Хеллмана* не применяется для шифрования сообщений или формирования электронной подписи. Его назначение – в распределении ключей. Он позволяет двум или более пользователям обменяться без посредников ключом, который может быть использован затем для симметричного шифрования. Это была первая криптосистема, которая позволяла защищать информацию без использования секретных ключей, передаваемых по защищенным каналам. Схема открытого распределения ключей, предложенная Диффи и Хеллманом, произвела настоящую революцию в мире шифрования, так как снимала основную проблему классической криптографии – проблему распределения ключей.

Алгоритм основан на трудности вычислений *дискретных логарифмов*. Попробуем разобраться, что это такое. В этом алгоритме, как и во многих других алгоритмах с открытым ключом, вычисления производятся по модулю некоторого большого простого числа Р. Вначале специальным образом подбирается некоторое натуральное число А, меньшее Р. Если мы хотим зашифровать значение X, то вычисляем

Y = AX mod P.

Причем, имея Х, вычислить Y легко. Обратная задача вычисления X из Y является достаточно сложной. Экспонента X как раз и называется *дискретным логарифмом Y*. Таким образом, зная о сложности вычисления *дискретного логарифма*, число Y можно открыто передавать по любому каналу связи, так как при большом модуле P исходное значение Х подобрать будет практически невозможно. На этом математическом факте основан *алгоритм Диффи-Хеллмана* для формирования ключа.

#### Формирование общего ключа

Пусть два пользователя, которых условно назовем пользователь 1 и пользователь 2, желают сформировать общий ключ для алгоритма симметричного шифрования. Вначале они должны выбрать большое простое число Р и некоторое специальное число А, 1 < A < P-1, такое, что все числа из интервала [1, 2, ..., Р-1] могут быть представлены как различные степени А mod Р. Эти числа должны быть известны всем абонентам системы и могут выбираться открыто. Это будут так называемые *общие параметры*.

Затем первый пользователь выбирает число Х1 (X1<P), которое желательно формировать с помощью датчика случайных чисел. Это будет закрытый ключ первого пользователя, и он должен держаться в секрете. На основе закрытого ключа пользователь 1 вычисляет число

Описание: Y_1 = A^{X_1}\: mod \: P

которое он посылает второму абоненту.

Аналогично поступает и второй пользователь, генерируя Х2 и вычисляя

Описание: Y_2 = A^{X_2}\: mod \: P

Это значение пользователь 2 отправляет первому пользователю.

После этого у пользователей должна быть информация, указанная в следующей таблице:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Общие параметры | Открытый ключ | Закрытый ключ |
| Пользователь 1 | Р, А | Y1 | Х1 |
| Пользователь 2 | Y2 | Х2 |

Из чисел Y1 и Y2, а также своих закрытых ключей каждый из абонентов может сформировать общий секретный ключ Z для сеанса симметричного шифрования. Вот как это должен сделать первый пользователь:

Описание: Z = (Y_2)^{X_1}\: mod \: P

Никто другой кроме пользователя 1 этого сделать не может, так как число Х1 секретно. Второй пользователь может получить то же самое число Z, используя свой закрытый ключ и открытый ключ своего абонента следующим образом:

Описание: Z = (Y_1)^{X_2}\: mod \: P

Если весь протокол формирования общего секретного ключа выполнен верно, значения Z у одного и второго абонента должны получиться одинаковыми. Причем, что самое важное, противник, не зная секретных чисел Х1 и Х2, не сможет вычислить число Z. Не зная Х1 и Х2, злоумышленник может попытаться вычислить Z, используя только передаваемые открыто Р, А, Y1 и Y2. Безопасность формирования общего ключа в алгоритме Диффи-Хеллмана вытекает из того факта, что, хотя относительно легко вычислить экспоненты по модулю простого числа, очень трудно вычислить дискретные логарифмы. Для больших простых чисел размером сотни и тысячи бит задача считается неразрешимой, так как требует колоссальных затрат вычислительных ресурсов.

Пользователи 1 и 2 могут использовать значение Z в качестве секретного ключа для шифрования и расшифрования данных. Таким же образом любая пара абонентов может вычислить секретный ключ, известный только им.

#### Пример вычислений по алгоритму

Пусть два абонента, желающие обмениваться через Интернет зашифрованными сообщениями, решили сформировать секретный ключ для очередного сеанса связи. Пусть они имеют следующие общие параметры:

Р = 11, А = 7.

Каждый абонент выбирает секретное число Х и вычисляет соответствующее ему открытое число Y. Пусть выбраны

Х1 = 3, Х2= 9.

Вычисляем

Y1 = 73 mod 11 = 2,

Y2= 79 mod 11 = 8.

Затем пользователи обмениваются открытыми ключами Y1 и Y2. После этого каждый из пользователей может вычислить общий секретный ключ:

пользователь 1: Z = 83 mod 11 = 6.

пользователь 2: Z = 29 mod 11 = 6.

Теперь они имеют общий ключ 6, который не передавался по каналу связи.

#### Вопросы практического использования алгоритма Диффи-Хеллмана

Для того, чтобы *алгоритм Диффи-Хеллмана* работал правильно, то есть оба пользователя, участвующих в протоколе, получали одно и то же число Z, необходимо правильным образом выбрать число А, используемое в вычислениях. Число А должно обладать следующим свойством: все числа вида

A mod P, A2 mod P, A3 mod P,... , AP-1 mod P

должны быть различными и состоять из целых положительных значений в диапазоне от 1 до Р-1 с некоторыми перестановками. Только в этом случае для любого целого Y < Р и значения A можно найти единственную экспоненту Х, такую, что

Y = AХmod P, где 0 <= X <= (P - 1)

При произвольно заданном Р задача выбора параметра А может оказаться трудной задачей, связанной с разложением на простые множители числа Р-1. На практике можно использовать следующий подход, рекомендуемый специалистами. Простое число Р выбирается таким, чтобы выполнялось равенство Р = 2q + l, где q — также простое число. Тогда в качестве А можно взять любое число, для которого справедливы неравенства

1<A<P-1 и Aq mod P ≠ 1

На подбор подходящих параметров А и Р необходимо некоторое время, однако это обычно не критично для системы связи и не замедляет ее работу. Эти параметры являются общими для целой группы пользователей. Они обычно выбираются один раз при создании сообщества пользователей, желающих использовать *протокол Диффи-Хеллмана*, и не меняются в процессе работы. А вот значения закрытых ключей рекомендуется каждый раз менять и выбирать их с помощью генераторов псевдослучайных чисел.

Следует заметить, что данный алгоритм, как и все *алгоритмы асимметричного шифрования*, уязвим для атак типа "man-in-the-middle" ("человек в середине"). Если противник имеет возможность не только перехватывать сообщения, но и заменять их другими, он может перехватить открытые ключи участников, создать свою пару открытого и закрытого ключа и послать каждому из участников свой открытый ключ. После этого каждый участник вычислит ключ, который будет общим с противником, а не с другим участником. Способы предотвращения такой атаки и некоторых других рассмотрены в конце этой лекции.

### Алгоритм Эль-Гамаля

#### Основные сведения

*Асимметричный алгоритм*, предложенный в 1985 году Эль-Гамалем (T. ElGamal), универсален. Он может быть использован для решения всех трех основных задач: для шифрования данных, для формирования цифровой подписи и для согласования общего ключа. Кроме того, возможны модификации алгоритма для схем проверки пароля, доказательства идентичности сообщения и другие варианты. Безопасность этого алгоритма, так же как и *алгоритма Диффи-Хеллмана*, основана на трудности вычисления *дискретных логарифмов*. Этот алгоритм фактически использует схему Диффи-Хеллмана, чтобы сформировать общий секретный ключ для абонентов, передающих друг другу сообщение, и затем сообщение шифруется путем умножения его на этот ключ.

И в случае шифрования, и в случае формирования цифровой подписи каждому пользователю необходимо сгенерировать пару ключей. Для этого, так же как и в схеме Диффи-Хеллмана, выбираются некоторое большое простое число Р и число А, такие, что различные степени А представляют собой различные числа по модулю Р. Числа Р и А могут передаваться в открытом виде и быть общими для всех абонентов сети.

Затем каждый абонент группы выбирает свое секретное число Хi, 1 < Хi < Р-1, и вычисляет соответствующее ему открытое число Описание: Y_i : Y_i = A^{X_i}\: mod \: P. Таким образом, каждый пользователь может сгенерировать закрытый ключ Хi и открытый ключ Yi.

Информация о необходимых параметрах системы сведена в следующую таблицу.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Общие параметры | Открытый ключ | Закрытый ключ |
| Пользователь 1 | Р, А | Y1 | Х1 |
| … | … | … |
| Пользователь i | Yi | Хi |

#### Шифрование

Теперь рассмотрим, каким образом производится шифрование данных. Сообщение, предназначенное для шифрования, должно быть представлено в виде одного числа или набора чисел, каждое из которых меньше Р. Пусть пользователь 1 хочет передать пользователю 2 сообщение m. В этом случае последовательность действий следующая.

1. Первый пользователь выбирает случайное число k, взаимно простое с Р-1, и вычисляет числа

Описание: r=A^k\: mod \: P, \qquad e=m \times Y_2^k \: mod \:P

где Y2 – открытый ключ пользователя 2. Число k держится в секрете.

1. Пара чисел (r, е), являющаяся шифротекстом, передается второму пользователю.
2. Второй пользователь, получив (r,e), для расшифрования сообщения вычисляет

Описание: m=e \times r^{P-1-X_2} \: mod \:P

где Х2 – закрытый ключ пользователя 2. В результате он получает исходное сообщение m.

Если злоумышленник узнает или перехватит Р, А, Y2, r, e, то он не сможет по ним раскрыть m. Это связано с тем, что противник не знает параметр k, выбранный первым пользователем для шифрования сообщения m. Вычислить каким-либо образом число k практически невозможно, так как это задача дискретного логарифмирования. Следовательно, злоумышленник не может вычислить и значение m, так как m было умножено на неизвестное ему число. Противник также не может воспроизвести действия законного получателя сообщения (второго абонента), так как ему не известен закрытый ключ Х2 (вычисление Х2 на основании Y2 — также задача дискретного логарифмирования).

По аналогичному алгоритму может производиться и согласование ключа, используемого для симметричного шифрования больших объемов данных. Более того, алгоритм Эль-Гамаля на практике целесообразно использовать именно для согласования общего *ключа сессии*, а не прямого шифрования больших сообщений. Это связано с тем, что в алгоритме используются операции возведения в степень и умножения по большому модулю. Так же как и в алгоритмах RSA и Диффи-Хеллмана, операции производятся над большими, состоящими из нескольких сотен или тысяч бит, числами. Поэтому шифрование больших сообщений производится крайне медленно.

#### Пример шифрования

Пусть два абонента, обменивающиеся через Интернет зашифрованными сообщениями, имеют следующие общие параметры:

Р = 11, А = 7.

Кроме того, пользователи 1 и 2 имеют пары закрытых и открытых ключей, вычисляемые также, как в п. 5.3.3:

Пользователь 1: закрытый ключ Х1= 3, открытый ключ Y1 = 73 mod 11 = 2,

Пользователь 2: закрытый ключ Х2 = 9, открытый ключ Y2 = 79mod 11 = 8.

Первый абонент желает передать второму сообщение. Для этого первый абонент запрашивает из *центра распределения ключей* открытый ключ второго абонента Y2 = 8. Теперь он может зашифровать свое сообщение, которое в числовом виде пусть имеет значение m=9.

Первый абонент выбирает случайно число k, например k = 7. Число k должно быть взаимно простым с Р-1. Значение k = 7 не имеет общих делителей с Р-1=10, значит, оно нам подходит. Первый абонент шифрует свое сообщение по формулам:

r = Ak mod P = 77 mod 11=6

e = m \* Y2k mod P = 9 \* 87 mod 11 = 7

Пара чисел (6, 7) будет представлять собой шифротекст и передается второму пользователю. Второй пользователь, получив (6,7) и используя свой закрытый ключ Х2 = 9 для расшифрования сообщения, вычисляет

Описание: m=e \times r^{P-1-X_2} \: mod \: P = 7 \times 6^{11-1-9} \: mod \: 11 = 7 \times 6^1 \: mod \: 11 = 9

В результате он действительно получает исходное сообщение m.

### Криптографические системы на эллиптических кривых

В 1985 году американские ученые Н. Коблиц (Neal Koblitz) и В. Миллер (Victor Miller) предложили использовать для криптосистем с открытым ключом теорию эллиптических кривых. Дальнейшие исследования подтвердили наличие подходящих свойств у этих математических функций и привели к созданию реальных криптографических систем, использующих математический аппарат эллиптических кривых. С 1998 года использование эллиптических кривых для решения криптографических задач, таких, как *цифровая подпись*, было закреплено в стандартах США *ANSI* X9.62 и *FIPS* 186-2, а в 2001 году аналогичный стандарт, ГОСТ Р34.10-2001, был принят и в России.

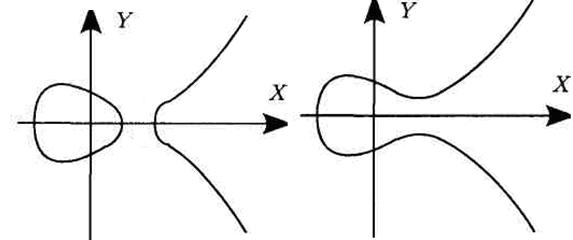
Основное достоинство **криптосистем на эллиптических кривых** состоит в том, что по сравнению с другими асимметричными криптосистемами, рассмотренными нами ранее, они обеспечивают существенно более высокую криптостойкость при равных затратах на обработку и вычисления. Это объясняется тем, что *вычисление* обратных функций на эллиптических кривых значительно сложнее, чем, например, *вычисление* *дискретных логарифмов* (алгоритмы Диффи-Хеллмана и Эль-Гамаля) или решение задачи факторизации (*алгоритм* *RSA*). В результате тот уровень стойкости, который достигается, скажем, в *RSA* при использовании 1024-битовых модулей, в системах на эллиптических кривых реализуется при размере модуля 160 *бит*, что обеспечивает более простую как программную, так и аппаратную реализацию.

*Криптография* эллиптических кривых использует достаточно сложный аппарат высшей алгебры, поэтому мы, в рамках данного учебного пособия, не сможем подробно рассмотреть используемые на практике алгоритмы и выполнить соответствующие примеры вычислений. Сформулируем основные принципы построения криптографических систем с использованием эллиптических кривых.

В криптографии используются *эллиптические кривые* на плоскости, определяемые уравнениями вида

Y2= X3+ аХ + b mod р,

где р – некоторое большое *простое число*, а a и b – *константы*. *График* эллиптической кривой при разных значениях параметров а и b имеет вид, как на [рис. 11.1](https://www.intuit.ru/studies/courses/691/547/lecture/12391?page=4" \l "image.11.1).



**Рис. 11.1.**Варианты графиков эллиптических кривых

Принцип использования эллиптических кривых следующий. Для группы пользователей выбирается общая *эллиптическая кривая* Е и некоторая точка G на ней. Закрытым ключом пользователя выступает некоторое *целое число* с, а открытым – точка D на кривой Е, полученная в результате специального преобразования *композиции* с использованием числа с. Параметры кривой и *список* открытых ключей абонентов, как и обычно, передаются всем пользователям сети. Открытые и закрытые ключи пользователей используются для выполнения операций шифрования и расшифрования в зависимости от назначения алгоритма.

С помощью эллиптических кривых могут быть реализованы многие известные протоколы с открытым ключом. Любая *криптосистема*, основанная на дискретном логарифмировании, легко может быть перенесена на *эллиптические кривые*. Например, можно заменить математические *операции* вида у = gхmod р на *операции* математического аппарата эллиптических кривых (*операции* вычисления композиции точек) в алгоритмах формирования ключа Диффи-Хеллмана или вычисления цифровой подписи Эль-Гамаля. В результате получатся те же алгоритмы, но с другими математическими операциями.

Несмотря на сложность математического аппарата эллиптических кривых, существуют эффективные вычислительные методы, позволяющие достаточно быстро реализовывать необходимые расчеты. За счет использования модуля меньшей длины *операции* генерации ключей и шифрования выполняются быстрее, чем, скажем, в алгоритме *RSA* или классическом алгоритме Диффи-Хеллмана. Криптографические методы на эллиптических кривых считаются перспективными и, закрепленные в различных стандартах, находят применение в современных системах защиты информации.

### Возможные атаки при использовании алгоритмов асимметричного шифрования

#### Атака "человек-в-середине"

Попробуем проанализировать простейший протокол шифрования с открытым ключом, рассмотренный в ["Введение в криптографию с открытым ключом"](https://www.intuit.ru/studies/courses/691/547/lecture/12387), с точки зрения возможности проведения злоумышленником различных атак. Вспомним, что этот протокол предусматривал следующие действия пользователей. Если пользователь А желает передать секретное сообщение пользователю Б так, чтобы никто другой не смог его прочитать, он должен получить от пользователя Б открытый ключ UБ и зашифровать свое сообщение этим открытым ключом. Зашифрованное сообщение может пересылаться по любому каналу связи, например, по электронной почте. Получив сообщение от пользователя А, пользователь Б может расшифровать его своим закрытым ключом RБ. Такая процедура обмена зашифрованными сообщениями с использованием *асимметричного алгоритма* не позволит противнику, контролирующему открытый канал связи, по перехваченным открытым ключам и зашифрованным сообщениям восстановить исходные сообщения. Это обеспечивается свойствами *односторонней функции*, а именно, сложностью вычисления обратной функции.

Однако такая схема уязвима для атак типа "man-in-the-middle" ( **"человек-в-середине"** ). Эта атака заключается в следующем. Допустим, злоумышленник может не только перехватывать сообщения, но и заменять их другими, т.е. имеет возможность осуществлять *активную атаку*. Это вполне возможно в современных сетях передачи данных, например в Интернете, где информация от одного пользователя передается другому через множество промежуточных узлов, не контролируемых этими пользователями. Злоумышленником может быть, например, системный администратор сети. Такой нарушитель может не только перехватывать сообщения пользователей, но изменять, удалять или заменять их своими. Он может выдавать себя за одного из участников сеанса связи. Вот как может производиться атака "man-in-the-middle":

1. Пользователь Б посылает пользователю А свой открытый ключ UБ. Противник перехватывает этот ключ, сохраняет его и заменяет его своим открытым ключом UП.
2. Пользователь А шифрует свое сообщение М полученным открытым ключом UП, предполагая, что использует открытый ключ абонента Б, и пересылает зашифрованное сообщение пользователю Б.
3. Злоумышленник перехватывает это сообщение, расшифровывает его своим закрытым ключом RП, читает или меняет, а затем зашифровывает открытым ключом пользователя Б и посылает пользователю Б.

Аналогично взломщик перехватит и открытый ключ пользователя А, чтобы читать ответы пользователя Б. В результате нарушитель сможет читать (а, возможно, и изменять) всю корреспонденцию абонентов. Пользователи А и Б, скорее всего, ничего не заподозрят, так как у них нет способа проверить, действительно ли они общаются друг с другом.

На практике разработано несколько способов предотвращения атаки "man-in-the-middle". Один из способов заключается в разделении каждого зашифрованного сообщения на две части, каждая из которых бесполезна без другой. Части сообщения пересылаются по очереди и не могут быть расшифрованы по отдельности. Вот как может выглядеть этот протокол для обмена сообщениями между двумя пользователями А и Б:

1. Пользователи А и Б обмениваются открытыми ключами.
2. Пользователь А шифрует свое сообщение открытым ключом пользователя Б и пересылает половину зашифрованного сообщения пользователю Б.
3. Пользователь Б шифрует свое сообщение открытым ключом пользователя А и пересылает половину зашифрованного сообщения пользователю А.
4. Пользователь А пересылает вторую половину зашифрованного сообщения пользователю Б.
5. Пользователь Б соединяет обе полученные половины сообщения от пользователя А и расшифровывает его своим закрытым ключом. Затем посылает вторую половину своего зашифрованного сообщения пользователю А.
6. Пользователь А складывает полученные от пользователя Б половины сообщения и расшифровывает его своим закрытым ключом.

Этот усовершенствованный протокол не позволит злоумышленнику читать или изменять корреспонденцию пользователей А и Б. Нарушитель, как и раньше, может подменить открытые ключи абонентов, а также перехватить передаваемые между ними данные. Однако, получив на шаге 2 протокола в свое распоряжение первую половину зашифрованного сообщения от А к Б, он не сможет расшифровать ее своим закрытым ключом и снова зашифровать открытым ключом абонента Б. Абоненты А и Б тоже не смогут прочитать сообщения до окончания протокола (шагов 5 и 6), но в этом нет ничего плохого, так как в результате они получат корректную корреспонденцию. Для осуществления протокола процесс разделения сообщения на две части может производиться разными способами, например, каждый нечетный байт помещается в первое сообщение, а каждый четный – во второе или как-то иначе.

Атаки "человек-в-середине" можно избежать и другими способами, например, добавляя к передаваемым открытым ключам цифровые подписи специального удостоверяющего центра.

#### Атака на основе выбранного открытого текста

Алгоритмы с открытым ключом чувствительны к атакам по выбранному открытому тексту. Как известно, такая атака имеет место, если криптоаналитик имеет возможность не только использовать предоставленные ему пары "текст-шифротекст", но и сам формировать нужные ему тексты и шифровать их.

Факт возможности проведения атаки по выбранному открытому тексту объясняется следующим образом. Предположим, мы используем *асимметричный алгоритм* F для согласования общего секретного ключа. Пусть один из абонентов отправил другому 64-битовый сеансовый ключ K, зашифрованный открытым ключом y другого абонента C=F(K, y). Злоумышленник, перехватив зашифрованное сообщение С, не сможет его, конечно, дешифровать, так как не имеет закрытого ключа x. Однако нарушитель может поступить по-другому, а именно, попытаться подобрать подходящее значение К. Для этого нужно зашифровать все возможные 64-битовые комбинации открытых текстов открытым ключом y и сравнить результаты с С. Это возможно, так значения y и C передавались в открытом виде. Особенно актуальна угроза такой атаки, если число возможных исходных сообщений не очень велико, например, если длина исходного сообщения мала или если не все исходные тексты допустимы на практике.

Для того, чтобы избежать возможности такой атаки, используют рандомизированные (или вероятностные) алгоритмы шифрования и формирования ЭЦП с открытым ключом. Такие алгоритмы шифруют одно и то же сообщение при наличии одинакового ключа каждый раз по-разному, так как используют некоторый случайный элемент. Примерами рандомизированных алгоритмов с открытым ключом могут служить алгоритмы Эль-Гамаля и алгоритмы формирования ЭЦП по ГОСТ Р34.10.

Другим вариантом предотвращения атаки на основе выбранного открытого текста является добавление в шифруемое сообщение некоторой дополнительной "случайной" информации, например, метки даты времени.

### Ключевые термины

**Алгоритм RSA** – *алгоритм* шифрования с открытым ключом. Название алгоритма составлено из первых букв фамилий авторов: Р.Райвеста (R.Rivest), А.Шамира (A.Shamir) и Л.Адлемана (L.Adleman). *Алгоритм* *RSA* основан на сложности задачи факторизации больших чисел. Данный *алгоритм* является, возможно, наиболее популярным и широко применяемым *асимметричным алгоритмом* в криптографических системах.

**Алгоритм Диффи-Хеллмана** – *алгоритм* шифрования с открытым ключом. Этот *алгоритм* основан на трудности вычислений *дискретных логарифмов*. *Алгоритм Диффи-Хеллмана* может использоваться для распределения ключей, которые могут быть использованы для симметричного шифрования.

**Алгоритм Эль-Гамаля** – *алгоритм* шифрования с открытым ключом, основанный на трудности вычислений *дискретных логарифмов*. *Алгоритм* Эль-Гамаля может быть использован для шифрования данных, для формирования цифровой подписи и для согласования общего ключа. Этот *алгоритм* фактически использует схему Диффи-Хеллмана, чтобы сформировать общий секретный *ключ* для абонентов, передающих друг другу сообщение, и затем сообщение шифруется путем умножения его на этот *ключ*.

**Атака "человек-в-середине"** (англ. "man-in-the-middle") – термин в криптографии, обозначающий ситуацию, когда атакующий способен читать и видоизменять по своей воле сообщения, которыми обмениваются корреспонденты, причём ни один из последних не может догадаться о его присутствии в канале связи.

**Криптосистемы на эллиптических кривых** – *группа* алгоритмов с открытым ключом, использующих в качестве математического аппарата свойства эллиптических кривых на плоскости.

### Краткие итоги

*Алгоритм* *RSA* – *алгоритм* шифрования с открытым ключом. *Алгоритм* *RSA* основан на сложности задачи факторизации больших чисел. Математические основы алгоритма *RSA* следующие. Выбираются два больших простых числа Р и Q и вычисляется *произведение* N = PQ. После этого определяется вспомогательное число f = (Р - l)(Q - 1). Затем случайным образом выбирается число d < f и взаимно простое с f. Далее необходимо найти число е, такое, что еd mod f = 1. Числа d и N будут открытым ключом пользователя, а *значение* е – закрытым ключом. Шифруемое сообщение должно быть представлено в цифровом виде и разбито на блоки m1, m2, m3, ... , где mi < N. Зашифрованное сообщение будет состоять из блоков ci = mid mod N. Расшифровывание производится по формуле mi = ce mod N. *Алгоритм* *RSA* может использоваться для шифрования данных небольшого размера, формирования электронной цифровой подписи, а также и в протоколах обмена ключами для симметричных систем шифрования.

*Алгоритм Диффи-Хеллмана* – *алгоритм* шифрования с открытым ключом. Этот *алгоритм* основан на трудности вычислений *дискретных логарифмов*. *Алгоритм Диффи-Хеллмана* может использоваться для распределения ключей. Принцип работы алгоритма следующий. Вначале выбираются большое *простое число* Р и число А, 1 < A < P-1, такое, что все числа из интервала [1, 2, ..., Р-l] могут быть представлены как различные степени А mod Р. Это – общие параметры. Затем два пользователя выбирают себе закрытые ключи Х1 и Х2 (Xi<P). На основе закрытых ключей пользователи вычисляют открытые ключи Описание: Y_i=A^{X_i} \: mod \: P, которыми они обмениваются. Из чисел Y1 и Y2, а также своих закрытых ключей каждый из абонентов может сформировать общий секретный *ключ* Z для сеанса симметричного шифрования: Описание: Z=(Y_2)^{X_1} \: mod \: P==(Y_1)^{X_2} \: mod \: P .

*Алгоритм* Эль-Гамаля может быть использован для шифрования данных, для формирования цифровой подписи и для согласования общего ключа. Этот *алгоритм* фактически использует схему Диффи-Хеллмана, чтобы сформировать общий секретный *ключ* для абонентов, передающих друг другу сообщение, и затем сообщение шифруется путем умножения его на этот *ключ*. Общими открытыми параметрами в криптосистеме системе Эль-Гамаля являются числа Р (большое *простое число*) и А (А< P). Закрытыми ключами абонентов являются числа Хi, 1 < Х i < Р-1, открытыми ключами – значения Описание: Y_i:Y_i=A^{X_i} \: mod \: P. *Пользователь*, передающий сообщение, выбирает случайное число k, взаимно простое с Р-1, и вычисляет числа Описание: r=A^k \: mod \: P, e=m\times Y_2^k \: mod \: P. Пара чисел (r, е), являющаяся шифротекстом, передается другому пользователю. Для расшифрования сообщения необходимо вычислить Описание: m=e \times r^{P-1-X_2}\: mod \: P. В результате получается исходное сообщение m.

Криптосистемы на эллиптических кривых – самая молодая *группа* алгоритмов с открытым ключом, использующих в качестве математического аппарата свойства эллиптических кривых на плоскости. Основное отличие таких систем состоит в том, что по сравнению с асимметричными криптосистемами, предложенными ранее, они обеспечивают существенно более высокую криптостойкость при равных затратах на обработку и вычисления. Это объясняется тем, что *вычисление* обратных функций на эллиптических кривых значительно сложнее, чем, например, *вычисление* *дискретных логарифмов* или решение задачи факторизации. В результате тот уровень стойкости, который достигается, скажем, в *RSA* при использовании 1024-битовых модулей, в системах на эллиптических кривых реализуется при размере модуля 160 *бит*, что обеспечивает более простую как программную, так и аппаратную реализацию.

Несмотря на достаточную *надежность* алгоритмов шифрования с открытым ключом, существует возможность проведения атак в системах, использующих асимметричное *шифрование*. Это связано с тем, что *атака* может быть направлена не на сам *алгоритм* шифрования, а на протокол, использующий этот *алгоритм*. Для исключения возможностей проведения различных атак в системах шифрования с открытым ключом применяют специальные меры, например, заверяют открытые ключи пользователей цифровыми подписями удостоверяющего центра или добавляют к шифруемым сообщениям некоторую случайную информацию.

#### Вопросы для самопроверки

1. Для каких целей может применяться алгоритм RSA?
2. Опишите процесс шифрования с использованием алгоритма RSA.
3. Для каких целей может применяться *алгоритм Диффи-Хеллмана*?
4. Опишите последовательность действий при использовании *алгоритма Диффи-Хеллмана*.
5. Для каких целей может применяться алгоритм Эль-Гамаля?.
6. Опишите последовательность действий при использовании алгоритма Эль-Гамаля.
7. Какие атаки возможны при использовании алгоритмов шифрования с открытым ключом?

**Лекция №6: Электронная цифровая подпись**

### ****Цель:****

Изучить назначение, принципы работы и алгоритмы электронной цифровой подписи, а также ознакомиться с её юридической значимостью и применением в ИКТ.

### ****Задание:****

#### ****1. Теоретическая часть:****

Изучить и законспектировать:

* Понятие и назначение ЭЦП
* Отличие от обычной подписи
* Принцип работы ЭЦП:
  + генерация ключей
  + создание подписи
  + проверка подписи
* Алгоритмы, применяемые для ЭЦП:
  + RSA
  + DSA
  + ГОСТ Р 34.10-2012
* Роль хеш-функций в ЭЦП
* Юридическая значимость ЭЦП в Республике Казахстан (на основе законодательства)

#### ****2. Практическая часть:****

* Составить блок-схему процесса формирования и проверки ЭЦП
* Рассмотреть пример применения ЭЦП в eGov (портал электронного правительства)
* Подготовить обзор одного программного средства для работы с ЭЦП (например, NCALayer, CryptoPro и т.д.)
* (Дополнительно) Выполнить создание и проверку ЭЦП в выбранной системе (если доступна)

#### ****3. Подготовить:****

* Презентацию (8–12 слайдов) с краткими тезисами и схемами
* Краткий отчёт (1–2 страницы) по изученным вопросам и практике

**Цель лекции**: познакомиться с некоторыми алгоритмами формирования и проверки электронной цифровой подписи (*ЭЦП*, ЭП).

### Электронная подпись на основе алгоритма RSA

Рассмотренная нами в Лекции 11 схема использования алгоритма *RSA* при большом модуле N практически не позволяет злоумышленнику получить закрытый *ключ* и прочитать зашифрованное сообщение. Однако она дает возможность злоумышленнику подменить сообщение от абонента А к абоненту Б, так как *абонент* А шифрует свое сообщение открытым ключом, полученным от Б по открытому каналу связи. А раз *открытый ключ* передается по открытому каналу, любой может получить его и использовать для подмены сообщения. Избежать этого можно, используя более сложные протоколы, например, следующий.

Пусть, как и раньше, *пользователь* А хочет передать пользователю Б сообщение, состоящее из нескольких блоков mi. Перед началом сеанса связи абоненты генерируют открытые и закрытые ключи, обозначаемые, как указано в следующей таблице:

|  |  |  |
| --- | --- | --- |
|  | **Открытый ключ** | **Закрытый ключ** |
| Пользователь А | NA, dA | eA |
| Пользователь Б | NБ, dБ | eБ |

В результате каждый *пользователь* имеет свои собственные открытый (состоящий из двух частей) и закрытый ключи. Затем пользователи обмениваются открытыми ключами. Это подготовительный этап протокола.

Основная часть протокола состоит из следующих шагов.

1. Сначала пользователь А вычисляет числа Описание: c_i=m_i^{e_A} \: mod \:N_A, то есть шифрует сообщение своим закрытым ключом. В результате этих действий пользователь А подписывает сообщение.
2. Затем пользователь А вычисляет числа Описание: g_i=c_i^{d_Б} \: mod \: N_Б, то есть шифрует то, что получилось на шаге 1 открытым ключом пользователя Б. На этом этапе сообщение шифруется, чтобы никто посторонний не мог его прочитать.
3. Последовательность чисел gi передается к пользователю Б.
4. Пользователь Б получает gi и вначале вычисляет последовательно числа Описание: c_i=g_i^{e_Б} \: mod \: N_Б, используя свой закрытый ключ. При этом сообщение расшифровывается.
5. Затем Б определяет числа Описание: m_i=c_i^{d_A} \: mod \: N_A
   , используя открытый ключ пользователя А. За счет выполнения этого этапа производится проверка подписи пользователя А.

В результате *абонент* Б получает исходное сообщение и убеждается в том, что его отправил именно *абонент* А. Данная схема позволяет защититься от нескольких видов возможных нарушений, а именно:

* пользователь А не может отказаться от своего сообщения, если он признает, что секретный ключ известен только ему;
* нарушитель без знания секретного ключа не может ни сформировать, ни сделать осмысленное изменение сообщения, передаваемого по линии связи.

Данная схема позволяет избежать многих конфликтных ситуаций. Иногда нет необходимости зашифровывать передаваемое сообщение, но нужно его скрепить электронной подписью. В этом случае из приведенного выше протокола исключаются шаги 2 и 4, то есть текст шифруется закрытым ключом отправителя, и полученная последовательность присоединяется к документу. Получатель с помощью открытого ключа отправителя расшифровывает прикрепленную подпись, которая, по сути, является зашифрованным повторением основного сообщения. Если расшифрованная подпись совпадает с основным текстом, значит, подпись верна.

Существуют и другие варианты применения алгоритма *RSA* для формирования *ЭЦП*. Например, можно шифровать (то есть подписывать) открытым ключом не само сообщение, а хеш-код от него.

Возможность применения алгоритма *RSA* для получения электронной подписи связана с тем, что секретный и открытый ключи в этой системе равноправны. Каждый из ключей, d или e, могут использоваться как для шифрования, так и для расшифрования. Это свойство выполняется не во всех криптосистемах с открытым ключом.

*Алгоритм* *RSA* можно использовать также и для обмена ключами.

### Цифровая подпись на основе алгоритма Эль-Гамаля

#### Принцип создания и проверки подписи

Алгоритм Эль-Гамаля также можно использовать для формирования цифровой подписи. Группа пользователей выбирает общие параметры Р и А. Затем каждый абонент группы выбирает свое секретное число Хi, 1 < Хi< Р-1, и вычисляет соответствующее ему открытое число Описание: Y_i \: : \: Y_i=A^{X_i} \: mod \: P. Таким образом, каждый пользователь получает пару (закрытый ключ; открытый ключ) = (Хi, Yi). Открытые ключи пользователей могут храниться в общей базе системы распределения ключей и при необходимости предоставляться всем абонентам системы.

Сообщение, предназначенное для подписи, должно быть представлено в виде числа, меньшего модуля Р. При большом размере сообщение разбивается на блоки необходимого размера. В некоторых случаях подписывается не само сообщение, а значение хеш-функции от него. В любом варианте цифровая подпись вычисляется в зависимости от некоторого числа m (m < P).

Пусть пользователь 1 хочет подписать свое сообщение цифровой подписью и передать его пользователю 2. В этом случае алгоритм действий следующий.

1. Первый пользователь выбирает случайное секретное число k, взаимно простое с Р-1, и вычисляет число Описание: a=A^k \: mod \: P
2. Затем с помощью расширенного алгоритма Евклида необходимо найти значение b в следующем уравнении:

m = (X1 \* a +k \* b) mod (P-1)

Пара чисел (a, b) будет цифровой подписью сообщения m.

1. Сообщение m вместе с подписью (a, b) отправляется пользователю 2.
2. Пользователь 2 получает сообщение m и с использованием открытого ключа первого абонента Y1 вычисляет два числа по следующим формулам: Описание: c_1=Y_1^a \times a^b \: mod \: P \\ c_2=A^m \: mod \: P  Если с1 = с2, то цифровая подпись первого пользователя верная. Для подписывания каждого нового сообщения должно каждый раз выбираться новое значение k.

Подписи, созданные с использованием алгоритма Эль-Гамаля, называются *рандомизированными*, так как для одного и того же сообщения с использованием одного и того же закрытого ключа каждый раз будут создаваться разные подписи (a,b), поскольку каждый раз будет использоваться новое значение k. Подписи, созданные с применением алгоритма RSA, называются *детерминированными*, так как для одного и того же сообщения с использованием одного и того же закрытого ключа каждый раз будет создаваться одна и та же подпись.

#### Пример вычисления и проверки цифровой подписи

Пусть абоненты, обменивающиеся через Интернет зашифрованными сообщениями, имеют следующие общие параметры: Р = 11, А = 7.

Один из пользователей этой системы связи хочет подписать свое сообщение m=5 цифровой подписью, сформированной по алгоритму Эль-Гамаля. Вначале он должен выбрать себе закрытый ключ, например, Х1=3 и сформировать открытый ключ Y1 = 73 mod 11 = 2. Открытый ключ может быть передан всем заинтересованным абонентам или помещен в базу данных открытых ключей системы связи.

Затем пользователь выбирает случайное секретное число k, взаимно простое с Р-1. Пусть k=9 ( 9 не имеет общих делителей с 10 ). Далее необходимо вычислить число

Описание: a=A^k \: mod \: P=7^9\: mod \: 11=8

После этого с помощью расширенного алгоритма Евклида находится значение b в уравнении:

Описание: m=(X_1 \times a + k \times b) \: mod \: (P-1),\\5=(3 \times 8 + 9 \times b) \: mod \: 10

Решением последнего уравнения будет значение b=9.

Таким образом, пара чисел (8, 9) будет цифровой подписью сообщения m=5.

Если любой другой пользователь сети желает проверить цифровую подпись в сообщении, он должен получить из базы данных открытый ключ первого пользователя (он равен 2 ), вычислить два числа с1 и с2 и сравнить их.

Описание: c_1-Y_1^a \times a^b \: mod \: P=2^8 \times 8^9 \: mod \: 11=10,\\c_2=A^m \: mod \: 11=10

Так как с1 = с2, то цифровая подпись первого пользователя в сообщения m=5 верная.

### Стандарты на алгоритмы цифровой подписи

#### Стандарт цифровой подписи DSS

Во многих странах сегодня существуют стандарты на *электронную (цифровую) подпись*. *Стандарт цифровой подписи* DSS (*Digital Signature Standard* – DSS) был принят в США в 1991 году и пересмотрен в 1994 году. В основе стандарта лежит алгоритм, называемый *DSA* (*Digital Signature Algorithm*) и являющийся вариацией подписи Эль-Гамаля. В алгоритме используется однонаправленная хеш-функция H(m). В качестве хэш-алгоритма стандарт DSS предусматривает использование алгоритма SHA-1.

Рассмотрим сам алгоритм генерации ЭЦП. Вначале для группы абонентов выбираются три общих (несекретных) параметра р, q и a:

* параметр р должен быть простым числом длиной от 512 до 1024 бит.
* q – простое число длиной 160 бит; между p и q должно выполняться соотношение p = bq + 1 для некоторого целого b. Старшие биты в р и q должны быть равны единице (таким образом 2159 < q < 2160 ).
* число а, удовлетворяющее неравенству 1 < a < p-1 и являющееся корнем уравнения aq mod p = 1.

Зная эти числа, каждый абонент системы случайно выбирает число х, удовлетворяющее неравенству 0 < х < q, и вычисляет

y = ax mod p.

Число х будет секретным ключом пользователя, а число у — открытым ключом. Вычислить у по известному х довольно просто. Однако, имея открытый ключ у, вычислительно невозможно определить х, который является *дискретным логарифмом* у по основанию a.

Предполагается, что открытые ключи всех пользователей указываются в некотором несекретном, но "сертифицированном" справочнике, который должен быть у всех, кто собирается проверять подписи. На этом этап выбора параметров заканчивается, и абоненты готовы к тому, чтобы формировать и проверять подписи.

Пусть имеется сообщение m, которое один из пользователей желает подписать. Для генерации подписи пользователь должен выполнить следующие действия:

1. Вычислить значение хеш-функции h = H(m) для сообщения m. Значение хеш-функции должно лежать в пределах 0 < h < q.
2. Затем сгенерировать случайное число k, 0 < k < q.
3. Вычислить r = (ak mod p) mod q.
4. Определить s = [k-1}(H(m) + x \* r)] mod q

В результате пользователь получит для сообщения m подпись, состоящую из пары чисел (r,s). Сообщение вместе с подписью может быть послано любому другому абоненту системы. Проверить подпись можно следующим образом:

1. Вычислить значение хеш-функции h = H(m) для сообщения m.
2. Проверить выполнение неравенств 0 < r < q, 0 < s < q.
3. Вычислить w = s-1 mod q ;
4. u1 = [H(m) \* w] mod q
5. u2 = r \* w mod q

v = [(аu1 \* yu2) mod p] mod q

1. Проверить выполнение равенства v = r. Если v = r, то подпись считается подлинной, иначе подпись считается недействительной.

В силу сложности вычисления дискретных логарифмов злоумышленник не может восстановить k из r или х из s, а следовательно, не может подделать подпись. По той же самой причине автор сообщения не сможет отказаться от своей подписи, так как никто кроме него не знает закрытого ключа х.

#### Стандарт цифровой подписи ГОСТ Р34.10-94

В России принят стандарт ГОСТ Р34.10-94 "Информационная технология. Криптографическая защита информации. Процедуры выработки и проверки электронной цифровой подписи на базе асимметричного криптографического алгоритма". В этом стандарте используется алгоритм, аналогичный алгоритму, реализованному в стандарте DSS. Рассмотрим вначале полностью алгоритм, описанный в ГОСТ Р34.10-94, а затем отметим его отличия от алгоритма *DSA*.

Вначале, так же как и по стандарту DSS, для группы абонентов выбираются три общих (несекретных) параметра р, q и a:

* параметр р должен быть простым числом длиной от 512 до 1024 бит. Старший бит в р должен быть равен единице.
* q – простое число длиной 254-256 бит; так же как и в *DSA*, q должно быть делителем числа (р-1). Старший бит в q должен быть равен единице.
* число а, удовлетворяющее неравенству 1 < a < p-1 и являющееся корнем уравнения aq mod p = 1.
* Затем каждый пользователь может сформировать закрытый и открытый ключи. В качестве закрытого ключа выбирается произвольное число х, 0 < x < q. Открытым ключом является число y, получаемое по формуле

y = аx mod p

Для создания каждой новой подписи каждый раз выбирается новое случайное число k, 0 < k < q.

Подпись сообщения m состоит из двух чисел (r, s), вычисляемых по следующим формулам:

r = (аk mod p) mod q,

s = (k \* H(m) + x \* r) mod q,

где H(m) – результат вычисления хеш-функции для сообщения m.

На этом формирование подписи закончено, и сообщение m вместе с ЭЦП (r,s) может быть отправлено получателю. Теперь отметим отличия алгоритма формирования ЭЦП по ГОСТ Р34.10-94 от алгоритма DSS.

1. Перед вычислением подписи исходное сообщение обрабатывается разными функциями хеширования: в ГОСТ Р34.10-94 применяется отечественный стандарт на хеш-функцию ГОСТ Р34.11-94, в DSS используется SHA-1, которые имеют разную длину хеш-кода. Отсюда и разные требования на длину простого числа q: в ГОСТ Р34.10-94 длина q должна быть от 254 до 256 бит, а в DSS длина q должна быть от 159 до 160 бит.

2. По-разному вычисляется компонента s подписи. В ГОСТ Р34.10-94 компонента s вычисляется по формуле

s = (k \* H(m) + x \* r) mod q,

а в DSS компонента s вычисляется по формуле

s = [k-1 (H(m) + x \* r)] mod q.

Последнее отличие приводит к соответствующим отличиям в формулах для проверки подписи.

В результате процедура проверки подписи по ГОСТ Р34.10-94 заключается в следующем. Получив [m, (r, s)], получатель вычисляет

w = H(m)-1mod q,

u1 = w \* s mod q,

u2 = (q-r) \* w mod q,

v = [(аu1 \* yu2) mod p] mod q.

Затем проверяется равенство вычисленного значения v и полученного в составе ЭЦП параметра r. Подпись считается корректной, если v = r.

В алгоритме создания ЭЦП по ГОСТ Р34.10-94, так же как и в алгоритме DSS, производятся достаточно сложные вычисления, требующие затрат вычислительных ресурсов. Для ускорения процесса генерации подписей по этим алгоритмам можно заранее вычислять некоторое количество значений параметра r, не зависящего от подписываемого сообщения. Затем эти значения можно использовать по мере необходимости для подписи документов. Для алгоритма DSS заранее может вычисляться и значение k-1.

#### Пример создания и проверки подписи по стандарту ГОСТ Р34.10-94

Пусть p = 23, q = 11, a =6 (проверяем: 611 mod 23 = 1 )

*Создание подписи*.

Предположим, пользователь А выбрал в качестве закрытого ключа число х=8. После этого он вычисляет открытый ключ по формуле y = аx mod p. То есть y = 68 mod 23 = 18.

Для создания подписи пользователь А выбирает случайное число k = 5.

Пусть результат вычисления хеш-функции для сообщения H(m) = 9.

Подпись сообщения состоит из двух чисел (r, s):

r = (аkmod p) mod q = (65 mod 23) mod 11 = 2,

s = (k\* H(m) + x \* r) mod q = (5 \* 9 + 8 \* 2) mod 11 = 6,

Таким образом, подпись сообщения состоит из пары чисел (2, 6).

*Проверка подписи*.

Получив сообщение вместе с подписью (2, 6), получатель вычисляет

w = H(m)-1mod q = H(m)-1mod q = 9-1mod 11 = 5,

u1 = w \* s mod q = 5 \* 6 mod 11 = 8,

u2 = (q-r) \* w mod q = (11-2) \* 5 mod 11 = 1,

v = [(аu1 \* yu2) mod p] mod q =[(68 \* 181) mod 23] mod 11 = 2

Так как v = r, то подпись считается верной.

Подписи, созданные с использованием стандартов ГОСТ Р34.10 или DSS, так же, как и подписи, полученные по алгоритму Эль-Гамаля, являются рандомизированными, так как для одинаковых сообщений с использованием одного и того же закрытого ключа х каждый раз будут создаваться разные подписи (r,s) благодаря использованию разных случайных значений k.

#### Новый отечественный стандарт ЭЦП

В 2001 г. был принят новый отечественный стандарт на алгоритм формирования и *проверки ЭЦП*. Его полное название следующее: "ГОСТ Р34.10-2001. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи".

Данный алгоритм был разработан главным управлением безопасности связи Федерального агентства правительственной связи и информации при Президенте Российской Федерации при участии Всероссийского научно-исследовательского института стандартизации. Новый стандарт разрабатывался с целью обеспечения большей *стойкости алгоритма* генерации ЭЦП.

В основе ГОСТ Р34.10-2001 лежат алгоритмы с использованием операций на эллиптических кривых. Стойкость ГОСТ Р34.10-2001 основывается на сложности взятия *дискретного логарифма* в группе точек эллиптической кривой, а также на стойкости хэш-функции по ГОСТ Р34.11-94. Размер формируемой цифровой подписи – 512 бит.

В целом алгоритм вычислений по алгоритму ГОСТ Р34.10-2001 аналогичен применяемому в предыдущем стандарте ГОСТ Р34.10-94. Сначала генерируется случайное число k, с его помощью вычисляется компонента r подписи. Затем на основе компоненты r, числа k, значения секретного ключа и хэш-значения подписываемых данных формируется s-компонента ЭЦП. При проверке же подписи аналогичным образом проверяется соответствие определенным соотношениям r, s, открытого ключа и хэш-значения информации, подпись которой проверяется. Подпись считается неверной, если соотношения неверны.

Старый ГОСТ Р34.10-94 не отменен, и в настоящее время параллельно действуют два отечественных стандарта на ЭЦП. Однако необходимо отметить, что для прежнего ГОСТа принято ограничение: при реализации ЭЦП по стандарту ГОСТ Р34.10-94 разрешено использовать только 1024-битные значения параметра p.

Использование математического аппарата группы точек эллиптической кривой в новом ГОСТ Р34.10-2001 позволяет существенно сократить порядок модуля p без потери криптостойкости. Так, в стандарте указано, что длина числа р может быть 256 или больше бит.

### Симметричная или асимметричная криптография?

Однозначного ответа на вопрос о том, какие алгоритмы - симметричные или асимметричные - предпочтительнее, конечно же, нет. Основным достоинством симметричной криптографии является высокая скорость обработки данных. Проблемы криптосистем с закрытым ключом обсуждались подробно в лекции 8. Попробуем теперь оценить особенности алгоритмов шифрования с открытым ключом.

Главным достоинством асимметричной криптографии является отсутствие необходимости в предварительном доверенном обмене ключевыми элементами при организации секретного обмена сообщениями. К основным недостаткам асимметричных криптосистем, мешающим им вытеснить симметричные методы шифрования, относят следующие.

1. Алгоритмы с открытым ключом работают намного (в сотни раз) медленнее классических алгоритмов с закрытым ключом. Это их самый главный недостаток. Связан он с тем, что основной операцией в системах с открытым ключом является возведение в степень по большому модулю 500-1000 битовых чисел, что при программной реализации производится намного медленнее, чем шифрование того же объема данных классическими способами.
2. Алгоритмы с открытым ключом требуют обеспечения достоверности открытых ключей, что порой превращается в довольно сложную задачу. То же самое относится и к протоколам цифровой подписи. Для управления открытыми ключами используют специальную инфраструктуру открытых ключей, обеспечивающую функции управления открытыми ключами.
3. Алгоритмы с открытым ключом чувствительны к атакам по выбранному открытому тексту.

Таким образом, с практической точки зрения системы с открытым ключом и *асимметричным шифрованием* целесообразно использовать лишь для распределения секретных ключей и организации цифровых подписей, так как для решения этих задач не требуется шифрования больших блоков данных.

Использование *асимметричных алгоритмов* позволяет создавать сеансовые ключи шифрования, которые удаляются после окончания сеанса связи. Это значительно снижает риск вскрытия зашифрованных сообщений, так как, если каждое передаваемое сообщение шифруется уникальным сеансовым ключом, задача взломщика существенно усложняется. Причем пользователям совсем необязательно выполнять *протокол обмена ключом* перед симметричным шифрованием. Вот возможный вариант протокола передачи зашифрованных данных одновременно с передачей ключа.

1. Пользователь А генерирует случайный *сеансовый ключ* К и зашифровывает им с помощью *симметричного алгоритма* Fсим свое сообщение M:

Cт = Fсим(M, K)

1. Пользователь А получает из базы данных открытый ключ U пользователя Б и зашифровывает им *сеансовый ключ* К:

Ck = Fасим(К, U)

1. Пользователь А посылает своему абоненту зашифрованное сообщение Cт и зашифрованный *сеансовый ключ* Ck. Для защиты от вскрытия "человек-в-середине" передаваемые данные могут быть дополнены цифровой подписью.
2. Пользователь Б расшифровывает полученный *сеансовый ключ* Ck с помощью своего закрытого ключа R:

K = Fасим-1(Ck, R)

1. Пользователь Б расшифровывает сообщение с помощью сеансового ключа К:

M = Fсим-1 (Cт, K)

Такая *криптографическая система* называется *смешанной*, так как в ней используется и асимметричное, и *симметричное шифрование*. Смешанные криптосистемы широко применяются на практике: в банковских и платежных сетях передачи данных, в мобильной связи, в системах электронной почты и др. Для лучшего обеспечения безопасности они могут быть дополнены цифровыми подписями пользователей и удостоверяющего центра, метками времени. *Цифровая подпись* в сочетании с открытым распределением ключей позволяют организовывать защищенный обмен электронными документами.

### Управление открытыми ключами

Благодаря асимметричной криптографии проблема распределения секретных ключей, рассмотренная нами в лекции 8, была решена, вернее, ликвидирована, однако, появилась новая проблема – проблема подтверждения подлинности открытых ключей. Эта проблема заключается в том, что, получая *открытый ключ* некоторого абонента А, *пользователь* должен быть уверен, что *ключ* принадлежит именно абоненту А, а не кому-то другому. Дело в том, что в исходном виде система распределения ключей, предлагавшаяся Диффи и Хеллманом, давала возможность проведения различного рода атак, основанных на перехвате и подмене открытых ключей абонентов. Так, например, в лекции 11 нами была рассмотрена *атака* "человек-в-середине", позволявшая злоумышленнику осуществлять полный *контроль* над передаваемой в системе связи информацией. На практике возможны и другие более сложные варианты атак, связанные с подменой открытых ключей абонентов, отказом от закрытого ключа, дублированием сообщений и т.д.

Большую роль в решении проблемы *сертификации открытых ключей* сыграло создание цифровой подписи. В системах связи с большим количеством абонентов, применяющих асимметричные криптосистемы, стали использовать специальные организационные структуры, выполняющие функции управления ключами абонентов и занимающиеся сертификацией открытых ключей. Эти организационные структуры играют роль доверенной третьей стороны и заверяют открытые ключи абонентов своими цифровыми подписями. Таким образом, в распределенных системах связи, использующих криптосистемы с открытыми ключами, вводится понятие **инфраструктуры открытых ключей** (*Public* Key *Infrastructure* - *PKI*), включающей комплекс программно-аппаратных средств, а также организационно-технических и административных мероприятий, обеспечивающих абонентам системы связи необходимый сервис для управления их открытыми ключами.

Основным элементом *инфраструктуры открытых ключей* является **центр сертификации** ( *удостоверяющий центр* ) (*Certification authority*, *CA*), который обеспечивает *контроль* за выполнением всех процедур, связанных с изготовлением, регистрацией, хранением и обновлением ключей, **сертификатов открытых ключей** и списков отозванных сертификатов. Сертификат представляет собой информацию, заверенную цифровой подписью центра, и включающую *открытый ключ* и другие данные об абоненте. Такими данными являются, например, *идентификатор* алгоритма электронной подписи, имя удостоверяющего центра, срок годности сертификата, *имя пользователя*, которому принадлежит сертификат. *Международный стандарт* *ISO* X.509 определяет структуру сертификатов открытых ключей и правила их использования для аутентификации в распределенных системах связи.

Сертификат обладает следующими свойствами:

* каждый пользователь центра сертификации, имеющий доступ к открытому ключу центра, может извлечь открытый ключ, включенный в сертификат;
* ни одна сторона, помимо центра сертификации, не может изменить сертификат так, чтобы это не было обнаружено (сертификаты нельзя подделать).

Так как сертификаты не могут быть подделаны, их можно опубликовать, поместив в общедоступный справочник.

Каждый *пользователь* системы связи может быть владельцем одного или нескольких сертификатов, сформированных удостоверяющим центром. *Открытый ключ* абонента может быть извлечен из сертификата любым пользователем, знающим *открытый ключ* администратора удостоверяющего центра. В качестве администратора центра сертификации выступает обычно не физическое лицо (человек), а высокопроизводительная *автоматизированная система*.

В распределенных системах связи с большим числом абонентов может быть создано несколько центров сертификации. Центры сертификации объединяются в древовидную структуру, в корне которой находится главный *удостоверяющий центр*. Главный центр выдает сертификаты подчиненным ему центрам сертификации, тем самым заверяя открытые ключи этих центров.

*Открытый ключ* пользователя формируется на основе закрытого ключа. Каждый *пользователь* должен хранить свой закрытый *ключ* таким образом, чтобы никто другой не смог узнать его *значение*. Если же у владельца ключа есть основания полагать, что *ключ* стал известен кому-либо еще, то такой закрытый *ключ* считается *скомпрометированным*, и потерпевший, допустивший компрометацию своего ключа, должен оповестить всех остальных абонентов системы связи, что его *открытый ключ* следует считать недействительным.

Сертификаты открытых ключей имеют *период действия*, однако любой сертификат может быть выведен из обращения (отозван) до истечения этого периода, если соответствующий сертификату *секретный ключ* пользователя скомпрометирован или скомпрометирован *ключ* удостоверяющего центра, использованный при формировании сертификата. *Удостоверяющий центр* должен информировать своих абонентов об отозванных сертификатах. Для этой цели он поддерживает *список* отозванных сертификатов или *список* отмены.

При такой организации *администратор* удостоверяющего центра не имеет доступа к секретным ключам пользователей, а значит, и к защищаемой с их помощью информации. *Администратор* может лишь подменить в справочнике сертификатов открытые ключи одного из абонентов или включить фиктивного абонента, от его имени войти в контакт и получить предназначенное ему сообщение. Для исключения таких конфликтов может применяться следующая схема подготовки и рассылки ключей.

1. Администратор удостоверяющего центра генерирует пару (закрытый ключ, открытый ключ) и сообщает свой открытый ключ всем своим абонентам.
2. Пользователь А выбирает закрытые ключи для выполнения операций шифрования и формирования ЭЦП, а также вычисляет соответствующие открытые ключи.
3. Открытые ключи шифрования и подписи зашифровываются открытым ключом администратора и предъявляются в удостоверяющий центр для регистрации.
4. Администратор удостоверяющего центра проверяет (расшифровывает своим закрытым ключом) открытые ключи пользователя А; изготавливает и подписывает сертификаты открытых ключей пользователя А и помещает их в справочники открытых ключей шифрования и открытых ключей подписей. Каждый из справочников предоставляется в распоряжение абонентов удостоверяющего центра.
5. Любой пользователь системы может извлечь из справочника сертификат необходимого абонента, проверить подпись администратора под сертификатом (расшифровать его открытым ключом администратора) и извлечь открытый ключ.

Такая схема подготовки и распределения открытых ключей выглядит несколько тяжеловесной, однако она защищает абонентов системы связи от разнообразных конфликтных ситуаций. На практике рассмотренная схема дополняется метками времени в цифровых подписях, проверками дополнительных полей в сертификатах (например, срока действия) и другими проверками, повышающими *безопасность* функционирования всей системы в целом.

Необходимо отметить, что в настоящее время в связи с широким использованием асимметричных криптоалгоритмов для банковских, платежных и других систем *инфраструктура* открытых ключей постоянно совершенствуется.

### Ключевые термины

**DSS** (*Digital Signature Standard*) – стандарт США на цифровую подпись. В основе стандарта лежит *алгоритм*, называемый *DSA* (*Digital Signature Algorithm*) и являющийся вариацией подписи Эль-Гамаля.

**ГОСТ Р34.10-2001** – новый российский стандарт на *алгоритм* формирования и *проверки ЭЦП*. Основан на сложности взятия *дискретного логарифма* в группе точек эллиптической кривой, а также на стойкости хэш-функции по ГОСТ Р34.11-94. Размер формируемой цифровой подписи – 512 *бит*.

**ГОСТ Р34.10-94** – российский стандарт на *алгоритм* формирования и *проверки ЭЦП*, действующий с 1995 года. В стандарте используется модификация схемы шифрования с открытым ключом Эль-Гамаля и *алгоритм* выработки хэш-функции по ГОСТ Р34.11-94.

**Инфраструктура открытых ключей** – комплекс программно-аппаратных средств, организационно-технических и административных мероприятий, обеспечивающих абонентам системы связи необходимый сервис для управления их открытыми ключами.

**Сертификат открытого ключа** – *информация*, заверенная цифровой подписью центра, и включающая *открытый ключ* и другие данные об абоненте (*идентификатор* алгоритма электронной подписи, имя удостоверяющего центра, срок годности сертификата, *имя пользователя*, которому принадлежит сертификат и др).

**Центр сертификации** – организация или подразделение организации, которая выпускает сертификаты ключей электронной цифровой подписи и отвечает за управление ключами пользователей. Открытые ключи и другая *информация* о пользователях хранится удостоверяющими центрами в виде цифровых сертификатов.

### Краткие итоги

Для формирования электронной цифровой подписи могут быть применены алгоритмы *RSA*, Эль-Гамаля и другие. Во многих странах сегодня существуют стандарты на электронную цифровую подпись. Так, в Российской Федерации действуют стандарты на алгоритмы формирования и *проверки ЭЦП* ГОСТ Р34.10-94 и ГОСТ Р34.10-2001.

В системах связи с большим количеством абонентов, применяющих асимметричные криптосистемы, стали использовать специальные организационные структуры, выполняющие функции управления ключами абонентов и занимающиеся сертификацией открытых ключей. Эти организационные структуры играют роль доверенной третьей стороны и заверяют открытые ключи абонентов своими цифровыми подписями. Основным элементом *инфраструктуры открытых ключей* является *центр сертификации* (*удостоверяющий центр*), который обеспечивает *контроль* за выполнением всех процедур, связанных с изготовлением, регистрацией, хранением и обновлением ключей, сертификатов открытых ключей и списков отозванных сертификатов.

*Сертификат открытого ключа* представляет собой информацию, заверенную цифровой подписью центра, и включающую *открытый ключ* и другие данные об абоненте.

В целом, системы с открытым ключом и *асимметричным шифрованием* целесообразно использовать лишь для распределения секретных ключей и организации цифровых подписей, так как для решения этих задач не требуется шифрования больших блоков данных. *Шифрование* больших массивов данных обычно выполняется симметричными алгоритмами, скорость работы которых значительно выше, чем скорость алгоритмов с открытым ключом. Такие криптографические системы называются смешанными, так как в них используется и асимметричное, и *симметричное шифрование*. Смешанные криптосистемы широко применяются на практике: в банковских и платежных сетях передачи данных, в мобильной связи, в системах электронной почты и др. Для лучшего обеспечения безопасности они могут быть дополнены цифровыми подписями пользователей и удостоверяющего центра, метками времени. *Цифровая подпись* в сочетании с открытым распределением ключей позволяют организовывать защищенный обмен электронными документами.

#### Вопросы для самопроверки

1. Какие асимметричные алгоритмы и как могут применяться для формирования и проверки электронной цифровой подписи?
2. Опишите процесс создания и проверки цифровой подписи с использованием различных *асимметричных алгоритмов*.
3. Какие стандарты действуют на алгоритмы формирования и проверки электронной цифровой подписи в России?
4. Какие цифровые подписи называются рандомизированными?
5. В чем заключается проблема сертификации открытых ключей?
6. Что включается в понятие инфраструктуры открытых ключей?
7. Каковы функции центра сертификации открытых ключей?
8. Что такое сертификат открытого ключа?
9. Какая схема распределения открытых ключей абонентов может использоваться в системе связи, имеющей в своем составе центр сертификации открытых ключей?

**Лекция №7: Совершенно секретные системы**

### ****Цель:****

Изучить концепцию совершенно секретных (идеально защищённых) систем и понять критерии, которым должна соответствовать такая система для обеспечения абсолютной криптографической безопасности.

### ****Задание:****

#### ****1. Теоретическая часть:****

Изучить следующие вопросы:

* Определение совершенно секретной системы (по К. Шеннону)
* Принципы идеальной криптографической защиты
* Условия совершенной секретности:
  + Равновероятность ключей
  + Независимость зашифрованного текста от открытого
* Пример: Шифр Вернама (одноразовый блокнот)
* Проблемы и ограничения идеальных систем (практическая неприменимость, сложность генерации и хранения ключей)
* Отличие от просто стойких шифров

#### ****2. Практическая часть:****

* Составить таблицу сравнения:
  + Шифр Вернама vs. Симметричные шифры (AES, DES)
* Рассчитать пример использования шифра Вернама (на коротком сообщении)
* Ответить письменно: возможна ли реализация абсолютно секретной системы в современном мире?

#### ****3. Подготовить:****

* Доклад (1 страница) с кратким изложением теории и выводами
* Презентацию (5–8 слайдов) с примерами и схемами
* Таблицу сравнения и пример шифрования — приложить отдельно

**Цель лекции:** познакомиться с основными положениями теории информации, используемыми в криптографии и с принципами построения совершенно секретных систем.

Основные положения теории информации, используемые в криптографии, были сформулированы в середине ХХ века. Большой вклад в эти исследования внес американский ученый К. Шеннон. Для исследования принципов передачи информации по коммуникационным каналам Шеннон предложил вероятностный подход к оценке количества передаваемой информации. Кроме того, он показал, что в принципе возможны так называемые **совершенно секретные криптографические системы**, которые не могут быть "взломаны". Рассмотрим основные идеи теории Шеннона.

### Основные подходы к измерению информации

Вопросы получения, обработки, передачи и защиты информации тесно связаны с проблемой ее количественного измерения. Выделяют несколько различных подходов к решению подобной проблемы, одним из которых является так называемый *статистический* (или *алфавитный*) подход. Его суть заключается в том, что количественная оценка объема передаваемой информации осуществляется на основе анализа статистических характеристик источника информации. В этом способе учитывается способ представления информации с помощью какого-либо языка.

Очевидно, что информационная ценность некоторого сообщения связана с разнообразием вариантов генерируемых сообщений, или, другими словами, с разнообразием состояний источника информации. Ввиду этого можно считать, что объем информации, передаваемый отдельным сообщением, пропорционален общему числу N различных сообщений (или числу состояний источника информации). На практике, однако, в качестве меры информационной емкости принимается не само число N, а логарифм по основанию 2 от него:

Описание: I= \log_2 N

Данная формула позволяет получить результат в битах. Подобная *мера* получила название формально-логической логарифмической меры информации, или *меры информации по Хартли*.

Для однобуквенных сообщений N равно числу букв в алфавите информационного устройства. В частности, при числе букв равном двум *количество информации*

I = log 22 = 1 биту

Описание: I= \log_2 2 = 1 \text{ биту}

Например, рассмотрим источник, который генерирует сообщения, состоящие из одиночных букв русского языка, в алфавите которого 33 буквы. Определим, сколько информации несет отдельное сообщение, состоящее из одной буквы:

Описание: I_1= \log_2 33 \approx 5 \text{ бит}

При таком подходе для определения количества информации в некотором сообщении надо количество символов в нем умножить на *количество информации*, содержащееся в одном символе, то есть на его информационный *вес*. Подсчитаем, какое *количество информации* несет сообщение, состоящее из четырех букв русского алфавита:

Описание: I_4= 4 \log_2 33 \approx 20 \text{ бит}

Таким образом, при алфавитном подходе объем информации в сообщении оценивается не по смыслу сообщения, а по статистическим характеристикам (количеству символов).

*Мера* Хартли не всегда является верной характеристикой сообщения, так как подразумевает равную *вероятность* любого из возможных сообщений. Так как ценность информации заключается в устранении имеющейся неопределенности, сообщения, имеющие высокую *вероятность*, менее ценны. Их можно в какой-то степени предвидеть заранее. И наоборот, маловероятные сообщения представляют большую ценность. Поэтому иногда используют другой способ количественной оценки информации, учитывающий вероятности генерации конкретных сообщений, – *меру информации по Шеннону*. Этот метод измерения информации называют также *содержательным подходом*.

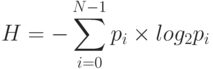
Согласно этому методу состояние источника информации до получения сообщения потребителем характеризуется некоторой неопределенностью. При этом получение информации снимает эту неопределенность (полностью или частично):

I = H нач - Н кон

Описание: I= H_{нач} - Н_{кон}

где H нач– неопределенность, характеризующая источник сообщений до получения сообщения, а Н кон– неопределенность после получения сообщения.

Неопределенность состояния источника информации оценивается по формуле



где p i – *вероятность* *i-го* состояния источника. Знак "минус" перед суммой введен потому, что величины вероятностей являются правильными дробями и имеют отрицательные логарифмы, а оценку неопределенности нужно получить со знаком "плюс".

Рассмотрим пример. При вынимании шаров из урны, где находится один черный и один белый шар, неопределенность составляет

Описание: H = - \frac {1}{2}log_2 \frac {1}{2}- \frac {1}{2}log_2 \frac {1}{2} = - log_2 \frac {1}{2} = -(-1) = 1

Неопределенность оказалась равной одному биту.

Рассмотрим другой пример. В урне находятся семь черных шаров и один белый. На этот раз неопределенность составит

Описание: H = - \frac {7}{8}log_2 \frac {7}{8}- \frac {1}{8}log_2 \frac {1}{8} = \frac {7}{8}( log_2 8-log_27) + \frac {1}{8}(log_2 8)=\frac{7(3-log_27)+3}{8} \approx \frac {7(3-2,8)+3)}{8} \approx 0,55бит

*Мера* неопределенности уменьшилась почти вдвое по сравнению с первым примером.

Неопределенность достигает максимального значения при равенстве вероятностей каждого из состояний друг другу и уменьшается с увеличением разброса этих вероятностей. Также следует заметить, что при равенстве вероятностей между собой Описание: p_i=p_j, \forall i,j=\overline{0 \dots N-1}, *мера* информации по Шеннону совпадает с мерой информации по Хартли.

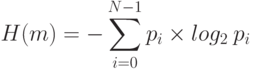
Во многих случаях алфавитный подход к измерению информации оказывается предпочтительнее. Именно *мера* информации по Хартли используется, если мы говорим, что некоторый *файл* содержит 1,5 мегабайта информации или что на одной странице некоторой книги умещается 17 *килобайт* информации.

Основной единицей количества информации является *бит*. Однако для практического применения это слишком мелкая *единица*. Более удобной единицей является *байт* (*byte*), равный восьми битам. Прибавляя к слову "*байт*" децимальные приставки "кило", "мега" и так далее, можно получать более крупные единицы измерения. Нужно только помнить об условности таких обозначений, так как их связывает *множитель*, равный не 1000, а 1024 =2 10.

### Энтропия и неопределенность

Таким образом, мы выяснили, что измерение количества информации в сообщении можно проводить на основе учета изменения неопределенности. К. Шеннон ввел понятие **энтропии** как меры неопределенности. *Энтропия* H(m) определяет *количество информации* в сообщении m и является мерой его неопределенности.

Пусть источник сообщений может создавать n разных сообщений m 1, m 2, ... m nс вероятностями p 1, p 2,... , p n. В этом случае *энтропия* сообщения будет определяться формулой



Так как в данной формуле используется двоичный логарифм, то *энтропия* измеряется в битах, что общепринято в криптографии, теории информации и в компьютерных науках.

"Физический" смысл энтропии состоит в том, что *энтропия* — это количественная *мера* неопределенности. В качестве примера рассмотрим три источника сообщений, каждый из которых может генерировать только по два разных сообщения m1 и m2. Пусть известно, что для первого источника *вероятность* появления первого сообщения р(m1)=0, а *вероятность* второго сообщения р(m1)=1. Для второго источника вероятности сообщений равны, то есть р(m1)=0,5 и р(m2)=0,5. Для третьего источника вероятности сообщений следующие: р(m1)=0,9 и р(m1)=0,1. Определим энтропию каждого из источников сообщений. Для первого источника:

H1 = -0 \* log2 0 – 1 \* log2 1 = 0 – 0 = 0

*Энтропия* или неопределенность первого источника равна нулю. И действительно, если заранее известно, что из двух сообщений всегда генерируется только одно, то никакой неопределенности нет.

Определим энтропию второго источника:

Описание: H_2=-\frac {1}{2} log_2 \frac {1}{2} - \frac {1}{2} log_2 \frac {1}{2} =-log_2 \frac {1}{2}=-(-1)=1

Неопределенность оказалась равной одному биту. Найдем теперь энтропию третьего источника:

Описание: H_3=-0,9log_2\:0,9-0,1log_2\:0,1 \approx -0,9 \times (-0,152)-0,1 \times (-3,322) \approx 0,47

Неопределенность у третьего источника меньше, чем у второго, так как из двух возможных сообщений, генерируемых третьим источником, одно более вероятно, чем другое.

Понятие энтропии играет важную роль во многих задачах теории передачи и хранения информации. В частности, *энтропия* может использоваться для определения максимальной степени сжатия данных. Точнее, если источник сообщений порождает текст достаточно большой длины n с определенной предельной энтропией h на *бит* сообщения, то этот текст теоретически может быть сжат до величины n\*h *бит*. Например, если h = 1/2, то текст может сжиматься вдвое и т.д. *Значение* n\*h является пределом и на практике достигается редко.

С точки зрения криптографии, *энтропия* определяет количество символов, которые необходимо раскрыть, чтобы узнать содержание сообщения. Так, если некоторый 8-битовый *блок данных* хранит одно из двух возможных сообщений (например, ответы "Да" или "Нет" ), то достаточно правильно узнать один *бит*, чтобы определить *значение* исходного сообщения. Сколько бы *бит* мы не отводили для шифрования слов "Да" и "Нет", *энтропия* или неопределенность всегда будет меньше или равна 1.

### Норма языка и избыточность сообщений

Для каждого языка можно ввести величину, называемую **нормой языка** r и определяемую по формуле

r = H(m)/N,

где H(m) – это *энтропия* сообщения, а N – *длина* сообщения в символах используемого языка. Норму языка можно рассматривать как *количество информации*, приходящееся на один символ сообщения. *Норма* языка будет различной для разных языков, а также для сообщений с разной длиной и содержанием. Так, например, различные исследователи оценивают норму английского языка в диапазоне от 1,0 до 1,5 *бит* на символ. Будем считать, что *норма* русского языка примерно равна 1,5 *бит* на символ.

**Абсолютной нормой языка** R называют максимальное количество *бит* информации, которое может быть передано одним символом рассматриваемого языка, при условии, что все последовательности символов равновероятны. Абсолютная *норма* языка, *алфавит* которого состоит из L символов, может быть вычислена как

R = log2 L

Для русского языка, *алфавит* которого состоит из 33 букв, абсолютная *норма* языка

Описание: R_{РУС} = \log_2 33 \approx 5 \text{ бит}

Таким образом, видно, что абсолютная *норма* русского языка значительно больше, чем реальная. В этом нет ничего удивительного, так как все естественные языки обладают значительной избыточностью. Это связано с несколькими факторами. Во-первых, некоторые буквы алфавита встречаются в сообщениях чаще других. Некоторая *статистика* по символам русского алфавита приведена в лекции 2, где рассматривается процесс криптоанализа сообщения на основе статистических данных языка. Второй причиной избыточности является то, что некоторые *сочетания* букв в словах недопустимы. Например, в русском языке нет слов, в которых стояли бы подряд буквы "ц" и "й" или "я" и "ь". Кроме того, естественные языки устроены таким образом, что иногда, зная фрагмент слова или фразы, мы может восстановить недостающую часть. Например, в приветствии

Зд.авствуй, до.огой д.уг!

мы легко сможет восстановить недостающие буквы "р".

**Избыточность языка** D оценивают как

D =R – r.

*Избыточность* русского языка получается равной 3,5 бит на символ. Это означает, что в среднем каждая буква русского языка содержит 3,5 бита неиспользуемой информации. Примерно такую же *избыточность* имеют и другие естественные языки, например, английский.

Минимальной избыточностью сообщений D = 0 обладал бы язык, в котором все символы равновероятны и могут встречаться в сообщениях независимо друг от друга в любом порядке.

### Понятие совершенно секретной системы

*Криптографическая система* называется **совершенно секретной**, если *анализ* зашифрованного текста не может дать никакой информации об открытом тексте, кроме, возможно, его длины.

Если *криптографическая система* не является совершенно секретной, то *знание* шифротекста сообщения предоставляет некоторую информацию относительно соответствующего открытого текста. Для большинства простых систем шифрования, например, методов однократной замены или перестановки, по мере увеличения длины перехваченного зашифрованного сообщения можно делать некоторые выводы о ключе шифрования или об открытом тексте. Это связано с большой избыточностью естественных языков. Так, например, если перехвачено сообщение, зашифрованное методом перестановки, то противник может узнать, какие символы и в каком количестве встречались в исходном сообщении, а после этого может попробовать провести какой-либо более сложный *анализ* с целью определения правила перестановки. Если нам известно зашифрованное методом моноалфавитной замены сообщение ДКДК, мы, конечно, без дополнительной информации не сможем однозначно определить, что содержалось в исходном тексте. Однако, получив в свое распоряжение ДКДК, мы сможем сделать *вывод*, что

1. в исходном сообщении использовалось всего две буквы алфавита
2. первая и третья, а также вторая и четвертые буквы открытого текста были одинаковы.

Можно также предположить, что либо Д, либо К заменяют гласную букву. Может быть, исходное сообщение представляло собой *слово* МАМА, а может быть ПАПА, а может быть что-нибудь еще. Однозначно дешифровать его нельзя, однако некоторую информацию по шифротексту мы смогли определить. Таким образом, можно сделать *вывод*, что методы перестановки или замены не являются совершенно секретными криптографическими шифрами.

На практике возможна следующая реализация совершенно секретной системы, называемая **одноразовая лента** (или **одноразовый блокнот**, или **шифр Вернама** по имени американского инженера, предложившего эту систему в первой половине ХХ века). Будем предполагать, что процессу шифрования подвергаются двоичные данные. На передающей и приемной сторонах подготавливаются две одинаковые ленты, например, магнитные. Они содержат *ключ* шифрования. На передающей стороне лента помещается в устройство шифрования, а на принимающей стороне – в идентичное устройство, используемое для расшифрования. Когда отправитель хочет передать сообщение, он складывает по модулю два один *бит* исходного сообщения и один *бит* с магнитной ленты. После этого лента перемещается в следующее положение и можно шифровать второй *бит* сообщения, используя второй *бит* ключа. Таким образом шифруется все сообщение. На принимающей стороне лента с ключом используется аналогично.

Например, пусть исходное сообщение m содержит следующие двоичные цифры:

m = 1100101110...

Предположим, в качестве ключевой используется последовательность:

k = 1001100111...

Выполним *шифрование* по методу одноразовой ленты, сложив цифры в каждом столбике по модулю 2:

исходный текст m = 1100101110...

биты ключевой послед-ти k = 1001100111...

-------------------

зашифрованный текст с = 0101001001...

Этот процесс напоминает наложение гаммы на *поток* входных данных. *Шифр* с одноразовой лентой действительно является гаммированием, однако, в отличие от всех рассмотренных до этого криптосистем в нем предполагается *бесконечная гамма*.

В одноразовой ленте все буквы встречаются с одинаковой частотой. Поэтому, сколько бы знаков гаммы нам ни было известно, мы не сможем предсказать, какой будет следующая буква. Из этого следует, что все последовательности знаков гаммы равновероятны. Это означает, что сообщение, зашифрованное с помощью шифра Вернама, может быть "дешифровано" в любой *открытый текст* подходящей длины, поскольку предполагаемая последовательность знаков гаммы не имеет никаких свойств, позволяющих отличить её от любой другой.

В шифре Вернама, конечно, совсем необязательно использовать в качестве носителя ключевых данных именно ленту. Главное, чтобы у отправителя и получателя был секретный *ключ* размера не меньшего, чем *длина* исходного сообщения. Проблемы могут возникнуть при шифровании большого объема данных, так как запасы ключевых цифр должны быть заблаговременно доставлены получателю информации и храниться у него.

Совершенно секретные системы могут быть реализованы на практике. Почему же они не используются во всех случаях? Это объясняется несколькими причинами. Во-первых, как и любые системы шифрования с закрытым ключом в них существует проблема распределения ключей. Во-вторых, в совершенной секретной системе *ключ* шифрования должен иметь по крайней мере такую же длину, как и *открытый текст*. Кроме того, для шифрования каждого сообщения должен применяться свой новый *ключ*. Все эти факторы делают реализацию совершенно секретной системы очень дорогой и не слишком удобной. Такие системы имеет смысл использовать лишь для самых важных линий связи, например, правительственных.

### Расстояние единственности

Если *криптографическая система* не является совершенно секретной, то зашифрованное сообщение может дать криптоаналитику некоторую информацию об исходном сообщении. Специалист, может быть, и не сможет сразу однозначно дешифровать шифротекст, однако он будет иметь возможность делать некоторые предположения о ключе или открытом тексте. После получения каждого следующего зашифрованного тем же ключом сообщения криптоаналитик будет расширять свои знания относительно ключа шифрования и в конце концов сможет дешифровать сообщения.

Возникает вопрос, а какой должна быть *длина* зашифрованного сообщения, чтобы можно было однозначно дешифровать его. В примере, рассмотренном в лекции 2 проводилась попытка дешифрования сообщения из 11 символов, *закрытого методом* простой замены. На основе анализа статистических закономерностей русского языка удалось подобрать несколько подходящих вариантов открытых текстов, однако для выбора из них одного "правильного" не хватило информации. По-видимому, существует некоторая *длина* перехваченного сообщения, после которой сообщение может быть дешифровано с вероятностью, близкой к единице.

Шеннон ввел понятие **расстояния единственности** шифра (или **расстояния уникальности** ) U, которое показывает, сколько букв зашифрованного сообщения необходимо перехватить для однозначного восстановления ключа.

Для вычисления расстояния единственности необходимо знать энтропию ключа Н(К). Для симметричных шифров *энтропия* ключа примерно равна логарифму числа ключей NK по основанию 2:

Н(К) = log2 NK

Например, для шифра простой замены, применяемого к русскому языку, число возможных ключей определяется количеством всех возможных таблиц замен и равно Описание: N_K = 33! \approx 8,68*10^{36} поэтому *энтропия* ключа будет равна

Описание: Н(К) = log_2  8,68 * 10^{36} \approx 122,7

Если нам известна *энтропия* ключа Н(К) для некоторого шифра, то *расстояние* единственности U для него вычисляется по формуле

U = H(K) / D,

где D – *избыточность* шифруемого сообщения.

Рассчитаем *расстояние* единственности для шифра простой замены, применяемого к сообщению на русском языке:

Описание: U = H(K) / D = 122,7 / 3,5 \approx 35,1

То есть если *длина* перехваченного зашифрованного сообщения превышает 35 символов, то его, скорее всего, можно будет однозначно дешифровать. А при длине зашифрованного текста меньше 35 символов возможно неоднозначное вскрытие.

*Расстояние* единственности показывает нам не то, какой размер должен иметь перехваченный шифротекст, чтобы его было *легко* дешифровать, а то, насколько большим он должен быть, чтобы было возможно в принципе его *однозначное* *дешифрование*.

Для того чтобы затруднить противнику *определение* ключа и *дешифрование* наших секретных сообщений, необходимо увеличивать (а желательно доводить до бесконечности) *расстояние* единственности в применяемых шифрах. Проанализировав формулу для вычисления расстояния единственности, определим, что это можно сделать двумя способами.

Если *энтропия* ключа равна бесконечности, то *расстояние* единственности шифра будет тоже равно бесконечности. *Энтропия* ключа тем больше, чем длиннее *ключ*. В случае использования системы одноразовой ленты *ключ* теоретически бесконечен и все его символы равновероятны, поэтому *энтропия* ключа такого шифра будет бесконечно большой. Следовательно, *расстояние* единственности шифра Вернама равно бесконечности.

Как уже отмечалось выше, использовать *шифр* с бесконечно большим ключом практически нецелесообразно. Однако на практике можно менять иногда *ключ* шифрования, тем самым затрудняя жизнь противнику. Специалисты рекомендуют использовать системы, в которых *ключ* меняется задолго до достижения расстояния единственности шифра. Этого можно достичь, например, используя *сеансовые ключи* шифрования, то есть применять для шифрования каждого сообщения новый *ключ*.

Второй способ увеличения расстояния единственности состоит в уменьшении избыточности исходного текста. Если *избыточность* сообщения равна нулю, то *ключ* никогда не будет определен, а зашифрованное сообщение вскрыто, так как *расстояние* единственности будет равно бесконечности. К сожалению, на практике такая ситуация невозможна, так как любое осмысленное сообщение будет иметь некоторую отличную от нуля *избыточность*.

Однако возможно уменьшение избыточности в сообщениях за счет сжатия данных. Дело в том, что при сжатии данных *энтропия* "сжатого" текста сохраняется, а *длина* уменьшается. Следовательно, *энтропия* на букву в сжатом тексте больше, чем в исходном, а *избыточность* – меньше. Значит, после сжимающего кодирования *расстояние* единственности шифра увеличивается.

### Ключевые термины

**Абсолютная норма языка** – максимальное количество *бит* информации, которое может быть передано одним символом некоторого языка, при условии, что все последовательности символов в языке равновероятны.

**Избыточность языка** – статистическая величина, обозначающая *избыточность* информации, содержащейся в тексте на определённом языке.

**Норма языка** – величина, характеризующая *количество информации*, приходящееся на один символ сообщения.

**Одноразовая лента (или одноразовый блокнот, или шифр Вернама)** – один из возможных вариантов реализации совершенно секретной системы. Может быть реализован как гаммирование с бесконечной гаммой.

**Расстояние единственности шифра (или расстояния уникальности)** – величина, показывающая, сколько букв зашифрованного сообщения необходимо перехватить для однозначного восстановления ключа.

**Совершенно секретная система** – *криптографическая система*, для которой *анализ* зашифрованного текста не дает никакой информации об открытом тексте, кроме, возможно, его длины.

**Энтропия сообщения** – характеристика, введенная Шенноном. Определяет *количество информации*, приходящейся на одно элементарное сообщение источника, вырабатывающего статистически независимые сообщения. Является мерой неопределённости или непредсказуемости информации.

### Краткие итоги

Основные положения теории информации, используемые в криптографии, были сформулированы в середине ХХ века. Большой вклад в эти исследования внес американский ученый К. Шеннон. Для исследования принципов передачи информации по коммуникационным каналам Шеннон предложил вероятностный подход к оценке количества передаваемой информации. Кроме того, он показал, что в принципе возможны так называемые совершенно секретные криптографические системы, которые не могут быть "взломаны".

*Криптографическая система* называется совершенно секретной, если *анализ* зашифрованного текста не может дать никакой информации об открытом тексте, кроме, возможно, его длины. Одним из возможных вариантов реализации совершенно секретной системы является *шифр* Вернама (или одноразовый блокнот, или одноразовая лента). *Шифрование* по этому методу выполняется как *сложение* по модулю два открытого текста с бесконечным ключом (называемым одноразовым блокнотом или шифроблокнотом). При этом *ключ* должен обладать тремя важными свойствами: быть истинно случайным; совпадать по размеру с заданным открытым текстом; применяться только один раз. Данный метод, к сожалению, не слишком удобен на практике. Во-первых, как и любые системы шифрования с закрытым ключом в них существует проблема распределения ключей. Во-вторых, в совершенной секретной системе *ключ* шифрования должен иметь по крайней мере такую же длину, как и *открытый текст*. Кроме того, для шифрования каждого сообщения должен применяться свой новый *ключ*. Все эти факторы делают реализацию совершенно секретной системы очень дорогой и не слишком удобной. Такие системы имеет смысл использовать лишь для самых важных линий связи, например, правительственных.

Если *криптографическая система* не является совершенно секретной, то *знание* шифротекста сообщения предоставляет некоторую информацию относительно соответствующего открытого текста. Большинство используемых на практике криптографических систем не является совершенно секретными.

Для увеличения безопасности реальных криптографических систем специалисты рекомендуют:

1. использовать *сеансовые ключи* шифрования, то есть применять для шифрования каждого сообщения новый ключ;
2. уменьшать избыточность шифруемых массивов данных, например, выполнять сжатие сообщений перед криптографическим шифрованием.

#### Вопросы для самопроверки

1. Как определяется энтропия источника сообщений?
2. Что характеризует энтропия источника сообщений?
3. Что определяет норма языка?
4. Как вычисляется абсолютная норма языка?
5. Что характеризует избыточность языка?
6. Почему в сообщениях, составленных на естественных языках, всегда имеется избыточность?
7. Дайте определение совершенно секретной криптографической системы.
8. Приведите примеры шифров, которые не являются совершенно секретными системами.
9. Почему шифр одноразовой ленты (шифр Вернама) является совершенно секретной системой?
10. Почему совершенно секретные системы не используются повсеместно на практике для защиты информации?
11. Как определяется энтропия ключа шифра?
12. Каким образом вычисляется расстояние единственности для шифра?

**Лекция №8: Шифрование, помехоустойчивое кодирование и сжатие информации**

### ****Цель:****

Изучить взаимосвязь между процессами шифрования, помехоустойчивого кодирования и сжатия информации в системах передачи и хранения данных, а также рассмотреть их совместное применение для повышения надёжности и безопасности.

### ****Задание:****

#### ****1. Теоретическая часть:****

Изучить и кратко изложить следующие темы:

* Понятие шифрования и его цель
* Основы помехоустойчивого (корректирующего) кодирования: Hamming, BCH, Reed-Solomon и др.
* Принципы сжатия информации: без потерь (Huffman, LZ77) и с потерями (JPEG, MP3)
* Влияние порядка применения: «сжатие → шифрование → кодирование»
* Примеры использования всех трёх этапов в реальных системах (например, в мобильной связи, спутниковых каналах, архивации и передаче данных)

#### ****2. Практическая часть:****

* Рассмотреть задачу: передать текстовый файл по нестабильному каналу. Предложить и обосновать порядок применения трёх процедур: сжатие, шифрование, кодирование.
* Найти и описать минимум одну современную систему, в которой применяются все три технологии.
* Выполнить пример: закодировать короткий текст сначала сжатием (любой метод), затем зашифровать (например, Цезаря или AES), после чего применить простой код Хэмминга.

#### ****3. Подготовить:****

* Краткий доклад (1–2 страницы) с описанием теоретических основ и анализа задачи
* Презентацию (5–7 слайдов)
* Решение практической задачи с пояснением

**Цель лекции**: познакомиться с принципами совместного использования шифрования, помехоустойчивого кодирования и сжатия информации для комплексной защиты информации в процессе ее передачи и хранения.

### Проблемы передачи информации и их комплексное решение

В процессе передачи информации от источника к потребителю на информацию воздействуют различные неблагоприятные факторы. Криптографические методы защищают информацию только от одного вида разрушающих воздействий – от предумышленного разрушения или искажения информации. Однако на практике при передаче информации от абонента к абоненту возможны случайные помехи на линиях связи, ошибки и сбои аппаратуры, частичное разрушение носителей данных и т.д. Таким образом, в реальных системах связи существует *проблема защиты информации от случайных воздействий*.

В связи с появлением сетей передачи данных высокой пропускной способности и развитием мультимедиа-технологий возникает *проблема шифрования больших объемов информации*. Если раньше основным типом шифруемых и передаваемых сообщений было текстовое сообщение, то в ХХI веке криптографическая защита все чаще применяется при передаче цифровых видео- и речевых сообщений, карт местности, для организации видеоконференций. Именно поэтому в последнее время возникает проблема шифрования огромных информационных массивов. Для интерактивных систем типа телеконференций, организации аудио- или видеосвязи, такое *шифрование* должно осуществляться в реальном режиме времени и по возможности быть незаметным для пользователей.

Решение указанных проблем, в том числе и защита от несанкционированного доступа, может быть достигнуто при комплексном использовании достижений теории информации.

В принципе в теории информации выделяют три вида преобразования информации: криптографическое *шифрование*, помехоустойчивое *кодирование* и сжатие (или компрессия). В некоторых научных работах ХХ века все три вида преобразования информации называли кодированием: криптографическое *кодирование*, помехоустойчивое *кодирование* и эффективное *кодирование* (сжатие данных). Общим для всех трех видов преобразования является то, что *информация* каким-либо образом меняет форму представления, но не смысл. Отличия разных видов кодирования связаны с целью проводимых преобразований.

Так, целью криптографического преобразования является, как известно, защита от несанкционированного доступа, *аутентификация* и защита от преднамеренных изменений. Помехоустойчивое *кодирование* выполняется с целью защиты информации от случайных помех при передаче и хранении. Эффективное *кодирование* производится с целью минимизации объема передаваемых или хранимых данных.

На практике эти три вида преобразования информации обычно используются совместно. Так, например, некоторые программные пакеты перед шифрованием архивируют обрабатываемые данные. С другой стороны, реальные системы передачи информации, будь то локальные и *глобальные сети* передачи данных, или компьютерные носители информации (CD или DVD-диски) всегда имеют в составе системы защиты информации средства контроля и коррекции случайных ошибок.

Таким образом, криптографическое *шифрование*, помехоустойчивое *кодирование* и сжатие отчасти дополняют друг друга и их комплексное использование помогает эффективно использовать каналы связи для надежной защиты передаваемой информации.

Для того, чтобы более эффективно использовать на практике криптографические методы защиты информации, рассмотрим основные положения теорий помехоустойчивого и эффективного кодирования, используемые в системах защиты информации.

### Помехоустойчивое кодирование

Как уже отмечалось, вопросы криптографического преобразования информации тесно связаны с вопросами помехоустойчивого кодирования сообщений. Это обусловлено, с одной стороны (теоретической), тем, что и при криптографическом шифровании, и при помехоустойчивом кодировании используются одни и те же законы теории информации. С другой стороны (практической) процессы накопления, хранения и передачи информации протекают в условиях воздействия помех, способных исказить хранимые и обрабатываемые данные. Это обуславливает актуальность разработки и использования методов, позволяющих обнаруживать и корректировать подобные ошибки. С математической точки зрения задача сводится к синтезу так называемых **помехоустойчивых кодов**.

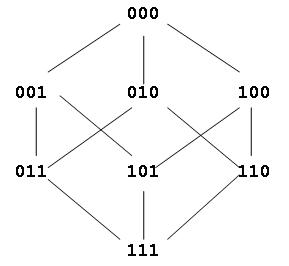
Аналогично понятию шифра в криптографии при обсуждении помехоустойчивого кодирования и вопросов сжатия сообщений вводят понятие кода. Вообще **кодом** называется совокупность знаков, а также система правил, позволяющая представлять информацию в виде набора таких знаков. **Кодовым словом** называют любой ряд допустимых знаков. Например, двоичное число 1100 можно считать двоичным 4-разрядным кодовым словом.

Общая идея помехоустойчивого кодирования состоит в том, что из всех возможных кодовых слов считаются допустимыми не все, а лишь некоторые из них. Например, в коде с контролем по четности считаются допустимыми лишь слова с четным числом единиц. Ошибка превращает *допустимое* *слово* в *недопустимое* и поэтому обнаруживается.

Помехоустойчивые коды делятся на блоковые, делящие информацию на фрагменты постоянной длины и обрабатывающие каждый из них в отдельности, и свёрточные, работающие с данными как с непрерывным потоком.

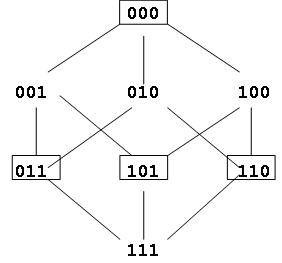
Блоковые коды характеризуются так называемым минимальным кодовым расстоянием. Вообще, **расстоянием по Хэммингу** (по имени американского *математика* Р.У. Хэмминга) между двумя кодовыми словами называется число разрядов, в которых они различны. При этом в качестве **минимального кодового расстояния** выбирается наименьшее из всех расстояний по Хэммингу для любых пар различных кодовых слов, образующих код.

Например, пусть мы используем только трехразрядные двоичные слова. Всего таких кодовых слов может быть восемь. Те кодовые слова, которые отличаются только на одну единицу, называются **соседними**. Например, кодовые слова 101 и 111 – соседние, так как отличаются только средним разрядом, а слова 101 и 110 – не соседние, так как у них отличаются два последних разряда. Изобразим все трехразрядные двоичные комбинации и соединим линией соседние кодовые слова. Тогда мы получим схему, как на [рис. 14.1](https://www.intuit.ru/studies/courses/691/547/lecture/12397?page=1" \l "image.14.1). *Минимальное кодовое расстояние* между словами обычного, не *помехоустойчивого кода* равно единице.



**Рис. 14.1.**Трехразрядные двоичные кодовые слова

В случае использования всех трехразрядных двоичных слов для передачи сообщений все они будут считаться допустимыми. Применим *контроль* по условию четности. Тогда допустимыми будут только выделенные рамками слова с четным числом единиц (см. [рис. 14.2](https://www.intuit.ru/studies/courses/691/547/lecture/12397?page=1" \l "image.14.2)).



**Рис. 14.2.**Допустимые трехразрядные кодовые слова при контроле по четности