1. **Необходимость защиты информации**

Защита информации – комплекс правовых, организационных и технических мероприятий и действий по предотвращению угроз информационной безопасности и устранению их последствий в процессе сбора, хранения, обработки и передачи информации в информационных системах. Под субъектами информационных отношений понимаются как владельцы, так и пользователи информации и поддерживающей инфраструктуры. Спектр интересов субъектов, связанных с использованием информационных систем, можно разделить на следующие составляющие: обеспечение доступности, целостности и конфиденциальности информационных ресурсов и поддерживающей инфраструктуры.

Доступность информации – свойство системы обеспечивать своевременный беспрепятственный доступ правомочных (авторизованных) субъектов к интересующей их информации или осуществлять своевременный информационный обмен между ними. Информационные системы создаются (приобретаются) для получения определенных информационных услуг. Если по тем или иным причинам предоставить эти услуги пользователям становится невозможно, это, очевидно, наносит ущерб всем субъектам информационных отношений.

Целостность информации – свойство информации, характеризующее ее устойчивость к случайному или преднамеренному разрушению или несанкционированному изменению. Целостность можно подразделить на статическую (понимаемую как неизменность информационных объектов) и динамическую (относящуюся к корректному выполнению сложных действий (транзакций)). Средства контроля динамической целостности применяются, в частности, при анализе потока финансовых сообщений с целью выявления кражи, переупорядочения или дублирования отдельных сообщений.

Конфиденциальность информации – свойство информации быть известной и доступной только правомочным субъектам системы (пользователям, программам, процессам). Конфиденциальность – самый проработанный у нас в стране аспект информационной безопасности.

Все каналы утечки данных можно разделить на косвенные и прямые.

Примеры косвенных каналов утечки:

* Кража или утеря носителей информации, исследование не уничтоженного мусора;
* Дистанционное фотографирование, прослушивание;
* Перехват электромагнитных излучений.

Примеры прямых каналов утечки:

* Инсайдеры (человеческий фактор). Утечка информации вследствие несоблюдения коммерческой тайны;
* Прямое копирование.

Каналы утечки информации можно также разделить по физическим свойствам и принципам функционирования:

* акустические — запись звука, подслушивание и прослушивание;
* акустоэлектрические — получение информации через звуковые волны с дальнейшей передачей её через сети электропитания;
* виброакустические — сигналы, возникающие посредством преобразования информативного акустического сигнала при воздействии его на строительные конструкции и инженерно-технические коммуникации защищаемых помещений;
* оптические — визуальные методы, фотографирование, видео съемка, наблюдение;
* электромагнитные — копирование полей путём снятия индуктивных наводок;
* радиоизлучения или электрические сигналы от внедренных в технические средства и защищаемые помещения специальных электронных устройств съема речевой информации «закладных устройств», модулированные информативным сигналом;
* материальные — информация на бумаге или других физических носителях информации

1. **История криптографии. Основные исторические события, шифры, устройства.**

Криптография изучает построение и использование систем шифрования, в том числе их стойкость, слабости и степень уязвимости относительно различных методов вскрытия.

Основные определения криптографии:

Шифр – совокупность заранее оговоренных способов преобразования исходного секретного сообщения с целью его защиты.

Исходные сообщения обычно называют открытыми текстами. В иностранной литературе для открытого текста используют термин plaintext.

Символ - это любой знак, в том числе буква, цифра или знак препинания.

Алфавит - конечное множество используемых для кодирования информации символов. Например, русский алфавит содержит 33 буквы от А до Я . Однако этих тридцати трех знаков обычно бывает недостаточно для записи сообщений, поэтому их дополняют символом пробела, точкой, запятой и другими знаками. Алфавит арабских цифр – это символы 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 . Этот алфавит содержит 10 знаков и с его помощью можно записать любое натуральное число. Любое сообщение может быть записано также с помощью двоичного алфавита , то есть с использованием только нулей и единиц.

Сообщение, полученное после преобразования с использованием любого шифра, называется шифрованным сообщением (закрытым текстом, криптограммой). В иностранной литературе для закрытого текста используют термин ciphertext.

Преобразование открытого текста в криптограмму называется зашифрованием. Обратное действие называется расшифрованием. В англоязычной литературе терминам "зашифрование/ расшифрование" соответствуют термины "enciphering/deciphering".

Ключ – информация, необходимая для шифрования и расшифрования сообщений.

С точки зрения русского языка термины "расшифрование" и "дешифрование" являются синонимами. Однако в работах по криптографии последних десятилетий часто эти слова различают. Будем считать, что термины "расшифрование" и "дешифрование" не являются синонимами. Примем, что расшифрованием занимается легальный получатель сообщения (тот, кто знает ключ), а человек, которому послание не предназначено, пытаясь понять его смысл, занимается дешифрованием .

Система шифрования, или шифрсистема, – это любая система, которую можно использовать для обратимого изменения текста сообщения с целью сделать его непонятным для всех, кроме тех, кому оно предназначено.

Криптостойкостью называется характеристика шифра, определяющая его стойкость к дешифрованию без знания ключа (т.е. способность противостоять криптоанализу).

Первый период (приблизительно с 3-го тысячелетия до н. э.) характеризуется господством моноалфавитных шифров (основной принцип — замена алфавита исходного текста другим алфавитом через замену букв другими буквами или символами). Шифр Скитала, Квадрат Полибия, Шифр Цезаря.

Второй период (хронологические рамки — с IX века на Ближнем Востоке (Ал-Кинди) и с XV века в Европе (Леон Баттиста Альберти) — до начала XX века) ознаменовался введением в обиход полиалфавитных шифров. Шифр Виженера, Книжный шифр.

Третий период (с начала и до середины XX века) характеризуется внедрением электромеханических устройств в работу шифровальщиков. При этом продолжалось использование полиалфавитных шифров. Шифровальная машина Энигма, машина Лоренса.

Четвёртый период — с середины до 70-х годов XX века — период перехода к математической криптографии. В работе Шеннона появляются строгие математические определения количества информации, передачи данных, энтропии, функций шифрования. Обязательным этапом создания шифра считается изучение его уязвимости к различным известным атакам — линейному и дифференциальному криптоанализу. Однако до 1975 года криптография оставалась «классической» или же, более корректно, криптографией с секретным ключом.

Современный период развития криптографии (с конца 1970-х годов по настоящее время) отличается зарождением и развитием нового направления — криптография с открытым ключом. Её появление знаменуется не только новыми техническими возможностями, но и сравнительно широким распространением криптографии для использования частными лицами. Правовое регулирование использования криптографии частными лицами в разных странах сильно различается — от разрешения до полного запрета.

1. **Секретные системы. Типы секретных систем**

Первая серьезная попытка систематизировать и подвести общую математическую базу под теорию криптографии была предпринята Клодом Шенноном в 1945 году. Статья «Теория связи в секретных системах» первоначально составляла содержание секретного доклада «Математическая теория криптографии», датированного 1 сентября 1945 г. Статья в настоящее время рассекречена.

Согласно К. Шеннону существуют три общих типа секретных систем:

* Системы маскировки, при помощи которых скрывается сам факт наличия сообщения (стеганография). Например, невидимые чернила или маскировка сообщения за безобидным текстом.
* Тайные системы, в которых для раскрытия сообщения требуется специальное оборудование. Например, инвертирование речи.
* Криптографические системы, где смысл сообщения скрывается при помощи шифра, кода и т. п., но само существование сообщения не скрывается.

Ограничимся рассмотрением только третьего вида систем и только для случая, когда информация имеет дискретный вид.

Секретная система – это некоторое множество отображений одного пространства (множества возможных сообщений) в другое пространство (множество возможных криптограмм), где каждое конкретное отображение из этого множества соответствует способу шифрования при помощи конкретного ключа.

Отображение является *взаимнооднозначным*, то есть если известен ключ, то в результате процесса дешифрования возможен лишь единственный ответ.

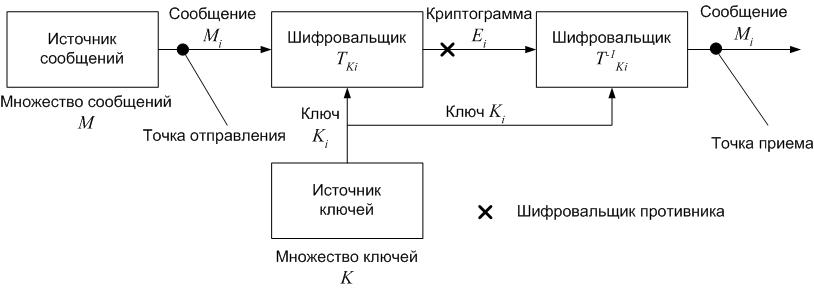
Каждому такому ключу соответствует некоторая *априорная вероятность –* вероятность выбрать этот ключ.

#### Понятие криптографической системы

Для использования такой системы для определенного сообщения *Mi* выбирается некоторый ключ *Ki­* из множества возможных ключей *K*. После чего при помощи ключа *Ki­* формируется криптограмма *Ei*. Эта криптограмма, полученная при помощи преобразования *ТKi*, по каналу передачи передается в точку приема. На приемном конце с помощью отображения , обратного выбранному, из криптограммы *Ei* восстанавливается исходное сообщение *Mi*.

Если противник перехватит криптограмму, то он не сможет ее расшифровать, если не знает ключа *Ki­*. Поэтому, чем больше мощность множества *K*, тем меньше вероятность того, что криптограмма будет расшифрована. Эта вероятность называется *апостериорной вероятностью*.

**Рисунок 2.6. Общая схема криптографической системы**



*Вычисление апостериорных вероятностей – есть общая задача дешифрования.*

#### Пример вычисления апостериорных вероятностей

Пусть имеется некоторый алфавит, состоящий из трех символов: «A», «B» и «C», и известны относительные частоты использования каждой буквы *P(A) = 50%, P(B) = 20%, P(C) = 30%.*

Возьмем произвольное сообщение, состоящее из *10* букв: «АВАСАСАВСА».

При использовании метода случайной подстановки мы получим множество возможных отображений, мощность которого равна *3! = 6*, то есть:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Множество возможных криптограмм | *P(A)*, % | *P(B)*, % | *P(C)*, % |
| 1 | ABACACABCA | 50 | 20 | 30 |
| 2 | BABCBCBACB | 20 | 50 | 30 |
| 3 | BCBABABCAB | 30 | 50 | 20 |
| 4 | CBCACACBAC | 30 | 20 | 50 |
| 5 | CACBCBCABC | 20 | 30 | 50 |
| 6 | ACABABACBA | 50 | 30 | 20 |

Допустим, что был выбран третий ключ для шифрования. Тогда закодированное сообщение будет «BCBABABCAB». Если противник знает, что мощность множества ключей равна *6* и ему известны относительные частоты использования букв, то он соответственно осуществит *6* перестановок и для каждой проверит *P(A), P(B)* и *P(C).*

Истинные относительные частоты он получит только в одном случае, во всех остальных они будут ложными.

Итак, для того чтобы найти решение задачи дешифрования необходимо знать:

1. Алфавит, используемый в исходном сообщении.
2. Мощность множества возможных ключей.
3. Вероятностные характеристики использования букв, слов.
4. Схему, по которой проводится шифрование.

На практике восстановление может быть очень сложной задачей, так как:

1. Информация об источнике сообщений неполная или ее вообще нет.
2. Мощность множества возможных ключей настолько велика, что перебор всех возможных значений займет слишком много времени (для алфавита в *256* символов мощность множества ключей в алгоритме простой перестановки составит ).
3. Вероятность использования символов может быть либо неизвестной (неизвестный язык источника сообщений), либо выражаться нечетко (метод использования имитовставок, когда в шифруемый текст преднамеренно вводится лишняя информация с целью «потопления статистики»).
4. Схема, по которой осуществлялось шифрование, неизвестна, либо достаточно сложна.

**4. Общая постановка задачи дешифрирования**

#### **5.Основные характеристики секретных систем. Комбинирование**

Имеется несколько различных критериев, которые можно было бы использовать для оценки качества предлагаемой секретной системы.

1. *Количество секретности* – определяется признаком существования единственности правильного решения; чем больше количество возможных решений с одинаковыми апостериорными вероятностями, тем выше секретность системы.
2. *Объем ключа –* чем меньше размер ключа, тем лучше, так как его легче запомнить.
3. *Сложность операции шифрования и дешифрования –* по возможности эти операции должны быть как можно проще для снижения затрат времени.
4. *Разрастание числа ошибок –* в некоторых шифрах ошибка в одной букве, допущенная при шифровании или передаче, приводит к большому числу ошибок в расшифрованном тексте, требуя повторной передачи криптограммы, следовательно, желательно минимизировать это разрастание.
5. *Увеличение объема сообщения –* в некоторых системах объем сообщения увеличивается в результате операции шифрования с целью «потопления статистики», этот нежелательный эффект нужно пытаться минимизировать.

#### Комбинирование секретных систем

Если имеются две секретные системы *T* и *R*, их часто можно комбинировать различными способами для получения новой секретной системы *S*.

Чаще всего используется два способа комбинирования: *взвешенная сумма* и *произведение*.

Взвешенная сумма

Если имеются две секретные системы, которые имеют одно и то же пространство сообщений, то можно образовать взвешенную сумму:



где *p* – вероятность использования системы Т, *q* – вероятность использования системы R.

Выбор конкретной системы является частью ключа системы *S*. Полный ключ должен определять, какая из систем выбрана (*T* или *R*) и с каким ключом используется выбранная система, так как любую систему можно записать как сумму фиксированных операций:

,

где *Ti* – определенная операция шифрования в системе *Т*, соответствующая выбору ключа *i*, причем вероятность такого выбора равна *pi*.

Обобщая далее, можно образовать сумму нескольких систем:

.

Произведение

Образование произведения двух секретных систем осуществляется следующим образом:

*S = RT*, причем *RS = SR*, а *RS ≠ RS.*

1. **Криптографические методы защиты информации**

Криптографические методы защиты информации – это мощное оружие в борьбе за информационную безопасность.

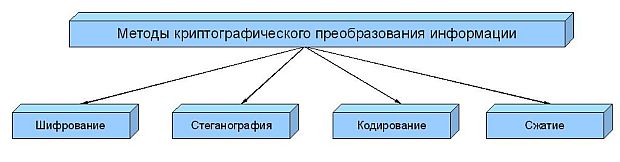
***Криптография*** (от древне-греч. κρυπτος – скрытый и γραϕω – пишу) – наука о методах обеспечения конфиденциальности и аутентичности информации.

Криптография представляет собой совокупность методов преобразования данных, направленных на то, чтобы сделать эти данные бесполезными для злоумышленника. Такие преобразования позволяют решить два главных вопроса, касающихся безопасности информации:

* защиту конфиденциальности;
* защиту целостности.

Проблемы защиты конфиденциальности и целостности информации тесно связаны между собой, поэтому методы решения одной из них часто применимы для решения другой.

Известны различные подходы к классификации методов криптографического преобразования информации. По виду воздействия на исходную информацию методы криптографического преобразования информации могут быть разделены на четыре группы:

[](https://www.intuit.ru/EDI/06_02_17_3/1486333223-7320/tutorial/1346/objects/2/files/2_1.jpg)

**Рис. 2.1.**Классификация методов криптографического преобразования информации

Процесс ***шифрования*** заключается в проведении обратимых математических, логических, комбинаторных и других преобразований исходной информации, в результате которых зашифрованная информация представляет собой хаотический набор букв, цифр, других символов и двоичных кодов.

Для шифрования информации используются **алгоритм преобразования** и **ключ**. Как правило, алгоритм для определенного метода шифрования является неизменным. Исходными данными для алгоритма шифрования служит информация, подлежащая зашифрованию, и ключ шифрования. Ключ содержит управляющую информацию, которая определяет выбор преобразования на определенных шагах алгоритма и величины операндов, используемых при реализации алгоритма шифрования. Операнд – это константа, переменная, функция, выражение и другой объект языка программирования, над которым производятся операции.

В отличие от других методов криптографического преобразования информации, методы ***стеганографии*** позволяют скрыть не только смысл хранящейся или передаваемой информации, но и сам факт хранения или передачи закрытой информации. В основе всех методов стеганографии лежит маскирование закрытой информации среди открытых файлов, т.е. скрываются секретные данные, при этом создаются реалистичные данные, которые невозможно отличить от настоящих. Обработка мультимедийных файлов в информационных системах открыла практически неограниченные возможности перед стеганографией.

Графическая и звуковая информация представляются в числовом виде. Так, в графических объектах наименьший элемент изображения может кодироваться одним байтом. В младшие разряды определенных байтов изображения в соответствии с алгоритмом криптографического преобразования помещаются биты скрытого файла. Если правильно подобрать алгоритм преобразования и изображение, на фоне которого помещается скрытый файл, то человеческому глазу практически невозможно отличить полученное изображение от исходного. С помощью средств стеганографии могут маскироваться текст, изображение, речь, цифровая подпись, зашифрованное сообщение.

Скрытый файл также может быть зашифрован. Если кто-то случайно обнаружит скрытый файл, то зашифрованная информация будет воспринята как сбой в работе системы. Комплексное использование стеганографии и шифрования многократно повышает сложность решения задачи обнаружения и раскрытия конфиденциальной информации.

Содержанием процесса ***кодирование*** информации является замена исходного смысла сообщения (слов, предложений) кодами. В качестве кодов могут использоваться сочетания букв, цифр, знаков. При кодировании и обратном преобразовании используются специальные таблицы или словари. В информационных сетях кодирование исходного сообщения (или сигнала) программно-аппаратными средствами применяется для повышения достоверности передаваемой информации.

Часто кодирование и шифрование ошибочно принимают за одно и тоже, забыв о том, что для восстановления закодированного сообщения, достаточно знать правило замены, в то время как для расшифровки сообщения помимо знания правил шифрования, требуется ключ к шифру.

***Сжатие*** информации может быть отнесено к методам криптографического преобразования информации с определенными оговорками. Целью сжатия является сокращение объема информации. В то же время сжатая информация не может быть прочитана или использована без обратного преобразования. Учитывая доступность средств сжатия и обратного преобразования, эти методы нельзя рассматривать как надежные средства криптографического преобразования информации. Даже если держать в секрете алгоритмы, то они могут быть сравнительно легко раскрыты статистическими методами обработки. Поэтому сжатые файлы конфиденциальной информации подвергаются последующему шифрованию. Для сокращения времени передачи данных целесообразно совмещать процесс сжатия и шифрования информации.

Основным видом криптографического преобразования информации в компьютерных сетях является ***шифрование***. Под шифрованием понимается процесс преобразования открытой информации в зашифрованную информацию (шифртекст) или процесс обратного преобразования зашифрованной информации в открытую. Процесс преобразования открытой информации в закрытую получил название зашифрование, а процесс преобразования закрытой информации в открытую – расшифрование.

За многовековую историю использования шифрования информации человечеством изобретено множество методов шифрования или шифров. ***Методом шифрования (шифром)*** называется совокупность обратимых преобразований открытой информации в закрытую информацию в соответствии с алгоритмом шифрования. Большинство методов шифрования не выдержали проверку временем, а некоторые используются и до сих пор. Появление компьютеров и компьютерных сетей инициировало процесс разработки новых шифров, учитывающих возможности использования компьютерной техники как для зашифрования/расшифрования информации, так и для атак на шифр. ***Атака на шифр (криптоанализ, криптоатака)*** – это процесс расшифрования закрытой информации без знания ключа и, возможно, при отсутствии сведений об алгоритме шифрования.

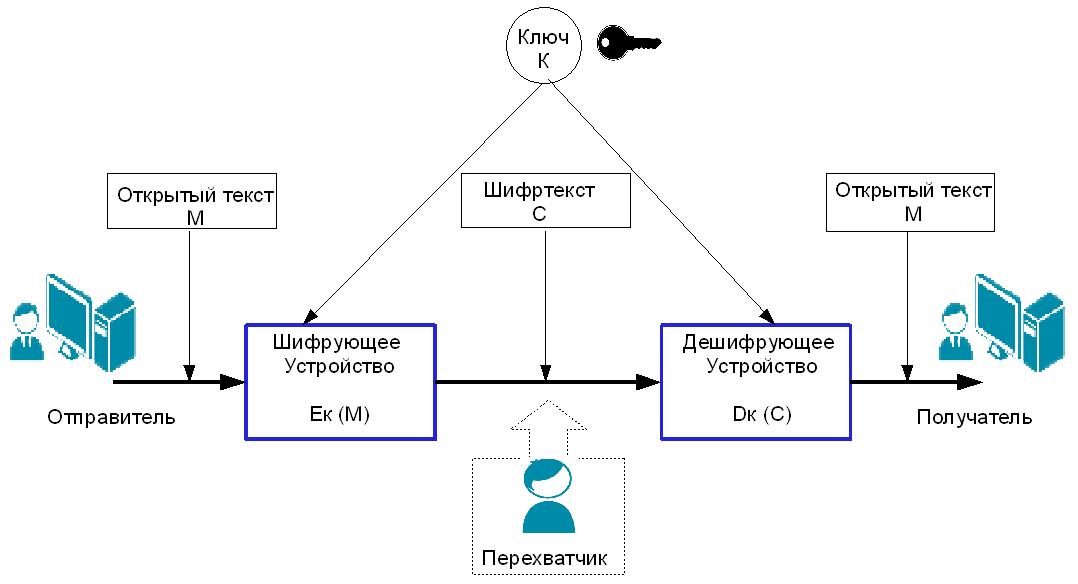
Современные методы шифрования должны отвечать следующим требованиям:

* стойкость шифра противостоять криптоанализу (криптостойкость) должна быть такой, чтобы вскрытие его могло быть осуществлено только путем решения задачи полного перебора ключей;
* криптостойкость обеспечивается не секретностью алгоритма шифрования, а секретностью ключа;
* шифртекст не должен существенно превосходить по объему исходную информацию;
* ошибки, возникающие при шифровании, не должны приводить к искажениям и потерям информации;
* время шифрования не должно быть большим;
* стоимость шифрования должна быть согласована со стоимостью закрываемой информации.

***Криптостойкость шифра*** является его основным показателем эффективности. Она измеряется временем или стоимостью средств, необходимых криптоаналитику для получения исходной информации по шифртексту, при условии, что ему неизвестен ключ.

Сохранить в секрете широко используемый алгоритм шифрования практически невозможно. Поэтому алгоритм не должен иметь скрытых слабых мест, которыми могли бы воспользоваться криптоаналитики. Если это условие выполняется, то криптостойкость шифра определяется длиной ключа, так как единственный путь вскрытия зашифрованной информации – перебор комбинаций ключа и выполнение алгоритма расшифрования. Таким образом, время и средства, затрачиваемые на криптоанализ, зависят от длины ключа и сложности алгоритма шифрования.

Работа простой криптосистемы проиллюстрирована на [рис. 2.2](https://www.intuit.ru/studies/courses/16655/1300/lecture/25505?page=2#image.2.2).



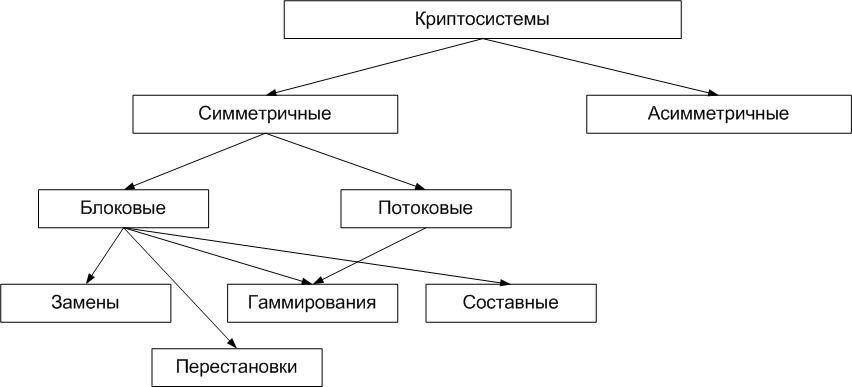
1. **Классификация современных криптосистем**

По характеру использования ключа все криптосистемы можно разделить на *симметричные* (одноключевые с секретным ключом) и *асимметричные* (несимметричные, с открытым ключом). В первом случае как для шифрования, так и для дешифрования применяется один и тот же ключ. Он является секретным и передается отправителем получателю по каналу связи, исключающем перехват. В асимметричных системах для шифрования и дешифрования используются разные ключи, связанные между собой некоторой математической зависимостью. Причем зависимость является такой, что из одного ключа вычислить другой ключ очень трудно за приемлемый промежуток времени.

Функции шифрования и дешифрования в зависимости от алгоритма могут быть одинаковыми или, что чаще всего, разными, причем процесс дешифрования является инверсией процесса шифрования.

Все многообразие симметричных криптографических систем) основывается на следующих базовых классах.

**Рисунок 2.8. Классификация криптосистем**



*Блочные шифры.*

Представляют собой семейство обратимых преобразований блоков (частей фиксированной длины) исходного текста. Фактически блочный шифр – это система подстановки блоков. После разбиения текста на блоки каждый блок шифруется отдельно независимо от его положения и входной последовательности.

Одним из наиболее распространенных способов задания блочных шифров является использование так называемых *сетей Фейстела* [13]. Сеть Фейстела представляет собой общий метод преобразования произвольной функции в перестановку на множестве блоков.

К алгоритмам блочного шифрования относятся: американский стандарт шифрования DES и его модификации, российский стандарт шифрования ГОСТ 28147–89, Rijndael, RC6, SAFFER+ и многие другие.

*Шифры замены (подстановки).*

Шифры замены (подстановки) – это наиболее простой вид преобразований, заключающийся в замене символов исходного текста на другие (того же алфавита) по более или менее сложному правилу. Подстановки различают моноалфавитные и многоалфавитные. В первом случае каждый символ исходного текста преобразуется в символ шифрованного текста по одному и тому же закону. При многоалфавитной подстановке закон меняется от символа к символу. К этому классу относится так называемая система с одноразовым ключом.

*Шифры перестановки.*

Перестановки – метод криптографического преобразования, заключающийся в перестановке местами символов исходного текста по некоторому правилу. Шифры перестановки в настоящее время не используются в чистом виде, так как их криптостойкость недостаточна.

*Гаммирование.*

Гаммирование – представляет собой преобразование, при котором символы исходного текста складываются по модулю, равному мощности алфавита, с символами псевдослучайной последовательности, вырабатываемой по некоторому правилу. В принципе, гаммирование нельзя выделить в отдельный класс криптопреобразований, так как эта псевдослучайная последовательность может вырабатываться, например, при помощи блочного шифра.

*Потоковые шифры.*

Потоковые шифры представляют собой разновидность гаммирования и преобразуют открытый текст в шифрованный последовательно, по одному биту. *Генератор ключевой последовательности,* иногда называемый *генератором бегущего ключа*, выдает последовательность бит . Эта ключевая последовательность складывается по модулю *2* с последовательностью бит исходного текста для получения шифрованного текста . На приемной стороне текст складывается по модулю *2* с идентичной ключевой последовательностью для получения исходного текста. Такое преобразование называется гаммированием с помощью операции XOR. Однако при потоковом шифровании для повышения криптостойкости генератор ключевой последовательности «завязывается» на текущее состояние кодируемого символа. То есть значения, выдаваемые генератором, зависят не только от ключа, но и от номера шифруемого бита и входной последовательности.

*К известным потоковым шифрам можно отнести RC4, SEAL, WAKE, шифры Маурера и Диффи, большинство из которых реализовано на генераторах ключевых последовательностей, использующих сдвиговые регистры [13,14].*

1. **Шифры 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.

1. **Шифры 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"](http://www.intuit.ru/department/security/bcript/1/) 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](http://www.intuit.ru/department/security/bcript/2/2.html#image.2.3).



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

В таблице на [рис. 2.3](http://www.intuit.ru/department/security/bcript/2/2.html#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](http://www.intuit.ru/department/security/bcript/2/2.html#image.2.4)). Для этого берем первую букву исходного сообщения "В". В таблице на [рис. 2.3](http://www.intuit.ru/department/security/bcript/2/2.html#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](http://www.intuit.ru/department/security/bcript/2/2.html#image.2.4)).

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



*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.

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

1. БОЛЬШОЙ СЕКРЕТ

может быть зашифровано следующим образом:

1. 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](http://www.intuit.ru/department/security/bcript/2/3.html#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 |

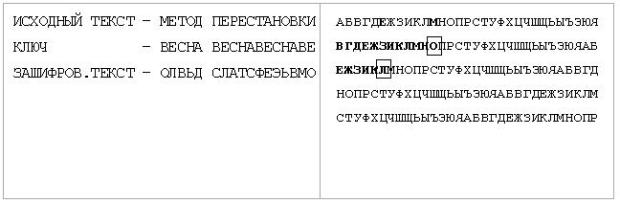
Для шифрования текста выбирают ключ, представляющий собой некоторое слово или набор символов исходного алфавита. Далее из полной матрицы выписывают подматрицу шифрования, включающую первую строку и строки матрицы, начальными буквами которых являются последовательно буквы ключа (например, если выбрать ключ "весна", то таблица шифрования будет такой, как на [таблица 2.7](http://www.intuit.ru/department/security/bcript/2/3.html#table.2.7)).

Таблица 2.7. Первый этап шифрования – составление подматрицы шифрования

|  |
| --- |
| 1. АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ |
| 1. ВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБ |
| 1. ЕЖЗИКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГД |
| 1. НОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИКЛМ |
| 1. СТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИКЛМНОПР |

В процессе шифрования ([рис. 2.5](http://www.intuit.ru/department/security/bcript/2/3.html#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](http://www.intuit.ru/department/security/bcript/2/3.html#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](http://www.intuit.ru/department/security/bcript/2/3.html#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

*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:

1. 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):

1. 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 , то есть можно записать:



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:



Сложим по модулю два двоичные числа 1110 и 1100:

1. Исходное число 1 1 1 0
2. Гамма 1 1 0 0
3. 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:

1. Зашифрованное число 0 0 1 0
2. Гамма 1 1 0 0
3. 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](http://www.intuit.ru/department/security/bcript/2/4.html#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](http://www.intuit.ru/department/security/bcript/2/4.html#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. 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.

**10. Матричное шифрование**

Для кодирования текста на русском языке занумеруем все буквы по месту их расположения в алфавите – от 1 до 33, добавив 34-ю пробел.

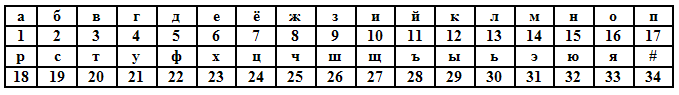


Рисунок 4.

Возьмем какое-нибудь простое предложение, например, «шёл снег», и каждую букву заменим соответствующей цифрой. Получим последовательность: 26,7, 13, 34, 19, 15, 6, 4.

 Построим из этой последовательности две таблички 2х2:

http://xn--i1abbnckbmcl9fb.xn--p1ai/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/503093/img5.gif

Такие таблички из четырех чисел называются матрицей.

Зашифруем эту последовательность с помощью еще одной матрицы http://xn--i1abbnckbmcl9fb.xn--p1ai/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/503093/img6.gif- кодирующей – по следующему правилу:

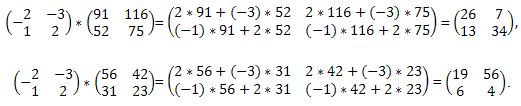
http://xn--i1abbnckbmcl9fb.xn--p1ai/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/503093/img7.gif

http://xn--i1abbnckbmcl9fb.xn--p1ai/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/503093/img8.gif

Такой способ шифрования и называют матричным. Ваш адресат получит текст: 91, 116, 52, 75, 56, 42, 31, 23.

А как же он его расшифрует? Оказывается, и это нетрудно: он должен взять декодирующую матрицу http://xn--i1abbnckbmcl9fb.xn--p1ai/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/503093/img9.gif

и проделать с полученным текстом то же самое, что делали мы с исходным текстом.



После замены матриц на последовательность 26,7, 13, 34, 19, 15, 6, 4, а затем - чисел на буквы дешифровальщик получит исходный текст «шёл снег».

Ясно, что никто посторонний, не знающий ни кодирующей, ни декодирующей матрицы, получить этот текст не сможет.

Однако матричный способ шифрования не смог бы существовать, если бы в качестве кодирующей можно было брать только матрицу http://xn--i1abbnckbmcl9fb.xn--p1ai/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/503093/img11.gif

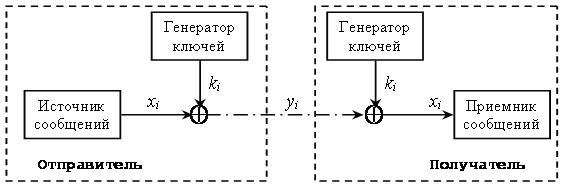
На самом деле таких матриц бесконечно много, придумывать их очень легко. И вообще вы можете менять свою систему «тайнописи» каждый день. Для этого нужно знать очень немного – уметь любые две матрицы «перемножать», т.е. по определенному правилу составлять из них третью.

http://xn--i1abbnckbmcl9fb.xn--p1ai/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/503093/img12.gif– правило умножения матриц.

**11. Потоковое шифрование**

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](http://www.intuit.ru/department/security/bcript/7/#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:



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



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.

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.

Структуру генератора ключевой последовательности можно представить в виде конечного автомата с памятью, состоящего из трех блоков: блока памяти, хранящего информацию о состоянии генератора, выходной функции, генерирующей бит ключевой последовательности в зависимости от состояния, и функции переходов, задающей новое состояние, в которое перейдет генератор на следующем шаге.

В настоящее время насчитывается несколько тысяч различных вариантов генераторов псевдослучайных чисел.

Рассмотрим основные методы получения псевдослучайных последовательностей, которые наиболее подходят для компьютерной криптографии.

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 задач

**12.Обзор программных методов получения псевдослучайных последовательностей**

В большинстве алгоритмов шифрования, особенно потоковых шифрах, используются генераторы ключевой последовательности. Генератор ключевой последовательности выдает поток битов, который выглядит случайными, но в действительности является детерминированным и может быть в точности воспроизведен на стороне получателя. Чем больше генерируемый поток похож на случайный, тем больше времени потребуется криптоаналитику для взлома шифра.

Однако если каждый раз при включении генератор будет выдавать одну и ту же последовательность, то взлом криптосистемы будет тривиальной задачей. Например, в случае использования потокового шифрования, перехватив два шифрованных текста, злоумышленник может сложить их по модулю 2 и получить два исходных текста, сложенных также по модулю 2. Такую систему раскрыть очень просто. Если же в руках противника окажется пара «исходный текст – шифрованный текст», то задача вообще становится тривиальной.

Поэтому все генераторы случайных последовательностей имеют зависимость от ключа. В этом случае простой криптоанализ будет невозможным.

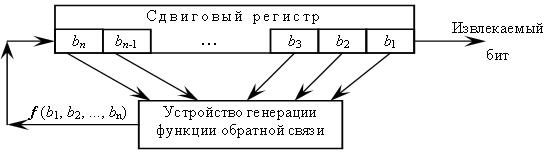
Структуру генератора ключевой последовательности можно представить в виде конечного автомата с памятью, состоящего из трех блоков: блока памяти, хранящего информацию о состоянии генератора, выходной функции, генерирующей бит ключевой последовательности в зависимости от состояния, и функции переходов, задающей новое состояние, в которое перейдет генератор на следующем шаге.

В настоящее время насчитывается несколько тысяч различных вариантов генераторов псевдослучайных чисел.

Рассмотрим основные методы получения псевдослучайных последовательностей, которые наиболее подходят для компьютерной криптографии.

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](http://www.intuit.ru/department/security/bcript/8/#image.8.1)).



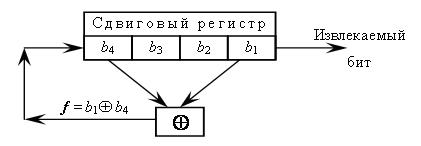
**Рис. 8.1.**  Сдвиговый регистр с обратной связью

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.

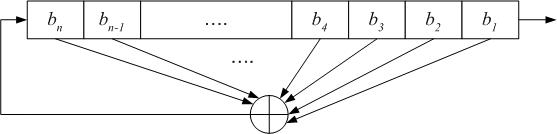
Для того, чтобы стало понятнее, как работает сдвиговый регистр с обратной связью, рассмотрим 4-битовый LFSR с отводом от первого и четвертого разрядов, представленный на [рис. 8.2](http://www.intuit.ru/department/security/bcript/8/#image.8.2).



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

Простейшим видом сдвигового регистра с обратной связью является *линейный сдвиговый регистр с обратной связью (LFSR – Left Feedback Shift Register)*. Обратная связь представляет собой просто XOR некоторых битов регистра, перечень этих битов называется *отводной последовательностью*.

**Рисунок 2.10. Сдвиговый регистр с линейной обратной связью (LFSR)**



*n*-битовый LFSR может находиться в одном из *2n-1* внутренних состояний. Это означает, что теоретически такой регистр может генерировать псевдослучайную последовательность с периодом *2n-1* битов. Число внутренних состояний и период равны, потому что заполнение регистра нулями приведет к тому, что он будет выдавать бесконечную последовательность нулей, что абсолютно бесполезно. Только при определенных отводных последовательностях LFSR циклически пройдет через все *2n-1* внутренних состояний. Такие LFSR называются *LFSR с максимальным периодом*.

Для того чтобы конкретный LFSR имел максимальный период, многочлен, образованный из отводной последовательности и константы *1,* должен быть примитивным по модулю *2*.

Вычисление примитивности многочлена – достаточно сложная математическая задача. Поэтому существуют готовые таблицы, в которых приведены номера отводных последовательностей, обеспечивающих максимальный период генератора. Например, для 32-битового сдвигового регистра можно найти такую запись: *(32,7,5,3,2,1,0)*. Это означает, что для генерации нового бита необходимо с помощью функции XOR просуммировать тридцать второй, седьмой, пятый, третий, второй и первый биты.

Код для такого LFSR на языке С++ будет таким:

int LFSR (void)

{

static unsigned long ShiftRegister = 1;

// Любое значение, кроме нуля

ShiftRegister = ((((ShiftRegister >> 31)

^ (ShiftRegister >> 6)

^ (ShiftRegister >> 4)

^ (ShiftRegister >> 2)

^ (ShiftRegister >> 1)

^ ShiftRegister)& 0x00000001) <<31)

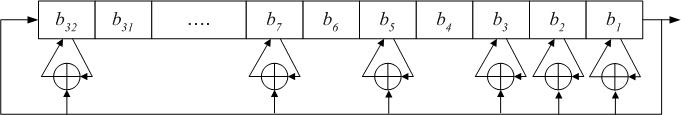
| (ShiftRegister >> 1);

return ShiftRegister & 0x00000001; }

Программные реализации LFSR генераторов достаточно медленны и быстрее работают, если они написаны на ассемблере, а не на языке С. Одним из решений является использование параллельно 16-ти LFSR (или 32 в зависимости от длины слова в архитектуре конкретного компьютера). В такой схеме используется массив слов, размер которого равен длине LFSR, а каждый бит слова массива относится к своему LFSR. При условии, что используются одинаковые номера отводных последовательностей, то это может дать заметный выигрыш в производительности.

Схему обратной связи также можно модифицировать. При этом генератор не будет обладать большей криптостойкостью, но его будет легче реализовать программно. Вместо использования для генерации нового крайнего левого бита битов отводной последовательности выполняется XOR каждого бита отводной последовательности с выходом генератора и замена его результатом этого действия, затем результат генератора становится новым левым крайним битом .

**Рисунок 2.11. LFSR Галуа**



Эту модификацию называют *конфигурацией Галуа*. На языке С это выглядит следующим образом:

static unsigned long ShiftRegister = 1;

void seed\_LFSR (unsigned long seed)

{

if (seed ==0)

seed = 1;

ShiftRegister = seed;

}

int Galua\_LFSR (void)

{

if (ShiftRegister & 0x00000001) {

ShiftRegister = (ShiftRegister ^ mask >> 1) | 0x8000000;

return 1;

} else {

ShiftRegister >>= 1;

return 0;

}

}

Выигрыш состоит том, что все XOR выполняются за одну операцию. Эта схема также может быть распараллелена.

Сами по себе LFSR являются хорошими генераторами псевдослучайных последовательностей, но они обладают некоторыми нежелательными неслучайными свойствами. Последовательные биты линейны, что делает их бесполезными для шифрования. Для LFSR длины *n* внутреннее состояние представляет собой предыдущие *n* выходных битов генератора. Даже если схема обратной связи хранится в секрете, то она может быть определена по *2n* выходным битам генератора при помощи специальных алгоритмов. Кроме того, большие случайные числа, генерируемые с использованием идущих подряд битов этой последовательности, сильно коррелированны и для некоторых типов приложений не являются случайными. Несмотря на это, LFSR часто используются для создания алгоритмов шифрования. Для этого используются несколько LFSR, обычно с различными длинами и номерами отводных последовательностей. Ключ является начальным состоянием регистров. Каждый раз, когда необходим новый бит, все регистры сдвигаются. Эта операция называется *тактированием*. Бит выхода представляет собой функцию, желательно нелинейную, некоторых битов LFSR. Эта функция называется *комбинирующей*, а генератор в целом – *комбинирующим генератором*. Многие из таких генераторов безопасны до сих пор.

**13. Генератор Парка-Миллера. Генератор Геффе**

Самая простая последовательность, которую можно предложить для реализации генератора равномерного распределения:

I(j+1)=a\*I(j)(mod m)

при соответствующем выборе констант. Константы были предложены Park и Miller:

a=75=16807, m=231-1=2147483647

и протестированы в исследованиях Lewis, Goodman, Miller (1969).

Прямое приложение этого метода возможно на языках ассемблера, но языки высокого уровня могут при этом зафиксировать переполнение. Для обхода этого Scharge предложил метод частичной факторизации модуля. Модуль разлагается в выражение:

m=a\*q+r

Если r<q и 0<z<m-1, то при этом величины a\*(z mod q) и r\*[z/q] всегда лежат в интервале 0,...,m-1. Для умножения (a\*z)(mod m) при этом используется алгоритм:

t = a(z mod q)-r[z/q]

если t<0, то t += m.

(a\*z)(mod m)=t.

В случае констант Парка-Миллера можно использовать q=12773 и r=2836.

/\* Minimal portable random generator by Park and Miller \*/

/\* Lewis-Goodman-Miller constants \*/

#define IA 16807

#define IM 2147483647

#define AM (1./IM)

/\* Scharge constants \*/

#define IQ 12773

#define IR 2836

/\* Special mask to be explained below \*/

#define MASK 123456789

static long dummy;

/\* initial seed, for all the generators here \*/

void Seed(long dum) {dummy=dum;}

/\* returns random uniformly distributed between 0 and 1 \*/

float unirand0(void) {

long k;

float ans;

dummy^=MASK; /\* avoid dummy==0 \*/

k=dummy/IQ;

if((dummy=IA\*(dummy-k\*IQ)-IR\*k)<0) dummy+=IM;

ans=AM\*dummy;

dummy^=MASK; /\* restore unmasked dummy \*/

return(ans);

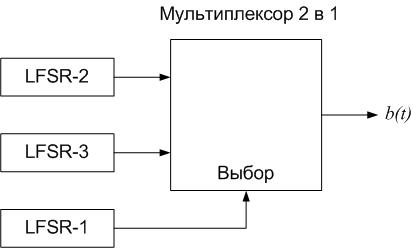
}

Использование маски связано с тем, что специфика алгоритма не позволяет устанавливать счетчик в нуль. Но, как показывает опыт, большинство пользователей счетчиков делают именно так. Маска гарантирует, что установленный счетчик не будет нулем. Если вы очень уверены в том, что человек не допустит подобной ошибки после вашего предупреждения, то можете убрать из программы все инструкции, связанные с маской.

*Генератор Геффа*

Одним из комбинирующих генераторов является генератор Геффа. В нем используются три LFSR, объединенные нелинейным способом. LFSR-2 и LFSR-3 являются входами мультиплексора *(рабочие регистры)*, а третий управляет входом мультиплексора. Если длины LFSR равны *n1*, *n2*, *n3* соответственно, то линейная сложность генератора равна .

**Рисунок 2.12. Генератор Геффа**



Период такого генератора будет равен наименьшему общему делителю периодов трех генераторов. При условии, что размеры регистров взаимно просты, то период этого генератора будет равен произведению периодов трех LFSR. В обобщенной схеме генератора Геффа используются несколько рабочих LFSR

**14. Линейный конгруэтный генератор. Аддитивный генератор**

**Линейный конгруэнтный метод** — один из методов генерации псевдослучайных чисел. Применяется в простых случаях и не обладает криптографической стойкостью.

В большинстве языков программирования именно этот метод используется в стандартной функции получения случайных чисел. Впервые этот метод был предложен Лехмером в 1949 году. Выбирается 4 числа:

Модуль m (m>0);

Множитель a (0<=a<m);

Приращение c (0<=c<m);

Начальное значение X0 (0<= X0<m)

Последовательность получается с использование следующей рекуррентной формулы: Xn+1=(a\* Xn+c) mod m.

Этот метод даёт действительно хорошие псевдослучайные числа, но, если взять числа m,a,c произвольно, то результат нас скорее всего разочарует. При m=7, X0=1, a=2, c=4 получится следующая последовательность: 1,6,2,1,6,2,1,…

Очевидно, что эта последовательность не совсем подходит под определение случайной. Тем не менее, этот провал позволил нам сделать два важных вывода:

Числа m,a,c, X0 не должны быть случайными;

Линейный конгруэнтный метод даёт нам повторяющиеся последовательности.

На самом деле любая функция, отображающая конечное множество X в X, будет давать циклически повторяемый значения. Т.о. наша задача состоит в том, чтобы максимально удлинить уникальную часть последовательности (кстати, очевидно, что длина уникальной части не может быть больше m).

Не вдаваясь в подробности доказательств, скажем, что период последовательности будет равен m только при выполнении следующих трех условий:

Числа c и m взаимно простые;

a-1 кратно p для каждого простого p, являющегося делителем m;

Если m кратно 4, то и a-1 должно быть кратно 4.

В завершении рассказа о линейном конгруэнтном методе надо сказать, что последовательности, получаемые с его помощью, хоть и являются в достаточном смысле случайными, тем не менее не являются криптографически стойкими. Т.к. зная 4 подряд идущих числа, криптоаналитик может составить систему уравнений, из которых можно найти a,c,m.

**Аддитивный генератор случайных чисел**

Генератор, формирующий очередное случайное число в соответствии с отношением (3), называется аддитивным:

Xn+1=(Xn+Xn-k) mod m. (3)

В трехтомнике Кнута обсуждаются подобные генераторы и рекомендован следующий вариант формулы (3):

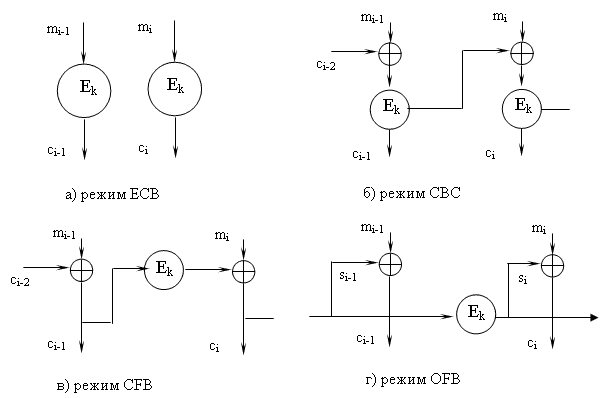
Xn+1=(Xn-24+Xn-55) mod m.(4)

Здесь n > 55, m=21, Хо, ..., Х54 — произвольные числа, среди которых есть и нечетные. При этих условиях длина периода последовательности равна 21-1 (255-1).   
Для генерации первых 55 случайных чисел можно использовать один из рассмотренных выше генераторов. Возьмем датчик линейной (смешанной) конгруэнтной последовательности случайных чисел (с > 0).

1. **Режимы использования блочных шифров. Объединения блочных шифров**

Применение блочного шифра для нужд практического кодирования информации возможно в одном из четырех основных режимов шифрования (рис. 2.8):

1. режим электронной кодовой книги (Electronic Code Book, ECB);
2. режим сцепления блоков шифра (Cipher Block Changing, CBC);
3. режим обратной связи по шифротексту (Electronic Feedback, CFB);
4. режим обратной связи по выходу (Output Feedback, OFB).

  
Рис. 2.8. Режимы шифрования блочных шифров

В режиме ECB шифрование/дешифрование i-го блока открытого текста/шифротекста выполняется независимо от остальных блоков:

ci = Ek(mi), mi = Dk(ci).

Недостатком данного режима шифрования является то, что одинаковые блоки входного текста будут кодироваться в одинаковые блоки шифротекста, что дает возможность злоумышленнику, во-первых, делать предположения о характере информации в открытом тексте, а во-вторых, подменить один или несколько блоков шифротекста. Достоинством режима можно назвать простоту реализации, а также возможность распараллеливания процедуры шифрации.

Режим CBC предполагает следующие алгоритмы шифрации/дешифрации:

ci = Ek(mi ⊕ ci-1),mi = Dk(ci) ⊕ ci-1

В режиме CBC каждый блок открытого текста складывается с блоком шифротекста, полученным на предыдущем этапе. Таким образом, происходит сцепление блоков друг с другом и независимая манипуляция с каждым из них невозможна, а одинаковые входные блоки будут давать на выходе разные блоки. Однако, задача распараллеливания процедуры кодирования в этом режиме затруднена. Дополнительным параметром процедур шифрования/дешифрования является параметр c0.

В режиме CFB также происходит «маскировка» блока открытого текста уже зашифованными блоками:

ci = mi ⊕ Ek(ci-1), mi = Dk(ci-1) ⊕ ci

По своим возможностям данный режим похож на режим CBC, но если длина сообщения не кратна размеру блока шифра, то в режиме CBC необходимо дополнять последний блок дополнительными битами и сообщать на принимающую сторону истинный размер сообщения, а режим CFB позволяет сформировать шифротекст того же размера, что и исходное сообщение.

В режиме OFB исходное сообщение вообще не подвергается криптопреобразованию, оно складывается с шифруемыми на секретном ключе блоками si (s0 является задаваемым несекретным параметром режима):

ci = mi ⊕ si, mi = ci ⊕ si, si = Ek(si-1)

В этом режиме, как и в режиме ECB, ошибки, которые могут возникнуть при передаче шифротекста по каналам связи, локализуются в блоке, не распространяясь на соседние, причем в режиме OFB ошибочными будут только биты, подвергшиеся изменению (в ECB изменится весь блок). Это дает возможность злоумышленнику незаметно для принимающей стороны подменить блок шифротекста. Возможности распараллеливания процедур шифрации/дешифрации затруднены.

Объединения блочных шифров

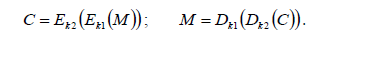
*Объединение блочных шифров*

Существует множество способов объединять блочные алгоритмы шифрования для получения новых алгоритмов. Стимулом создания подобных схем является желание повысить безопасность, не создавая новый алгоритм шифрования.

Одним из способов объединения является *многократное шифрование* – для шифрования одного и того же блока открытого текста алгоритм шифрования используется несколько раз с разными ключами. *Шифрование каскадом* похоже на многократное шифрование, но использует различные алгоритмы. Существуют и другие методы [14].

Повторное шифрование блока открытого текста одним и тем же ключом с помощью одного и того же алгоритма неразумно. Необходимо выбирать различные ключи, причем они должны быть независимыми.

*Двойное шифрование*. Наивным способом повысить безопасность алгоритма является шифрование блока дважды с двумя различными ключами. Сначала блок шифруется первым ключом, а затем получившийся шифротекст шифруется вторым ключом. Дешифрование является обратным процессом.



*Тройное шифрование с двумя ключами*. В более интересном методе, предложенном Тачменом, блок обрабатывается три раза с помощью двух ключей: первым ключом, вторым ключом и снова первым ключом. Предлагается чтобы отправитель сначала шифровал первым ключом, затем дешифровал вторым и снова шифровал первым. Получатель должен расшифровать первым ключом, затем зашифровать вторым и снова расшифровать первым.



Иногда такой режим называют *шифрование-дешифрование-шифрование* (encrypt-decrypt-encrypt, EDE). Если блочный алгоритм использует *n*-битный ключ, то длина ключа описанной схемы составляет *2n* бит.

*Тройное шифрование с тремя ключами*. В такой схеме обычно используется модификация алгоритма EDE, только на третьем шаге используется новый независимый ключ.



Еще одним вариантом объединения различных блочных шифров является *использование строки случайных бит*. Для двух алгоритмов и двух независимых ключей выполняется такая последовательность действий:

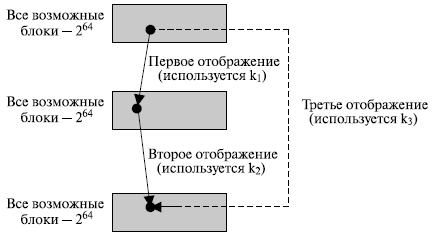
1. Генерируется строка случайных битов *R* того же размера, что и сообщение *M*.

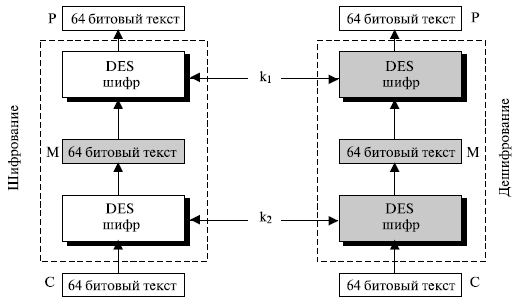
2. *R* шифруется первым алгоритмом.

3. шифруется вторым алгоритмом. R M 

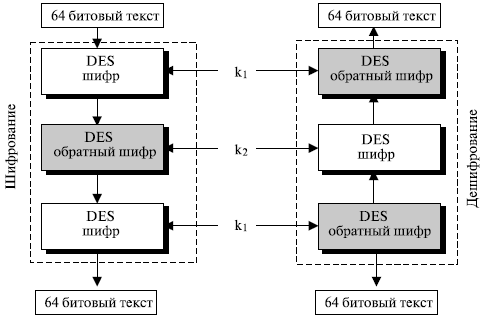
4. Шифротекст сообщения является объединением результатов этапов 2 и 3.

При условии, что строка случайных бит действительно случайна, то этот метод обладает хорошей криптостойкостью. Так как для восстановления исходного сообщения необходимо знать и первый и второй алгоритм шифрования, то криптоаналитику придется выполнять двойную работу. Недостатком является удвоение размера шифротекста по сравнению с открытым текстом. *обеспечение* и *аппаратные средства*.





**Рис. 8.14.**Атака сведения к середине в двукратном DES

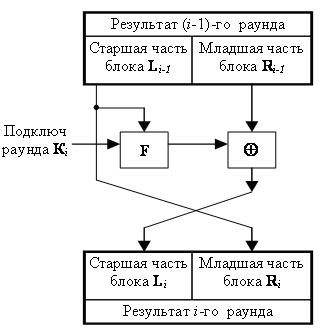
  
**Рис. 8.16.**Трехкратный DES с двумя ключами

1. **Сеть Фейстола. Алгоритмы, использующие эту сеть.**

На [рис. 3.1](http://www.intuit.ru/department/security/bcript/3/2.html#image.3.1) была представлена общая структура блочного алгоритма шифрования. Понятно, что само преобразование данных выполняется в раундах или шагах шифрования. Какие же действия надо выполнить в одном раунде, чтобы в результате выполнения всего алгоритма получить надежно зашифрованные данные?

Большой вклад в исследования принципов разработки блочных шифров внес американский ученый Х. Фейштель (Horst Feistel). Он, в частности, принимал участие в разработке системы шифрования "Люцифер" фирмы IBM. Фейштель предложил структуру, называемую в настоящее время сетью Фейштеля. Сети Фейштеля получили широкое распространение, так как, с одной стороны, они удовлетворяют всем требованиям к алгоритмам симметричного шифрования, а с другой стороны, достаточно просты и удобны в использовании.

Раунд, организованный по сети Фейштеля имеет следующую структуру. Входной блок делится на несколько частей равной длины. Эти части блока называются ветвями. Так, например, если блок имеет длину 64 бита, используются две ветви по 32 бита каждая. Ветви обрабатываются по отдельности, после чего осуществляется циклический сдвиг всех ветвей влево. В случае двух ветвей каждый раунд имеет структуру, показанную на [рис. 3.2](http://www.intuit.ru/department/security/bcript/3/3.html#image.3.2)



**Рис. 3.2.**  i-й раунд сети Фейштеля

Функция F называется образующей. Каждый раунд состоит из вычисления функции F для одной ветви и побитового выполнения операции "сумма по модулю 2" результата F с другой ветвью. После этого ветви меняются местами. Число раундов может быть различным для разных алгоритмов. В некоторых алгоритмах рекомендуется от 8 до 32 раундов, в других – больше. В целом увеличение количества раундов увеличивает криптостойкость алгоритма. Возможно, эта особенность и повлияла на столь активное распространение сети Фейштеля, так как для большей криптостойкости достаточно просто увеличить количество раундов, не изменяя сам алгоритм. В последнее время количество раундов не фиксируется, а лишь указываются рекомендуемые пределы.

В последнее время все чаще используются различные разновидности сети Фейштеля для 128-битного блока с четырьмя ветвями. Увеличение количества ветвей, а не размерности каждой ветви связано с тем, что наиболее популярными до сих пор остаются процессоры с 32-разрядными словами, следовательно, оперировать 32-разрядными словами эффективнее, чем с 64-разрядными.

В блочных алгоритмах, построенных на основе сети Фейштеля, основной операцией является вычисление образующей функции F. Эта функция использует подключ раунда и одну ветвь входного блока для вычисления результата. Именно тем, как определяется функция F, системы шифрования и отличаются друг от друга. В некоторых алгоритмах введены также начальные преобразования входного блока данных, придающие некоторую "случайность" входному тексту (это называется рандомизацией данных.) Рандомизация производится для того, чтобы уменьшить естественную избыточность входного сообщения.

**17. Системы шифрования с открытым ключом. Достоинства и недостатки**

Слабым местом любых симметричных систем является ключ, который необходимо передавать по абсолютно защищенному каналу связи. Если ключ станет известным, то любая симметричная система будет обречена на провал. Для передачи ключа необходимо определить *механизм распространения* ключей, который позволил бы разрешить задачу безопасной передачи ключевой информации.

Существуют криптосистемы, в которых для шифрования и дешифрования используются различные ключи (один ключ называют *открытым* ключом, другой *закрытым*) и необходимость шифровать канал передачи ключевой информации отпадает. Такие криптосистемы называются *системами с открытым ключом*.

*Механизм распространения открытых ключей*

В криптосистемах с открытым ключом в алгоритмах шифрования и дешифрования используются различные ключи. Один служит для шифрования – он называется *открытым ключом*, второй для дешифрования – *закрытый ключ*. Между открытым и закрытым ключом существует определенная математическая зависимость, но она такова, что из одного ключа невозможно получить второй ключ за приемлемое время. Основной принцип систем с открытым ключом основывается на применении односторонних (необратимых) преобразований. Чаще всего используются следующие необратимые преобразования:

1. Разложение произведения больших простых чисел на сомножители.

2. Вычисление логарифма в конечном поле.

3. Вычисление корней алгебраических уравнений.

Такие системы могут быть использованы по следующим назначениям:

1. Как самостоятельные средства защиты передаваемых и хранимых данных.

2. Как средства для распределения ключей.

3. Как средства аутентификации пользователей.

Распространение ключей происходит так (Рисунок 2.2221):

1. Получатель сообщения производит вычисление пары ключей *Кш* и *Кд* . Ключ *Кд* считается закрытым, *Кш –* открытым.

2. Закрытый ключ *Кд* получатель оставляет у себя, а открытый ключ *Кш* пересылает отправителю.

3. Пользуясь открытым ключом *Кш* любой абонент может зашифровать текст и отослать его получателю. Расшифровать зашифрованное сообщение может только тот, у кого есть закрытый ключ, а так как он

никуда не передается, то теоретически такой ключ может быть только у получателя.

4. Пользуясь закрытым ключом *Кд,* получатель расшифровывает полученное сообщение.

Возможна и другая схема распространения ключей (Рисунок 2.22):

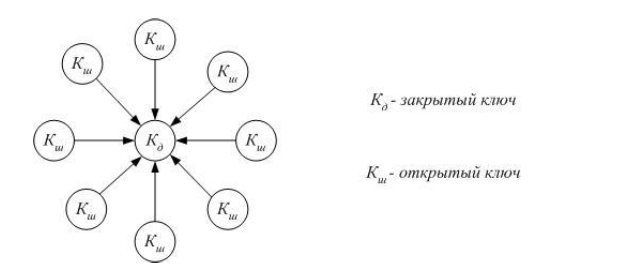
1. Отправитель сообщения производит вычисление пары ключей *Кш* и *Кд* . Ключ *Кш* считается закрытым, *Кд –* открытым.

2. Закрытый ключ *Кш* отправитель оставляет у себя, а открытый ключ *Кд* пересылает получателю.

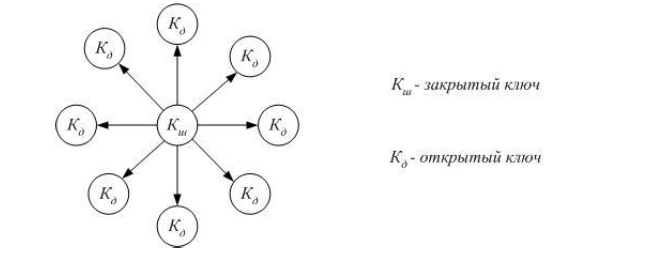
3. Пользуясь закрытым ключом *Кш*, отправитель производит шифрование сообщения и отправляет зашифрованный текст получателю.

4. Пользуясь открытым ключом *Кд,* любой абонент может расшифровать зашифрованный текст. Причем ключ дешифрования *Кд* может расшифровать только то сообщение, которое было зашифровано закрытым ключом *Кш*.

**Рисунок 2.21. Первая схема распространения открытых ключей**



**Рисунок 2.22. Вторая схема распространения открытых ключей**



1. **Механизм кодирования и декодирования по алгоритму RSA**

RSA – аббревиатура от сокращения фамилий 3-х ученых, разработавших

эту систему – Рональда Райвеста, Ади Шамира и Леонарда Адельмана.

Алгоритм был предложен в 1977 году. Он основывается на том, что нахождение

больших простых чисел осуществляется сравнительно легко, но разложение на

множители произведения двух таких чисел требует значительных

вычислительных затрат.

*Простым числом* называется число из натурального ряда, большее 1,

которое делится без остатка только на 1 и само на себя.

Аксиома. Число простых чисел бесконечно.

Число *x* называется *простым относительно y*, если его нельзя разложить

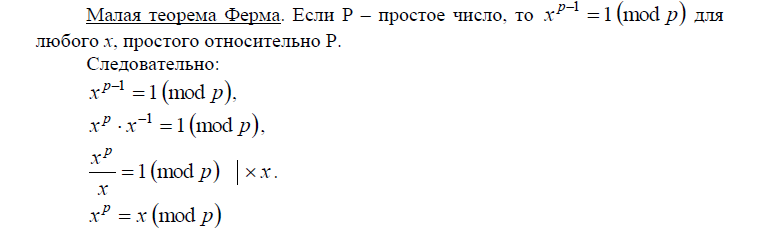
на сомножители, на которые число *y* не делится без остатка. Например число 4

является простым относительно 15, а число 6 не является простым

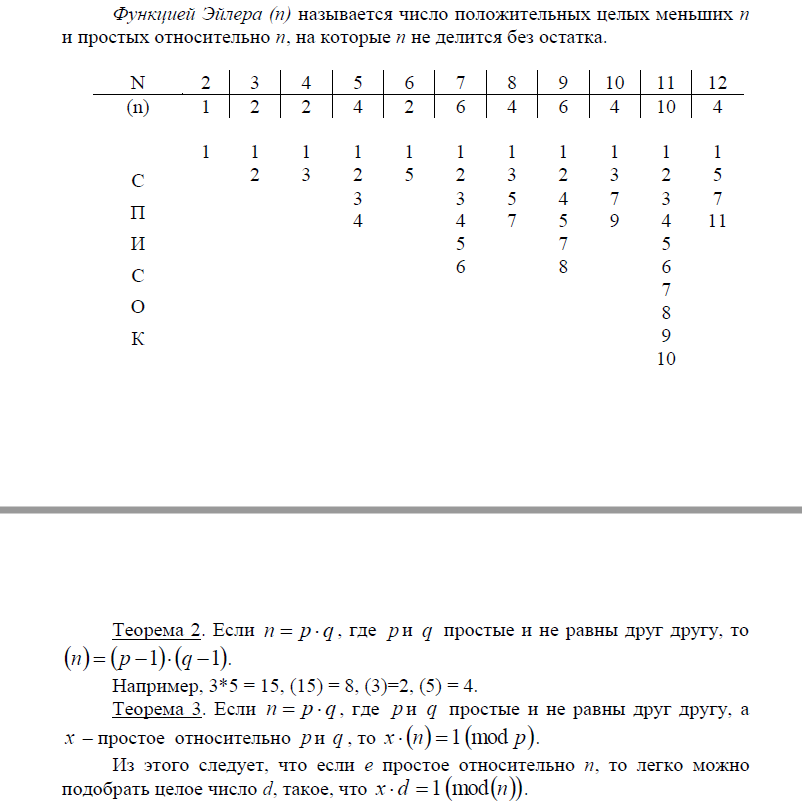
относительно числа 15.

Числа *x* и *y* называются *взаимно простыми*, если наименьший общий

делитель *НОД (x,y) = 1*.

*Функцией Эйлера (n)* называется число положительных целых меньших *n*

и простых относительно *n*, на которые *n* не делится без остатка.

*Алгоритм генерации ключей.*

1. Отправитель выбирает два очень больших простых числа *P* и *Q* и

вычисляет два произведения *N=P\*Q* и *M=(P-1)\*(Q-1)*.

2. Затем он выбирает случайное число *D*, взаимно простое с *M*, и вычисляет

*E*, удовлетворяющее условию *D\*E=1 mod M*.

3. После этого он публикует *D* и *N* как свой открытый ключ

шифрования, сохраняя *E* (в паре с *N*) как закрытый ключ.

4. Теперь, чтобы зашифровать данные по известному ключу *{D,N}*,

необходимо сделать следующее:

a. разбить шифруемый текст на блоки, каждый из которых может быть

представлен в виде числа *S(i)=0,1, ... ,N-1*;

b. зашифровать текст, рассматриваемый как последовательность

чисел *S(i)* по формуле *S'(i)=(S(i)D) mod N*.

5. Чтобы расшифровать эти данные, используют секретный ключ *{E,N}*,

необходимо выполнить следующие вычисления:

*S''(i)=S(i)=(S'(i)E) mod N.*

6. В результате будет получено множество чисел *S''(i)*, которые

представляют собой исходный текст.

1. **Механизм кодирования и раскодирования Эль-Гамаля.**

*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 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

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

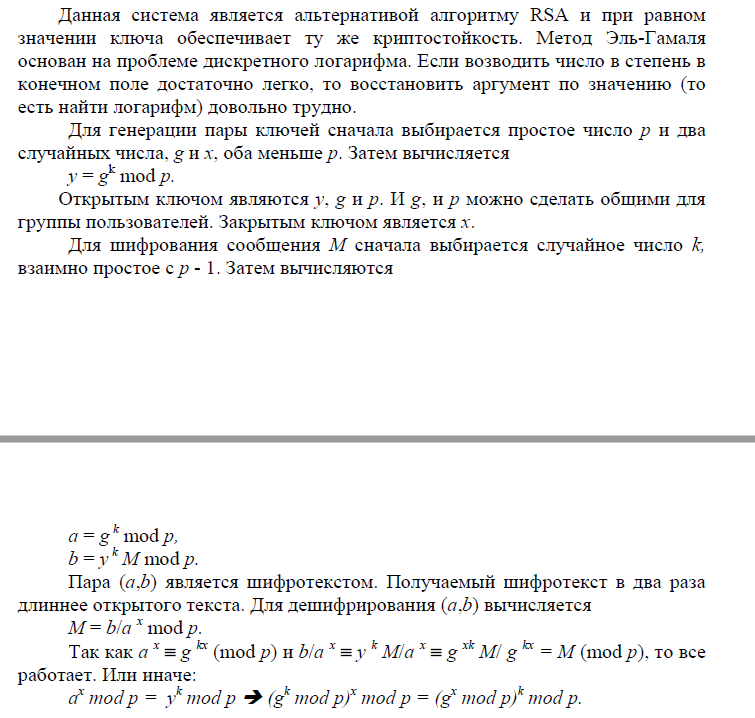
где Y2 – открытый ключ пользователя 2. Число k держится в секрете.

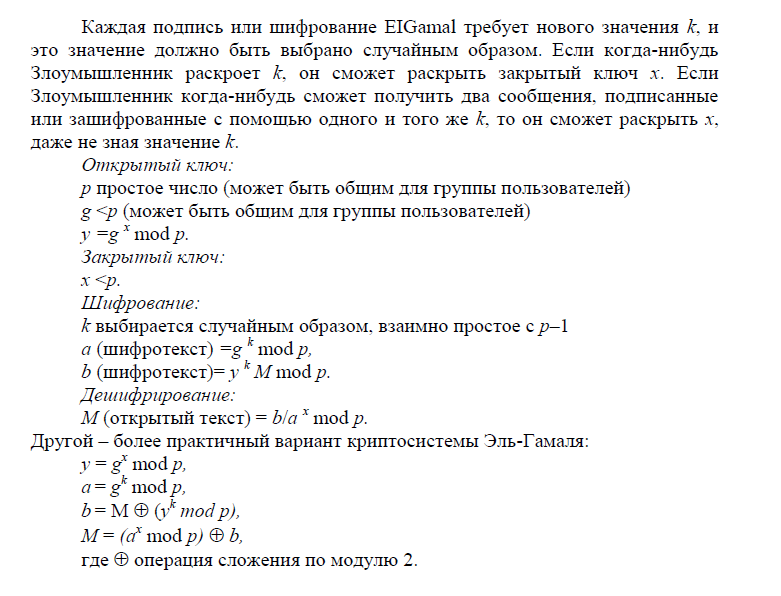
1. Пара чисел (r, е), являющаяся шифротекстом, передается второму пользователю.
2. Второй пользователь, получив (r,e), для расшифрования сообщения вычисляет

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 kпрактически 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).





1. **Оптимизация вычислений для RSA и Эль-Гамаля**

Криптосистема RSA была разработана в 1977 году и получила название в честь ее создателей: Рона Ривеста(Ron Rivest), Ади Шамира(Adi Shamir) и Леонарда Адлмана(Leonard Adleman). Они воспользовались тем фактом, что нахождение больших простых чисел в вычислительном отношении осуществляется легко, но разложение на множители произведения двух таких чисел практически невыполнимо. Доказано (теорема Рабина), что раскрытие шифра RSA эквивалентно такому разложению. Поэтому для любой длины ключа можно дать нижнюю оценку числа операций для раскрытия шифра, а с учетом производительности современных компьютеров оценить и необходимое на это время. Возможность гарантированно оценить защищенность алгоритма RSA стала одной из причин популярности этой криптосистемы на фоне десятков других схем. Поэтому алгоритм RSA используется в банковских компьютерных сетях, особенно для работы с удаленными клиентами (обслуживание кредитных карточек).   
Безопасность RSA основана на трудности разложения на множители больших чисел. Открытый и закрытый ключи являются функциями двух больших (100-200 разрядов или даже больше) простых чисел. Предполагается, что восстановление открытого текста по шифротексту и открытому ключу эквивалентно разложению на множители двух больших чисел.

**Алгоритм шифрования данных**

Для генерации двух ключей используются два больших случайных простых числа p и q. Для максимальной безопасности выбирайте p и q равной длины. Рассчитывается произведение:

n=p\*q

Затем случайным образом выбирается ключ шифрования e, такой что e и (p-1)(q-1) являются взаимно простыми числами. Наконец расширенный алгоритм Евклида используется для вычисления ключа дешифрирования d, такого что

e\*d=1mod(p-1)(q-1)

Заметим, что d и n также взаимно простые числа. Числа e и n - это открытый ключ, а число d - закрытый. Два простых числа p и q больше не нужны. Они должны быть отброшены, но не должны быть раскрыты.  
Для шифрования сообщения m оно сначала разбивается на цифровые блоки, меньшие n (для двоичных данных выбирается самая большая степень числа 2, меньшая n). То есть, если p и q - 100-разрядные простые числа, то n будет содержать около 200 разрядов, и каждый блок сообщения mi должен быть около 200 разрядов в длину. (Если нужно зашифровать фиксированное число блоков, их можно дополнить несколькими нулями слева, чтобы гарантировать, что блоки всегда будут меньше n. Зашифрованное сообщение c будет состоять из блоков ci той же самой длины.

***Оптимизация RSA***

Шифрование RSA выполняется намного быстрей, если вы правильно выберете значение e. Тремя наиболее частыми вариантами являются 3, 17 и 65537 (216 + 1). (Двоичное представление 65537 содержит только две единицы, поэтому для возведения в степень нужно выполнить только 17 умножений.) Не существует никаких проблем безопасности, связанных с использованием в качестве e любого из этих трех значений (при условии, что вы дополняете сообщения случайными числами), даже если одно и то же значение e используется целой группой пользователей.  
Операции с закрытым ключом можно ускорить при помощи китайской теоремы об остатках, если вы сохранили значения p и q, а также дополнительные значения: d mod (p-1), d mod (q-1) и q-1 mod p. Эти дополнительные числа можно легко вычислить по закрытому и открытому ключам.

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

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

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 kпрактически 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.

1. **Механизм распространения открытых ключей. Элект. подпись**

**Технологии цифровых подписей**

Как оказалось, теория асимметричного шифрования позволяет очень красиво решать еще одну проблему информационной безопасности – проверку подлинности автора сообщения. Для решения этой проблемы с помощью симметричной криптографии была разработана очень трудоемкая и сложная схема. В то же время с помощью, например, того же алгоритма RSA создать алгоритм проверки подлинности автора и неизменности сообщения чрезвычайно просто.

Предположим, что нам нужно передать какой-либо текст, не обязательно секретный, но важно то, чтобы в него при передаче по незащищенному каналу не были внесены изменения. К таким текстам обычно относятся различные распоряжения, справки, и тому подобная документация, не представляющая секрета. Вычислим от нашего текста какую-либо хеш-функцию – это будет число, которое более или менее уникально характеризует данный текст.

В принципе, можно найти другой текст, который дает то же самое значение хеш-функции, но изменить в нашем тексте десять-двадцать байт так, чтобы текст остался полностью осмысленным, да еще и изменился в выгодную нам сторону (например, уменьшил сумму к оплате в два раза) – чрезвычайно сложно. Именно для устранения этой возможности хеш-функции создают такими же сложными как и криптоалгоритмы – если текст с таким же значением хеш-функции можно будет подобрать только методом полного перебора, а множество значений будет составлять как и для блочных шифров 232–2128 возможных вариантов, то для поиска подобного текста злоумышленнику "потребуются" те же самые миллионы лет.

Таким образом, если мы сможем передать получателю защищенным от изменения методом хеш-сумму от пересылаемого текста, то у него всегда будет возможность самостоятельно вычислить хеш-функцию от текста уже на приемной стороне и сверить ее с присланной нами. Если хотя бы один бит в вычисленной им самостоятельно контрольной сумме текста не совпадет с соответствующим битом в полученном от нас хеш-значении, значит, текст по ходу пересылки подвергся несанкционированному изменению.

Представим теперь готовую к передаче хеш-сумму в виде нескольких k-битных блоков hi, где k – это размер сообщений по алгоритму RSA в предыдущем параграфе. Вычислим над каждым блоком значение si=((hi)d)mod n, где d – это тот самый закрытый ключ отправителя. Теперь сообщение, состоящее из блоков si можно "спокойно" передавать по сети. Никакой опасности по известным hi и si найти Ваш секретный ключ нет – это настолько же сложная задача, как и задача "логарифмирования в конечном поле". А вот любой получатель сообщения может легко прочесть исходное значение hi, выполнив операцию ((si)e)mod n = ((hi)d\*e)mod n = hi – Ваш открытый ключ (e,n) есть у всех, а то, что возведение любого числа в степень (e\*d) по модулю n дает исходное число, мы доказали в прошлом параграфе. При этом никто другой, кроме Вас, не зная Вашего закрытого ключа d не может, изменив текст, а следовательно, и хеш-сумму, вычислить такие s'i, чтобы при их возведении в степень e получилась хеш-сумма h'i, совпадающая с хеш-суммой фальсифицированного текста.

Таким образом, манипуляции с хеш-суммой текста представляют из себя "асимметричное шифрование наоборот": при отправке используется закрытый ключ отправителя, а для проверки сообщения – открытый ключ отправителя. Подобная технология получила название "электронная подпись". Информацией, которая уникально идентифицирует отправителя (его виртуальной подписью), является закрытый ключ d. Ни один человек, не владеющий этой информацией, не может создать такую пару (текст,si), что описанный выше алгоритм проверки дал бы положительный результат.

Подобный обмен местами открытого и закрытого ключей для создания из процедуры асимметричного шифрования алгоритма электронной подписи возможен только в тех системах, где выполняется свойство коммутативности ключей. Для других асимметричных систем алгоритм электронной подписи либо значительно отличается от базового, либо вообще не реализуем.

**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 . 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 
2. Затем с помощью расширенного алгоритма Евклида необходимо найти значение b в следующем уравнении:
3. 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: 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.

**Механизм распространения открытых ключей**

Казалось бы, асимметричные криптосистемы лишены одного из самых главных недостатков симметричных алгоритмов – необходимости предварительного обмена сторонами секретным ключом по защищенной схеме (например, из рук в руки или с помощью поверенного курьера). Вроде бы достаточно "раструбить" по всему свету о своем открытом ключе, и вот готова надежная линия передачи сообщений.

Но оказывается не все так просто: предположим я Ваш потенциальный собеседник. Для того чтобы отправить зашифрованное сообщение, я должен узнать Ваш открытый ключ. Если Вы не приносили мне его лично на дискете, значит я его просто взял из информационной сети. А теперь главный вопрос: где доказательство, что данный набор байт является именно Вашим открытым ключом? Ведь злоумышленник может сгенерировать произвольную пару (закрытый ключ, открытый ключ), затем активно распространять или пассивно подменять при запросе Ваш открытый ключ созданным им. В этом случае при отправке сообщения 1) я зашифрую его тем ключом, который думаю, что является Вашим, 2) злоумышленник, перехватив сообщение дешифрует его парным закрытым ключом, прочтет и более того: 3) может переслать дальше, зашифровав действительно уже Вашим открытым ключом. Точно так же, но по инверсной схеме, он может подменить и мою электронную подпись под моим письмом.

Таким образом, если между отправителем и получателем нет конфиденциальной схемы передачи асимметричных ключей, то возникает серьезная опасность появления злоумышленника-посредника. Но асимметричная криптография нашла изящный способ очень значительного снижения риска подобной атаки. Если задуматься, то неправильно говорить, что между Вами и Вашим собеседником нет гарантированной линии связи. Несомненно у Вас найдется трое-четверо надежных знакомых в столице или за рубежом, у них в свою очередь также найдется множество знакомых во многих точках страны и мира. В конце концов, Вы пользуетесь программным обеспечением фирм, если не центры, то хотя бы филиалы которых находятся в той стране или в том городе, куда Вы хотите отправить письмо. Проблема только в том, что начиная, со второго от Вас звена ни Вы не знаете человека, ни он Вас, и вероятность того, что он, или более того, крупная компания, будут что-либо делать ради Вас, очень мала.

Но в принципе, если множество единомышленников объединятся с целью создать надежную сеть распространения ключей, то это будет им вполне под силам. А сама асимметричная криптография поможет им в этом следующим образом: на самом деле никуда ходить с дискетой, получив просьбу от своего знакомого передать открытый ключ мистера V.M.B. мистеру R.H.J., не нужно. Ведь Вы общаетесь с Вашим знакомым, значит, у Вас есть его открытый ключ, полученный каким-либо надежным способом. А следовательно, он может Вам прислать этот открытый ключ мистера V.M.B., подписав сообщение своей электронной подписью. А от Вас в свою очередь требуется всего лишь отправить этот ключ дальше по цепочке в направлении мистера R.H.J., подписав уже своей электронной подписью. Таким образом, минуя несколько переподписываний, открытый ключ дойдет от места отправления к месту требования по надежному пути. В принципе от Вас даже может не требоваться никаких действий – просто поставьте на Вашей ЭВМ специальный сервер распространения ключей, и он все только что описанные действия будет выполнять автоматически.

На сегодняшний день не существует единой сети распространения открытых ключей, и дело, как это часто бывает, заключается в войне стандартов. Развиваются несколько независимых систем, но ни одна из них не получила довлеющего превосходства над другими, которое назывется "мировым стандартом".

Необходимо отметить, что цепочка распространения ключей в реальных случаях не очень велика. Обычно она состоит из двух-четырех звеньев. С привлечением к процессу распространения ключей крупных фирм-производителей программных продуктов она становится еще короче. Действительно, если на компакт-диске (не пиратском !) с купленным программным обеспечением уже находится открытый ключ этой фирмы, а сама она имеет крупный рынок сбыта, то цепочка будет состоять либо из одного звена (если ПО этой же фирмы стоит и у Вашего потенциального собеседника), либо из двух (вторым станет какой-нибудь другой гигантский концерн, чье ПО установлено у собеседника – уж между собой-то все крупные компании обменялись ключами электронных подписей достаточно давно). Открытый ключ, подписанный какой-либо третьей стороной, называется заверенным с помощью сертификата. Сертификатом называется информационный пакет, содержащий какой-либо объект (обычно ключ) и электронную подпись, подтверждающую этот объект от имени чьего-либо лица.

1. **Алгоритм подписи по стандарту 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

1. 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
6. v = [(аu1 \* yu2) mod p] mod q
7. 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.

1. **Алгоритм подписи RSA наоборот**

Рассмотренная нами в "Криптографические алгоритмы с открытым ключом и их использование"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 В 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 , то есть шифрует сообщение своим закрытым ключом. В результате этих действий пользователь А подписывает сообщение.
2. Затем пользователь А вычисляет числа , 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 , используя свой закрытый ключ. При этом сообщение расшифровывается.
5. Затем Б определяет числа , 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.

1. **Обмен ключами по Диффи-Хелману**

Данный параграф посвящен еще одному интересному алгоритму, который достаточно трудно классифицировать. Он помогает обмениваться секретным ключом для симметричных криптосистем, но использует метод, очень похожий на асимметричный алгоритм RSA. Алгоритм назван по фамилиям его создателей Диффи (Diffie) и Хеллмана (Hellman).

Определим круг его возможностей. Предположим, что двум абонентам необходимо провести конфиденциальную переписку, а в их распоряжении нет первоначально оговоренного секретного ключа. Однако, между ними существует канал, защищенный от модификации, то есть данные, передаваемые по нему, могут быть прослушаны, но не изменены (такие условия имеют место довольно часто). В этом случае две стороны могут создать одинаковый секретный ключ, ни разу не передав его по сети, по следующему алгоритму.

Предположим, что обоим абонентам известны некоторые два числа v и n. Они, впрочем, известны и всем остальным заинтересованным лицам. Например, они могут быть просто фиксированно "зашиты" в программное обеспечение. Для того, чтобы создать неизвестный более никому секретный ключ, оба абонента генерируют случайные или псевдослучайные простые числа: первый абонент – число x, второй абонент – число y. Затем первый абонент вычисляет значение (vx) mod n и пересылает его второму, а второй вычисляет (vy) mod n и передает первому. Злоумышленник получает оба этих значения, но модифицировать их (вмешаться в процесс передачи) не может. На втором этапе первый абонент на основе имеющегося у него x и полученного по сети (vy) mod n вычисляет значение (((vy) mod n)x)mod n, а второй абонент на основе имеющегося у него y и полученного по сети (vx) mod n вычисляет значение (((vx) mod n)y)mod n. На самом деле операция возведения в степень переносима через операцию взятия модуля по простому числу (то есть коммутативна в конечном поле), то есть у обоих абонентов получилось одно и то же число: ((vx\*y) mod n. Его они и могут использовать в качестве секретного ключа, поскольку здесь злоумышленник снова встретится с проблемой RSA при попытке выяснить по перехваченным (vx) mod n и (vy) mod n сами числа x и y – это очень и очень ресурсоемкая операция, если числа v,n,x,y выбраны достаточно большими.

Необходимо еще раз отметить, что алгоритм Диффи-Хеллмана работает только на линиях связи, надежно защищенных от модификации. Если бы он был применим на любых открытых каналах, то давно снял бы проблему распространения ключей и, возможно, заменил собой всю асимметричную криптографию. Однако, в тех случаях, когда в канале возможна модификация данных, появляется очевидная возможность вклинивания в процесс генерации ключей "злоумышленника-посредника" по той же самой схеме, что и для асимметричной криптографии.

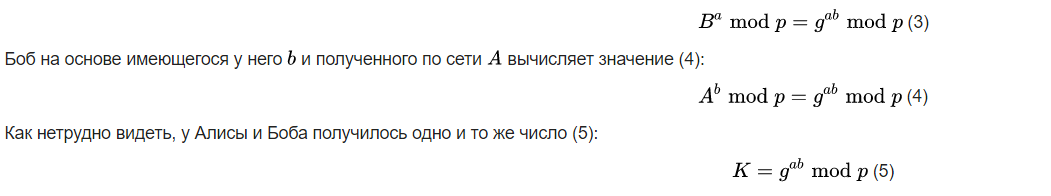
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

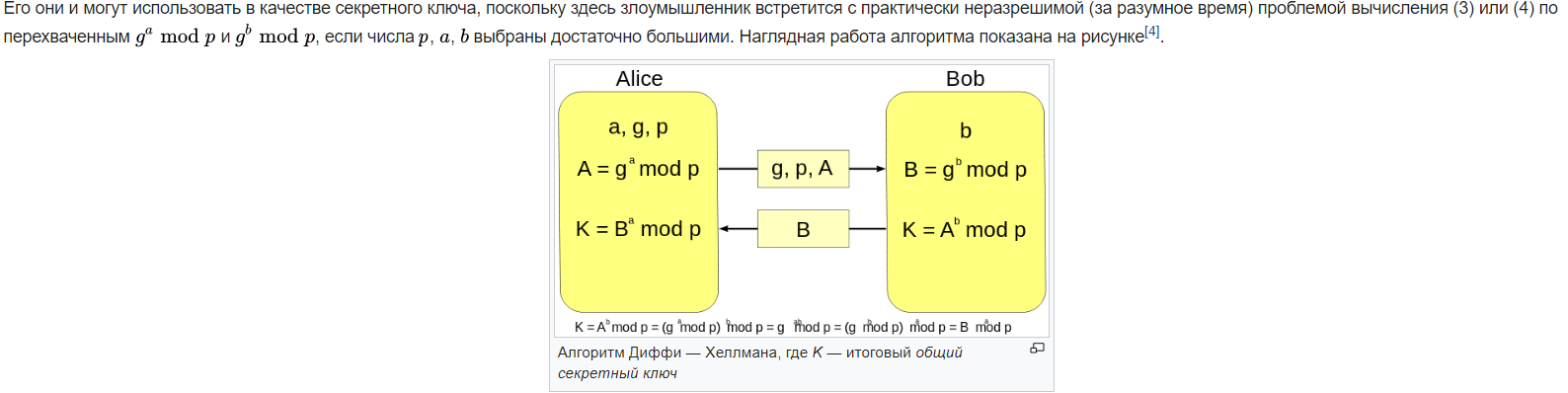
Предположим, существует два абонента: Алиса и Боб. Обоим абонентам известны некоторые два числа {\displaystyle g}g и {\displaystyle p}p, которые не являются секретными и могут быть известны также другим заинтересованным лицам. Для того, чтобы создать неизвестный более никому секретный ключ, оба абонента генерируют большие случайные числа: Алиса — число *{\displaystyle a}*, Боб — число *{\displaystyle b}*. Затем Алиса вычисляет значение (1):

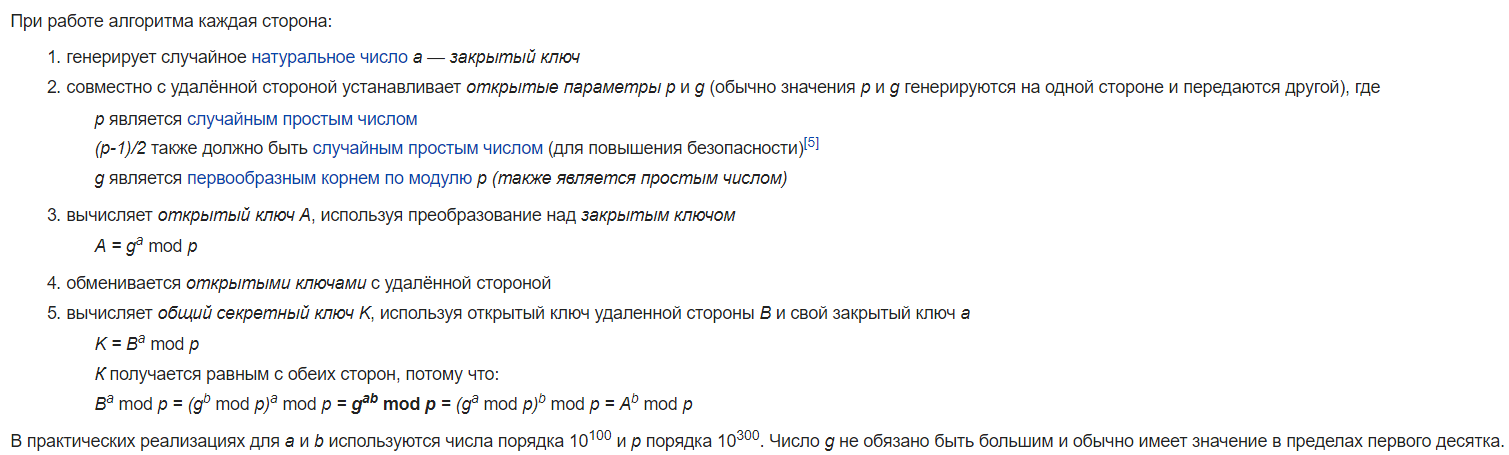


и передаёт Алисе. Предполагается, что злоумышленник может получить оба этих значения, но не модифицировать их (то есть, у него нет возможности вмешаться в процесс передачи).

На втором этапе Алиса на основе имеющегося у неё *{\displaystyle a}a*  и полученного по сети *{\displaystyle B}B* вычисляет значение (3):







1. **Понятие хэш-функции. Хэширование паролей**

**Хеш-функцией (hash function)** называется математическая или иная функция, которая для строки произвольной длины вычисляет некоторое целое значение или некоторую другую строку фиксированной длины. Математически это можно записать так:

1. h=H(M),

где М – исходное сообщение, называемое иногда прообразом, а h – результат, называемый значением хеш-функции (а также хеш-кодом или дайджестом сообщения (от англ. message digest )).

Смысл хеш-функции состоит в определении характерного признака прообраза – значения хеш-функции. Это значение обычно имеет определенный фиксированный размер, например, 64 или 128 бит. Хеш-код может быть в дальнейшем проанализирован для решения какой-либо задачи. Так, например, хеширование может применяться для сравнения данных: если у двух массивов данных хеш-коды разные, массивы гарантированно различаются; если одинаковые — массивы, скорее всего, одинаковы. В общем случае однозначного соответствия между исходными данными и хеш-кодом нет из-за того, что количество значений хеш-функций всегда меньше, чем вариантов входных данных. Следовательно, существует множество входных сообщений, дающих одинаковые хеш-коды (такие ситуации называются коллизиями ). Вероятность возникновения коллизий играет немаловажную роль в оценке качества хеш-функций.

Хеш-функции широко применяются в современной криптографии.

Простейшая хеш-функция может быть составлена с использованием операции "сумма по модулю 2" следующим образом: получаем входную строку, складываем все байты по модулю 2 и байт-результат возвращаем в качестве значения хеш-фукнции. Длина значения хеш-функции составит в этом случае 8 бит независимо от размера входного сообщения.

Например, пусть исходное сообщение, переведенное в цифровой вид, было следующим (в шестнадцатеричном формате):

1. 3E 54 A0 1F B4

Переведем сообщение в двоичный вид, запишем байты друг под другом и сложим биты в каждом столбике по модулю 2:

1. 0011 1110
2. 0101 0100
3. 1010 0000
4. 0001 1111
5. 1011 0100
6. ----------
7. 0110 0101

Результат ( 0110 0101(2) или 65(16) ) и будет значением хеш-функции.

Однако такую хеш-функцию нельзя использовать для криптографических целей, например для формирования электронной подписи, так как достаточно легко изменить содержание подписанного сообщения, не меняя значения контрольной суммы.

Поэтому рассмотренная хеш-функция не годится для криптографических применений. В криптографии хеш-функция считается хорошей, если трудно создать два прообраза с одинаковым значением хеш-функции, а также, если у выхода функции нет явной зависимости от входа.

Сформулируем основные требования, предъявляемые к криптографическим хеш-функциям:

* хеш-функция должна быть применима к сообщению любого размера;
* вычисление значения функции должно выполняться достаточно быстро;
* при известном значении хеш-функции должно быть трудно (практически невозможно) найти подходящий прообраз М ;
* при известном сообщении М должно быть трудно найти другое сообщение М’ с таким же значением хеш-функции, как у исходного сообщения;
* должно быть трудно найти какую-либо пару случайных различных сообщений с одинаковым значением хеш-функции.

Создать хеш-функцию, которая удовлетворяет всем перечисленным требованиям – задача непростая. Необходимо также помнить, что на вход функции поступают данные произвольного размера, а хеш-результат не должен получаться одинаковым для данных разного размера.

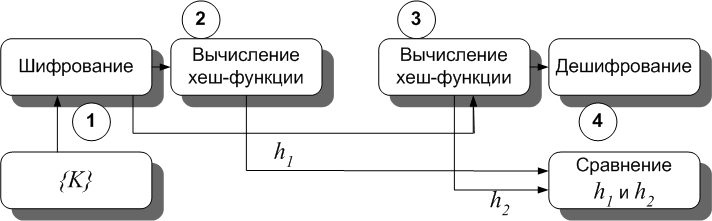
В настоящее время на практике в качестве хеш-функций применяются функции, обрабатывающие входное сообщение блок за блоком и вычисляющие хеш-значение hi для каждого блока Mi входного сообщения по зависимостям вида

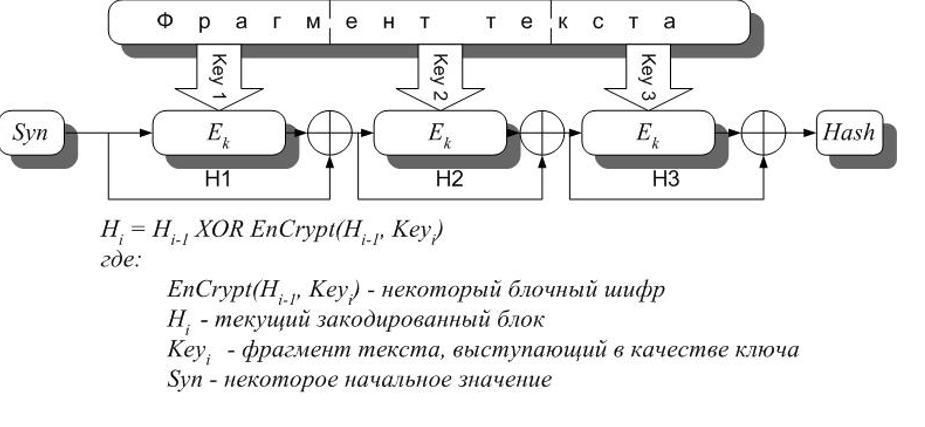
1. hi=H(Mi,hi-1),

где hi-1 – результат, полученный при вычислении хеш-функции для предыдущего блока входных данных.

В результате выход хеш-функции hn является функцией от всех n блоков входного сообщения.

1. **Методы вычисления хэш-функции**

****



Известные алгоритмы хэширования:



В настоящее время предложены и практически используются различные специальные алгоритмы для вычисления хеш-функции. Наиболее известными алгоритмами являются MD5, SHA-1, SHA-2 и другие версии SHA, а также отечественный алгоритм, изложенный в ГОСТ Р 34.11-94.

Алгоритм MD5 появился в начале 90-х годов ХХ века в результате усовершенствования алгоритма формирования хеш-функции MD4. Символы в названии "MD" означают Message Digest – краткое изложение сообщения. Автор алгоритмов MD4 и MD5 – Р. Ривест (R.Rivest). В результате использования MD5 для произвольного сообщения формируется 128-битное хеш-значение. Входные данные обрабатываются блоками по 512 бит. В алгоритме используются элементарные логические операции (инверсия, конъюнкция, сложение по модулю 2, циклические сдвиги и др.), а также обыкновенное арифметическое сложение. Комплексное повторение этих элементарных функций алгоритма обеспечивает то, что результат после обработки хорошо перемешан. Поэтому маловероятно, чтобы два сообщения, выбранные случайно, имели одинаковый хеш-код. Алгоритм MD5 имеет следующее свойство: каждый бит полученного хеш-значения является функцией от каждого бита входа. Считается, что MD5 является наиболее сильной хеш-функцией для 128-битного хеш-значения.

Алгоритм SHA (Secure Hash Algorithm – Безопасный хеш-алгоритм) был разработан национальным институтом стандартов и технологии (NIST) США и опубликован в качестве американского федерального информационного стандарта в 1993 году. SHA-1, как и MD5, основан на алгоритме MD4. SHA-1 формирует 160-битное хеш-значение на основе обработки исходного сообщения блоками по 512 бит. В алгоритме SHA-1 также используются простые логические и арифметические операции. Наиболее важным отличием SHA-1 от MD5 является то, что хеш-код SHA-1 на 32 бита длиннее, чем хеш-код MD5. Если предположить, что оба алгоритма одинаковы по сложности для криптоанализа, то SHA-1 является более стойким алгоритмом. Используя атаку методом грубой силы (лобовую атаку), труднее создать произвольное сообщение, имеющее данный хеш-код, а также труднее создать два сообщения, имеющие одинаковый хеш-код.

В 2001 году национальный институт стандартов и технологии США принял в качестве стандарта три хеш-функции с большей длиной хеш-кода, чем у SHA-1. Часто эти хеш-функции называют SHA-2 или SHA-256, SHA-384 и SHA-512 (в названии указывается длина создаваемого алгоритмами хеш-кода). Эти алгоритмы отличаются не только длиной создаваемого хеш-кода, но и используемыми внутренними функциями и длиной обрабатываемого блока (у SHA-256 длина блока – 512, а у SHA-384 и SHA-512 длина блока – 1024 бита). Постепенные усовершенствования алгоритма SHA ведут к увеличению его криптостойкости. Несмотря на отличия рассматриваемых алгоритмов друг от друга, все они являются дальнейшим развитием SHA-1 и MD4 и имеют похожую структуру.

В России принят **ГОСТ Р34.11-94**, который является отечественным стандартом для хеш-функций. Его структура довольно сильно отличается от структуры алгоритмов SHA-1,2 или MD5, в основе которых лежит алгоритм MD4. Длина хеш-кода, создаваемого алгоритмом ГОСТ Р 34.11-94, равна 256 битам. Алгоритм последовательно обрабатывает исходное сообщение блоками по 256 бит справа налево. Параметром алгоритма является стартовый вектор хеширования – произвольное фиксированное значение длиной также 256 бит. В алгоритме ГОСТ Р 34.11-94 используются операции перестановки, сдвига, арифметического сложения, сложения по модулю 2. В качестве вспомогательной функции в ГОСТ 34.11-94 используется алгоритм по ГОСТ 28147-89 в режиме простой замены.

1. **Алгоритмы DES. Режимы работы, достоинства, недостатки**

**Стандарты шифрования данных** — совокупность правил, используемых в мощной алгоритмической технике кодировании информации.

Сегодня самым известным и простым алгоритмом, который используется в системах защиты информации организаций и предприятий, является [DES (Data Encryption Standart)](https://ru.wikipedia.org/wiki/DES). Он был описан в 1977 году Национальным бюро стандартов [Соединённых Штатов Америки](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B5%D0%B4%D0%B8%D0%BD%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%A8%D1%82%D0%B0%D1%82%D1%8B_%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B8). Он необходим для защиты от незаконного доступа к важной информации в организациях. Особенности алгоритма DES — используется ключ длиной 56 бит; зашифровать можно сообщение с помощью одной программы, а расшифровать — используя любую другую программу, соответствующую DES; высокая скорость обработки достигается за счёт несложного алгоритма и высокой стойкости.

Алгоритм Data Encryption Standart использует множество подстановок и перестановок, производит шифрование 64-битовых блоков с помощью ключа размером 64 бит. Основными являются 56 бит, остальные 8 [бит](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) — контрольные. Дешифрование в Data Encryption Standart — процедура, обратная шифрованию, производится повтором операций в обратном порядке. Шифрование заключается в перестановке бит 64-битового блока, 16 циклах, и в конечной перестановке битов. Расшифрование является обратным процессу шифрования.

Data Encryption Standart подходит для шифрования и [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) данных. Он позволяет преобразовывать открытый текст в 64- битовый выходной зашифрованный текст.

Для того чтобы пользоваться алгоритмом Data Encryption Standart для решения задач разработаны 4 режима: электронная кодовая книга ECB (Electronic Code Book), сцепление блоков шифра CFB (Cipher Feed Back) и обратная связь по выходу OFB (Output Feed Back). Обратная связь по зашифрованному тексту и по выходу используются в качестве поточных шрифтов. Это происходит так: каждый бит из последующего потока шифруется отдельно с использованием ключа и ранее закодированной информации.

Одной из наиболее известных криптографических систем с закрытым ключом является *DES – Data Encryption Standard*. Эта система первой получила статус государственного стандарта в области шифрования данных. Она разработана специалистами фирмы *IBM* и вступила в действие в США 1977 году. *Алгоритм* *DES* широко использовался при хранении и передаче данных между различными вычислительными системами; в почтовых системах, в электронных системах чертежей и при электронном обмене *коммерческой информацией*. Стандарт *DES* реализовывался как программно, так и аппаратно. Предприятиями разных стран был налажен массовый выпуск цифровых устройств, использующих *DES* для шифрования данных. Все устройства проходили обязательную сертификацию на соответствие стандарту.

Несмотря на то, что уже некоторое время эта система не имеет статуса государственного стандарта, она по-прежнему широко применяется и заслуживает внимания при изучении блочных шифров с закрытым ключом.

*Длина ключа* в алгоритме *DES* составляет 56 *бит*. Именно с этим фактом связана основная полемика относительно способности *DES*противостоять различным атакам. Как известно, любой блочный *шифр* с закрытым ключом можно взломать, перебрав все возможные комбинации ключей. При длине ключа 56 *бит* возможны 256 разных ключей. Если *компьютер* перебирает за одну секунду 1 000 000 ключей (что примерно равно 220), то на перебор всех 256 ключей потребуется 236 секунд или чуть более двух тысяч лет, что, конечно, является неприемлемым для злоумышленников.

Однако возможны более дорогие и быстрые вычислительные системы, чем *персональный компьютер*. Например, если иметь возможность объединить для проведения параллельных вычислений миллион процессоров, то максимальное время подбора ключа сокращается примерно до 18 часов. Это время не слишком велико, и криптоаналитик, оснащенный подобной дорогой техникой, вполне может выполнить вскрытие данных, зашифрованных DESза приемлемое для себя время.

Вместе с этим можно отметить, что систему *DES* вполне можно использовать в небольших и средних приложениях для шифрования данных, имеющих небольшую ценность. Для шифрования данных государственной важности или имеющих значительную коммерческую *стоимость* система *DES* в настоящее время, конечно, не должна использоваться. В 2001 году после специально объявленного конкурса в США был принят новый стандарт на блочный *шифр*, названный *AES (Advanced Encryption Standard)*, в основу которого был положен *шифр* *Rijndael*, разработанный бельгийскими специалистами. Этот *шифр* рассматривается в конце лекции.

Основные параметры *DES*: размер блока 64 бита, *длина ключа* 56 *бит*, количество раундов – 16. *DES* является классической сетью Фейштеля с двумя ветвями. *Алгоритм* преобразует за несколько раундов 64-битный *входной* *блок данных* в 64-битный выходной блок. Стандарт *DES* построен на комбинированном использовании перестановки, замены и гаммирования. Шифруемые данные должны быть представлены в двоичном виде.

**Шифрование**

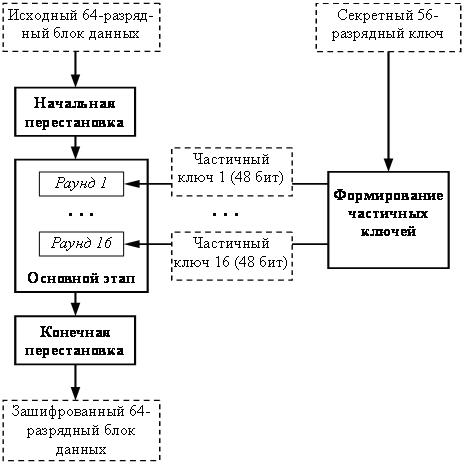
Общая структура *DES* представлена на [рис. 4.1](https://www.intuit.ru/studies/courses/691/547/lecture/12377?page=1#image.4.1). Процесс шифрования каждого 64-битового блока исходных данных можно разделить на три этапа:

1. начальная подготовка блока данных;
2. 16 раундов "основного цикла";
3. конечная обработка блока данных.

На первом этапе выполняется *начальная перестановка* 64-битного исходного блока текста, во время которой биты определенным образом переупорядочиваются.

На следующем (основном) этапе блок делится на две части (ветви) по 32 бита каждая. Правая *ветвь* преобразуется с использованием некоторой функции F и соответствующего *частичного ключа*, получаемого из основного ключа шифрования по специальному алгоритму преобразования ключей. Затем производится *обмен данными* между левой и правой ветвями блока. Это повторяется в цикле 16 раз.

Наконец, на третьем этапе выполняется *перестановка* результата, полученного после шестнадцати шагов основного *цикла*. Эта *перестановка* обратна начальной перестановке.



**Рис. 4.1.**Общая схема DES

Рассмотрим более подробно все этапы криптографического преобразования по стандарту *DES*.

На первом этапе 64-разрядный блок исходных данных подвергается начальной перестановке. В литературе эта операция иногда называется "забеливание" – *whitening*. При начальной перестановке биты блока данных определенным образом переупорядочиваются. Эта операция придает некоторую "хаотичность" исходному сообщению, снижая возможность использования криптоанализа статистическими методами.

Одновременно с начальной перестановкой блока данных выполняется начальная *перестановка* 56 *бит* ключа. Из [рис. 4.1](https://www.intuit.ru/studies/courses/691/547/lecture/12377?page=1#image.4.1). видно, что в каждом из раундов используется соответствующий 48-битный частичный *ключ* Ki. Ключи Ki получаются по определенному алгоритму, используя каждый из битов начального ключа по нескольку раз. В каждом раунде 56-битный *ключ* делится на две 28-битовые половинки. Затем половинки сдвигаются влево на один или два бита в зависимости от номера раунда. После сдвига определенным образом выбирается 48 из 56 битов. Так как при этом не только выбирается *подмножество* битов, но и изменяется их порядок, то эта операция называется "*перестановка* со сжатием". Ее результатом является набор из 48 битов. В среднем каждый *бит*исходного 56-битного ключа используется в 14 из 16 подключей, хотя не все биты используются одинаковое количество раз.

Далее выполняется основной цикл преобразования, организованный по сети Фейштеля и состоящий из 16 одинаковых раундов. При этом в каждом раунде ([рис. 4.2](https://www.intuit.ru/studies/courses/691/547/lecture/12377?page=1#image.4.2)) получается промежуточное 64-битное *значение*, которое затем обрабатывается в следующем раунде.

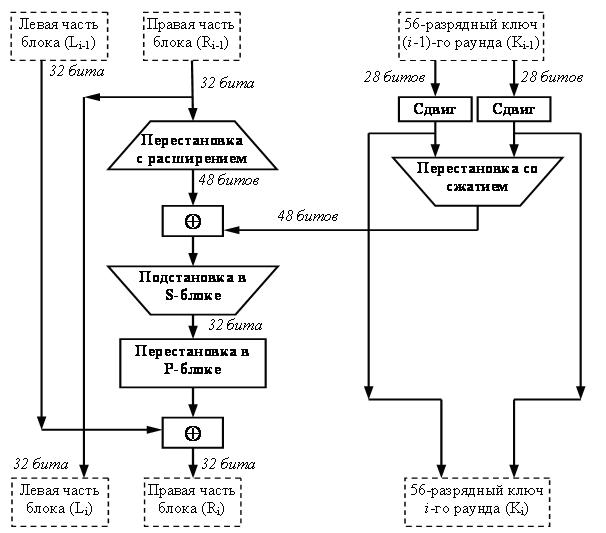
Основные ***достоинства алгоритма DES***:

* используется только один ключ длиной 56 битов;
* зашифровав сообщение с помощью одного пакета, для расшифровки вы можете использовать любой другой;
* относительная простота алгоритма обеспечивает высокую скорость обработки информации;
* достаточно высокая стойкость алгоритма.

Проблемы DES

Так как длина ключа равна 56 битам, существует 256 возможных ключей. На сегодня такая длина ключа недостаточна, поскольку допускает успешное применение лобовых атак. Альтернативой DES можно считать тройной DES, IDEA, а также алгоритм Rijndael, принятый в качестве нового стандарта на алгоритмы симметричного шифрования.

Также без ответа пока остается вопрос, возможен ли криптоанализ с использованием существующих характеристик алгоритма DES. Основой алгоритма являются восемь таблиц подстановки, или S-boxes, которые применяются в каждой итерации. Существует опасность, что эти S-boxes конструировались таким образом, что криптоанализ возможен для взломщика, который знает слабые места S-boxes. В течение многих лет обсуждалось как стандартное, так и неожиданное поведение S-boxes, но все-таки никому не удалось обнаружить их фатально слабые места.



**Рис. 4.2.**Структура одного раунда DES

Левая и правая ветви каждого промежуточного значения обрабатываются как отдельные 32-битные значения, обозначенные L и R.

Вначале правая часть блока Ri расширяется до 48 битов, используя таблицу, которая определяет перестановку плюс расширение на 16 битов. Эта операция приводит размер правой половины в соответствие с размером ключа для выполнения *операции* *XOR*. Кроме того, за счет выполнения этой *операции* быстрее возрастает зависимость всех битов результата от битов исходных данных и ключа (это называется "лавинным эффектом"). Чем сильнее проявляется лавинный эффект при использовании того или иного алгоритма шифрования, тем лучше.

После выполнения перестановки с расширением для полученного 48-битного значения выполняется операция XOR с 48-битным подключом Ki. Затем полученное 48-битное *значение* подается на вход блока подстановки S (от англ. *Substitution* - *подстановка), результатом* которой является 32-битное *значение*. *Подстановка* выполняется в восьми блоках подстановки или восьми S-блоках (S-boxes). При выполнении этой *операции* 48 битов данных делятся на восемь 6-битовых подблоков, каждый из которых по своей таблице замен заменяется четырьмя битами. *Подстановка* с помощью S-блоков является одним из важнейших этапом *DES*. Таблицы замен для этой *операции* специально спроектированы специалистами так, чтобы обеспечивать максимальную *безопасность*. В результате выполнения этого этапа получаются восемь 4-битовых блоков, которые вновь объединяются в единое 32-битовое *значение*.

Далее полученное 32-битовое *значение* обрабатывается с помощью перестановки Р (от англ. *Permutation* – *перестановка*), которая не зависит от используемого ключа. Целью перестановки является максимальное переупорядочивание битов такое, чтобы в следующем раунде шифрования каждый *бит* с большой вероятностью обрабатывался другим S-блоком.

И, наконец, результат перестановки объединяется с помощью *операции* XOR с левой половиной первоначального 64-битового блока данных. Затем левая и правая половины меняются местами, и начинается следующий раунд.

После шестнадцати раундов шифрования выполняется конечная *перестановка* результата. Эта *перестановка* инверсна (обратна) начальной перестановке.

После выполнения всех указанных шагов *блок данных* считается полностью зашифрованным и можно переходить к шифрованию следующего блока исходного сообщения.

1. **Алгоритм ГОСТ. Режимы работы**

**Российский стандарт шифрования данных**

Российский стандарт шифрования данных представлен в [ГОСТ](https://ru.wikipedia.org/wiki/%D0%93%D0%9E%D0%A1%D0%A2)[[1]](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D1%8B_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#cite_note-gost-1) 28147-89. «Система обработки информации. Защита криптографическая. Алгоритм криптографического преобразования.» Алгоритм является блочным шифром. Он управляет блоками из 64 бит, используя функции преобразования. Преимущество данного алгоритма — длина ключа 128 бит. Операции производятся над 16-битовыми подблоками. Один алгоритм используют для кодирования и для дешифрования. Как и для остальных шифров, при шифровке производятся процессы перемешивания и рассеивания, и процессы могут реализованы программными средствами. Преимущества алгоритма — высокая криптографическая стойкость, удобство программного и аппаратного представления.

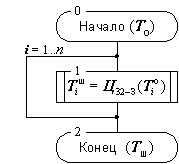
Режимы работы

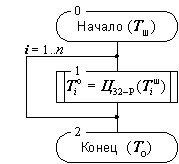
ГОСТ 28147-89 — блочный шифр с 256-[битным](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) [ключом](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) и 32 циклами (называемыми раундами) преобразования, оперирующий 64-битными блоками. Основа [алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) шифра — [сеть Фейстеля](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D1%8C_%D0%A4%D0%B5%D0%B9%D1%81%D1%82%D0%B5%D0%BB%D1%8F). Выделяют четыре режима работы ГОСТ 28147-89:

* [простой замены](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC_%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2%D0%BE%D0%B9_%D0%BA%D0%BD%D0%B8%D0%B3%D0%B8)
* [гаммирование](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)
* [гаммирование с обратной связью](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D0%B8_%D0%BF%D0%BE_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82%D1%83)
* режим выработки [имитовставки](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%B8%D1%82%D0%BE%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0).

**Простая замена.**

Зашифрование в данном режиме заключается в применении цикла 32-З к блокам открытых данных, расшифрование – цикла 32-Р к блокам зашифрованных данных. Это наиболее простой из режимов, 64-битовые блоки данных обрабатываются в нем независимо друг от друга. Схемы алгоритмов зашифрования и расшифрования в режиме простой замены приведены на рисунках 3а и б соответственно, они тривиальны и не нуждаются в комментариях.

  
*Рисунок. 3а. Алгоритм зашифрования данных в режиме простой замены*

  
*Рисунок. 3б. Алгоритм расшифрования данных в режиме простой замены*

Размер массива открытых или зашифрованных данных, подвергающийся соответственно зашифрованию или расшифрованию, должен быть кратен 64 битам: |***T***о|=|***T***ш|=64·***n***, после выполнения операции размер полученного массива данных не изменяется.

Режим шифрования простой заменой имеет следующие особенности:

* Так как блоки данных шифруются независимо друг от друга и от их позиции в массиве данных, при зашифровании двух одинаковых блоков открытого текста получаются одинаковые блоки шифртекста и наоборот. Отмеченное свойство позволит криптоаналитику сделать заключение о тождественности блоков исходных данных, если в массиве зашифрованных данных ему встретились идентичные блоки, что является недопустимым для серьезного шифра.
* Если длина шифруемого массива данных не кратна 8 байтам или 64 битам, возникает проблема, чем и как дополнять последний неполный блок данных массива до полных 64 бит. Эта задача не так проста, как кажется на первый взгляд. Очевидные решения типа «дополнить неполный блок нулевыми битами» или, более обще, «дополнить неполный блок фиксированной комбинацией нулевых и единичных битов» могут при определенных условиях дать в руки криптоаналитика возможность методами перебора определить содержимое этого самого неполного блока, и этот факт означает снижение стойкости шифра. Кроме того, длина шифртекста при этом изменится, увеличившись до ближайшего целого, кратного 64 битам, что часто бывает нежелательным.

На первый взгляд, перечисленные выше особенности делают практически невозможным использование режима простой замены, ведь он может применяться только для шифрования массивов данных с размером кратным 64 битам, не содержащим повторяющихся 64-битовых блоков. Кажется, что для любых реальных данных гарантировать выполнение указанных условий невозможно. Это почти так, но есть одно очень важное исключение: вспомните, что размер ключа составляет 32 байта, а размер таблицы замен – 64 байта. Кроме того, наличие повторяющихся 8-байтовых блоков в ключе или таблице замен будет говорить об их весьма плохом качестве, поэтому в реальных ключевых элементах такого повторения быть не может. Таким образом, мы выяснили, что режим простой замены вполне подходит для шифрования ключевой информации, тем более, что прочие режимы для этой цели менее удобны, поскольку требуют наличия дополнительного синхронизирующего элемента данных – синхропосылки (см. следующий раздел). Наша догадка верна, ГОСТ предписывает использовать режим простой замены исключительно для шифрования ключевых данных.

**Гаммирование.**

Как же можно избавиться от недостатков режима простой замены? Для этого необходимо сделать возможным шифрование блоков с размером менее 64 бит и обеспечить зависимость блока шифртекста от его номера, иными словами, ***рандомизировать***процесс шифрования. В ГОСТе это достигается двумя различными способами в двух режимах шифрования, предусматривающих ***гаммирование***. ***Гаммирование***– это наложение (снятие) на открытые (зашифрованные) данные криптографической гаммы, то есть последовательности элементов данных, вырабатываемых с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных. Для наложения гаммы при зашифровании и ее снятия при расшифровании должны использоваться взаимно обратные бинарные операции, например, сложение и вычитание по модулю 264 для 64-битовых блоков данных. В ГОСТе для этой цели используется операция побитового сложения по модулю 2, поскольку она является обратной самой себе и, к тому же, наиболее просто реализуется аппаратно. Гаммирование решает обе упомянутые проблемы: во-первых, все элементы гаммы различны для реальных шифруемых массивов и, следовательно, результат зашифрования даже двух одинаковых блоков в одном массиве данных будет различным. Во-вторых, хотя элементы гаммы и вырабатываются одинаковыми порциями в 64 бита, использоваться может и часть такого блока с размером, равным размеру шифруемого блока.

Теперь перейдем непосредственно к описанию режима гаммирования. Гамма для этого режима получается следующим образом: с помощью некоторого алгоритмического рекуррентного генератора последовательности чисел (РГПЧ) вырабатываются 64-битовые блоки данных, которые далее подвергаются преобразованию по циклу 32-З, то есть зашифрованию в режиме простой замены, в результате получаются блоки гаммы. Благодаря тому, что наложение и снятие гаммы осуществляется при помощи одной и той же операции побитового исключающего или, алгоритмы зашифрования и расшифрования в режиме гаммирования идентичны, их общая схема приведена на рисунке 4.

РГПЧ, используемый для выработки гаммы, является рекуррентной функцией: http://kaf401.rloc.ru/Criptfiles/gost28147/symb07.jpg– элементы рекуррентной последовательности, *f* – функция преобразования. Следовательно, неизбежно возникает вопрос о его инициализации, то есть об элементе http://kaf401.rloc.ru/Criptfiles/gost28147/symb08.jpg В действительности, этот элемент данных является параметром алгоритма для режимов гаммирования, на схемах он обозначен как *S*, и называется в криптографии ***синхропосылкой***, а в нашем ГОСТе – ***начальным заполнением***одного из регистров шифрователя. По определенным соображениям разработчики ГОСТа решили использовать для инициализации РГПЧ не непосредственно синхропосылку, а результат ее преобразования по циклу 32-З: http://kaf401.rloc.ru/Criptfiles/gost28147/symb09.jpg. Последовательность элементов, вырабатываемых РГПЧ, целиком зависит от его начального заполнения, то есть элементы этой последовательности являются функцией своего номера и начального заполнения РГПЧ: http://kaf401.rloc.ru/Criptfiles/gost28147/symb10.jpg где *fi*(*X*)=*f*(*fi*–1(*X*)), *f* 0(*X*)=*X*. С учетом преобразования по алгоритму простой замены добавляется еще и зависимость от ключа:

http://kaf401.rloc.ru/Criptfiles/gost28147/symb11.jpgгде *Гi*– *i*-тый элемент гаммы, *K* – ключ.

Таким образом, последовательность элементов гаммы для использования в режиме гаммирования однозначно определяется ключевыми данными и синхропосылкой. Естественно, для обратимости процедуры шифрования в процессах за- и расшифрования должна использоваться одна и та же синхропосылка. Из требования уникальности гаммы, невыполнение которого приводит к катастрофическому снижению стойкости шифра, следует, что для шифрования двух различных массивов данных на одном ключе необходимо обеспечить использование различных синхропосылок. Это приводит к необходимости хранить или передавать синхропосылку по каналам связи вместе с зашифрованными данными, хотя в отдельных особых случаях она может быть предопределена или вычисляться особым образом, если исключается шифрование двух массивов на одном ключе.

Теперь подробно рассмотрим РГПЧ, используемый в ГОСТе для генерации элементов гаммы. Прежде всего, надо отметить, что к нему не предъявляются требования обеспечения каких-либо статистических характеристик вырабатываемой последовательности чисел. РГПЧ спроектирован разработчиками ГОСТа исходя из необходимости выполнения следующих условий:

* период повторения последовательности чисел, вырабатываемой РГПЧ, не должен сильно (в процентном отношении) отличаться от максимально возможного при заданном размере блока значения 264;
* соседние значения, вырабатываемые РГПЧ, должны отличаться друг от друга в каждом байте, иначе задача криптоаналитика будет упрощена;
* РГПЧ должен быть достаточно просто реализуем как аппаратно, так и программно на наиболее распространенных типах процессоров, большинство из которых, как известно, имеют разрядность 32 бита.

Исходя из перечисленных принципов, создатели ГОСТа спроектировали весьма удачный РГПЧ, имеющий следующие характеристики:

* в 64-битовом блоке старшая и младшая части обрабатываются независимо друг от друга:http://kaf401.rloc.ru/Criptfiles/gost28147/15form.jpg; фактически, существуют два независимых РГПЧ для старшей и младшей частей блока.
* рекуррентные соотношения для старшей и младшей частей следующие:

http://kaf401.rloc.ru/Criptfiles/gost28147/16form.jpg, где *C* 0=101010116;

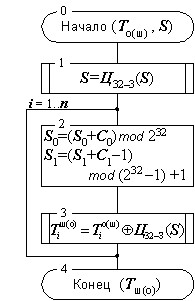
http://kaf401.rloc.ru/Criptfiles/gost28147/17form.jpg, где *C* 1=101010416;

Нижний индекс в записи числа означает его систему счисления, таким образом, константы, используемые на данном шаге, записаны в 16-ричной системе счисления.

Второе выражение нуждается в комментариях, так как в тексте ГОСТа приведено нечто другое: http://kaf401.rloc.ru/Criptfiles/gost28147/18form.jpg, с тем же значением константы *C* 1. Но далее в тексте стандарта дается комментарий, что, оказывается, под операцией взятия остатка по модулю 232–1 *там* понимается не то же самое, что и в математике. Отличие заключается в том, что согласно ГОСТу (232–1)*mod*(232–1)=(232–1), а не 0. На самом деле, это упрощает реализацию формулы, а математически корректное выражение для нее приведено выше.

* период повторения последовательности для младшей части составляет 232, для старшей части 232–1, для всей последовательности период составляет 232(232–1), доказательство этого факта, весьма несложное, получите сами. Первая формула из двух реализуется за одну команду, вторая, несмотря на ее кажущуюся громоздкость, за две команды на всех современных 32-разрядных процессорах – первой командой идет обычное сложение по модулю 232 с запоминанием бита переноса, а вторая команда прибавляет бит переноса к полученному значению.

Схема алгоритма шифрования в режиме гаммирования приведена на рисунке 4, ниже изложены пояснения к схеме:

  
*Рисунок 4. Алгоритм зашифрования (расшифрования) данных в режиме гаммирования.*

**Шаг 0**

Определяет исходные данные для основного шага криптопреобразования:

* *T* о(ш) – массив открытых (зашифрованных) данных произвольного размера, подвергаемый процедуре зашифрования (расшифрования), по ходу процедуры массив подвергается преобразованию порциями по 64 бита;
* ***S***– ***синхропосылка***, 64-битовый элемент данных, необходимый для инициализации генератора гаммы;

**Шаг 1**

Начальное преобразование синхропосылки, выполняемое для ее «рандомизации», то есть для устранения статистических закономерностей, присутствующих в ней, результат используется как начальное заполнение РГПЧ;

**Шаг 2**

Один шаг работы РГПЧ, реализующий его рекуррентный алгоритм. В ходе данного шага старшая (*S* 1) и младшая (*S* 0) части последовательности данных вырабатываются независимо друг от друга;

**Шаг 3**

Гаммирование. Очередной 64-битовый элемент, выработанный РГПЧ, подвергается процедуре зашифрования по циклу 32–З, результат используется как элемент гаммы для зашифрования (расшифрования) очередного блока открытых (зашифрованных) данных того же размера.

**Шаг 4**

Результат работы алгоритма – зашифрованный (расшифрованный) массив данных.

Ниже перечислены особенности гаммирования как режима шифрования:

1. Одинаковые блоки в открытом массиве данных дадут при зашифровании различные блоки шифртекста, что позволит скрыть факт их идентичности.
2. Поскольку наложение гаммы выполняется побитно, шифрование неполного блока данных легко выполнимо как шифрование битов этого неполного блока, для чего используется соответствующие биты блока гаммы. Так, для зашифрования неполного блока в 1 бит согласно стандарту следует использовать самый младший бит из блока гаммы.
3. Синхропосылка, использованная при зашифровании, каким-то образом должна быть передана для использования при расшифровании. Это может быть достигнуто следующими путями:

* хранить или передавать синхропосылку вместе с зашифрованным массивом данных, что приведет к увеличению размера массива данных при зашифровании на размер синхропосылки, то есть на 8 байт;
* использовать предопределенное значение синхропосылки или вырабатывать ее синхронно источником и приемником по определенному закону, в этом случае изменение размера передаваемого или хранимого массива данных отсутствует;

Оба способа дополняют друг друга, и в тех редких случаях, где не работает первый, наиболее употребительный из них, может быть использован второй, более экзотический. Второй способ имеет гораздо меньшее применение, поскольку сделать синхропосылку предопределенной можно только в том случае, если на данном комплекте ключевой информации шифруется заведомо не более одного массива данных, что бывает не так уж часто. Генерировать синхропосылку синхронно у источника и получателя массива данных также не всегда представляется возможным, поскольку требует жесткой привязки к чему-либо в системе. Так, здравая на первый взгляд идея использовать в качестве синхропосылки в системе передачи зашифрованных сообщений номер передаваемого сообщения не подходит, поскольку сообщение может потеряться и не дойти до адресата, в этом случае произойдет рассинхронизация систем шифрования источника и приемника. Поэтому в рассмотренном случае нет альтернативы передаче синхропосылки вместе с зашифрованным сообщением.

С другой стороны, можно привести и обратный пример. Допустим, шифрование данных используется для защиты информации на диске, и реализовано оно на низком уровне, для обеспечения независимого доступа данные шифруются по секторам. В этом случае невозможно хранить синхропосылку вместе с зашифрованными данными, поскольку размер сектора нельзя изменить, однако ее можно вычислять как некоторую функцию от номера считывающей головки диска, номера дорожки (цилиндра) и номера сектора на дорожке. В этом случае синхропосылка привязывается к положению сектора на диске, которое вряд ли может измениться без переформатирования диска, то есть без уничтожения данных на нем.

Режим гаммирования имеет еще одну интересную особенность. В этом режиме биты массива данных шифруются независимо друг от друга. Таким образом, каждый бит шифртекста зависит от соответствующего бита открытого текста и, естественно, порядкового номера бита в массиве: http://kaf401.rloc.ru/Criptfiles/gost28147/20form.jpg. Из этого вытекает, что изменение бита шифртекста на противоположное значение приведет к аналогичному изменению бита открытого текста на противоположный:

http://kaf401.rloc.ru/Criptfiles/gost28147/21form.jpg,

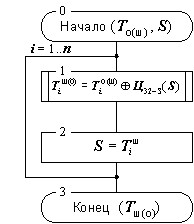
где http://kaf401.rloc.ru/Criptfiles/gost28147/22form.jpg обозначает инвертированное по отношению к *t* значение бита (http://kaf401.rloc.ru/Criptfiles/gost28147/221form.jpg).

Данное свойство дает злоумышленнику возможность воздействуя на биты шифртекста вносить предсказуемые и даже целенаправленные изменения в соответствующий открытый текст, получаемый после его расшифрования, не обладая при этом секретным ключом. Это иллюстрирует хорошо известный в криптологии факт, что ***секретность и аутентичность суть различные свойства******криптографических систем***. Иными словами, свойства криптосистемы обеспечивать защиту от несанкционированного ознакомления с содержимым сообщения и от несанкционированного внесения изменений в сообщение являются независимыми и лишь в отдельных случаях могут пересекаться. Сказанное означает, что существуют криптографические алгоритмы, обеспечивающие определенную секретность зашифрованных данных и при этом никак не защищающие от внесения изменений и наоборот, обеспечивающие аутентичность данных и никак не ограничивающие возможность ознакомления с ними. По этой причине рассматриваемое свойство режима гаммирования не должно рассматриваться как его недостаток.

**Гаммирование с обратной связью.**

Данный режим очень похож на режим гаммирования и отличается от него только способом выработки элементов гаммы – очередной элемент гаммы вырабатывается как результат преобразования по циклу 32-З предыдущего блока зашифрованных данных, а для зашифрования первого блока массива данных элемент гаммы вырабатывается как результат преобразования по тому же циклу синхропосылки. Этим достигается зацепление блоков – каждый блок шифртекста в этом режиме зависит от соответствующего и всех предыдущих блоков открытого текста. Поэтому данный режим иногда называется ***гаммированием с зацеплением блоков***. На стойкость шифра факт зацепления блоков не оказывает никакого влияния.

Схема алгоритмов за- и расшифрования в режиме гаммирования с обратной связью приведена на рисунке 5 и ввиду своей простоты в комментариях не нуждается.

  
*Рисунок 5. Алгоритм зашифрования (расшифрования) данных в режиме гаммирования с обратной связью.*

Шифрование в режиме гаммирования с обратной связью обладает теми же особенностями, что и шифрование в режиме обычного гаммирования, за исключением влияния искажений шифртекста на соответствующий открытый текст. Для сравнения запишем функции расшифрования блока для обоих упомянутых режимов:

http://kaf401.rloc.ru/Criptfiles/gost28147/24form.jpg, гаммирование;

http://kaf401.rloc.ru/Criptfiles/gost28147/25form.jpg, гаммирование с обратной связью;

Если в режиме обычного гаммирования изменения в определенных битах шифртекста влияют только на соответствующие биты открытого текста, то в режиме гаммирования с обратной связью картина несколько сложнее. Как видно из соответствующего уравнения, при расшифровании блока данных в режиме гаммирования с обратной связью, блок открытых данных зависит от соответствующего и предыдущего блоков зашифрованных данных. Поэтому, если внести искажения в зашифрованный блок, то после расшифрования искаженными окажутся два блока открытых данных – соответствующий и следующий за ним, причем искажения в первом случае будут носить тот же характер, что и в режиме гаммирования, а во втором случае – как в режиме простой замены. Другими словами, в соответствующем блоке открытых данных искаженными окажутся те же самые биты, что и в блоке шифрованных данных, а в следующем блоке открытых данных все биты независимо друг от друга с вероятностью 1*/*2 изменят свои значения.

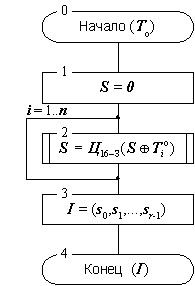
**Выработка имитовставки к массиву данных.**

В предыдущих разделах мы обсудили влияние искажения шифрованных данных на соответствующие открытые данные. Мы установили, что при расшифровании в режиме простой замены соответствующий блок открытых данных оказывается искаженным непредсказуемым образом, а при расшифровании блока в режиме гаммирования изменения предсказуемы. В режиме гаммирования с обратной связью искаженными оказываются два блока, один предсказуемым, а другой непредсказуемым образом. Значит ли это, что с точки зрения защиты от навязывания ложных данных режим гаммирования является плохим, а режимы простой замены и гаммирования с обратной связью хорошими? – Ни в коем случае. При анализе данной ситуации необходимо учесть то, что непредсказуемые изменения в расшифрованном блоке данных могут быть обнаружены только в случае избыточности этих самых данных, причем, чем больше степень избыточности, тем вероятнее обнаружение искажения. Очень большая избыточность имеет место, например, для текстов на естественных и искусственных языках, в этом случае факт искажения обнаруживается практически неизбежно. Однако в других случаях, например, при искажении сжатых оцифрованных звуковых образов, мы получим просто другой образ, который сможет воспринять наше ухо. Искажение в этом случае останется необнаруженным, если, конечно, нет априорной информации о характере звука. Вывод здесь такой: поскольку способность некоторых режимов шифрования обнаруживать искажения, внесенные в шифрованные данные, существенным образом опирается на наличие и степень избыточности шифруемых данных, эта способность не является имманентным свойством соответствующих режимов и не может рассматриваться как их достоинство.

Для решения задачи обнаружения искажений в зашифрованном массиве данных с заданной вероятностью в ГОСТе предусмотрен дополнительный режим криптографического преобразования – выработка имитовставки. Имитовставка – это контрольная комбинация, зависящая от открытых данных и секретной ключевой информации. Целью использования имитовставки является обнаружение всех случайных или преднамеренных изменений в массиве информации. Проблема, изложенная в предыдущем пункте, может быть успешно решена с помощью добавления к шифрованным данным имитовставки. Для потенциального злоумышленника две следующие задачи практически неразрешимы, если он не владеет ключом:

* вычисление имитовставки для заданного открытого массива информации;
* подбор открытых данных под заданную имитовставку;

Схема алгоритма выработки имитовставки приведена на рисунке 6.

  
*Рисунок 6. Алгоритм выработки имитовставки для массива данных.*

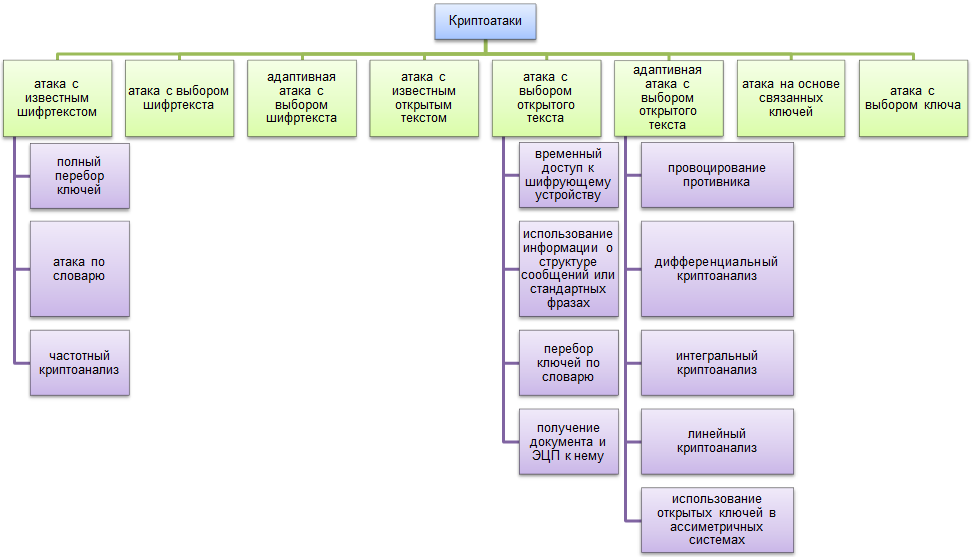
В качестве имитовставки берется часть блока, полученного на выходе, обычно – 32 его младших бита. При выборе размера имитовставки надо принимать во внимание, что вероятность успешного навязывания ложных данных равна величине 2–|***I***| на одну попытку подбора, если в распоряжении злоумышленника нет более эффективного метода подбора, чем простое угадывание. При использовании имитовставки размером 32 бита эта вероятность равна http://kaf401.rloc.ru/Criptfiles/gost28147/symb12.jpg

1. **Основы криптоанализа. Парадокс «Дней Рождения»**

Криптография – наука о методах обеспечения конфиденциальности (невозможности прочтения информации посторонним) и аутентичности (целостности и подлинности авторства, а также невозможности отказа от авторства) информации.

Криптоанализ – наука, занимающаяся вопросами оценки сильных и слабых сторон методов шифрования, а также разработкой методов, позволяющих взламывать криптосистемы.

Классификация криптоатак приведена на следующем рисунке

****

I. Атака с известным шифртекстом (ciphertext only attack). Предполагается, что противник знает алгоритм шифрования, но не знает секретный ключ. Кроме того, в его распоряжении имеется набор перехваченных шифрограмм. Разновидности:

- полный перебор ключей;

- атака по словарю, перебор ключей по словарю (dictionary attack). Использование Интернет-червя в 1988 г. предоставляет хорошо документированный случай взлома паролей. Интернет-червь пытался взломать пароли, работая с серией словарей. На первом этапе атаки было использовано множество слов, содержащее имена пользователей, взятых из файла паролей системы Unix. Если это не имело успеха, использовался внутренний словарь 432 общепринятых, используемых в Интернет-жаргоне, слов. Если второй этап не имел успеха, использовался Unix словарь, состоящий из 24474 слов. Червь также проверял на пустой пароль. Сайты, на которые производилась атака, сообщили, что около 50 % паролей было успешно взломано, используя данную стратегию.

- частотный криптоанализ - метод вскрытия шифра, основывающийся на предположении о существовании зависимости между частотой появления символов в открытых сообщениях и соответствующих шифрозамен в шифрограммах. Метод частотного криптоанализа известен с IX-го века (работы Ал-Кинди), хотя наиболее известным случаем его применения в реальной жизни, возможно, является дешифровка египетских иероглифов Ж.-Ф. Шампольоном в 1822 г. В художественной литературе наиболее известными упоминаниями являются рассказы «Золотой жук» Эдгара По, «Пляшущие человечки» Конан Дойля, а также роман «Дети капитана Гранта» Жюль Верна [17].

II. Атака с выбором шифртекста (chosen ciphertext attack). Криптоаналитик имеет возможность выбрать необходимое количество криптограмм и получить соответствующие им открытые тексты. Криптоаналитик может воспользоваться устройством расшифрования один или несколько раз для получения шифротекста в расшифрованном виде. Используя полученные данные, он может попытаться восстановить секретный ключ для расшифровки.

III. Адаптивная атака с выбором шифртекста (adaptive chosen ciphertext attack). Криптоаналитик имеет возможность выбирать новые шифрограммы для расшифровки с учетом того, что ему известна некоторая информация из предыдущих сообщений. В некоторых криптографических протоколах при получении шифрограммы, несоответствующей стандарту (содержащей ошибки), отправитель получает ответное сообщение, иногда с детализированным описанием этапа проверки и причины возникновения ошибки. Криптоаналитик может использовать эту информацию для последовательной посылки и уточнения параметров криптосистемы.

IV. Атака с известным открытым текстом (known plaintext attack). То же, что и предыдущая, но противник для некоторых шифрограмм получает в свое распоряжение соответствующие им открытые сообщения. Во время «холодной войны» американские дипломаты в Москве пользовались шифровальными машинами, получившими кодовое название «бриллианты». Оборудование располагалось на девятом этаже здания посольства США в особо защищенном помещении, называвшемся узлом связи. Внутри этого помещения была еще одна комната, где шифровальные машины зашифровывали и расшифровывали шифрограммы, которыми обменивались посольство и штаб-квартира ЦРУ в Лэнгли (штат Виргиния). В другой части узла связи другие машины передавали перехваченную АНБ информацию или вели обмен сообщениями между Госдепартаментом и его представителями в Москве. Чтобы обеспечить как можно более полную защиту этого помещения, были предприняты особые меры безопасности в отношении его стен, электропроводки, линий электропитания и подачи воздуха. Сами шифровальные машины представляли собой новейшую разработку АНБ с цифровыми кодами на магнитных лентах, которые менялись каждые 24 часа. Кроме того, каждое сообщение пропускалось через две машины, зашифровываясь дважды. Все это создавало впечатление неуязвимости как узла связи, так и «бриллиантов». Но, несмотря на все меры безопасности, русские сумели тайно подменить «чистые» кабели электропитания шифровальных машин. Таким образом они смогли обойти электронные фильтры защиты и улавливали сигналы напрямую с «бриллиантов». В печатающих устройствах шифровальных машин были заменены печатные платы и кремниевые микросхемы, что позволило агентам КГБ снимать и записывать незашифрованные сигналы с линий связи и принтеров. Получив в свое распоряжение открытый текст, они имели возможность затем сравнить его с зашифрованными сообщениями и найти ключи шифрования. Сверхсекретное шифровальное оборудование в Москве превратилось в огромное подслушивающее устройство [43].

V. Атака с выбором открытого текста (chosen plaintext attack). Криптоаналитик обладает определённым числом открытых текстов и соответствующих шифротекстов, кроме того, он имеет возможность зашифровать несколько предварительно выбранных открытых текстов (до начала атаки). Разновидности [17]:

- временный доступ к шифрующему устройству - американским спецслужбам удалось, подкупив охрану, выкрасть из японского посольства шифровальную машину на два дня. Возможность взлома машины была исключена, так как это бы привело к раскрытию операции и к смене всех ключей. Зато, американцы получили возможность осуществить атаку японского шифра на основе подобранного открытого текста;

- использование информации о структуре сообщений или стандартных фразах – криптоаналитики из Блетчли-Парка могли определить открытый текст сообщений в зависимости от того, когда эти сообщения были посланы. Например, ежедневный отчет о погоде посылался немецкими связистами в одно и то же время. По той причине, что военные доклады имеют определенную структуру, криптоаналитикам удавалось расшифровывать остальную информацию, пользуясь данными о погоде в той местности. Другим ярким примером являлись перехваченные сообщения офицера африканского корпуса, который постоянно отсылал «Нечего докладывать». Другие операторы тоже часто использовали стандартные ответы или приветствия;

- перебор ключей по словарю (dictionary attack) – криптоаналитик шифрует слова и фразы, наличие которых предполагается в шифрограмме, с использованием различных ключей. Совпадение зашифрованных слов и фраз с частями шифрограммы может говорить об обнаружении ключа;

- получение документа и ЭЦП к нему, сгенерированной с помощью закрытого ключа.

VI. Адаптивная атака с выбором открытого текста (adaptive chosen plaintext attack). Криптоаналитик имеет возможность выбирать новые открытые тексты с учетом того, что ему известна некоторая информация из предыдущих сообщений. Т.е. он может получить пары «открытое сообщение – шифрограмма», в т.ч. и после начала атаки. Разновидности [17]:

- провоцирование противника на использование в сообщениях определенных слов или фраз – во время Второй мировой в Блетчли-Парк был придуман следующий способ заставить немцев отсылать определенные сообщения. По просьбе криптоаналитиков Королевские военновоздушные силы Великобритании минировали определенные участки Северного моря, этот процесс был назван «Садоводством» (англ. «gardening»). Практически сразу после этого немцами посылались зашифрованные сообщения, включающие слово «мины» и названия мест, где они были сброшены;

- дифференциальный криптоанализ – метод вскрытия симметричных блочных шифров (и других криптографических примитивов, в частности, хеш-функций), основанный на изучении разностей между шифруемыми значениями на различных раундах для пары подобранных открытых сообщений при их шифровании с одним и тем же ключом. Предложен в 1990 г. израильскими специалистами Эли Бихамом и Ади Шамиром. Является статистической атакой, в результате работы которой предлагается список наиболее вероятных ключей шифрования. Дифференциальный криптоанализ применим для взлома DES, FEAL и некоторые других шифров, как правило, разработанных ранее начала 90-х. Количество раундов современных шифров (AES, Camellia и др.) рассчитывалось с учетом обеспечения стойкости, в т.ч. и к дифференциальному криптоанализу;

- интегральный криптоанализ – аналогичен дифференциальному криптоанализу, но в отличие от него рассматривает воздействие алгоритма не на пару, а сразу на множество открытых текстов. Впервые применен в 1997 г. Ларсом Кнудсеном;

- линейный криптоанализ – метод вскрытия шифра, использующий линейные приближения для описания его работы. Линейный криптоанализ был изобретён японским криптологом Мицуру Мацуи (Mitsuru Matsui). Предложенный им в 1993 г. алгоритм был изначально направлен на вскрытие DES и FEAL. Впоследствии линейный криптоанализ был распространён и на другие алгоритмы. На сегодняшний день наряду с дифференциальным криптоанализом является одним из наиболее распространённых методов вскрытия блочных шифров. Разработаны атаки на блочные и потоковые шифры. Криптоанализ выполняется в два шага. Первый – построение линейных соотношений между открытым текстом, шифрограммой и ключом, которые справедливы с высокой вероятностью. Второй – использование этих соотношений вместе с известными парами «открытый текст – шифрограмма» для получения битов ключа;

- использование открытых ключей в асиметричных системах – криптоаналитик имеет возможность получить шифротекст, соответствующий выбранному сообщению, на основе открытого ключа.

VII. Атака на основе связанных ключей (related key attack). Криптоаналитик знает не сами ключи, а некоторые различия (соотношения) между ними. Множество реальных систем используют разные ключи, связанные известным соотношением. Например, для каждого нового сообщения предыдущее значение ключа увеличивается на единицу.

VIII. Атака с выбором ключа (chosen key attack). Криптоаналитик задает часть ключа, а на оставшуюся часть ключа выполняет атаку на основе связанных ключей.

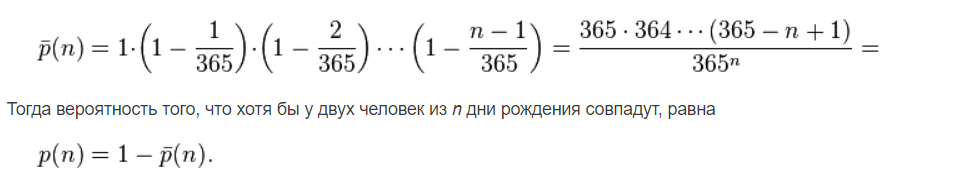
**Парадокс Дней Рождения**

Парадо́кс дней рожде́ния — утверждение, гласящее, что если дана группа из 23 или более человек, то вероятность того, что хотя бы у двух из них дни рождения (число и месяц) совпадут, превышает 50 %. Для группы из 60 или более человек вероятность совпадения дней рождения хотя бы у двух её членов составляет более 99 %, хотя 100 % она достигает, только когда в группе не менее 367 человек (с учётом високосных лет).

Такое утверждение может показаться противоречащим здравому смыслу, так как вероятность одному родиться в определённый день года довольно мала, а вероятность того, что двое родились в конкретный день — ещё меньше, но является верным в соответствии с теорией вероятностей. Таким образом, оно не является парадоксом в строгом научном смысле — логического противоречия в нём нет, а парадокс заключается лишь в различиях между интуитивным восприятием ситуации человеком и результатами математического расчёта.

В данном примере для расчёта вероятности того, что в группе из n человек как минимум у двух дни рождения совпадут, примем, что дни рождения распределены равномерно, то есть нет високосных лет, близнецов, рождаемость не зависит от дня недели, времени года и других факторов. В действительности, это не совсем так — обычно летом рождается больше детей; кроме того, в некоторых странах из-за особенностей работы больниц больше детей рождается в определённые дни недели. Однако неравномерность распределения может лишь увеличить вероятность совпадения дней рождения, но не уменьшить: если бы все люди рождались только в 3 дня из 365, то вероятность совпадения дней рождения была бы очень высокой.

Рассчитаем сначала, какова вероятность p (n) того, что в группе из n человек дни рождения всех людей будут различными. Если n > 365, то в силу принципа Дирихле вероятность равна нулю. Если же n ≤ 365, то будем рассуждать следующим образом. Возьмём наугад одного человека из группы и запомним его день рождения. Затем возьмём наугад второго человека, при этом вероятность того, что у него день рождения не совпадёт с днем рождения первого человека, равна 1 — 1/365. Затем возьмём третьего человека, при этом вероятность того, что его день рождения не совпадёт с днями рождения первых двух, равна 1 — 2/365. Рассуждая по аналогии, мы дойдём до последнего человека, для которого вероятность несовпадения его дня рождения со всеми предыдущими будет равна 1 — (n — 1)/365. Перемножая все эти вероятности, получаем вероятность того, что все дни рождения в группе будут различными:



Значение этой функции превосходит 1/2 при n = 23 (при этом вероятность совпадения равна примерно 50.7 %). Вероятности для некоторых значений n иллюстрируются следующей таблицей:

n p (n)

10 12 %

20 41 %

30 70 %

50 97 %

100 99.99996 %

200 99.9999999999999999999999999998 %

300 (1 − 7×10−73) × 100 %

350 (1 − 3×10−131) × 100 %

366 100 %

1. **Понятие криптографического протокола. Известные протоколы**

**Протокол** – это порядок действий, предпринимаемых двумя или более сторонами, предназначенный для решения определенной задачи. **Криптографический протокол** – это протокол, использующий криптографию. Криптографические методы в протоколе применяются для в предотвращении или обнаружении вредительства и мошенничества.

Условные обозначения участников протоколов:

Алиса – Первый участник всех протоколов

Боб – Второй участник всех протоколов

Мэллори – Взломщик протоколов

Трент – Заслуживающий доверия посредник.

Криптографические попытки взлома могут быть направлены против криптографических алгоритмов, используемых в протоколах, против криптографических методов, используемых для реализации алгоритмов и протоколов или непосредственно против протоколов.

**Протокол обмена сеансовыми ключами, использующий криптографию с открытыми ключами**

1. Алиса получает открытый ключ Боба из центра распределения ключей.
2. Алиса генерирует случайный сеансовый ключ, зашифровывает его открытым ключом Боба и посылает его Бобу.
3. Боб расшифровывает сообщение Алисы с помощью своего закрытого ключа.
4. Алиса и Боб шифруют свой обмен информацией этим сеансовым ключом.

**Вскрытие "человек-в-середине"**

Вскрытие осуществляется следующим образом:

1. Алиса посылает Бобу свой открытый ключ. Мэллори перехватывает его и посылает Бобу свой собственный открытый ключ.
2. Боб посылает Алисе свой открытый ключ. Мэллори перехватывает его и посылает Алисе собственный открытый ключ.
3. Когда Алиса посылает сообщение Бобу, зашифрованное подмененным открытым ключом "Боба", Мэллори перехватывает его, расшифровывает своим закрытым ключом его, снова зашифровывает открытым ключом Боба и посылает Бобу. Аналогично осуществляется перехват сообщений посланных Бобом Алисе.

**Протокол «держась за руки»**

Протокол "держась за руки", позволяет избежать вскрытия «человек-в-середине». Вот как он работает:

1. Алиса посылает Бобу свой открытый ключ.
2. Боб посылает Алисе свой открытый ключ.
3. Алиса зашифровывает свое сообщение открытым ключом Боба и отправляет Бобу половину зашифрованного сообщения.
4. Боб зашифровывает свое сообщение открытым ключом Алисы и отправляет ей половину зашифрованного сообщения.
5. Алиса отправляет Бобу вторую половину зашифрованного сообщения.
6. Боб складывает две части сообщения Алисы и расшифровывает его с помощью своего закрытого ключа, а затем отправляет Алисе вторую половину своего зашифрованного сообщения.
7. Алиса складывает две части сообщения Боба и расшифровывает его с помощью своего закрытого ключа.

Суть метода заключается в том, что половина зашифрованного сообщения не может быть дешифрирована без второй половины. Боб не сможет прочитать ни одной части сообщения Алисы до этапа (6), а Алиса до этапа (7). Но и Мелори, перехватив половину сообщения Алисы на этапе (3), не сможет расшифровать ее своим закрытым ключом и снова зашифровать открытым ключом Боба.

Примеры разбиения на части:

* Передача четных и нечетных битов
* Первая половина сообщения является хэш-функцией шифрованного сообщения, а во вторая половина – собственно шифрованным сообщением.

**Разделение секрета.**

Если необходимо хранить некоторый секрет так, чтобы воспользоваться им могла только определенная группа людей, собравшись вместе, то применяется протокол разделения секрета.

Например Трент желает разделить сообщение между Алисой и Бобом:

1. Трент генерирует строку случайных битов, *R*, такой же длины, что и сообщение, *M*.
2. Трент выполняет "исключающее или" (XOR) над *M* и *R*, создавая *S*.
3. *R ******M = S*
4. Трент передает Алисе *R*, а Бобу - *S*.
5. Чтобы получить сообщение, Алисе и Бобу нужно выполнить единственное действие
6. Алиса и Боб выполняют операцию над имеющимися у них частями, восстанавливая сообщение *R ******S = M*

Для расширения схемы на большее число людей необходимо выполнить операцию XOR с большим числом строк случайных битов. В следующем примере Трент делит сообщение на четыре части:

*M ******R ******S ******T = U.*

*R*, *S*, *T*, *U* передается держателям секрета. Для восстановления необходимо выполнить

*R ******S ******T ******U = M.*

Однако, если любая из частей будет потеряна, восстановить сообщение будет уже невозможно.

В практическом плане применяется также разделение ключей для цифровой подписи, например чтобы три заместителя могли подписать контракт в отсутствии начальника.

**Метки времени**

Во многих ситуациях нужно убедиться, что определенный документ уже существовал в определенный момент времени. Например, при споре об авторских правах или патенте. Для этой применяется следующий протокол меток времени:

1. Алиса вычисляет значение однонаправленной хэш-функции для документа.
2. Алиса передает это значение Тренту.
3. Трент добавляет время и дату получения этого значения и затем подписывает результат цифровой подписью.
4. Трент отправляет подписанное значение хэш-функции вместе с меткой времени Алисе.

Достоинство этого протокола в том, что Трент не знает тайны подписываемого документа и не обязан хранить ни копию самого документа, ни копию хэш-функции.

**Пример протокола защиты базы данных**

Существует открытая база данных адресов сотрудников, любой сотрудник может получить адрес коллеги, по его фамилии, однако получить список адресов всех сотрудников, например с целью рассылки спама, должно быть невозможно.

Выбирается однонаправленная хэш-функцию и симметричный алгоритм шифрования. У каждой записи в базе данных два поля. Индексным полем является фамилия сотрудника, обработанная хэш-функцией. Поле данных адреса шифруется с помощью используемой в качестве ключа фамилии. Не зная фамилии, невозможно расшифровать поле данных.

Для поиска по фамилии, она сначала хэшируется, и выполняется поиск значения хэш-функции в базе данных, затем расшифровывается поле адреса.

1. **Понятие вируса, история, причины возникновения.**

**Компьютерным вирусом** называется программа (некоторая совокупность исполняемого кода и данных), которая обладает способностью создавать свои копии (не обязательно полностью совпадающие с оригиналом) и внедрять их в различные объекты и ресурсы компьютерных систем, сетей и т.д. без ведома пользователя. При этом копии сохраняют способность дальнейшего распространения.

«Доисторический» (70-80 гг)

* Первые теоретические труды
* Вирусы-легенды
* Возникающие инциденты

«До-интернетовский» (80-90 гг)

* Появление первых вирусов
* Классические вирусы MS-DOS

Интернет-этап (90-2000 гг)

* Черви
* Трояны

Современный этап (2000 - современность)

* Криминализация
* Использование интернета в преступных целях

 Как это ни странно, но основную массу вирусных программ пишут школьники и студенты, которые только что изучили язык Assembler. Они просто пробуют свои силы, не найдя для них более достойного применения. Эти вирусы пишутся в основном для самоутверждения.

Во второй группе присутствуют также молодые люди, которые пишут вирусы уже не ради самоутверждения, а исключительно из компьютерного хулиганства. Став опытнее, они становятся талантливыми программистами и пишут уже более опасные вирусные программы. Причиной, заставляющей писать эти программы, является обычно комплекс неполноценности. Плюс к этому еще бывает и неуравновешенная психика.

Еще одна группа авторов вирусов – это так называемые «исследователи». В этой группе состоят довольно толковые программисты, которые занимаются изобретением методов скрытия, заражения и противодействия антивирусам. Они же придумывают и способы появления вирусов в новых операционных системах. Эти программисты пишут вирусные программы не ради вирусов, а ради того, чтобы исследовать тонкости компьютера. Обычно эти программы так и остаются неиспользованными. Но попав в руки программистов из предыдущей группы, эти задумки очень быстро претворяются в жизнь в новых вирусах.

**Программа Creeper**

Как отмечается на сайте [Discovery](http://news.discovery.com/tech/first-computer-virus-creeper-was-no-bug-110316.htm), программа Creeper, о которой часто говорят как о первом вирусе, была создана в 1971 году сотрудником компании BBN Бобом Томасом. По факту, Creeper был создан как тестовая программа, чтобы проверить, возможна ли в принципе самовоспроизводящаяся программа. Оказалось, что в некотором смысле возможна. Заразив новый жесткий диск, Creeper пытался удалить себя с предыдущего компьютера. Creeper не совершал никаких вредоносных действий – он только выводил простое сообщение: "I'M THE CREEPER. CATCH ME IF YOU CAN!" (Я CREEPER. ПОЙМАЙ МЕНЯ, ЕСЛИ СМОЖЕШЬ!)

**Вирус Rabbit**

Согласно сайту [InfoCarnivore](http://infocarnivore.com/the-very-first-viruses-creeper-wabbit-and-brain/), вирус Rabbit (также известный как Wabbit) был создан в 1974 г. с вредоносной целью и мог самовоспроизводиться. Попав на компьютер, он делал большое количество копий себя, значительно ухудшал работоспособность системы и в итоге приводил к отказу компьютера. Имя («Кролик») было дано вирусу из-за того, что он очень быстро самовоспроизводился.

**Первый троянец**

Согласно сайту [Fourmilab](http://www.fourmilab.ch/documents/univac/animal.html), первый троянец по названием ANIMAL (хотя есть [споры](https://securelist.com/threats/1970s/)относительно того, были ли это действительно троянец или просто вирус) был разработан компьютерным программистом Джоном Уолкером в 1975 г. В то время были очень популярны компьютерные игры, в которых пользователь загадывал какое-нибудь животное, а программа должна была его угадать за 20 вопросов. Уолкер написал одну из таких игр, и она стала популярной. Чтобы поделиться ее со своими друзьями, Уолкер записывал и передавал ее на магнитной ленте. Чтобы упростить эту процедуру, Уолкер создал программу PERVADE, которая устанавливалась на компьютер вместе с игрой ANIMAL. Пока пользователь играл в игру, PREVADE проверял все доступные пользователю директории на компьютере, а затем копировал ANIMAL во все директории, где этой программы не было. Вредоносной цели здесь не было, но ANIMAL и PREVADE подпадают под определение троянца: по сути, внутри программы ANIMAL была запрятана другая программа, которая выполняла действия без согласия пользователя.

**Вирус загрузочного сектора Brain**

Brain, первый вирус для IBM-совместимых компьютеров, появился в 1986 году – он заражал пятидюймовые дискеты. Как сообщает [Securelist](https://securelist.ru/threats/1980-e/), вирус был написан двумя братьями – Баситом и Амджадом Фаруком Алви, которые держали компьютерный магазин в Пакистане. Братьям надоело, что покупатели нелегально копировали купленное у них ПО, и они создали этот вирус, которые заражал загрузочные сектора дискет. Brain заодно оказался и первым вирусом-невидимкой: при обнаружении попытки чтения зараженного сектора диска вирус незаметно подставлял его незараженный оригинал. Также он записывал на дискету фразу «(c) Brain», но при этом не портил никаких данных.

**Вирус ILoveYou**

В начале 21 века появился надежный высокоскоростной интернет-доступ, и это изменило методы распространения вредоносных программ. Теперь они не были ограничены дискетами и корпоративными сетями и могли очень быстро распространяться через электронную почту, популярные веб-сайты и даже напрямую через интернет. Начало формироваться вредоносное ПО в современном виде. Ландшафт угроз оказался заселенным вирусами, червями и троянцами. Возник собирательный термин «вредоносное ПО». Одна из самых серьезных эпидемий новой эры была вызвана червем ILoveYou, который появился 4 мая 2000 г.

Как указывает [Securelist](https://securelist.ru/threats/2001/), ILoveYou следовал модели ранее существовавших вирусов, распространявшихся по почте. При этом, в отличие от макровирусов, широко распространенных с 1995 года, ILoveYou распространялся не в виде зараженного документа Word, а в виде VBS-файла (такое расширение имеют скрипты, написанные в Visual Basic). Метод оказался простым и действенным – пользователи еще не привыкли остерегаться незапрошенных электронных писем. В качестве темы письма была строчка «I Love You», а в приложении к каждому письму был файл «LOVE-LETTER-FOR-YOU-TXT.vbs». По задумке создателя Онеля де Гузмана, червь стирал существующие файлы и поверх них записывал собственные копии, благодаря которым червь рассылался по всем адресам из списка контактов пользователя. Поскольку письма, как правило, приходили со знакомых адресов, получатели обычно открывали их – и заражали червем свой компьютер. Таким образом, ILoveYou на практике подтвердил эффективность методов социальной инженерии.

**Червь Code**

Червь Code Red был так называемым бестелесным червем – он существовал только в памяти и не предпринимал попыток заразить файлы в системе. Используя брешь в системе безопасности Microsoft Internet Information Server, червь всего за несколько часов распространился по всему миру и вызвал хаос, внедряясь в протоколы обмена информацией между компьютерами.

Как пишет сайт [Scientific American](http://www.scientificamerican.com/article/code-red-worm-assault-on/), зараженные компьютеры в итоге были использованы для проведения DDoS-атаки на веб-сайт Белого дома – Whitehouse.gov.

**Heartbleed**

Угроза Heartbleed появилась в 2014 г. и сразу поставила под угрозу множество интернет-серверов. В отличие от вирусов и червей, Heartbleed – это уязвимость в OpenSSL – криптографической библиотеке универсального применения, широко использующейся по всему миру. OpenSSL периодически рассылает соединенным устройствам специальные сигналы, подтверждающие актуальность соединения. Пользователи могут отослать некоторый объем данных и в ответ запросить такое же количество данных – например, отослать один байт и получить в ответ тоже один байт. Максимальное количество данных, отправляемых за один раз – 64 килобайта. Как объясняет специалист по безопасности [Брюс Шнайер](https://www.schneier.com/blog/archives/2014/04/heartbleed.html), пользователь может объявить, что отсылает 64 килобайта, а по факту отправить только один байт – в этом случае сервер в ответ пришлет 64 килобайта данных, хранящихся в его оперативной памяти, в которых может оказаться все что угодно – от имен пользователей до паролей и криптостойких ключей.

1. **Классификация вирусов. Вредоносное действие вируса**

Три условия существования вредоносных программ:

* Популярность - широкое распространение и известность данной системы
* Документированность - наличие разнообразной и достаточно полной документации по системе
* Незащищенность системы или существование известных уязвимостей в ее безопасности и приложениях

Способы проникновения вредоносных программ в систему:

* Социальная инженерия - тем или иным способом заставляют пользователя запустить заражённый файл или открыть ссылку на заражённый веб-сайт
* Технические приемы внедрения - осуществляется это через уязвимости в системе безопасности операционных систем и в программном обеспечении. Наличие уязвимостей позволяет изготовленному злоумышленником сетевому червю или троянской программе проникнуть в компьютер-жертву и самостоятельно запустить себя на исполнение без ведома пользователя
* Одновременное использование социальной инженерии и технических методов

Классификация вредоносных программ:

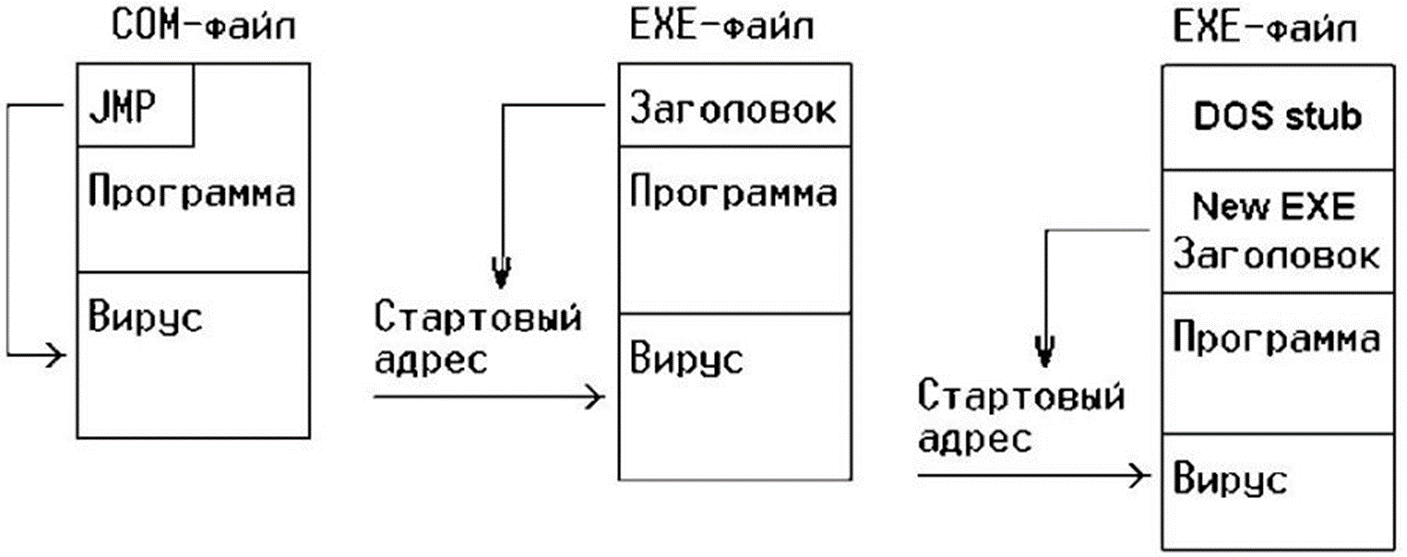
* Вирусы и черви - вредоносные программы, которые обладают способностью к несанкционированному пользователем саморазмножению в компьютерах или компьютерных сетях, при этом полученные копии также обладают этой возможностью.
* Троянские программы - вредоносные программы, которые созданы для осуществления несанкционированных пользователем действий, направленных на уничтожение, блокирование, модификацию или копирование информации, нарушение работы компьютеров или компьютерных сетей
* Вредоносные утилиты - программы, разработанные для автоматизации создания других вирусов, червей или троянских программ, организации DoS-атак на удаленные сервера, взлома других компьютеров и т.п.

Классификация компьютерных вирусов:

* **По среде обитания вируса**
  + Файловые вирусы
  + Загрузочные вирусы
  + Сетевые вирусы
  + Макро вирусы
  + Flash-вирусы
* **По способу заражения**
  + Резидентные вирусы
  + Нерезидентные вирусы
* **По деструктивным возможностям**
  + Безвредные
  + Нарушение работоспособности компьютера
  + Потеря или кража информации
  + Отказ работы «железа»

Особенности современных вирусов и алгоритмов их работы:

* **Stealth** (стелс, невидимость) – способность вируса заражать файлы скрытно, не давая пользователю повода заподозрить неладное
* **Polymorph** (полиморфизм) – способность вируса шифровать свое тело так, чтобы никакие две копии вируса не были похожи друг на друга
* **Armored** (защита, бронирование) – способность вируса сопротивляться отладке и дизассемблированию
* **Multipartite** (многосторонность) – способность вируса заражать и программы, и загрузочные сектора дисков



**Вредоносное воздействие (действие)** — любые действия вредоносных программ, приводящие к нарушению работоспособности компьютерных систем (в том числе за счет исчерпанию их ресурсов), потерям данных, получения несанкционированного доступа к данным или причинения иного ущерба владельцу или пользователю компьютерной системы. Распространение вредоносной программы также является вредоносным воздействием.

**Пояснение**

Уже сам факт распространения компьютерных вирусов и других вредоносных программ следует рассматривать как вредоносное воздействие.

Спектр вредоносных воздействий очень широк и зависит только от желаний и возможностей создателей вирусов. Ниже приведен далеко не полный список возможных вредоносных воздействий:

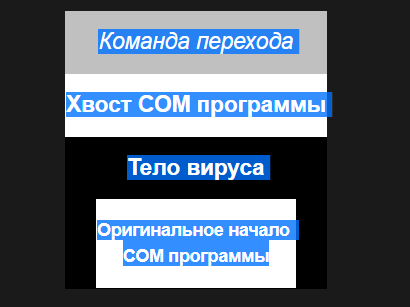
* визуальные и звуковые эффекты;
* удаление файлов и каталогов;
* изменение содержимого файлов и каталогов;
* изменение содержимого секторов диска;
* воздействие на базы данных;
* воздействие на аппаратное обеспечение компьютера;
* воздействие на систему в целом (замедление и полное блокирование работы компьютерных систем);
* получение несанкционированного доступа и похищение информации;
* компрометация пользователя;
* провоцирование пользователя

1. **Структура COM и EXE программ. Структура PSP. Область DTA**

Структура COM - файла проста . В файлах данного типа,обычно не имеющими даже заголовка файла,содержатся только машинный код и данные программы.

Размер COM - файла ограничен 64 кб, т.е. размером одного сегмента памяти

В основном COM файлы пишут на языке Ассемблера, но это не обязательно. Нописать файл можно на любом языке, который можно потом компилировать.

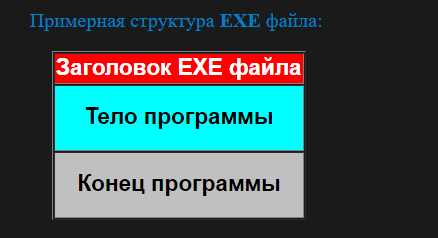


Структура EXE файла

COM файлы пишут в основном на языке Ассемблера, но они постепенно устаревают и на смену им приходят огромные по своим размерам и сложные по своей структуре EXE файлы.

Состоять EXE файлы могут из нескольких сегментов, следовательно их размер не ограничен 64 кб. По структуре EXE файл сложнее, кроме кода программы в файле также содержется: заголовок файла, таблица настройки адресов, данные и т.п.

Примерная структура EXE файла:



Заголовок EXE файла - содержет данные необходимые для загрузки программы

Тело программы - основная часть программы, выполняющая какие-либо полезные действия

Конец программы - завершающая часть программы, которая сохраняет нужные и удаляет ненужные данные из ОЗУ, закрывает все открытые данной программой файлы и т.п.

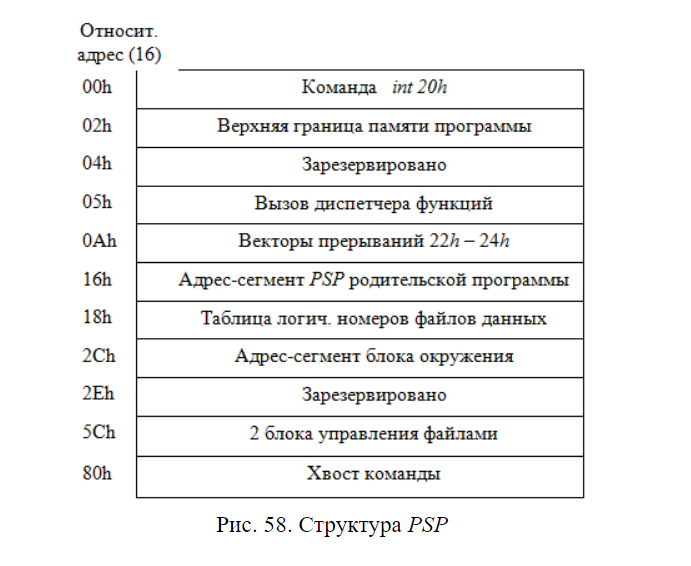
Первая и последняя часть программы являются обязательными для всех EXE файлов, а вторая часть вовсе не обязательна. Вы можете просто взять её и пропустить.

Приводить виды "инфицированного" и "не инфицированного" EXE файла я не буду, т.к. в отличие от COM вирусов, у EXE вирусов есть множество способов заражения. Подробно о каждом из них я расскажу Вам в следующих выпусках рассылки, а сейчас только перечислю названия основных из них:

* OVERWRITE - вирусы, замещающие программный код
* COMPANION - вирусы-спутники
* PARASITIC - вирусы, внедряющиеся в программу
* метод переименования EXE файла
* внедрение способом переноса
* внедрение способом сдвига

Структура psp

Префикс программного сегмента PSP (Program Segment Prefics) – специальная область оперативной памяти размером 256 (100h) байт. PSP может использоваться в программе для определения имен файлов и параметров из командной строки, введенной при запуске программы на выполнение, объема доступной памяти, переменных окружения системы и так далее. После загрузки программы в память сегментные регистры DS и ES указывают на начало PSP этой программы.



Вызов этой функции заполняет данными область памяти DTA (область передачи данных), которая начинается по умолчанию со смещения 0080h от начала блока данных PSP (при запуске СОМ- и ЕХЕ-программ сегменты DS и ES содержат сегментный адрес начала PSP), но ее можно переопределить с помощью функции 1Ah.

Функции поиска файлов заполняют DTA следующим образом:

* +00h: байт — биты 0 – 6: ASCII-код буквы диска; бит 7: диск сетевой
* +01h: 11 байт — маска поиска (без пути)
* +0СН: байт — атрибуты для поиска
* +0Dh: слово — порядковый номер файла в каталоге
* +0Fh: слово — номер кластера начала внешнего каталога
* +11h: 4 байта — зарезервировано
* +15h: байт — атрибут найденного файла
* +16h: слово — время создания файла в формате DOS:
* биты 15 – 11: час (0 — 23)
* биты 10 – 5: минута
* биты 4 – 0: номер секунды, деленный на 2 (0 – 30)
* +18h: слово — дата создания файла в формате DOS:
* биты 15 – 9: год, начиная с 1980
* биты 8 – 5: месяц
* биты 4 – 0: день
* +1Ah: 4 байта — размер файла
* +1Eh: 13 байт — ASCIZ-имя найденного файла с расширением

После того как DTA заполнена данными, для продолжения поиска следует вызывать функцию 4Fh, пока не будет возвращена ошибка.

Классические определения компьютерного вируса приведены выше.

Жизненный цикл

Поскольку отличительной особенностью вирусов в традиционном смысле является способность к размножению в рамках одного компьютера, деление вирусов на типы происходит в соответствии со способами размножения.Сам процесс размножения может быть условно разделен на несколько стадий:

1. Проникновение на компьютер 2. Активация вируса3. Поиск объектов для заражения4. Подготовка вирусных копий5. Внедрение вирусных копий

Особенности реализации каждой стадии порождают атрибуты, набор которых фактически и определяет класс вируса.

Проникновение

Вирусы проникают на компьютер вместе с зараженными файлами или другими объектами (загрузочными секторами дискет), никак, в отличие от червей, не влияя на процесс проникновения. Следовательно, возможности проникновения полностью определяются возможностями заражения и классифицировать вирусы по этим стадиям жизненного цикла отдельно смысла нет.

Активация

Для активации вируса необходимо, чтобы зараженный объект получил управление. На данной стадии деление вирусов происходит по типам объектов, которые могут быть заражены:

1.Загрузочные вирусы - вирусы, заражающие загрузочные сектора постоянных и сменных носителей.

2.Файловые вирусы —вирусы, заражающие файлы. Эта группа дополнительно делится на три, в зависимости от среды в которой выполняется код:

· Собственно файловые вирусы — те, которые непосредственно работают с ресурсами операционной системы.Из последних вредоносных программ, обладающих вирусной функциональностью, можно отметить Email-Worm.Win32.Bagle.p (а также его модификации .q и .r).

· Макровирусы — вирусы, написанные на языке макрокоманд и исполняемые в среде какого-либо приложения. В подавляющем большинстве случаев речь идет о макросах в документах Microsoft Office.

Подготовка вирусных копий

Внедрение

Внедрение вирусных копий может осуществляться двумя принципиально разными методами:

· Внедрение вирусного кода непосредственно в заражаемый объект · Замена объекта на вирусную копию. Замещаемый объект, как правило, переименовывается

Для вирусов характерным является преимущественно первый метод. Второй метод намного чаще используется червями и троянами, а точнее троянскими компонентами червей, поскольку трояны сами по себе не распространяются.

Ущерб от вредоносных программ

Черви и вирусы могут осуществлять все те же действия, что и трояны. На уровне реализации это могут быть как отдельные троянские компоненты, так и встроенные функции. Кроме этого, за счет массовости, для вирусов и червей характерны также другие формы вредоносных действий

1. **Примем COM-вируса.**

**Загрузка и выполнение COM-программы:**

* Запускаемой программе отводится вся свободная в данный момент ОП. Сегментная часть начального адреса этой памяти обычно называется начальным сегментом программы.
* По нулевому смещению в сегменте, определяемом начальным сегментом программы, EXEC строит специальную служебную структуру PSP (Program Segment Prefix), в котором содержится информация,необходимая для правильной работы программы . Заполняет PSP операционная система ( ОС ), а его размер всегда равен 100h ( 256 ) байт .
* Сразу вслед за PSP загружается сама COM - программа
* EXEC выполняет настройку регистров процессора:
  + CS = DS = SS = ES указывают на начальный сегмент программы,
  + регистр IP инициализируется числом 100h
  + регистр SP инициализируется числом 0fffeh .
* Теперь загруженную COM - программу можно исполнить . Для этого EXEC передает управление по адресу CS : 100h.

Работа вируса в зараженной программе:

* Восстанавливает в памяти компьютера исходные три байтa зараженной программы
* Ищет на диске подходящий COM - файл
* Записывает свое тело в конец этого файла
* Заменяет первые три байта заражаемой программы командой перехода на свой код, сохранив предварительно исходные три байта в своей области данных
* Выполняет вредные действия, предусмотренные автором
* Передает управление зараженной программе . Поскольку в COM - файле точка входа всегда равна CS : 100h, можно не выполнять сложных расчетов, а просто выполнить переход на этот адрес

1. **Пример EXE-Вируса.**

Вирусы-спутники (Companion)

Эти вирусы получили свое название из-за алгоритма размножения:

к каждому инфицированному файлу создается файл-спутник. Рассмотрим более подробно два типа вирусов этой группы:

Вирусы первого типа размножается следующим образом. Для каждого ин-фицируемого ЕХЕ-файла в том же каталоге создается файл с вирусным кодом, имеющий такое же имя, что и ЕХЕ-файл, но с расширением СОМ. Вирус активируется, если при запуске программы в командной строке указано только имя исполняемого файла. Дело в том, что, если не указано расширение файла, DOS сначала ищет в текущем каталоге файл с заданным именем и расширением СОМ. Если СОМ-файл с таким именем не найден, ведется поиск одноименного ЕХЕ-файла. Если не найден и ЕХЕ-файл, DOS попробует обнаружить ВАТ (пакетный) файл. В случае отсутствия в текущем каталоге исполняемого файла с указанным именем поиск ведется во всех каталогах, доступных по переменной PATH. Другими словами, когда пользователь хочет запустить программу и набирает в командной строке только ее имя (в основном так все и делают), первым управление получает вирус, код которого находится в СОМ-файле. Он создает СОМ-файл еще к одному или нескольким ЕХЕ-файлам (распространяется), а затем исполняет ЕХЕ-файл с указанным в командной строке именем. Пользователь же думает, что работает только запущенная ЕХЕ-программа.

Вирус-спутник обезвредить довольно просто - достаточно удалить СОМ-файл.

Вирусы второго типа действуют более тонко. Имя инфицируемого ЕХЕ-файла остается прежним, а расширение заменяется каким-либо другим, отличным от исполняемого (СОМ, ЕХЕ и ВАТ), Например, файл может получить расширение DAT (файл данных) или OVL (программный оверлей). Затем на место ЕХЕ-файла копируется вирусный код. При запуске такой инфицированной программы управление получает вирусный код, находящийся в ЕХЕ-файле. Инфицировав еще один или несколько ЕХЕ-файлов таким же образом, вирус возвращает оригинальному файлу исполняемое расширение (но не ЁХЕ, а СОМ, поскольку ЕХЕ-файл с таким именем занят вирусом), после чего исполняет его. Когда работа инфицированной программы закончена, ее запускаемому файлу возвращается расширение неисполняемого. Лечение файлов, зараженных вирусом этого типа, может быть затруднено, если вирус-спутник шифрует часть или все тело инфицируемого файла, а перед исполнением его расшифровывает.

1. **Общая стратегия защиты от вирусов. Структура антивируса**

*Стратегия защиты позволяет реализовать следующий комплекс задач:*

1. Упреждение эпидемий.   
   Системой проводятся профилактика проверок, на базе которых создаются правила для существующей сети. Имеется автономная связь со всеми узлами сети, отслеживающая подключения извне. При появлении потенциальной опасности компоненты автоматически обновляются по глобальной рассылке. Стратегия упреждает как вирусные заражения, так и другие сетевые угрозы.
2. Централизованное управление системой безопасности.  
   Правила создаются для всех компонентов сети и для ее отдельных узлов. Каждый узел периодически проверяется на уязвимость (при обнаружении активируются меры борьбы). Обеспечивается централизованный контроль сетевого трафика внутри/вовне сети, что экономит человеческие и материальные ресурсы на обеспечение безопасности.
3. Комплексное противодействие вирусной вспышке.   
   В случае обнаружения вируса система активирует меры по локализации угрозы: карантин пораженных станций (ограничение и остановка роста эпидемии - вредоносный код не сможет быть передан с уже пораженной машины или извне); отключение внешних соединений (важно, что пораженные станции будут в состоянии продолжать работу); локальное/глобальное предупреждение (все действия осуществляются автономно, что снижает угрозу эпидемии и сокращает затраты на ликвидацию). При обновлении антивируса меры действуют до полной ликвидации последствий. Контролировать рост эпидемии можно уже до получения обновления.
4. Ликвидация последствий.   
   Обычно борьба с вирусом заканчивается с его ликвидацией. Но фрагменты/рабочие контуры вируса способны сохраняться в скрытых либо не используемых пользователем данных — реестрах системы, приложениях, временных Интернет-файлах. Избежать повторного заражения поможет служба поиска/уничтожения фрагментов. С ее помощью сеть полностью очищается, что снижает стоимость ликвидации. Поиск/очистка не останавливают сеть, позволяя пользователям продолжать работу. Действие стратегии заканчивается с полной очисткой станций.

*Стратегия защиты включает:*

* комплексное решение, состоящее из антивирусных программ для станций и серверов;
* автономный системный анализатор уязвимости;
* службы принужденного отключения соединений;
* службы очистки остатков вредоносного кода;
* центр управления и настройки системы защиты.

Антиви́русная програ́мма (антиви́рус) — специализированная программа для обнаружения компьютерных вирусов, а также нежелательных (считающихся вредоносными) программ вообще и восстановления заражённых (модифицированных) такими программами файлов, а также для профилактики — предотвращения заражения (модификации) файлов или операционной системы вредоносным кодом.

Антивирусные программы различаются по виду (способу) защиты от вирусов:

* Программы-детекторы, или сканеры, находят вирусы в оперативной памяти и на внешних носителях, выводя сообщение при обнаружении вируса.
* Программы-доктора, (фаги, программы-вакцины) находят зараженные файлы и "лечат" их. Среди этого вида программ существуют полифаги, которые способны удалять разнообразные виды вирусов, самые известные из антивирусов-полифагов Norton AntiVirus, Doctor Web, Kaspersky Antivirus.
* Программы-ревизоры являются наиболее надежными в плане защиты от вирусов. Ревизоры запоминают исходное состояние программ, каталогов, системных областей диска до момента инфицирования компьютера, затем сравнивают текущее состояние с первоначальным, выводя найденные изменения на дисплей.
* Программы-мониторы (файерволы, брандмауэры) начинают свою работу при запуске операционной системы, постоянно находятся в памяти компьютера и осуществляют автоматическую проверку файлов по принципу "здесь и сейчас".
* Программы-фильтры (сторожа) обнаруживают вирус на ранней стадии, пока он не начал размножаться. Программы-сторожа - небольшие резидентные программы, целью которых является обнаружение действий, характерных для вирусов.

Существует несколько основных методов поиска вирусов, которые применяются антивирусными программами. К ним относятся:

* сканирование;
* эвристический анализ;
* обнаружение изменений на диске;
* постоянное наблюдение.

Сканирование - наиболее традиционный метод поиска вирусов, который заключается в поиске кодов известных вирусов. Программы, которые работают на основе сканирования, называются полифагами.

Полифаги могут обнаружить только уже известные и предварительно изученные вирусы. Поэтому программы-сканеры не защищают компьютер от проникновения новых неизвестных вирусов.

Эвристический анализ используется для поиска шифрующихся и полиморфных вирусов. Эвристический анализатор позволяет обнаруживать ранее неизвестные вирусы, хотя их лечение при этом бывает невозможным.

Обнаружение изменений. Заражая компьютер, вирус делает изменения на жестком диске: изменяет файлы или загрузочные записи (например, у файлов может измениться размер, дата и время создания). Антивирусные программы-ревизоры находят такие изменения и сообщают об этом пользователю.

Однако у программ, использующих данную технологию, имеется недостаток. Он заключается в том, что не все изменения на компьютере вызываются вторжением вируса.

Постоянное наблюдение используется программами, получившими название резидентные мониторы. Резидентные мониторы - это программы, которые постоянно находятся в памяти компьютера и отслеживают все подозрительные действия, выполняемыми другими программами.