Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и радиоэлектроники

Факультет непрерывного и дистанционного обучения

Кафедра информатики

Электронный учебно-методический комплекс

по дисциплине

**Методы защиты информации**

Для студентов специальности

**1-31 03 04 “Информатика“**

Минск 2011

# Общие сведения

## Сведения об ЭУМК

Данный комплекс специально разработан для студентов, обучающихся дистанционно. Целью его создания было сделать процесс изучения данной дисциплины максимально удобным и комфортным для студентов. Кроме того, обладание комплексом должно позволить студенту минимизировать свои непосредственные контакты с университетом и компьютером, что особенно удобно для иногородних студентов, а также тех, кто по различным причинам вынужден на долгое время уезжать из города. В принципе данный комплекс позволяет студенту изучить дисциплину и подготовиться к сдаче экзамена по ней “автономно”

Выше приведён перечень и описание составных частей данного комплекса. Они выполнены в виде гиперссылок и для перехода к нужной части требуется, удерживая клавишу “Ctrl”, щёлкнуть левой кнопкой мыши выбранную гиперссылку. Впрочем, поскольку весь комплекс представляет собой файл Microsoft Word, его можно просматривать в обычном режиме, а также полностью или частично печатать

**Составитель: Стержанов М.В.**, ассистент кафедры информатики Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники».

Рассмотрен и рекомендован к изданию на заседании кафедры информатики, протокол № \_\_ от \_\_.\_\_.2011.

## Методические рекомендации по изучению дисциплины

В соответствии с учебным планом студенты дистанционной формы обучения инженерных специальностей изучают курс «Методы защиты информации».

Учебным планом по данному курсу предусмотрено изучение теоретических вопросов, решение задач, выполнение 2 контрольных работ и 2 ИПР с ИКТ. Изучение курса заканчивается сдачей экзамена. К сдаче экзамена студенты допускаются только при условии выполненных и защищенных контрольных работ и ИПР с ИКТ.

Рекомендуется изучать курс «Методы защиты информации» в соответствии с рабочей программой. Сначала необходимо ознакомиться с содержанием курса, затем изучить рекомендуемую литературу, обращая внимание на вопросы, выделенные в рабочей программе, после чего изучить теоретическое изложение курса по приведенным разделам, темам и вопросам, ответить на контрольные вопросы, выполнить задачи для решения (выполнения контрольных работ) в соответствии с заданием.

Так как теоретический материал излагается в строгой логической последовательности, рекомендуется изучать данную дисциплину, придерживаясь данной логики.

## Рабочая учебная программа

**Учреждение образования**

**«Белорусский государственный университет**

**информатики и радиоэлектроники»**

УТВЕРЖДАЮ

Декан факультета непрерывного и дистанционного обучения

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. М. Бондарик

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2011 г.

Регистрационный № -\_\_\_/.

**Методы защиты информации**

Рабочая учебная программа

**для направления специальности 1-31 03 04**

**“Информатика“**

Факультет **непрерывного и дистанционного обучения**

Кафедра **информатики**

Курс **пятый (семестр 10)**

**Контрольные работы** **2 работы**

**ИПР с ИКТ 2 работы**

Всего часов **146 часов**

Экзамен **5 курс**

Форма получения

высшего образования **дистанционная**

Минск 2011

Составиль Стержанов М.В.

Учебная программа составлена на основе типовой учебной  *«Методы защиты информации*» для специальности 1-31 03 04 Информатика, утвержденной Министерством образования Республики Беларусь 14. 04. 2010 регистрационный № ТД-G.266/тип.

Рассмотрена и рекомендована к утверждению на заседании кафедры информатики

протокол № \_\_ от \_\_\_\_\_ \_\_\_\_

Заведующий кафедрой Минченко Л.И.

Одобрена и рекомендована к утверждению Научно-методической комиссией факультета компьютерных систем и сетей Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники»

протокол № \_\_ от \_\_\_\_\_ \_\_\_

Председатель Лукашевич М.М.

СОГЛАСОВАНО

Начальник отдела методического обеспечения

учебного процесса \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ц. С. Шикова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Цель преподавания дисциплины. Целью преподавания дисциплины "Методы защиты информации" является ознакомление с кругом проблем и решений в области информационной безопасности, а также изучение наиболее важных сервисов и механизмов защиты информации. В курсе рассматриваются: терминологический базис, модели информационной безопасности, наиболее важные криптографические алгоритмы и протоколы.

Задачи изучения дисциплины. Изучение курса преследует цель получения студентами знаний о комплексном подходе к защите информации, используемых средствах защиты информации в современных операционных системах, использовании шифрования и криптографических систем.

В результате изучения дисциплины выпускник должен

знать:

- основные виды угроз безопасности информации в компьютерных системах;

- математические основы методов защиты информации;

- методы криптографической защиты информации в компьютерных системах.

уметь:

-использовать криптографические средства защиты информации.

Перечень дисциплин, усвоение которых необходимо для изучения данной дисциплины

Дисциплина «Методы защиты информации» непосредственно связана и базируется на знании дисциплин «Основы алгоритмизации и программирования», «Конструирование программ и языки программирования», «Теория вероятности и математическая статистика».

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ку  р  с | Название и содержание тем (*по типовой или учебной программе*) | Контрольная работа (номер и тема по п.2) | | Лабораторная работа с указанием вида 1  (по п.1) | Оснащение контрольных и лабораторных работ  (по п.4) | Литература (по п.3) | Рекомендуемый объем для изучения (в часах)2 | Форма контроля знаний (зачет по контрольной работе, тесты, защита лабораторной работы, защита курсового проекта, экзамен, зачет) |
| 1 | 2 | 3 | | 4 | 5 | 6 | 7 | 8 |
|  | | | | | | | | |
|  | 1. ***Информационная безопасность компьютерных систем.***  Основные понятия и определения. Анализ угроз информационной безопасности. Модель сетевой безопасности. Обеспечение безопасности АСОИ. | |  |  |  | 3.4, 3.5 | 12 |  |
|  | ***2. Криптографическая защита информации.*** Основные понятия и определения. Понятие стойкости шифра. Классификация криптоалгоритмов и типы криптоатак. Современные приложения криптографии. Скремблирование. | |  |  |  | 3.1- 3.5 | 12 |  |
|  | ***3. Алгоритмы симметричного шифрования.*** [Основные понятия и определения](file:///G:\Other-documents\Dima\Ð´Ð¸Ð¼Ð°\ÐÐ¸Ð¼Ð°%20-%20ÐÐÐ£ÐÐ %202009-2013\6%20ÑÐµÐ¼\ÐÐ§Ð\ÐÐ¾Ð¿.ÐÐ°Ð´Ð°Ð½Ð¸Ðµ\1\Ð­Ð£ÐÐ%20ÐÐ¡Ð¸Ð¡%2013\Ð¢ÐµÐ¾ÑÐ¸Ñ\ÐÐ¡Ð¸Ð¡_Ð). Сеть Фейштеля. Алгоритм DES. Алгоритм Blowfish. Алгоритм IDEA. Алгоритм ГОСТ 28147. Режимы выполнения алгоритмов симметричного шифрования.3.8 Способы создания псевдослучайных чисел. Алгоритм AES. | |  | ИПР с ИКТ №1 |  | 3.1- 3.3 | 20 | Защита ИПР c ИКТ №1. |
|  | ***4. Криптография с открытым ключом*** [Основные требования к алгоритмам асимметричного шифрования. Криптоанализ алгоритмов с открытым ключом. Алгоритм RSA](file:///G:\Other-documents\Dima\Ð´Ð¸Ð¼Ð°\ÐÐ¸Ð¼Ð°%20-%20ÐÐÐ£ÐÐ %202009-2013\6%20ÑÐµÐ¼\ÐÐ§Ð\ÐÐ¾Ð¿.ÐÐ°Ð´Ð°Ð½Ð¸Ðµ\1\Ð­Ð£ÐÐ%20ÐÐ¡Ð¸Ð¡%2013\Ð¢ÐµÐ¾ÑÐ¸Ñ\ÐÐ¡Ð¸Ð¡_Ð). Алгоритм Диффи-Хеллмана. | |  | ИПР с ИКТ №2 |  | 3.1- 3.3 | 20 | Защита ИПР c ИКТ №2. |
|  | ***5. Хэш-функции.*** Требования к хэш-функциям[. Простые хэш-функции. Парадокс дня рождения. Использование цепочки зашифрованных блоков. Хэш-функция MD5. Хэш-функция SHA-1. Хэш-функция SHA-2](file:///G:\Other-documents\Dima\Ð´Ð¸Ð¼Ð°\ÐÐ¸Ð¼Ð°%20-%20ÐÐÐ£ÐÐ %202009-2013\6%20ÑÐµÐ¼\ÐÐ§Ð\ÐÐ¾Ð¿.ÐÐ°Ð´Ð°Ð½Ð¸Ðµ\1\Ð­Ð£ÐÐ%20ÐÐ¡Ð¸Ð¡%2013\Ð¢ÐµÐ¾ÑÐ¸Ñ\ÐÐ¡Ð¸Ð¡_Ð). Хэш-функции ГОСТ 3411 | | 2.1 |  |  | 3.1- 3.3 | 16 | Зачет по контрольной работе. |
|  | ***6.*** ***Коды аутентификации сообщений – МАС.*** Требования к MAC. МАС на основе алгоритма симметричного шифрования. МАС на основе хэш-функции. | |  |  |  | 3.1- 3.3 | 16 |  |
|  | ***7.*** ***Цифровая подпись.*** Требования к цифровой подписи. Прямая и арбитражная цифровые подписи. Стандарт цифровой подписи DSS. Отечественный стандарт цифровой подписи ГОСТ 3410. | | 2.2 |  |  | 3.1- 3.3 | 16 | Зачет по контрольной работе. |
|  | ***8. Криптография с использованием эллиптических кривых.***  Математические понятия. Аналог алгоритма Диффи-Хеллмана обмена ключами. Алгоритм цифровой подписи на основе эллиптических кривых ECDSA. Шифрование/дешифрование с использованием эллиптических кривых. | |  |  |  | 3.1- 3.3 | 16 |  |
|  | ***9. Алгоритмы обмена ключей и протоколы аутентификации.*** Алгоритмы распределения ключей с использованием третьей доверенной стороны. Взаимная аутентификация. | |  |  |  | 3.1- 3.3 | 16 |  |

1.ИНДИВИДУАЛЬНЫЕ ПРАКТИЧЕСКИЕ РАБОТЫ, ИХ ХАРАКТЕРИСТИКИ

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Тема | Характеристика | Рекомендуемый объем в часах |
| 1. | Алгоритмы шифрования DES и ГОСТ | Цель работы – реализовать на языке программирования C++ алгоритмы симметричного шифрования DES и ГОСТ 28147. | 8 |
| 2. | Алгоритм шифрования RSA | Цель работы – продемонстрировать программную реализацию алгоритма симметричного шифрования RSA. | 8 |

2. КОНТРОЛЬНЫЕ РАБОТЫ, ИХ ХАРАКТЕРИСТИКИ

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название темы | Содержание | Объем в часах |
| 1 | 2 | 3 | 4 |
| Десятый семестр | | | |
| 1. | Реализация алгоритма хэширования | Цель контрольной работы **–** реализовать алгоритм Хэш-функции ГОСТ 3411 | 4 |
| 2. | Реализация алгоритма цифровой подписи. | Цель контрольной работы **–** реализовать алгоритм цифровой подписи ГОСТ 3410. | 4 |
| Итого: | | | 8 |

3. ЛИТЕРАТУРА

3.1.ОСНОВНАЯ

1. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. / Б. Шнайер / М: Триумф, 600 с.

2. Лапонина, О.Р. Основы сетевой безопасности: криптографические алгоритмы и протоколы взаимодействия: учебное пособие, 2-е изд. испр., М. ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», БИНОМ. Лаборатория знаний, 2007г., 531с.

3. Menezes A., Handbook of Applied Cryptography. A. Menezes, P. van Oorschot, S. Vanstone / CRC Press, 1996, 816 p.

4. Харин, Ю.С. Математические и компьютерные основы криптологии. / Ю.С. Харин – Мн., 2003г.

5. Мельников, В.П. Информационная безопасность и защита информации / В.П. Мельников, С.А. Клейменов, А.М. Петраков – Издательство: Академия, 2006.

6. Мельников, В. Безопасность информации в автоматизированных системах / В. Мельников. – М: Финансы и статистика, 2003.

3.2.ДОПОЛНИТЕЛЬНАЯ

1. Завгородский, В.И. Комплексная защита информации в компьютерных системах / В.И. Завгородский. – Логос, 2001.

2. Романец, Ю.В. Защита информации в компьютерных системах и сетях / Ю.В. Романец, П.А. Тимофеев, В.Ф. Шаньгин. – М: "Радио и связь", 2001.

3. Казанцев, С. Я. Правовое обеспечение информационной безопасности: Учеб. пособие для студ. высш. учеб. заведений. – М: Издательский центр "Академия", 2005.

4. Козачок, В. Основы организационного обеспечения информационной безопасности объектов информатизации / В. Козачок. – Гелиос АРВ, 2005.

4. ПЕРЕЧЕНЬ КОМПЬЮТЕРНЫХ ПРОГРАММ, НАГЛЯДНЫХ И ДРУГИХ ПОСОБИЙ, МЕТОДИЧЕСКИХ УКАЗАНИЙ И МАТЕРИАЛОВ И ТЕХНИЧЕСКИХ СРЕДСТВ ОБУЧЕНИЯ

4.1. Среда программирования Microsoft Visual Studio.

4.2. Программные пакеты по математике MATLAB, MATHCAD, МATHEMATICA.

Теоретический раздел

# Теоретический раздел

## Лекции

### ВВЕДЕНИЕ

За несколько последних десятилетий требования к информационной безопасности существенно изменились. До начала широкого использования автоматизированных систем обработки данных безопасность информации достигалась исключительно физическими и административными мерами. С появлением компьютеров стала очевидной необходимость использования автоматических средств защиты файлов данных и программной среды. Следующий этап развития автоматических средств защиты связан с появлением распределенных систем обработки данных и компьютерных сетей, в которых средства сетевой безопасности используются в первую очередь для защиты передаваемых по сетям данных. В данном курсе рассматриваются основные методы криптографической защиты информации.

**ТЕМА 1.** **Информационная безопасность компьютерных систем**

**1.1. Основные понятия и определения**

Рассмотрим основные понятия ЗИ и информационной безопасности КсиС с учетом определений стандарта ГОСТ Р 50922-96.

*Защита информации(ЗИ)* – это деятельность по предотвращению утечки защищаемой информации, несанкционированных и непреднамеренных воздействий на защищаемую информацию.

*Объект защиты* – информация, или носитель информации, или информационный процесс, в отношении которого необходимо обеспечивать защитту в соответствии с поставленной целью ЗИ.

*Цель ЗИ* – это желаемый результат ЗИ. Целью ЗТ может быть предотвращение ущерба собственнику, влалельцу, пользователю инофрмации в результате возможной утечки информации и/или несанкционированного и непреднамеренного воздействия на информацию.

*Эффективность ЗИ* – степень воздействия результатов ЗИ поставленной цели.

*Система ЗИ* – совокупность органов и/или исполнителей, используемая ими техника ЗИ, а также объекты защиты, организованные и функционирующие по правилам, установленным соответствующими правовыми, организационно-распорядительными и нормативными документами по ЗИ.

Под *информационной безопасностью* понимают защищенность информации от незаконного ознакомления, преобразования и уничтожения, а также защищенность информационных ресурсов от воздействий, направленных на нарушение их работоспособности.

Компоненты [Автоматизированных систем обработки информации](https://www.google.by/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiGhuWRmIvTAhVI3iwKHTKZBVYQFggaMAA&url=https%3A%2F%2Fabitur.bsuir.by%2Favtomatizirovannye-sistemy-obrabotki-informatsii&usg=AFQjCNGpAWnqPAtqceRZMCFqrqcQZby41g) (далее – АСОИ) можно разбить на следующие группы:

* Аппаратные средства;
* Программное обеспечение;
* Данные;
* Персонал.

*Информационаая безопасность* компьютерных систем достигается обеспечением *конфиденциальности, целостности и достоверности* обрабатываемых данных, а также доступности и целостности информационных компонентов и ресурсов системы.

*Конфиденциальность данных* – этот статус, предоставленный данным и определяющий требуемую степень их защиты. К конфиденциальным данным можено отнести, например, следующие: личная информация пользователей; учетные записи; данные о кредитных картах; бухгалтерские сведения.

Под *целостностью информации* понимается свойство информации сохранять свою структуру и/или содержание в процессе передачи и хранения. Целостность информации обеспечивается в том случае, если данные не в системе не отличаются в семантическом отношении от исходных документов, то есть если не произошло их случайного или преднамеренного разрушения.

*Достоверность информации* – свойство информации, выражающееся в строгой принадлежности субъекту, который является ее источником, либо тому субъекту, от которого эта информация принята.

*Собственник информации* – субъект, в полном объеме реализующий полномочия владения, пользования, распоряжения информацией в соответствии с законодательными актами.

*Доступность данных*. Работа пользователя с данными возможна только в том случае, если он имеет к ним доступ.

*Доступ к информации* – получение субъектом овзможности ознакомления с информацией, в том числе при помощи технических средств.

Различают санкционированный и несанкционированный доступ к информации.

*Санкционированный доступ к информации* – это доступ к информации, не нарушающий установленные правила разграничения доступа. Правила разграничения доступа служат для регламентации права доступа к компонентам системы.

*Несанкционированный доступ (НСД)* характеризуется нарушением установленных правил разграничения доступа. Лицо или процесс, осуществляющие НСД к информации, являются нарушителями правил разграничения доступа. НСД является наиболее распространенным видом компьютерных нарушений.

С допуском к информации и ресурсам системы связана группа таких важных понятий, как идентификация, аутентификация, авторизация.

*Идентификация субъекта* – это процедура распознавания субъекта по его идентификатору. Идентификация выполняется при попытке субъекта войти в систему (сеть).

*Аутентификация субъекта* – это проверка подлинности субъекта с данным идентификатором. Процедура аутентификации устанавливает, является ли субъект именно тем, кем он себя объявил.

*Авторизация субъекта* – это процедура предоставления законному субъекту, успешно прошедшему идентификацию и аутентификацию, соответствующих полномочий и доступных ресурсов системы (сети).

Под *угрозой безопасности* АСОИ понимаются возможные действия, способные прямо или косвенно нанести ущерб ее безопасности. *Ущерб безопасности* подразумевает нарушение состояния защищенности информации, содержащейся и обрабатываемой в системе.

*Уязвимость компьютерной системы* – это присущее системе неудачное свойство, которое может привести к реализации угрозы.

*Атака на компьютерную систему* – это поиск и / или использование злоумышленником той или иной уязвимости системы.*Атака* – любое действие, нарушающее политику безопасности информационной системы. Иными словами, атака – это реализация угрозы безопасности.

Противодействие угрозам безопасности является целью средств защиты компьютерных систем и сетей.

*Безопасная* или *защищенная система* – это система со средствами защиты, которые успешно и эффективно противостоят угрозам безопасности.

*Комплекс средств защиты* представляет собой совокупность программных и технических средств, создаваемых и поддерживаемых для обеспечения информационной безопасности АСОИ.

*Политика безопасности* – это совокупность норм, правил и практических рекомендаций, регламентирующих работу средств защиты АСОИ от заданного множества угроз безопасности.

1.**2. Анализ угроз информационной безопасности**

Рассмотрение возможных угроз информационной безопасности приводится с целью определения полного набора требований к разрабатываемой системе защиты.

По цели воздействия различают три основных типа угроз безопасности АСОИ:

* угрозы нарушения конфиденциальности информации;
* угрозы нарушения целостности информации;
* угрозы нарушения работоспособности системы (отказы в обслуживании).

Классификация возможных угроз информационной безопасности АСОИ может быть приведена по ряду базовых признаков:

1. По *природе возникновения* различают:
   * естественные угрозы;
   * искусственные угрозы.
2. По *степени преднамеренности проявления* различают:
   * угрозы, вызванные ошибками или халатностью персонала;
   * угрозы преднамеренного действия.
3. По *непосредственному источнику угроз*. Источниками угроз могут быть:
   * природная среда;
   * человек;
   * санкционированные программно-аппаратные средства;
   * несанкционированные программно-аппаратные средства.
4. По *положению источника угроз*. Источник угроз может быть расположен:
   * вне контролируемой зоны АСОИ;
   * в пределах контролируемой зоны АСОИ;
   * непосредственно в АСОИ.
5. По *степени зависимости от активности* АС. Угрозы проявляются:
   * независимо от активности АСОИ;
   * только в процессе обработки данных.
6. По *степени воздействия на* АСОИ различают:
   * пассивные угрозы;
   * активные угрозы.
7. По *этап доступа пользователей или программ к ресурсам* АСОИ различают:
   * угрозы, проявляющиеся на этапе доступа к ресурсам АСОИ;
   * угрозы, проявляющиеся после разрешения доступа к ресурсам АСОИ.
8. По *способу доступа к ресурсам* АСОИ различают:
   * угрозы, с использованием стандартного пути доступа к ресурсам АСОИ;
   * угрозы с использованием скрытого нестандартного пути доступа к ресурсам АСОИ.
9. По *текущему месту расположения информации, хранимой и обрабатываемой в* АСОИ, различают:
   * угрозы доступа к информации на внешних запоминающих устройствах;
   * угрозы доступа к информации в оперативной памяти;
   * угрозы доступа к информации, циркулирующей в линиях связи;
   * угрозы доступа к информации, отображаемой на терминале или печатаемой на принтере.

Опасные воздействия на АСОИ можно подразделить на *случайные* и *преднамеренные*.

Исходя из возможности возникновения наиболее опасной ситуации, обусловленной действиями нарушителя, можно составить гипотетическую модель потенциального нарушителя:

* квалификация нарушителя может быть на уровне разработчика данной системы;
* нарушителем может быть как постороннее лицо, так и законный пользователь системы;
* нарушителю известна информация о принципах работы системы;
* нарушитель выберет наиболее слабое звено в защите.

*Несанкционированный доступ (НСД)* является наиболее распространенным и многообразным видом компьютерных нарушений. Суть НСД состоит в получении пользователем (нарушителем) доступа к объекту в нарушение правил разграничения доступа, установленных в соответствии с принятой в организации политикой безопасности. НСД использует любую ошибку в системе защиты и возможен при нерациональном выборе средств защиты, их некорректной установке и настройке. НСД может быть осуществлен как штатными средствами АСОИ, так и специально созданными аппаратными и программными средствами.

Перечислим основные каналы несанкционированного доступа, через которые нарушитель может получить доступ к компонентам АСОИ и осуществить хищение, модификацию и/или разрушение информации:

* все штатные каналы доступа к информации (терминалы пользователей, оператора, администратора системы; средства отображения и документирования информации; каналы связи) при их использовании нарушителями, а также законными пользователями вне пределов их полномочий;
* технологические пульты управления;
* линии связи между аппаратными средствами АСОИ;
* побочные электромагнитные излучения от аппаратуры, линий связи, сетей электропитания и заземления и др.

Из всего разнообразия способов и приемов несанкционированного доступа остановимся на следующих распространенных и связанных между собой нарушениях:

* перехват паролей;
* "маскарад";
* незаконное использование привилегий.

1.3**. Модель сетевой безопасности**

Все *атаки* можно разделить на два класса: *пассивные* и *активные*.

**I. Пассивная атака**

Пассивной называется такая *атака*, при которой *противник* не имеет возможности модифицировать передаваемые сообщения и вставлять в информационный канал между отправителем и получателем свои сообщения. Целью *пассивной атаки* может быть только прослушивание передаваемых сообщений и анализ трафика.

**II. Активная атака**

Активной называется такая *атака*, при которой *противник* имеет возможность модифицировать передаваемые сообщения и вставлять свои сообщения. Различают следующие типы *активных атак*:

1. **Отказ в обслуживании** - ***DoS-атака (Denial of Service)***

Отказ в обслуживании нарушает нормальное функционирование сетевых сервисов.*Противник* может перехватывать все сообщения, направляемые определенному адресату. Другим примером подобной *атаки* является создание значительного трафика, в результате чего сетевой сервис не сможет обрабатывать запросы законных клиентов. Классическим примером такой *атаки* в сетях TCP/IP является SYN-атака, при которой нарушитель посылает пакеты, инициирующие установление ТСР-соединения, но не посылает пакеты, завершающие установление этого соединения. В результате может произойти переполнение памяти на сервере, и серверу не удастся установить соединение с законными пользователями.

1. **Модификация потока данных** - ***атака "man in the middle"***

Модификация потока данных означает либо изменение содержимого пересылаемого сообщения, либо изменение порядка сообщений.

1. **Создание ложного потока (фальсификация)**

***Фальсификация*** (нарушение аутентичности) означает попытку одного субъекта выдать себя за другого.

1. **Повторное использование**

Повторное использование означает пассивный захват данных с последующей их пересылкой для получения несанкционированного доступа - это так называемая *replay-атака*. На самом деле *replay-атаки* являются одним из вариантов фальсификации, но в силу того, что это один из наиболее распространенных вариантов*атаки* для получения несанкционированного доступа, его часто рассматривают как отдельный тип *атаки*.

Перечисленные *атаки* могут существовать в любых типах сетей, а не только в сетях, использующих в качестве транспорта протоколы TCP/IP,и на любом уровне модели OSI.

1.4**.** Обеспечение безопасности АСОИ

Основным назначением АСОИ является переработка (сбор, хранение, обработка и выдача) информации, поэтому проблема обеспечения информационной безопасности является для АСОИ центральной. Обеспечение безопасности АСОИ предполагает организацию противодействия любому несанкционированному вторжению в процесс функционирования АСОИ, а также попыткам модификации, хищения, выведения из строя или разрушения ее компонентов, т.е. защиту всех компонентов АСОИ – аппаратных средств, программного обеспечения, данных и персонала.

Существуют два подхода к проблеме обеспечения безопасности АСОИ: "фрагментарный" и комплексный.

*"Фрагментарный" подход* направлен на противодействие четко определенным угрозам в заданных условиях. В качестве примеров реализации такого подхода можно указать отдельные средства управления доступом, автономные средства шифрования, специализированные антивирусные программы и т.п.

Достоинством такого подхода является высокая избирательность к конкретной угрозе. Существенным недостатком данного подхода является отсутствие единой защищенной среды обработки информации. Фрагментарные меры защиты информации обеспечивают защиту конкретных объектов АСОИ только от конкретной угрозы. Даже небольшое видоизменение угрозы ведет к потере эффективности защиты.

*Комплексный подход* ориентирован на создание защищенной среды обработки информации в АСОИ, объединяющей в единый комплекс разнородные меры противодействия угрозам. Организация защищенной среды обработки информации позволяет гарантировать определенный уровень безопасности АСОИ, что является несомненным достоинством комплексного подхода. К недостаткам этого подхода относятся: ограничения на свободу действий пользователей АСОИ, большая чувствительность к ошибкам установки и настройки средств защиты, сложность управления.

Комплексный подход применяют для защиты АСОИ крупных организаций или небольших АСОИ, выполняющих ответственные задачи или обрабатывающих особо важную информацию. Нарушение безопасности информации в АСОИ крупных организаций может нанести огромный материальный ущерб как самим организациям, так и их клиентам. Поэтому такие организации вынуждены уделять особое внимание гарантиям безопасности и реализовывать комплексную защиту. Комплексного подхода придерживаются большинство государственных и крупных коммерческих предприятий и учреждений. Этот подход нашел свое отражение в различных стандартах.

ТЕМА 2. Криптографическая защита информации

2.1 Основные понятия и определения

Проблемой защиты информации путем ее преобразования занимается криптология (kruptoV ‑ тайный, logoV ‑ наука (слово) (греч.)). Криптология разделяется на два направления – *криптографию* и *криптоанализ*. *Криптография* представляет собой совокупность методов преобразования данных, направленных на то, чтобы сделать эти данные бесполезными для противника.

Сфера интересов *криптоанализа* ‑ исследование возможности расшифровывания информации без знания ключей.

Ключ — некоторый неизвестный параметр шифра, позволяющий выбрать для шифрования и расшифрования конкретное преобразование из всего множества преобразований, составляющих шифр.

Незашифрованное сообщение будем обозначать *Р* или *M*, от слов plaintext и message. Зашифрованное сообщение будем называть шифртекстом или криптограммой и обозначать *С*, от слова ciphertext.

Шифрование (С = Ек [Р]) – процесс создания шифрованного текста при наличии ключа.

Дешифрование (Р = DK [С]) – восстановление открытого текста или ключа из шифрованного текста.

Противник – субъект (или физическое лицо), который не знает и не должен знать ключа или открытого текста, но стремящийся получить его.

При этом шифртекст можетсодержать как новые знаки, так и уже имеющиеся в исходном сообщении. Количество знаков в криптограмме и в исходном тексте в общем случае может различаться. Непременным требованием является возможность однозначного и в полном объеме восстановления исходного текста, используя лишь некоторые логическиедействия с символами шифртекста.

Криптографическая атака – попытка криптоаналитика вызвать отклонения в атакуемой защищенной системе обмена информацией. Успешную криптографическую атаку называют взлом или вскрытие.

Примитивные с позиции сегодняшнего дня криптографические методы известны с древнейших времен и некоторое время рассматривались скорее как некоторые ухищрения, чем строгая научная дисциплина. По утверждению ряда специалистов криптография по возрасту - ровесник египетских пирамид. В документах древних цивилизаций – Индии, Египта, Месопотамии – есть сведения о системах и способах составления шифрованных писем.

Пробуждение значительного интереса к криптографии и ее последующее развитие началось в XIX веке, что связано с зарождением электросвязи. В XX столетии секретные службы большинства развитых стран стали относиться к этой дисциплине как к обязательному инструменту своей деятельности.

Говоря об исторических аспектах научных исследований в области криптографии, необходимо отметить тот факт, что весь период с древних времен до 1949 года можно назвать донаучным периодом, поскольку средства закрытия письменной информации не имели строгого математического обоснования. Поворотным моментом, придавшим криптографии научность и выделившим ее в отдельное направление математики, явилась публикация в 1949 году статьи К. Э. Шеннона "Теория связи в секретных системах. Указанная работа послужила основой развития *одноключевых симметричных криптосистем****,*** в которых предполагается обмен секретными ключами между корреспондентами. Впоследствии с учетом особенностей построения симметричные шифры были разделены на две криптосистемы: *поточные*и *блочные шифры****.*** Отличительная особенность первых со­стоит в преобразовании каждого символа в потоке исходных данных, тогда как вторые осуществляют последовательное преобразование целых блоков данных.

Фундаментальнымвыводом из работы Шеннона стало определение зависимости *надежности*алгоритма от размера и качества секретного ключа, а также от *информационной избыточности*исходного текста. Шеннон ввел формальное определение информации и функции ненадежности ключа как его неопределенности при заданном количестве известных битов закрытого текста. Кроме того, им было введено важное понятие *расстояния единственности*как минимального размера текста, для которого еще возможно однозначное раскрытие исходного текста. Было показано, что расстояние единственности прямо пропорционально длине ключа и обратно пропорционально избыточности исходного текста.

Другим фундаментальным толчком развития криптографии явилась публикация и 1976 году статьи У. Диффи и М. Е. Хеллмана "Новые направления в криптографии". Вэтой работе впервые было показано, что секретность передачи информации может обеспечиваться без обмена секретными ключами. Тем самым была открыта эпоха *двухключевых (асимметричных) криптосистем****,*** разновидностями которых являются системы электронной цифровой подписи, тайного электронного голосования, защиты от навязывания ложных сообщений, электронной жеребьевки, идентификации и аутентификации удаленных пользователей и ряд других систем.

2.2. Понятие стойкости шифра

Вдалекие времена надежность сохранения информации в тайне определялась секретностью самого метода преобразования. Однако секретность алгоритма принципиально не может обеспечить его *безусловную стойкость****,*** т. е. невозможность чтения криптограммы противником, об­ладающим бесконечными вычислительными ресурсами. Поскольку секретные алгоритмыне доступны для проведения широкомасштабных криптоаналитических исследований, то по сравнению с открытыми алгоритмами имеется значительно более высокая вероятность того, что впоследствии будут найдены уязви­мыеместа и эффективные способы их взлома. Всвязи с этими обстоятельствами в настоящеевремя наиболее широко используются открытые алгоритмы, прошедшие длительное тестирование и обсуждение в открытой криптографической литературе.

Стойкость современных криптосистем основывается не на секретности алгорит­ма, а на секретности некоторой информации сравнительно малого размера, назы­ваемой *секретным ключом****.*** Ключ используется для управления процессом криптографического преобразования (шифрования) и является легко сменяемым элементом криптосистемы. Ключ может быть заменен пользователями в произвольный моментвремени, тогда как сам алгоритм шифровании является долгосрочным элементом криптосистемы и связан с длительным этапом разработки и тестирования.

Голландский криптограф Керкхофф (1835—1903) впервые сформулировал правило стойкости шифра***,*** в соответствии с которым:

* весьмеханизм преобразований считается известным злоумышленнику;
* надежность алгоритма должна определяться только неизвестным значением секретного ключа.

Второетребование означает, что оппонент не сможет разработать методы, позволяющиеснять защиту или определить истинный ключ, за время существенно меньшее, чем время *полного (тотального) перебора*всего множества возможных секретныхключей. Такой подход отражает очень важный принцип технологии защиты информации: защищенность системы не должна зависеть от секретности чего-либо такого, что невозможно быстро изменить в случае утечки секретной информации. Обычно криптосистема представляет собой совокупность аппаратных и программных средств, которую можно изменить только при значительных затратах времени и средств, тогда как ключ является легко изменяемым объектом. Именно поэтому стойкость криптосистемы определяется только секретностью ключа.

Безопасность, обеспечиваемая традиционной криптографией, зависит от нескольких факторов.

Во-первых, криптографический алгоритм должен быть достаточно сильным, чтобы передаваемое зашифрованное сообщение невозможно было расшифровать без ключа, используя только различные статистические закономерности зашифрованного сообщения или какие-либо другие способы его анализа.

Во-вторых, безопасность передаваемого сообщения должна зависеть от секретности ключа, но не от секретности алгоритма. Алгоритм должен быть проанализирован специалистами, чтобы исключить наличие слабых мест, при которых плохо скрыта взаимосвязь между незашифрованным и зашифрованным сообщениями. К тому же при выполнении этого условия производители могут создавать дешевые аппаратные чипы и свободно распространяемые программы, реализующие данный алгоритм шифрования.

В-третьих, алгоритм должен быть таким, чтобы нельзя было узнать ключ, даже зная достаточно много пар (зашифрованное сообщение, неза­шифрованное сообщение), полученных при шифровании с использова­нием данного ключа.

Принято различать криптоалгоритмы по степени доказуемости их безопасности. Существуют *безусловно стойкие*, *доказуемо стойкие* и *предположительно стойкие криптоалгоритмы*. Безопасность *безусловно стойких* *криптоалгоритмов* основана на доказанных теоремах о невозможности раскрытия ключа. Строго говоря, безусловно стойкими называются шифры (по Шеннону – совершенно секретными), для которых криптоаналитик не может улучшить оценку исходного сообщения *M* на основе знания криптограммы С по сравнению с оценкой при неизвестной криптограмме. При этом предполагается, что криптоаналитик обладает бесконечными вычислительными ресурсами. Примером безусловно стойкого криптоалгоритма является система с разовым использованием ключей (шифр Вернама) или система квантовой криптографии, основанная на квантовомеханическом принципе неопределенности.

Криптосистемы второго типа характеризуются тем, что по мере того, как объем доступной криптоаналитику криптограммы возрастает при определенном значении n=n0, существует единственное решение криптоаналитической задачи. Минимальный объем криптограммы, для которого имеется единственное решение, называется *расстоянием единственности*. В случае ленты однократного использования n0→∞. При конечной длине секретного ключа значение n0 конечно. Заранее известно, что по криптограмме, имеющей размер больше расстояния единственности можно найти единственное решение криптоаналитической задачи. Однако для криптоаналитика, обладающего ограниченными вычислительными ресурсами, вероятность найти это решение за время, в течение которого информация представляет ценность, чрезвычайно мала (10-30 и менее). Стойкость *доказуемо стойких криптоалгоритмов* определяется сложностью решения хорошо известной математической задачи, которую пытались решить многие математики и которая является общепризнанно сложной. Примером могут служить системы Диффи-Хеллмана или Ривеста-Шамира-Адельмана, основанные на сложностях соответственно дискретного логарифмирования и разложения целого числа на множители.

*Предположительно стойкие криптоалгоритмы* основаны на сложности решения частной математической задачи, которая не сводится к хорошо известным задачам и которую пытались решить один или несколько человек. Задачи такого типа называются трудными или вычислительно сложными, а об их решении говорится, что оно является вычислительно нереализуемым (или вычислительно неосуществимым). Примерами могут криптоалгоритмы *ГОСТ 28147-89, DES, FEAL.*

Клод Шеннон ввел понятия диффузии и конфузии для описания стойкости алгоритма шифрования.

*Диффузия* – это рассеяние статистических особенностей и законо­мерностей незашифрованного текста в широком диапазоне статистичес­ких особенностей и закономерностей зашифрованного текста. *Конфузия* – это уничтожение статистической взаимосвязи между зашифрованным текстом и ключом.

2.3. Классификация криптоалгоритмов и типы криптоатак

В отношении криптоалгоритмов существует несколько схем классификации, каждая из которых основана на группе характерных признаков. Таким образом, один и тот же алгоритм "проходит" сразу по нескольким схемам, оказываясь в каждой из них в какой-либо из подгрупп.

Основной схемой классификации всех криптоалгоритмов является следующая:

* Тайнопись.

Отправитель и получатель производят над сообщением преобразования, известные только им двоим. Сторонним лицам неизвестен сам алгоритм шифрования. Некоторые специалисты считают, что тайнопись не является криптографией вообще, и автор находит это совершенно справедливым.

* Криптография с ключом.

Алгоритм воздействия на передаваемые данные известен всем сторонним лицам, но он зависит от некоторого параметра – "ключа", которым обладают только отправитель и получатель.

Симметричные криптоалгоритмы

Для зашифровки и расшифровки сообщения используется один и тот же блок информации (ключ).

Асимметричные криптоалгоритмы

Алгоритм таков, что для зашифровки сообщения используется один ("открытый") ключ, известный всем желающим, а для расшифровки – другой ("закрытый"), существующий только у получателя.

В зависимости от *характера воздействий*, производимых над данными, алгоритмы подразделяются на:

1. Перестановочные  
   Блоки информации (байты, биты, более крупные единицы) не изменяются сами по себе, но изменяется их порядок следования, что делает информацию недоступной стороннему наблюдателю.
2. Подстановочные  
   Сами блоки информации изменяются по законам криптоалгоритма. Подавляющее большинство современных алгоритмов принадлежит этой группе.

В зависимости от размера блока информации криптоалгоритмы делятся на:

1. *Потоковые шифры*.

Единицей кодирования является один бит. Результат кодирования не зависит от прошедшего ранее входного потока. Наиболее распространенными предствателями поточных шифров являются *скремблеры*.

1. *Блочные шифры*.

Единицей кодирования является блок из нескольких байтов (в настоящее время 4-32). Результат кодирования зависит от всех исходных байтов этого блока. Схема применяется при пакетной передаче информации и кодировании файлов.

Существует четыре основных типа криптоаналитических атак. Конечно, все они формулируются в предположении, что криптоаналитику известны применяемый алгоритм шифрования и шифртексты сообщений.

1. Криптоаналитическая атака при наличии только известного шифртекста.
2. Криптоаналитическая атака при наличии известного открытого текста.
3. Криптоаналитическая атака при возможности выбора открытого текста.
4. Криптоаналитическая атака с адаптивным выбором открытого текста.

Кроме перечисленных основных типов криптоаналитических атак, можно отметить, по крайней мере, еще два типа.

1. Криптоаналитическая атака с использованием выбранного шифртекста.
2. Криптоаналитическая атака методом полного перебора всех возможных ключей.

**2.4. Современные приложения криптографии**

Значение криптографии выходит далеко за рамки обеспечения секретности данных. По мере все большей автоматизации процессов передачи и обработки информации и интенсификации информационных потоков криптографические методы приобретают уникальное значение. Новые информационные технологии в своей основе имеют двухключевую криптографию, которая позволяет реализовать протоколы, предполагающие, что секретный ключ известен только одному пользователю, т. е. протоколы, ориентированные на взаимное недоверие взаимодействующих сторон. Отметим основные приложения современной криптографии.

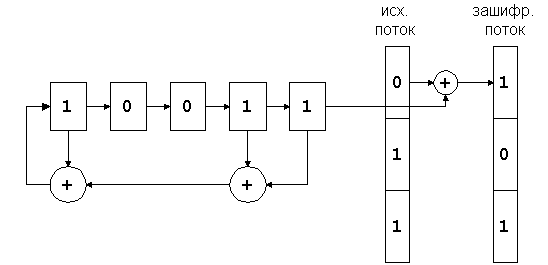
* Защита от несанкционированного чтения (или обеспечение конфндециальности информации).
* Защита от навязывания ложных сообщений (умышленных и непреднамеренных).
* Аутентификация законных пользователей.
* Контроль целостности информации.
* Аутентификация информации.
* Электронная цифровая подпись.
* Системы тайного электронного голосования.
* Электронные деньги.
* Электронная жеребьевка.
* Защита от отказа факта приема сообщения.

2.5. Скремблирование

Скремблерами называются программные или аппаратные реализации алгоритма, позволяющего шифровать побитно непрерывные потоки информации. Сам скремблер представляет из себя набор бит, изменяющихся на каждом шаге по определенному алгоритму. После выполнения каждого очередного шага на его выходе появляется шифрующий бит – либо 0, либо 1, который накладывается на текущий бит информационного потока операцией XOR.

Суть скремблирования заключается в побитном изменении проходящего через систему потока данных. Практически единственной операцией, используемой в скремблерах является XOR – "побитное исключающее ИЛИ". Параллельно прохождению информационного потока в скремблере по определенному правилу генерируется поток бит – кодирующий поток. Как прямое, так и обратное шифрование осуществляется наложением по XOR кодирующей последовательности на исходную.

Генерация кодирующей последовательности бит производится циклически из небольшого начального объема информации – ключа по следующему алгоритму. Из текущего набора бит выбираются значения определенных разрядов и складываются по XOR между собой. Все разряды сдвигаются на 1 бит, а только что полученное значение ("0" или "1") помещается в освободившийся самый младший разряд. Значение, находившееся в самом старшем разряде до сдвига, добавляется в кодирующую последовательность, становясь очередным ее битом (см. рис.1).

  
Рисунок 2.1 – Принцип работы скремблера

Из теории передачи данных криптография заимствовала для записи подобных схем двоичную систему записи. По ней изображенный на рисунке скремблер записывается комбинацией "100112" – единицы соответствуют разрядам, с которых снимаются биты для формирования обратной связи.

Рассмотрим пример кодирования информационной последовательности 0101112 скремблером 1012 с начальным ключом 1102.

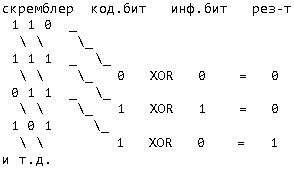


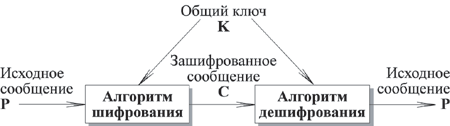
Рисунок 2.2 – Принцип работы скремблера

Декодирование заскремблированных последовательностей происходит по той же самой схеме, что и кодирование. Именно для этого в алгоритмах применяется результирующее кодирование по "исключающему ИЛИ" – схема, однозначно восстановимая при раскодировании без каких-либо дополнительных вычислительных затрат.

**ТЕМА 3. Алгоритмы симметричного шифрования**

3.1. Основные понятия и определения

Рассмотрим общую схему симметричного шифрования.



**Рисунок 3.1 – Схема симметричного шифрования**

**Симметричные алгоритмы** можно классифицировать на:

1. *Моно- и много­алфавитные подстано­ки.*
2. *Перестановки*
3. *Гаммирование.*
4. *Блочные шифры.*

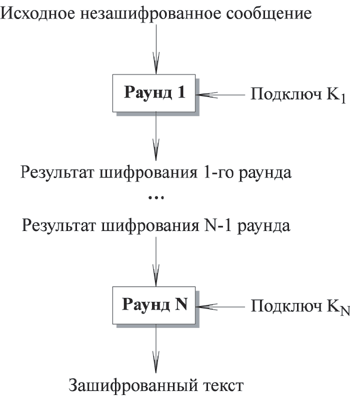
Алгоритмы симметричного шифрования различаются способом, которым обрабатывается исходный текст. Возможно шифрование блоками или шифрование потоком.

Все действия, производимые над данными блочным криптоалгоритмом, основаны на том факте, что преобразуемый блок может быть представлен в виде целого неотрицательного числа из диапазона, соответствующего его разрядности. Так, например, 32-битный блок данных можно интерпретировать как число из диапазона 0..4'294'967'295. Кроме того, блок, разрядность которого обычно является "степенью двойки", можно трактовать как несколько независимых неотрицательных чисел из меньшего диапазона (рассмотренный выше 32-битный блок можно также представить в виде 2 независимых чисел из диапазона 0..65535 или в виде 4 независимых чисел из диапазона 0..255).

Над этими числами блочным криптоалгоритмом и производятся по определенной схеме следующие действия:

* Биективные математические функции (cложение, исключающее ИЛИ, умножение по модулю 2N+1, умножение по модулю 2N);
* Битовые сдвиги (арифметические и циклические сдвиги влево и вправо);
* Табличные подстановки.

Описанные выше операции циклически повторяются в алгоритме, образуя так называемые *раунды*. Входом каждого *раунда* является выход предыдущего *раунда* и ключ, который получен по определенному алгоритму из ключа шифрования K. Ключ *раунда* называется *подключом*. Каждый алгоритм шифрования может быть представлен следующим образом:



**Рисунок 3.2 – Структура алгоритма симметричного шифрования**

Характерным признаком блочных алгоритмов является многократное и косвенное использование материала ключа. Это диктуется в первую очередь требованием невозможности обратного декодирования в отношении ключа при известных исходном и зашифрованном текстах. Для решения этой задачи в приведенных выше преобразованиях чаще всего используется не само значение ключа или его части, а некоторая, иногда необратимая (небиективная) функция от материала ключа. Более того, в подобных преобразованиях один и тот же блок или элемент ключа используется многократно. Это позволяет при выполнении условия обратимости функции относительно величины X сделать функцию необратимой относительно ключа Key.

Поскольку операция зашифровки или расшифровки отдельного блока в процессе кодирования пакета информации выполняется многократно (иногда до сотен тысяч раз), а значение ключа и, следовательно, функций *Vi*(Key) остается неизменным, то иногда становится целесообразно заранее однократно вычислить данные значения и хранить их в оперативной памяти совместно с ключом. Поскольку эти значения зависят только от ключа, то оин в криптографии называются материалом ключа. Необходимо отметить, что данная операция никоим образом не изменяет ни длину ключа, ни криптостойкость алгоритма в целом. Здесь происходит лишь оптимизация скорости вычислений путем кеширования (англ. caching) промежуточных результатов. Описанные действия встречаются практически во многих блочных криптоалгоритмах и носят название расширение ключа (англ. key scheduling).

**Области применения**

Стандартный алгоритм шифрования должен быть применим во многих приложениях:

* Шифрование данных.
* Создание случайных чисел.
* Хэширование.

**Платформы**

Стандартный алгоритм шифрования должен быть реализован на различных платформах:

* Специализированная аппаратура.
* Большие процессоры.
* Процессоры среднего размера.
* Малые процессоры.

**Используемые критерии при разработке алгоритмов**

Считается, что алгоритм симметричного шифрования должен:

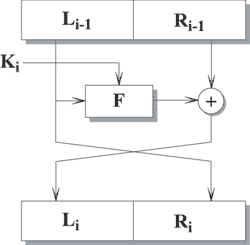
* Иметь размер блока 64 или 128 бит.
* Иметь масштабируемый ключ до 256 бит.
* Использовать простые операции, которые эффективны на микропро­цессорах, Не должно использоваться сдвигов переменной длины, побитных перестановок или условных переходов.
* Должна быть возможность реализации алгоритма на 8-битном про­цессоре с минимальными требованиями к памяти.
* Использовать заранее вычисленные подключи.
* Состоять из переменного числа итераций.
* По возможности не иметь слабых ключей.
* Задействовать подключи, которые являются односторонним хэшем ключа.
* Не иметь линейных структур, которые уменьшают комплексность и не обеспечивают исчерпывающий поиск.
* Использовать простую для понимания разработку.

3.2. Сеть Фейштеля

Блочный алгоритм преобразовывает n-битный блок незашифрованного текста в n-битный блок зашифрованного текста. Число блоков длины n равно 2n. Для того чтобы преобразование было обратимым, каждый из таких блоков должен преобразовываться в свой уникальный блок зашифрованного текста. При маленькой длине блока такая подстановка плохо скрывает статистические особенности незашифрованного текста. Если блок имеет длину 64 бита, то он уже хорошо скрывает статистические особенности исходного текста. Но в данном случае преобразование текста не может быть произвольным в силу того, что ключом будет являться само преобразование, что исключает эффективную как программную, так и аппаратную реализации.

Наиболее широкое распространение получили сети Фейштеля, так как, с одной стороны, они удовлетворяют всем требованиям к алгорит­мам симметричного шифрования, а с другой стороны, достаточно просты и компактны.

Сеть Фейштеля имеет следующую структуру. Входной блок делится на несколько равной длины подблоков, называемых ветвями. В случае, если блок имеет длину 64 бита, используются две ветви по 32 бита каждая. Каждая ветвь обрабатывается независимо от другой, после чего осуществляется циклический сдвиг всех ветвей влево. Такое преобразование вы­полняется несколько циклов или раундов. В случае двух ветвей каждый раунд имеет структуру, показанную на рис.



**Рисунок 3.2 –I-ый раунд сети Фейштеля**

Каждый раунд состоит из вычисления функции *F* для одной ветви и побитового выполнения операции XOR результата *F* с другой ветвью. После этого ветви меняются местами. Считается, что оптимальное число раундов – от 8 до 32. Важно то, что увеличение количества раундов значительно 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 допустимые пределы.

Сеть Фейштеля является обратимой даже в том случае, если функция *F* не является таковой, так как для расшифрования не требуется вычислять *F-1*.Для расшифрования используется тот же алгоритм, но на вход подается зашифрованный текст, и ключи используются в обратном порядке.

Достоинства сети Фейштеля:

* процедуры шифрования и расшифрования совпадают, с тем исключением, что ключевая информация при расшифровании используется в обратном порядке;
* хорошая изученность алгоритмов на основе сетей Фейштеля;
* для построения устройств шифрования можно использовать те же блоки в цепях шифрования и расшифрования.

Недостатком является то, что на каждой итерации изменяется только половина блока обрабатываемого текста, что приводит к необходимости увеличивать число итераций для достижения требуемой стойкости.

В настоящее 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.

Основной характеристикой алгоритма, построенного на основе сети Фейштеля, является функция *F*. Различные варианты касаются также на­чального и конечного преобразований. Подобные преобразования, назы­ваемые забеливанием (whitening), осуществляются для того, чтобы вы­полнить начальную рандомизацию входного текста.

3.3. Дифференциальный и линейный криптоанализ

Понятие *дифференциального криптоанализа* было введено Эли Бихамом (Biham) и Ади Шамиром (Shamir) в 1990 году. Конечная задача ***дифференциального криптоанализа*** - используя свойства алгоритма, в основном свойства *S-box*, определить *подключ* *раунда*. Конкретный способ *дифференциального криптоанализа* зависит от рассматриваемого алгоритма шифрования.

Если в основе алгоритма лежит *сеть Фейштеля*, то можно считать, что блок m состоит из двух половин - m0 и m1. *Дифференциальный криптоанализ* рассматривает отличия, которые происходят в каждой половине при шифровании. (Для алгоритма *DES* "отличия" определяются с помощью операции XOR, для других алгоритмов возможен иной способ). Выбирается пара незашифрованных текстов с фиксированным отличием. Затем анализируются отличия, получившиеся после шифрования одним *раундом* алгоритма, и определяются вероятности различных ключей. Если для многих пар входных значений, имеющих одно и то же отличие Х, при использовании одного и того же *подключа* одинаковыми (Y) оказываются и отличия соответствующих выходных значений, то можно говорить, что Х влечет Y с определенной вероятностью. Если эта вероятность близка к единице, то можно считать, что *подключ* *раунда* найден с данной вероятностью. Так как *раунды* алгоритма независимы, вероятности определения *подключа* каждого *раунда* следует перемножать. Как мы помним, считается, что результат шифрования данной пары известен. Результаты *дифференциального криптоанализа* используются как при разработке конкретных *S-box*, так и при определении оптимального числа *раундов*.

Другим способом криптоанализа является *линейный криптоанализ*, который использует линейные приближения преобразований, выполняемых алгоритмом шифрования. Данный метод позволяет найти ключ, имея достаточно большое число пар (незашифрованный текст, зашифрованный текст). Рассмотрим основные принципы, на которых базируется *линейный криптоанализ*. Обозначим

|  |  |
| --- | --- |
| P[1], … , P[n] - незашифрованный блок сообщения. | |
| C[1], … , C[n] - зашифрованный блок сообщения. | |
| K[1], … , K[m] - ключ. |
| A[i, j, …, k] = A[i] A[j] … A[k] |

Целью *линейного криптоанализа* является поиск линейного уравнения вида

P[α1, α 2, …, α a]C[β1, β2, …, βb ] = K[γ1, …, γc]

Выполняющееся с вероятностью р <> 0.5. α i, βi и γi - фиксированные позиции в блоках сообщения и ключе. Чем больше р отклоняется от 0.5, тем более подходящим считается уравнение.

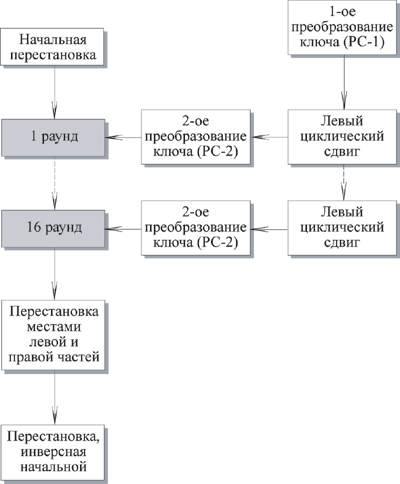
Это уравнение означает, что если выполнить операцию XOR над некоторыми битами незашифрованного сообщения и над некоторыми битами зашифрованного сообщения, получится бит, представляющий собой XOR некоторых битов ключа. Это называется линейным приближением, которое может быть верным с вероятностью р.

Уравнения составляются следующим образом. Вычисляются значения левой части для большого числа пар соответствующих фрагментов незашифрованного и зашифрованного блоков. Если результат оказывается равен нулю более чем в половине случаев, то полагают, что K[γ1, …, γс] = 0. Если в большинстве случаев получается 1, полагают, что K[γ1, …, γс] = 1. Таким образом получают систему уравнений, решением которой является ключ.

3.4. Алгоритм DES

Самым распространенным и наиболее известным алгоритмом симметричного шифрования является *DES* (Data Encryption Standard). Алгоритм был разработан в 1977 году, в 1980 году был принят NIST (National Institute of Standards and Technology США) в качестве стандарта (FIPS PUB 46).

*DES* является классической *сетью Фейстеля* с двумя ветвями. Данные шифруются 64-битными блоками, используя 56-битный ключ. Алгоритм преобразует за несколько *раундов* 64-битный вход в 64-битный выход. Длина ключа равна 56 битам. Процесс шифрования состоит из четырех этапов. На первом из них выполняется начальная перестановка (*IP*) 64-битного исходного текста (забеливание), во время которой биты переупорядочиваются в соответствии со стандартной таблицей. Следующий этап состоит из 16 *раундов* одной и той же функции, которая использует операции сдвига и подстановки. На третьем этапе левая и правая половины выхода последней (16-й) итерации меняются местами. Наконец,на четвертом этапе выполняется перестановка IP-1 результата, полученного на третьем этапе. Перестановка IP-1 инверсна начальной перестановке.



**Рисунок 3.3 –** **Общая схема DES**

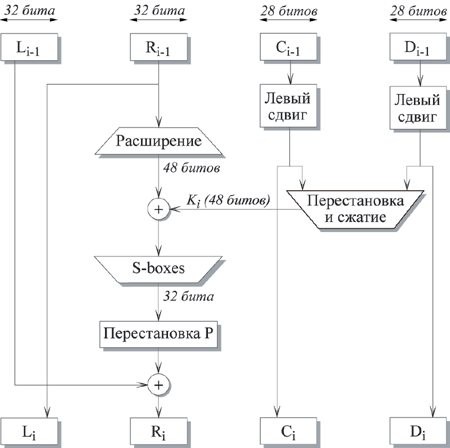
**Шифрование**

**Начальная перестановка**

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если М- это произвольные 64 бита, то X = IP (M)-переставленные 64 бита. Если применить обратную функцию перестановки Y = IP-1 (X) = IP-1 (IP(M)), то получится первоначальная последовательность бит.

**Последовательность преобразований отдельного раунда**

Теперь рассмотрим последовательность преобразований, используемую в каждом *раунде*.



**Рисунок 3.4 –I-ый раунд DES**

64-битный входной блок проходит через 16 *раундов*, при этом на каждой итерации получается промежуточное 64-битное значение. Левая и правая части каждого промежуточного значения трактуются как отдельные 32-битные значения, обозначенные *L* и *R*. Каждую итерацию можно описать следующим образом:

Li = Ri-1

Ri = Li-1  F(Ri-1, Ki)

Где обозначает операцию XOR.

Таким образом, выход левой половины *Li* равен входу правой половины *Ri-1*. Выход правой половины *Ri*является результатом применения операции XOR к *Li-1* и функции *F*, зависящей от *Ri-1* и *Ki*.

Рассмотрим функцию *F* более подробно.

*Ri*, которое подается на вход функции *F*, имеет длину 32 бита. Вначале Riрасширяется до 48 бит, используя таблицу, которая определяет перестановку плюс расширение на 16 бит. Расширение происходит следующим образом. 32 бита разбиваются на группы по 4 бита и затем расширяются до 6 бит, присоединяя крайние биты из двух соседних групп. Например, если часть входного сообщения

. . . efgh ijkl mnop . . .

то в результате расширения получается сообщение

. . . defghi hijklm lmnopq . . .

После этого для полученного 48-битного значения выполняется операция XOR с 48-битным *подключом Ki*. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32-битное значение.

Подстановка состоит из восьми *S-boxes,*каждый из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения *S-box* определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

Далее полученное 32-битное значение обрабатывается с помощью перестановки *Р*, целью которой является максимальное переупорядочивание бит, чтобы в следующем *раунде* шифрования с большой вероятностью каждый бит обрабатывался другим *S-box*.

**Создание подключей**

Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа,используемого на входе алгоритма, вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1). Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как C0 и D0 соответственно. На каждом *раунде Ci* и *Di*независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *раунда*. Полученные значения являются входом следующего *раунда*. Они также представляют собой вход в Permuted Choice 2 (РС-2), который создает 48-битное выходное значение, являющееся входом функции *F*(*Ri-1*, *Ki*).

**Дешифрование**

Процесс дешифрования аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности. *K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

**Проблемой DES** является малая длина ключа. 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.

**Двойной DES**

Простейший способ увеличить длину ключа состоит в повторном применении *DES* с двумя разными ключами. Используя незашифрованное сообщение P и два ключа K1 и K2, зашифрованное сообщение С можно получить следующим образом:

C = Ek2 [Ek1 [P]]

При дешифрования два ключа применяются в обратном порядке:

P = Dk1 [Dk2 [C]]

В этом случае длина ключа равна 56 \* 2 = 112 бит.

**Атака "встреча посередине"**

Для приведенного выше алгоритма двойного *DES* существует так называемая атака "встреча посередине". Она основана на следующем свойстве алгоритма. Мы имеем

С = Ek2 [Ek1 [P]]

Тогда

X = Ek1 [P] = Dk2 [C].

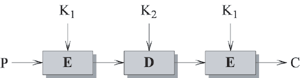
Атака состоит в следующем. Требуется, чтобы атакующий знал хотя бы одну пару незашифрованный текст и соответствующий ему зашифрованный текст: (Р, С). В этом случае, во-первых, шифруется Р для всех возможных 256 значений K1. Этот результат запоминается в таблице, и затем таблица упорядочивается по значению Х. Следующий шаг состоит в дешифровании С, с применением всех возможных 256 значений K2. Для каждого выполненного дешифрования ищется равное ему значение в первой таблице. Если соответствующее значение найдено, то считается, что эти ключи могут быть правильными, и они проверяются для следующей известной пары незашифрованный текст, зашифрованный текст.

**Тройной DES с двумя ключами**

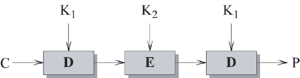
Очевидное противодействие атаке "встреча посередине" состоит в использовании третьей стадии шифрования с тремя различными ключами. Это поднимает стоимость лобовой атаки до 2168, которая на сегодняшний день считается выше практических возможностей. Но при этом длина ключа равна 56 \* 3 = 168 бит, что иногда бывает громоздко.

В качестве альтернативы предлагается метод тройного шифрования,использующий только два ключа. В этом случае выполняется последовательность зашифрование-расшифрование-зашифрование (EDE).

C = EK1 [DK2 [EK1 [P]]]



**Рисунок 3.5 – Шифрование тройным DES**



**Рисунок 3.6 – Дешифрование тройным DES**

Не имеет большого значения, что используется на второй стадии: шифрование или дешифрование. В случае использования дешифрования существует только то преимущество, что можно *тройной DES* свести к обычному одиночному *DES*, используя K1 = K2:

C = EK1 [DK1 [EK1 [P]]] = EK1 [P]

Известных криптографических атак на *тройной DES* не существует. Цена подбора ключа в *тройном DES* равна 2112.

3.5. Алгоритм Blowfish

***Blowfish*** является сетью Фейштеля, у которой количество итераций равно 16. Длина блока равна 64 битам, ключ может иметь любую длину в пределах 448 бит. Хотя перед началом любого шифрования выполняется сложная фаза инициализации, само шифрование данных выполняется достаточно быстро.

Алгоритм предназначен в основном для приложений, в которых ключ меняется нечасто, к тому же существует фаза начального рукопожатия, во время которой происходит аутентификация сторон и согласование общих параметров и секретов. Классическим примером подобных приложений является сетевое взаимодействие. При реализации на 32-битных микропроцессорах с большим кэшем данных *Blowfish* значительно быстрее DES.

Алгоритм состоит из двух частей: расширение ключа и шифрование данных. Расширение ключа преобразует ключ длиной, по крайней мере, 448 бит в несколько массивов *подключей* общей длиной 4168 байт.

В основе алгоритма лежит сеть Фейштеля с 16 итерациями. Каждая итерация состоит из перестановки, зависящей от ключа, и подстановки, зависящей от ключа и данных. Операциями являются XOR и сложение 32-битных слов.

*Blowfish* использует большое количество *подключей*. Эти ключи должны быть вычислены заранее, до начала любого шифрования или дешифрования данных. Элементы алгоритма:

1. Р - массив, состоящий из восемнадцати 32-битных *подключей*:

Р1, Р2, ..., Р18.

1. Четыре 32-битных *S-boxes* c 256 входами каждый. Первый индекс означает номер *S-box*, второй индекс - номер входа.
2. S1,0, S1,1, … S1,255;
3. S2,0, S2,1, … S2,255;
4. S3,0, S3,1, … S3,255;
5. S4,0, S4,1, … S4,255;

Метод, используемый для вычисления этих *подключей*, будет описан ниже.

**Шифрование**

Входом является 64-битный элемент данных X, который делится на две 32-битные половины, Xl и Xr.

Xl = Xl XOR Pi

Xr = F (Xl) XOR Xr

Swap Xl and Xr

**Функция F**

Разделить Xl на четыре 8-битных элемента A, B, C, D.

F (Xl) = ((S1,А + S2,B mod 232) XOR S3,C) + S4,D mod 232

Дешифрование отличается от шифрования тем, что Pi используются в обратном порядке.

**Генерация подключей**

*Подключи* вычисляются с использованием самого *алгоритма Blowfish*.

1. Инициализировать первый Р-массив и четыре *S-boxes* фиксированной строкой.
2. Выполнить операцию XOR P1 с первыми 32 битами ключа, операцию XOR P2 со вторыми 32 битами ключа и т.д. Повторять цикл до тех пор, пока весь Р-массив не будет побитово сложен со всеми битами ключа. Для коротких ключей выполняется конкатенация ключа с самим собой.
3. Зашифровать нулевую строку *алгоритмом Blowfish*, используя *подключи*, описанные в пунктах (1) и (2).
4. Заменить Р1 и Р2 выходом, полученным на шаге (3).
5. Зашифровать выход шага (3), используя *алгоритм Blowfish* с модифицированными *подключами*.
6. Заменить Р3 и Р4 выходом, полученным на шаге (5).
7. Продолжить процесс, заменяя все элементы Р-массива, а затем все четыре *S-boxes*, выходами соответствующим образом модифицированного *алгоритма Blowfish*.

Для создания всех *подключей* требуется 521 итерация.

3.5. Алгоритм IDEA

***IDEA*** (International Data Encryption Algorithm) является блочным симметричным алгоритмом шифрования, разработанным Сюдзя Лай и Джеймсом Массей из швейцарского федерального института технологий. Первоначальная версия была опубликована в 1990 году. Пересмотренная версия алгоритма, усиленная средствами защиты от дифференциальных криптографических атак, была представлена в 1991году и подробно описана в 1992 году.

*IDEA* является блочным алгоритмом, который использует 128-битовый *ключ для*

Каждая операция *IDEA* выполняется над двумя 16-битными входами и создает один 16-битный выход. Этими операциями являются:

1. Побитовое исключающее OR, обозначаемое как .
2. Сумма целых по модулю 216 (по модулю 65536), при этом входы и выходы трактуются как беззнаковые 16-битные целые. Эту операцию обозначим как +.
3. Умножение целых по модулю 216 + 1 (по модулю 65537), при этом входы и выходы трактуются как беззнаковые 16-битные целые, за исключением того, что блок из одних нулей трактуется как 216. Эту операцию обозначим как •.

Эти три операции являются несовместимыми в том смысле, что:

1. Не существует пары из трех операций, удовлетворяющих дистрибутивному закону. Например a • (b + c) <> (a • b) + (a • c)
2. Не существует пары из трех операций, удовлетворяющих ассоциативному закону. Например a + (b c) <> (a + b)  c

**Шифрование**

*Алгоритм IDEA* состоит из восьми *раундов*, за которыми следует заключительное преобразование. Алгоритм разделяет блок на четыре 16-битных подблока. Каждый *раунд* получает на входе четыре 16-битных подблока и создает четыре 16-битных выходных подблока. Заключительное преобразование также получает на входе четыре 16-битных подблока и создает четыре 16-битных подблока. Каждый *раунд* использует шесть16-битных ключей, заключительное преобразование использует четыре *подключа*, т.е. всего в алгоритме используется 52 *подключа*.

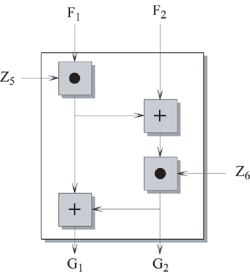


**Рисунок 3.7 – Алгоритм IDEA**

**Последовательность преобразований отдельного раунда**

Рассмотрим последовательность преобразований отдельного *раунда*.

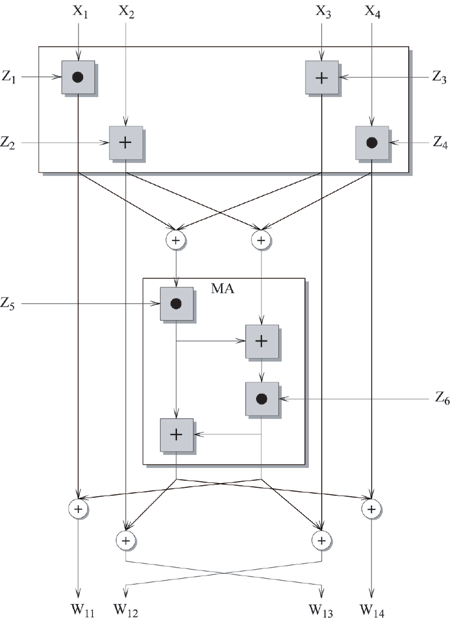
Одним из основных элементов алгоритма, обеспечивающих диффузию,является структура, называемая МА (умножение/сложение):



**Рисунок 3.8 – Структура МА (умножение/сложение)**

На вход этой структуре подаются два 16-битных значения и два 16-битных *подключа*, на выходе создаются два 16-битных значения.

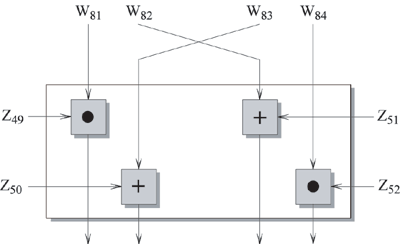
*Раунд* начинается с преобразования, которое комбинирует четыре входных подблока с четырьмя *подключами*, используя операции сложения и умножения. Четыре выходных блока этого преобразования комбинируются,используя операцию XOR для формирования двух 16-битных блоков,которые являются входами МА структуры. Кроме того, МА структура имеет на входе еще два *подключа* и создает два 16-битных выхода.



**Рисунок 3.9 – I-ый раунд IDEA**

В заключении четыре выходных подблока первого преобразования комбинируются с двумя выходными подблоками МА структуры, используя XOR для создания четырех выходных подблоков данной итерации. Заметим,что два выхода, которые частично создаются вторым и третьим входами (Х2 и Х3), меняются местами для создания второго и третьего выходов (W12 и W13). Это увеличивает перемешивание бит и делает алгоритм более стойким для дифференциального криптоанализа.

Рассмотрим девятый *раунд алгоритма*, обозначенный как заключительное преобразование. Это та же структура, что была описана выше. Единственная разница состоит в том, что второй и третий входы меняются местами. Это сделано для того, чтобы дешифрование имело ту же структуру, что и шифрование. Заметим, что девятая стадия требует только четыре входных *подключа*, в то время как для первых восьми стадий для каждой из них необходимо шесть входных *подключей*.



**Рисунок 3.10 – Заключительное преобразование**

**Создание подключей**

Пятьдесят два 16-битных *подключа* создаются из 128-битного *ключа шифрования* следующим образом. Первые восемь *подключей*, которые обозначим как Z1, Z2, ..., Z8, получаются непосредственно из ключа, при этом Z1 равен первым 16 битам, Z2 равен следующим 16 битам и т.д. Затем происходит циклический сдвиг ключа влево на 25 бит, и создаются следующие восемь *подключей*. Эта процедура повторяется до тех пор, пока не будут созданы все 52 *подключа*.

Заметим, что каждый первый *подключ раунда* получен из своего подмножества бит ключа. Если весь ключ обозначить как Z[1..128], то первыми ключами в восьми *раундах* будут:

**Z1 = Z [1..16] Z25 = Z [76..91]**

**Z7 = Z [97..112] Z31 = Z [44..59]**

**Z13 = Z [90..105] Z37 = Z [37..52]**

**Z19 = Z [83..98] Z43 = Z [30..45]**

Хотя на каждом *раунде* за исключением первого и восьмого используются только 96 бит *подключа*, множество бит ключа на каждой итерации не пересекаются, и не существует отношения простого сдвига между *подключами* разных *раундов*. Это происходит потому, что на каждом *раунде* используется только шесть *подключей*, в то время как при каждой ротации ключа получается восемь *подключей*.

**Дешифрование**

Процесс дешифрования аналогичен процессу шифрования. Дешифрование состоит в использовании зашифрованного текста в качестве входа в ту же самую структуру *IDEA*, но с другим набором ключей. Дешифрующие ключи U1,...,U52 получаются из шифрующих ключей следующим образом:

1. Первые четыре *подключа* i-ого *раунда* дешифрования получаются из первых четырех *подключей* (10-i)-го *раунда* шифрования, где стадия заключительного преобразования считается 9-м *раундом*. Первый и четвертый ключи дешифрования эквивалентны мультипликативной инверсии по модулю (216 + 1) соответствующих первого и четвертого *подключей* шифрования. Для *раундов* со 2 по 8 второй и третий *подключи* дешифрования эквивалентны аддитивной инверсии по модулю (216) соответствующих третьего и второго *подключей* шифрования. Для *раундов* 1 и 9 второй и третий *подключи* дешифрования эквивалентны аддитивной инверсии по модулю (216) соответствующих второго и третьего *подключей* шифрования.
2. Для первых восьми *раундов* последние два *подключа* i *раунда* дешифрования эквивалентны последним двум *подключам* (9-i) *раунда* шифрования.

Для мультипликативной инверсии используется нотация Zj-1, т.е.:

**Zj • Zj-1 =1 mod (216 + 1)**

Так как 216 + 1 является простым числом, каждое ненулевое целое Zj <= 216имеет уникальную мультипликативную инверсию по модулю ( 216 + 1 ). Для аддитивной инверсии используется нотация ( -Zj ), таким образом, мы имеем: -Zj + Zj = 0 mod (216)

Для доказательства того, что алгоритм дешифрования с соответствующими *подключами* имеет корректный результат, рассмотрим одновременно процессы шифрования и дешифрования. Каждый из восьми *раундов* разбит на две стадии преобразования, первая из которых называется трансформацией, а вторая шифрованием.

При шифровании поддерживаются следующие соотношения на выходе трансформации:

Y1 = W81 • Z49 Y3 = W82 + Z51

Y2 = W83 + Z50 Y4 = W84 • Z52

Первая стадия первого *раунда* процесса дешифрования поддерживает следующие соотношения:

J11 = Y1 • U1 J13 = Y3 + U3

J12 = Y2 + U2 J14 = Y4 • U4

Подставляя соответствующие значения, получаем:

J11 = Y1 • Z49-1 = W81 • Z49 • Z49-1 = W81

J12 = Y2 + -Z50 = W83 + Z50 = W83 + Z50 + -Z50 = W83

J13 = Y3 + -Z51 = W82 + Z51 + -Z51 = W82

J14 = Y4 • Z52-1 = W84 • Z52 • Z52-1 = W84

Таким образом, выход первой стадии процесса дешифрования эквивалентен входу последней стадии процесса шифрования за исключением чередования второго и третьего блоков. Теперь рассмотрим следующие отношения:

W81 = I81   MAR(I81   I83, I82   I84)

W82 = I83   MAR(I81   I83, I82   I84)

W83 = I82   MAL(I81   I83, I82   I84)

W84 = I84   MAL(I81   I83, I82   I84)

Где MAR(X, Y)есть правый выход МА структуры с входами Х и Y, и MAL(X, Y) есть левый выход МА структуры с входами Х и Y. Теперь получаем

V11 = J11   MAR (J11   J13, J12   J14) =

W81   MAR(W81   W82, W83   W84) =

I81   MAR(I81   I83, I82   I84) 

MAR[ I81   MAR(I81   I83, I82   I84)  I83  

MAR(I81   I83, I82   I84), I82  

MAL(I81   I83, I82   I84)  I84  

MAL(I81   I83, I82   I84) ] =

I81   MAR(I81   I83, I82   I84) 

MAR(I81   I83, I82  I84) =I81

Аналогично мы имеем

V12 = I83

V13 = I82

V14 = I84

Таким образом, выход второй стадии процесса дешифрования эквивалентен входу предпоследней стадии процесса шифрования за исключением чередования второго и третьего подблоков. Аналогично можно показать,что

V81 = I11

V82 = I13

V83 = I12

V84 = I14

Наконец, так как выход трансформации процесса дешифрования эквивалентен первой стадии процесса шифрования за исключением чередования второго и третьего подблоков, получается, что выход всего процесса шифрования эквивалентен входу процесса шифрования.

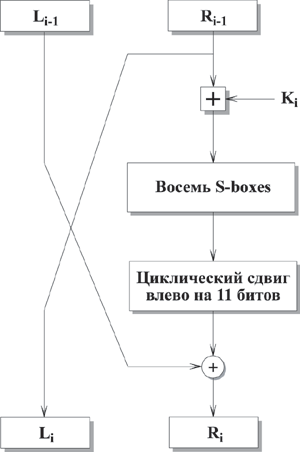
3.5. Алгоритм *ГОСТ 28147*

*Алгоритм ГОСТ 28147* является отечественным стандартом для алгоритмов симметричного шифрования. *ГОСТ 28147* разработан в 1989 году, является блочным алгоритмом шифрования, длина блока равна 64 битам, длина ключа равна 256 битам, количество *раундов* равно 32. Алгоритм представляет собой классическую сеть Фейштеля.

Li = Ri-1

Ri = Li  f (Ri-1, Ki)

Функция F проста. Сначала правая половина и i-ый *подключ* складываются по модулю 232. Затем результат разбивается на восемь 4-битовых значений, каждое из которых подается на вход *S-box*. *ГОСТ 28147* использует восемь различных *S-boxes*, каждый из которых имеет 4-битовый вход и 4-битовый выход. Выходы всех *S-boxes* объединяются в 32-битное слово, которое затем циклически сдвигается на 11 битов влево. Наконец, с помощью XOR результат объединяется с левой половиной, в результате чего получается новая правая половина.



**Рисунок 3.11 – I-ый раунд ГОСТ 28147**

Генерация ключей проста. 256-битный ключ разбивается на восемь 32-битных подключей. Алгоритм имеет 32 раунда, поэтому каждый подключ используется в четырех раундах по следующей схеме:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Раунд** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Подключ** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Раунд** | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| **Подключ** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Раунд** | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| **Подключ** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Раунд** | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| **Подключ** | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Считается, что стойкость алгоритма ГОСТ 28147 во многом определяется структурой S-boxes. Входом и выходом S-box являются 4-битные числа, поэтому каждый S-box может быть представлен в виде строки чисел от 0 до 15, расположенных в некотором порядке. Тогда порядковый номер числа будет являться входным значением S-box, а само число - выходным значением S-box.

3.6. Режимы выполнения методов симметричного шифрования

Для любого симметричного блочного алгоритма шифрования определено четыре режима выполнения.

***ECB*** - Electronic Codebook - каждый блок из 64 битов незашифрованного текста шифруется независимо от остальных блоков, с применением одного и того же *ключа шифрования*. Типичные приложения - безопасная передача одиночных значений.

***CBC*** - Cipher Block Chaining - вход криптографического алгоритма является результатом применения операции XOR к следующему блоку незашифрованного текста и предыдущему блоку зашифрованного текста. Типичные приложения - общая блокоориентированная передача.

***CFB*** - Cipher Feedback - при каждом вызове алгоритма обрабатывается J битов входного значения. Предшествующий зашифрованный блок используется в качестве входа в алгоритм; к J битам выхода алгоритма и следующему незашифрованному блоку из J битов применяется операция XOR, результатом которой является следующий зашифрованный блок из J битов. Типичные приложения - потокоориентированная передача, аутентификация.

***OFB*** - Output Feedback - аналогичен *CFB*, за исключением того, что на вход алгоритма при шифровании следующего блока подается результат шифрования предыдущего блока; только после этого выполняется операция XOR с очередными J битами незашифрованного текста. Типичные приложения - потокоориентированная передача по зашумленному каналу (например, спутниковая связь).

3.7. Алгоритмы создания случайных чисел

Большинство алгоритмов сетевой безопасности, основанных на криптографии, используют случайные числа. Например:

1. Схемы взаимной аутентификации.
2. Ключ сессии, созданный KDC или кем-либо из участников.

Двумя основными требованиями к последовательности случайных чисел являются *случайность* и *непредсказуемость*.

Шифрующие приложения используют для создания случайных чисел специальные алгоритмы. Эти алгоритмы детерминированы и, следовательно, создают последовательность чисел, которая не является статистически случайной. Тем не менее, если алгоритм хороший, полученная последовательность будет проходить много тестов на случайность. Такие числа часто называют *псевдослучайными числами*.

**Генераторы псевдослучайных чисел**

Первой широко используемой технологией создания случайного числа был алгоритм, предложенный Лехмером, который известен как метод линейного конгруента. Этот алгоритм параметризуется четырьмя числами следующим образом:

|  |  |  |
| --- | --- | --- |
| **M** | Модуль (основание системы) | m > 0 |
| **A** | Множитель | 0 ≤ a < m |
| **C** | Приращение | 0 ≤ с < m |
| **Х0** | Начальное значение или зерно (seed) | 0 ≤ Х0 < m |

Последовательность случайных чисел {Xn} получается с помощью следующего итерационного равенства:

1. Xn+1 = (a Xn + c) mod m

Если m, а и с являются целыми, то создается последовательность целых чисел в диапазоне 0 ≤ Xn < m.

Существует три критерия, используемые при выборе генератора случайных чисел:

1. Функция должна создавать полный период, т.е. все числа между 0 и m до того, как создаваемые числа начнут повторяться.
2. Создаваемая последовательность должна появляться случайно. Последовательность не является случайной, так как она создается детерминированно, но различные статистические тесты, которые могут применяться, должны показывать, что последовательность случайна.
3. Функция должна эффективно реализовываться на 32-битных процессорах.

Значения а, с и m должны быть выбраны таким образом, чтобы эти три критерия выполнялись. В соответствии с первым критерием можно показать, что если m является простым и с = 0, то при определенном значении а период, создаваемый функцией, будет равен m-1. Для 32-битной арифметики соответствующее простое значение m = 231 - 1. Таким образом, функция создания *псевдослучайных чисел* имеет вид:

Xn+1 = (a Xn) mod (231 - 1)

Сила алгоритма линейного конгруента в том, что если сомножитель и модуль (основание) соответствующим образом подобраны, то результирующая последовательность чисел будет статистически неотличима от последовательности, являющейся случайной из набора 1, 2, ..., m-1. Но не может быть случайности в последовательности, полученной с использованием алгоритма, независимо от выбора начального значения Х0. Если значение выбрано, то оставшиеся числа в последовательности будут предопределены. Это всегда учитывается при криптоанализе.

Если противник знает, что используется алгоритм линейного конгруента, и если известны его параметры (а = 75, с = 0, m = 231 - 1), то, если раскрыто одно число, вся последовательность чисел становится известна. Даже если противник знает только, что используется алгоритм линейного конгруента, знания небольшой части последовательности достаточно для определения параметров алгоритма и всех последующих чисел. Предположим, что противник может определить значения Х0, Х1, Х2, Х3. Тогда :

Х1 = (а Х0 + с ) mod m

Х2 = (а Х1 + с ) mod m

Х3 = (а Х2 + с ) mod m

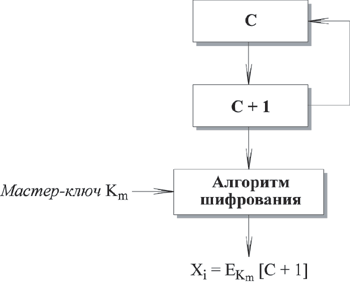
Эти равенства позволяют найти а, с и m.

Таким образом, хотя алгоритм и является хорошим генератором *псевдослучайной последовательности чисел*, желательно, чтобы реально используемая последовательность была непредсказуемой, поскольку в этом случае знание части последовательности не позволит определить будущие ее элементы. Эта цель может быть достигнута несколькими способами. Например, использование внутренних системных часов для модификации потока случайных чисел. Один из способов применения часов состоит в перезапуске последовательности после N чисел, используя текущее значение часов по модулю m в качестве нового начального значения. Другой способ состоит в простом добавлении значения текущего времени к каждому случайному числу по модулю m.

**Криптографически созданные случайные числа**

В криптографических приложениях целесообразно шифровать получающиеся случайные числа. Чаще всего используется три способа.

***Циклическое шифрование***



**Рисунок 3.12 – Циклическое шифрование**

В данном случае применяется способ создания ключа сессии из мастер-ключа. Счетчик с периодом N используется в качестве входа в шифрующее устройство. Например, в случае использования 56-битного ключа DES может применяться счетчик с периодом 256. После каждого созданного ключа значение счетчика увеличивается на 1. Таким образом, псевдослучайная последовательность, полученная по данной схеме, имеет полный период: каждое выходное значение Х0, Х1,...ХN-1 основано на различных значениях счетчика и, следовательно, Х0 neX1 neXN-1. Так как мастер-ключ защищен, легко показать, что любой секретный ключ не зависит от знания одного или более предыдущих секретных ключей.

***Режим Output Feedback DES***

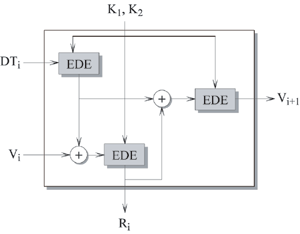
Режим *OFB* DES может применяться для генерации ключа, аналогично тому, как он используется для потокового шифрования. Заметим, что выходом каждой стадии шифрования является 64-битное значение, из которого только левые j битов подаются обратно для шифрования. 64-битные выходы составляют последовательность *псевдослучайных чисел* с хорошими статистическими свойствами.

***Генератор псевдослучайных чисел ANSI X9.17***

Один из наиболее сильных генераторов *псевдослучайных чисел* описан в ANSI X9.17. В число приложений, использующих эту технологию, входят приложения финансовой безопасности и PGP.

Алгоритмом шифрования является тройной DES. Генератор ANSI X9.17 состоит из следующих частей:

1. **Вход:** генератором управляют два псевдослучайных входа. Один является 64-битным представлением текущих даты и времени, которые изменяются каждый раз при создании числа. Другой является 64-битным начальным значением; оно инициализируется некоторым произвольным значением и изменяется в ходе генерации последовательности *псевдослучайных чисел*.
2. **Ключи:** генератор использует три модуля тройного DES. Все три используют одну и ту же пару 56-битных ключей, которая должна держаться в секрете и применяться только для генерации *псевдослучайного числа*.
3. **Выход:** выход состоит из 64-битного *псевдослучайного числа* и 64-битного значения, которое будет использоваться в качестве начального значения при создании следующего числа.



**Рисунок 3.13 – Генератор псевдослучайных чисел ANSI X9.17**

|  |
| --- |
| DTi - значение даты и времени на начало i-ой стадии генерации. |
| Vi - начальное значение для i-ой стадии генерации. |
| Ri - *псевдослучайное число*, созданное на i-ой стадии генерации. |
| K1, K2 - ключи, используемые на каждой стадии. |

Тогда:

Ri = EDEK1,K2 [ EDEK1,K2 [ DTi]  Vi ]

Vi+1 = EDEK1,K2 [ EDEK1,K2 [ DTi] Ri]

Схема включает использование 112-битного ключа и трех EDE-шифрований. На вход подаются два *псевдослучайных значения*: значение даты и времени и начальное значение очередной итерации, на выходе создаются начальное значение для следующей итерации и очередное *псевдослучайное значение*. Даже если *псевдослучайное число* Ri будет скомпрометировано, вычислить Vi+1 из Ri невозможно, и, следовательно, следующее *псевдослучайное значение* Ri+1, так как для получения Vi+1 дополнительно выполняются три операции EDE.

3.7. Алгоритм АЕS

Инициатива в разработке *AES* принадлежит NIST. В августе 1999 года были представлены пять финалистов. Ими стали *MARS*, *RC6™*, *Rijndael*, *Serpent* и *Twofish*. В итоге в качестве нового стандарта был принят *Rijndael.*

Практически все операции Rijndael определяются на уровне байта. Байты можно рассматривать как элементы конечного поля GF (28). Некоторые операции определены в терминах четырехбайтных слов.

**Поле GF(28)**

Элементы конечного поля могут быть представлены несколькими различными способами. Для любой степени простого числа существует единственное конечное поле, поэтому все представления GF (28) являются изоморфными. Несмотря на подобную эквивалентность, представление влияет на сложность реализации. Выберем классическое полиномиальное представление.

Байт b, состоящий из битов b7, b6, b5, b4, b3, b2, b1, b0, представляется в виде полинома с коэффициентами из {0, 1}:

b7х7 + b6х6 + b5х5 + b4х4 + b3х3 + b2х2 + b1х1 + b0

В полиномиальном представлении сумма двух элементов является полиномом с коэффициентами, которые равны сумме по модулю 2 коэффициентов слагаемых.

В полиномиальном представлении умножение в GF (28) соответствует умножению полиномов по модулю неприводимого двоичного полинома степени 8. Полином является неприводимым, если он не имеет делителей, кроме 1 и самого себя. Для Rijndael такой полином называется m(x) и определяется следующим образом:

m(x) = x8 + x4 + x3 + x + 1

Умножение, определенное выше, является ассоциативным, и существует единичный элемент ('01'). Для любого двоичного полинома b(x) не выше 8-й степени можно использовать расширенный алгоритм Евклида для вычисления полиномов a(x) и c(x) таких, что

b(x) a(x) + m(x) c(x) = 1

Следовательно,

a(x) • b(x) mod m(x) = 1

или

b-1(x) = a(x) mod m(x)

Более того, можно показать, что

a(x) • (b(x) + c(x)) = a(x) • b(x) + a(x) • c(x)

Из всего этого следует, что множество из 256 возможных значений байта образует конечное поле GF (28) c XOR в качестве сложения и умножением, определенным выше.

Если умножить b(x) на полином х, мы будем иметь:

b7x8 + b6x7 + b5x6 + b4x5 + b3x4 + b2x3 + b1x2 + b0x

x • b(x) получается понижением предыдущего результата по модулю m(x). Если b7 = 0, то данное понижение является тождественной операцией. Если b7 = 1, m(x) следует вычесть (т.е. XORed). Из этого следует, что умножение на х может быть реализовано на уровне байта как левый сдвиг и последующий побитовый XOR c '1B'. Данная операция обозначается как b = xtime (a).

Полиномы могут быть определены с коэффициентами из GF(28). В этом случае четырехбайтный вектор соответствует полиному степени 4.

Полиномы могут быть сложены простым сложением соответствующих коэффициентов. Как сложение в GF(28) является побитовым XOR, так и сложение двух векторов является простым побитовым XOR.

**Умножение на х**

При умножении b(x) на полином х будем иметь:

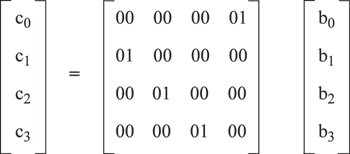
b3x4 + b2x3 + b1x2 + b0x

x b(x) получается понижением предыдущего результата по модулю 1 + х4.

Это дает

b2x3 + b1x2 + b0x + b3

Умножение на х эквивалентно умножению на матрицу, как описано выше со всеми ai = '00' за исключением а1 = '01'. Имеем:



Следовательно, умножение на х соответствует циклическому сдвигу байтов внутри вектора.

В большинстве алгоритмов шифрования преобразование каждого раунда имеет структуру сети Фейштеля . В этом случае обычно часть битов в каждом промежуточном состоянии просто перемещается без изменения в другую половину. Преобразование раунда алгоритма Rijndael не имеет структуру сети Фейштеля. Вместо этого преобразование каждого раунда состоит из четырех различных преобразований, называемых слоями.

Каждый слой разрабатывался с учетом противодействия линейному и дифференциальному криптоанализу. В основу каждого слоя положена своя собственная функция:

1. Нелинейный слой состоит из параллельного применения S-boxes для оптимизации нелинейных свойств в наихудшем случае.
2. Слой линейного перемешивания строк гарантирует высокую степень диффузии для нескольких раундов.
3. Слой линейного перемешивания столбцов также гарантирует высокую степень диффузии для нескольких раундов.
4. Дополнительный слой ключа состоит из простого XOR промежуточного состояния с ключом раунда.

Перед первым раундом применяется дополнительное забеливание с использованием ключа. Причина этого состоит в следующем. Любой слой после последнего или до первого добавления ключа может быть просто снят без знания ключа и тем самым не добавляет безопасности в алгоритм (например, начальная и конечная перестановки в DES). Начальное или конечное добавление ключа применяется также в некоторых других алгоритмах, например IDEA, SAFER и Blowfish.

**Спецификация алгоритма**

Rijndael является блочным алгоритмом шифрования с переменной длиной блока и переменной длиной ключа. Длина блока и длина ключа могут быть независимо установлены в 128, 192 или 256 бит.

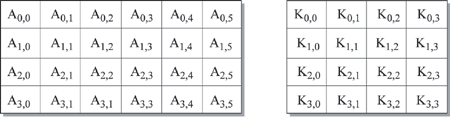
**Состояние, ключ шифрования и число раундов**

Различные преобразования выполняются над промежуточным результатом, называемым состоянием.

Состояние можно рассматривать как двумерный массив байтов. Этот массив имеет четыре строки и различное число столбцов, обозначаемое как Nb, равное длине блока, деленной на 32. Ключ также можно рассматривать как двумерный массив с четырьмя строками. Число столбцов ключа шифрования, обозначаемое как Nk, равно длине ключа, деленной на 32.

В некоторых случаях эти блоки также рассматриваются как одномерные массивы четырехбайтных векторов, где каждый вектор состоит из соответствующего столбца. Такие массивы имеют длину 4, 6 или 8 соответственно, и индексы в диапазонах 0 … 3, 0 … 5 или 0 … 7. Четырехбайтные вектора иногда мы будем называть словами.

Если необходимо указать четыре отдельных байта в четырехбайтном векторе, будет использоваться нотация (a, b, c, d), где a, b, c и d являются байтами в позициях 0, 1, 2 и 3, соответственно, в рассматриваемом столбце, векторе или слове.



**Рисунок 3.14 – Пример состояния (с Nb = 6) и ключа шифрования**

**(с Nk = 4)**

Входы и выходы Rijndael считаются одномерными массивами из 8 байтов, пронумерованными от 0 до 4\* Nb - 1. Следовательно, эти блоки имеют длину 16, 24 или 32 байта, и массив индексируется в диапазонах 0 … 15, 0 … 23 или 0 … 31. Ключ считается одномерным массивом 8-битных байтов, пронумерованных от 0 до 4\* Nk - 1. Следовательно, эти блоки имеют длину 16, 24 или 32 байта, и массив индексируется в диапазонах 0 … 15, 0 … 23 или 0 … 31.

Входные байты алгоритма отображаются в байты состояния в следующем порядке: А0,0, А1,0, А2,0, А3,0, А0,1, А1,1, А2,1, А3,1, … Байты ключа шифрования отображаются в массив в следующем порядке: K0,0, K1,0, K2,0, K3,0, K0,1, K1,1, K2,1, K3,1, … После выполнения операции шифрования выход алгоритма получается из байтов состояния аналогичным образом.

Следовательно, если одноразмерный индекс байта в блоке есть n, и двухмерный индекс есть (i,j), то мы имеем:

I = n mod 4

J = 

N = i + 4\*j

Более того, индекс i является также номером байта в четырехбайтном векторе или слове, j является индексом вектора или слова во вложенном блоке.

Число раундов обозначается Nr и зависит от значений Nb и Nk, что показано в следующей таблице.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.1. Число раундов как функция от длины блока и длины ключа | | | |
| **Nr** | **Nb = 4** | **Nb = 6** | **Nb = 8** |
| **Nk = 4** | 10 | 12 | 14 |
| **Nk = 6** | 12 | 12 | 14 |
| **Nk = 8** | 14 | 14 | 14 |

**Преобразование раунда**

Преобразование раунда состоит из четырех различных преобразований. В нотации на псевдо С это можно записать следующим образом:

Round (State, RoundKey)

{

ByteSub (State);

ShiftRow (State);

MixColumn (State);

AddRoundKey (State, RoundKey);

}

Заключительный раунд алгоритма немного отличается и выглядит следующим образом:

FinalRound (State, RoundKey)

{

ByteSub (State);

ShiftRow (State);

AddRoundKey (State, RoundKey);

}

Как мы видим, заключительный раунд эквивалентен остальным, за исключением того, что отсутствует слой MixColumn.

**Создание ключей раунда**

Ключи раунда получаются из ключа шифрования с помощью преобразования, состоящего из двух компонентов: расширение ключа и выбор ключа раунда. Основной принцип состоит в следующем:

* Общее число битов ключа раунда равно длине блока, умноженной на количество раундов плюс 1. Например, для длины блока 128 бит и 10 раундов необходимо 1408 битов ключа раунда.
* Ключ шифрования расширяется в ExpandedKey.
* Ключи раунда получаются из этого ExpandedKey следующим способом: первый ключ раунда состоит из первых Nb слов, второй состоит из следующих Nb слов и т.д.

**Расширение ключа**

Expanded Key является линейным массивом четырехбайтных слов и обозначается как W [Nb \* (Nr + 1)]. Первые Nk слов состоят из ключа шифрования. Остальные слова определяются рекурсивно. Функция расширения ключа зависит от значения Nk: существует версия функции для Nk, равным или меньшим 6, и версия для Nk больше 6.

Для Nk ≤ 6 мы имеем:

KeyExpansion (byte Key [4\*Nk]

word W[Nb \* (Nr + 1)])

{

for (i = 0; i < Nk; i++)

W[i] =(Key [4\*i], Key [4\*i+1],

Key [4\*i+2], Key [4\*i+3]);

for (i = Nk; i < Nb \* (Nr + 1); i++) {

temp = W [i - 1];

if (i % Nk == 0)

temp = SubByte (RotByte (temp)) ^ Rcon [i / Nk];

W [i] = W [i- Nk] ^ temp;

}

}

В данном случае SubByte (W) является функцией, которая возвращает четырехбайтное слово, в котором каждый байт является результатом применения S-box Rijndael к байту в соответствующей позиции во входном слове. Функция RotByte (W) возвращает слово, в котором байты циклически переставлены таким образом, что для входного слова (a, b, c, d) создается выходное слово (b, c, d, a).

Можно заметить, что первые Nk слов заполняются ключом шифрования. Каждое следующее слово W[i] равно XOR предыдущего слова W[i-1] и позиций слова Nk до W[i - Nk]. Для слов в позициях, которые кратны Nk, сначала применяется преобразование XOR к W[i-1] и константой раунда. Данное преобразование состоит из циклического сдвига байтов в слове RotByte, за которым следут применение табличной подстановки для всех четырех байтов в слове (SubByte).

Для Nk > 6 мы имеем:

KeyExpansion (byte Key [4\*Nk]

word W [Nb\* (Nr+1)])

{

for (i=0; i < Nk; i++)

W[i]= (key [4\*i], key [4\*i+1],

key [4\*i+2], key [4\*i+3]);

for (i = Nk; i < Nb \* (Nr + 1); i++) {

temp = W [i-1];

if (i % Nk == 0)

temp = SubByte (RotByte (temp)) ^

Rcon [i / Nk];

else if (i % Nk == 4)

temp = SubByte (temp);

W[i] = W[i - Nk] ^ temp;

}

}

Отличие в схеме для Nk le6 состоит в том, что для i-4 кратных Nk, SubByte применяется для W[i-1] перед XOR.

Константы раунда не зависят от Nk и определяются следующим образом:

Rcon [i] = (RC [i], '00', '00', '00')

RC [i] являются элементами в GF (28) со значением x(i-1) таким, что:

RC [1] = 1 (т.е. '01')

RC [i] = x (т.е. '02') • (RC [i-1]) = x(i-1)

Ключ раунда i получается из слов буфера ключа раунда W [Nb \* i] до W [Nb \* (i+1)].

**Алгоритм шифрования**

Алгоритм шифрования Rijndael состоит из

* начального сложения с ключом;
* Nr - 1 раундов;
* заключительного раунда.

В С-подобном представлении это выглядит так:

Rijndael (State, CipherKey)

{

KeyExpansion (CipherKey, ExpandedKey);

AddRoundKey (State, ExpandedKey);

for (i=1; i < Nr; i++)

Round (State, ExpandedKey + Nb\*i);

FinalRound (State, ExpandedKey + Nb\*Nr)

}

Расширение ключа может быть выполнено заранее, и Rijndael может быть специфицирован в терминах расширенного ключа.

Rijndael (State, ExpandedKey)

{

AddRoundKey (State, ExpandedKey);

for (i=1; i < Nr; i++)

Round (State, ExpandedKey + Nb\*i);

FinalRound (State, ExpandedKey + Nb\*Nr)

}

Замечание: расширенный ключ всегда получается из ключа шифрования и никогда не специфицируется непосредственно. Тем не менее, на выбор самого ключа шифрования ограничений не существует.

**ТЕМА 4 Криптография с открытым ключом**

4.1 Основные требования к алгоритмам асимметричного шифрования

Алгоритмы шифрования с *открытым ключом* разрабатывались для того, чтобы решить две наиболее трудные задачи, возникшие при использовании симметричного шифрования: распределение ключа и цифровая подпись.

При описании симметричного шифрования и шифрования с *открытым ключом* будем использовать следующую терминологию. Ключ, используемый в симметричном шифровании, будем называть секретным ключом. Два ключа, используемые при шифровании с *открытым ключом*, будем называть *открытым ключом* и *закрытым ключом*. *Закрытый ключ* держится в секрете, но называть его будем *закрытым ключом*, а не секретным, чтобы избежать путаницы с ключом, используемым в симметричном шифровании. *Закрытый ключ* будем обозначать KR, *открытый ключ* - KU.

Будем предполагать, что все участники имеют доступ к *открытым ключам* друг друга, а *закрытые ключи* создаются локально каждым участником и, следовательно, распределяться не должны.

Диффи и Хеллман описывают требования, которым должен удовлетворять алгоритм шифрования с *открытым ключом*.

1. Вычислительно легко создавать пару *KU, KR*.
2. Вычислительно легко, имея *открытый ключ* и незашифрованное сообщение *М*, создать соответствующее зашифрованное сообщение:

С = ЕKU[М]

1. Вычислительно легко дешифровать сообщение, используя *закрытый ключ*:

М = DKR[C] = DKR[EKU[M]]

1. Вычислительно невозможно, зная *открытый ключ* *KU*, определить *закрытый ключ* *KR*.
2. Вычислительно невозможно, зная *открытый ключ* *KU* и зашифрованное сообщение С, восстановить исходное сообщение *М*.

Можно добавить шестое требование, хотя оно не выполняется для всех алгоритмов с *открытым ключом*:

1. Шифрующие и дешифрующие функции могут применяться в любом порядке:

М = ЕKU[DKR[M]]

Это достаточно сильные требования, которые вводят понятие *односторонней функции с люком*. ***Односторонней функцией*** называется такая функция, у которой каждый аргумент имеет единственное обратное значение, при этом вычислить саму функцию легко, а вычислить обратную функцию трудно.

|  |
| --- |
| Y = f(X) - легко |
| X = f-1(Y) - трудно |

Вернемся к определению *односторонней функции с люком*, которую, подобно *односторонней функции*, легко вычислить в одном направлении и трудно вычислить в обратном направлении до тех пор, пока недоступна некоторая дополнительная информация. При наличии этой дополнительной информации инверсию можно вычислить за полиномиальное время. Таким образом, *односторонняя функция* с люком принадлежит семейству *односторонних функций* fk таких, что

|  |
| --- |
| Y = fk(X) - легко, если k и Х известны |
| X = fk-1(Y) - легко, если k и Y известны |
| Х = fk-1(Y) - трудно, если Y известно, но k неизвестно |

.

5.2. Криптоанализ алгоритмов с открытым ключом

Как и в случае симметричного шифрования, алгоритм шифрования с *открытым ключом* уязвим для лобовой атаки. Контрмера стандартная: использовать большие ключи.

Криптосистема с *открытым ключом* применяет определенные неинвертируемые математические функции. Сложность вычислений таких функций не является линейной от количества битов ключа, а возрастает быстрее, чем ключ. Таким образом, размер ключа должен быть достаточно большим, чтобы сделать лобовую атаку непрактичной, и достаточно маленьким для возможности практического шифрования. На практике размер ключа делают таким, чтобы лобовая атака была непрактичной, но в результате скорость шифрования оказывается достаточно медленной для использования алгоритма в общих целях. Поэтому шифрование с *открытым ключом* в настоящее время в основном ограничивается приложениями управления ключом и подписи, в которых требуется шифрование небольшого блока данных.

Другая форма атаки состоит в том, чтобы найти способ вычисления *закрытого ключа*, зная *открытый ключ*. Невозможно математически доказать, что данная форма атаки исключена для конкретного алгоритма *открытого ключа*. Таким образом, любой алгоритм, включая широко используемый *алгоритм RSA*, является подозрительным.

Наконец, существует форма атаки, специфичная для способов использования систем с *открытым ключом*. Это атака вероятного сообщения. Предположим, например, что посылаемое сообщение состоит исключительно из 56-битного ключа сессии для алгоритма симметричного шифрования. Противник может зашифровать все возможные ключи, используя *открытый ключ*, и может дешифровать любое сообщение, соответствующее передаваемому зашифрованному тексту. Таким образом, независимо от размера ключа схемы *открытого ключа*, атака сводится к лобовой атаке на 56-битный симметричный ключ. Защита от подобной атаки состоит в добавлении определенного количества случайных битов в простые сообщения.

5.3. Основные способы использования алгоритмов с открытым ключом

Основными способами использования алгоритмов с *открытым ключом* являются шифрование/дешифрование, создание и проверка подписи и обмен ключа.

Шифрование с *открытым ключом* состоит из следующих шагов:

1. Пользователь В создает пару ключей KUb и KRb, используемых для шифрования и дешифрования передаваемых сообщений.
2. Пользователь В делает доступным некоторым надежным способом свой ключ шифрования, т.е. *открытый ключ* KUb. Составляющий пару *закрытый ключ* KRb держится в секрете.
3. Если А хочет послать сообщение В, он шифрует сообщение, используя *открытый ключ* В KUb .
4. Когда В получает сообщение, он дешифрует его, используя свой *закрытый ключ* KRb. Никто другой не сможет дешифровать сообщение, так как этот *закрытый ключ* знает только В.

Создание и проверка подписи состоит из следующих шагов:

1. Пользователь А создает пару ключей KRA и KUA, используемых для создания и проверки подписи передаваемых сообщений.
2. Пользователь А делает доступным некоторым надежным способом свой ключ проверки, т.е. *открытый ключ* KUA. Составляющий пару *закрытый ключ* KRA держится в секрете.
3. Если А хочет послать подписанное сообщение В, он создает подпись EKRa[M] для этого сообщения, используя свой *закрытый ключ* KRA.
4. Когда В получает подписанное сообщение, он проверяет подпись DKUa[M], используя *открытый ключ* А KUA. Никто другой не может подписать сообщение, так как этот *закрытый ключ* знает только А.

*Обмен ключей***:** две стороны взаимодействуют для обмена ключом сессии, который в дальнейшем можно использовать в алгоритме симметричного шифрования.

Некоторые алгоритмы можно задействовать тремя способами, в то время как другие могут использоваться одним или двумя способами.

5.4. Алгоритм RSA

Алгоритм, разработанный Ривестом, Шамиром и Адлеманом, использует выражения с экспонентами. Данные шифруются блоками, каждый блок рассматривается как число, меньшее некоторого числа n. Шифрование и дешифрование имеют следующий вид для некоторого незашифрованного блока М и зашифрованного блока С.

С = Ме (mod n)

M = Cd (mod n) = (Me)d (mod n) = Med (mod n)

Как отправитель, так и получатель должны знать значение n. Отправитель знает значение е, получатель знает значение d. Таким образом, *открытый ключ* есть KU = {e, n} и *закрытый ключ* есть KR = {d, n}. При этом должны выполняться следующие условия:

1. Возможность найти *е*, *d* и *n* такие, что *Med = M mod n* для всех М < n .
2. Относительная легкость вычисления Ме и Сd для всех значений М < n.
3. Невозможность определить d, зная е и n.

Рассмотрим некоторые математические понятия, свойства и теоремы, которые позволят нам определить e, d и n.

1. Если (а · b) ≡ (a · c) mod n, то b ≡ c mod n, если а и n взаимнопростые, т.е gcd (a, n) = 1.
2. Обозначим Zp - все числа, взаимнопростые с p и меньшие p. Если p - простое, то Zp - это все остатки. Обозначим w-1 такое число, что w · w-1 ≡ 1 mod p.

Тогдаw Zp z: w · z ≡1 mod p

1. Определим функцию Эйлера следующим образом: Φ(n) - число положительных чисел, меньших n и взаимнопростых с n. Если p - простое, то Φ(р) = p-1.

Если p и q - простые, то Φ(p · q) = (p-1) · (q-1).

1. *Теорема Ферма*.

an-1 ≡1 mod n, если n - простое.

1. *Теорема Эйлера*.

aΦ(n) ≡1 mod n для всех взаимнопростых a и n.

Теперь рассмотрим все элементы *алгоритма RSA*.

|  |  |
| --- | --- |
| p, q - два простых целых числа | -открыто, вычисляемо. |
| n = p · q | - закрыто, вычисляемо. |
| d, gcd (Φ(n), d) = 1; | - открыто, выбираемо. |
| 1 < d < Φ(n) |
| е ≡ d-1 mod Φ(n) | - закрыты, выбираемы. |

*Закрытый ключ* состоит из {d, n}, *открытый ключ* состоит из {e, n}. Предположим, что пользователь А опубликовал свой *открытый ключ*, и что пользователь В хочет послать пользователю А сообщение М. Тогда В вычисляет С = Ме (mod n) и передает С. При получении этого зашифрованного текста пользователь А дешифрует вычислением М = С d (mod n).

Суммируем *алгоритм RSA*:

*Создание ключей*

|  |
| --- |
| Выбрать простые р и q |
| Вычислить n = p · q |
| Выбрать d     gcd (Φ(n), d) = 1; 1 < d < Φ(n) |
| Вычислить е     е = d-1 mod Φ(n) |
| *Открытый ключ* KU = {e, n} |
| *Закрытый ключ* KR = {d, n} |

*Шифрование*

|  |
| --- |
| Незашифрованный текст: М < n |
| Зашифрованный текст: С = М е (mod n) |

*Дешифрование*

|  |
| --- |
| Зашифрованный текст: С |
| Незашифрованный текст: М = Сd (mod n) |

***Обсуждение криптоанализа***

Можно определить четыре возможных подхода для криптоанализа *алгоритма RSA*:

1. Лобовая атака: перебрать все возможные *закрытые ключи*.
2. Разложить n на два простых сомножителя. Это даст возможность вычислить Φ(n) = (p-1) · (q-1) и d = e-1 (mod Φ(n)).
3. Определить Φ(n) непосредственно, без начального определения р и q. Это также даст возможность определить d = e-1 (mod Φ(n)).
4. Определить d непосредственно, без начального определения Φ(n).

Защита от лобовой атаки для *RSA* и ему подобных алгоритмов приводится далее.

5.5. Алгоритм обмена ключа Диффи-Хеллмана

Цель алгоритма состоит в том, чтобы два участника могли безопасно обменяться ключом, который в дальнейшем может использоваться в каком-либо алгоритме симметричного шифрования. Сам *алгоритм Диффи-Хеллмана* может применяться только для обмена ключами.

Алгоритм основан на трудности вычислений *дискретных логарифмов*. *Дискретный логарифм* определяется следующим образом. Вводится понятие *примитивного корня простого числа* Q как числа, чьи степени создают все целые от 1 до Q - 1. Это означает, что если А является *примитивным корнем простого числа* Q, тогда числа

A mod Q, A2 mod Q, . . . , AQ - 1 mod Q

являются различными и состоят из целых от 1 до Q - 1 с некоторыми перестановками. В этом случае для любого целого Y < Q и *примитивного корня* A простого числа Q можно найти единственную экспоненту Х, такую, что

Y = AХ mod Q, где 0≤X≤(Q - 1)

Экспонента X называется *дискретным логарифмом*, или индексом Y, по основанию A mod Q. Это обозначается как

indA, Q (Y).

Теперь опишем алгоритм обмена ключей *Диффи-Хеллмана*.

**Общеизвестные элементы.** *Q*: простое число; *A*: *A* < *Q* и *A* является примитивным корнем Q.

**Создание пары ключей пользователем I.** Выбор случайного числа Хi (*закрытый ключ*), Xi < Q. Вычисление числа Yi (открытый ключ) Yi = A Хi mod Q.

**Создание *открытого ключа* пользователем J.** Выбор случайного числа Хj (закрытый ключ) *Xj* < *Q*. Вычисление случайного числа Yj (открытый ключ) Yj = A *Xj* mod *Q*.

**Создание общего секретного ключа пользователем I.** K = (Yj)Xi mod Q.

**Создание общего секретного ключа пользователем J.** K = (Yi)Xj mod Q.

Предполагается, что существуют два известных всем числа: простое число Q и целое A, которое является *примитивным корнем* Q. Теперь предположим, что пользователи I и J хотят обменяться ключом для алгоритма симметричного шифрования. Пользователь I выбирает случайное число Хi < Q и вычисляет Yi = AXi mod Q. Аналогично пользователь J независимо выбирает случайное целое число Хj < Q и вычисляет Yj = AXj mod Q. Каждая сторона держит значение Х в секрете и делает значение Y доступным для другой стороны. Теперь пользователь I вычисляет ключ как К = (Yj)Xi mod Q, и пользователь J вычисляет ключ как K = (Yi)Xj mod Q. В результате оба получат одно и то же значение:

K = (Yj)Xi mod Q = (AXj mod Q)Xi mod Q = (AXj )Xi mod Q = AXj Xi mod Q

= (AXj )Xj mod Q = (AXi mod Q)Xj mod Q = (Yi)Xj mod Q

Таким образом, две стороны обменялись секретным ключом. Так как Хi и Хj являются закрытыми, противник может получить только следующие значения: Q, A, Yi и Yj. Для вычисления ключа атакующий должен взломать *дискретный логарифм*, т.е. вычислить

Xj = inda, q (Yj)

Следует заметить, что данный алгоритм уязвим для атак типа "man-in-the-middle".

**ТЕМА 5. Хэш-функции**

5.1. Требования к хэш-функциям

Хэш-функцией называется односторонняя функция, предназначенная для получения дайджеста или "отпечатков пальцев" файла, сообщения или некоторого блока данных.

Хэш-код создается функцией Н: h = H (M)

Где М является сообщением произвольной длины и h является хэш-кодом фиксированной длины.

Хэш-функция Н, которая используется для аутентификации сообщений, должна обладать следующими свойствами:

1. Хэш-функция Н должна применяться к блоку данных любой длины.
2. Хэш-функция Н создает выход фиксированной длины.
3. Н (М) относительно легко (за полиномиальное время) вычисляется для любого значения М.
4. Для любого данного значения хэш-кода h вычислительно невозможно найти M такое, что Н (M) = h.
5. Для любого х вычислительно невозможно найти y nex, что H (y) = H (x).
6. Вычислительно невозможно найти произвольную пару (х, y) такую, что H (y) = H (x).

5.2. Простые хэш-функции

Все *хэш-функции* выполняются следующим образом. Входное значение (сообщение, файл и т.п.) рассматривается как последовательность n-битных блоков. Входное значение обрабатывается последовательно блок за блоком, и создается m-битное значение *хэш-кода*.

Одним из простейших примеров *хэш-функции* является побитный XOR каждого блока:

Сi = bi1  bi2 ...  bik

Где

|  |
| --- |
| Сi - i-ый бит *хэш-кода*, 1≤i≤n. |
| k - число n-битных блоков входа. |
| bij - i-ый бит в j-ом блоке. |
| - операция XOR. |

В результате получается *хэш-код* длины n, известный как продольный избыточный контроль. Это эффективно при случайных сбоях для проверки целостности данных.

Часто при использовании подобного продольного избыточного контроля для каждого блока выполняется однобитный циклический сдвиг после вычисления *хэш-кода*. Это можно описать следующим образом.

* Установить n-битный *хэш-код* в ноль.
* Для каждого n-битного блока данных выполнить следующие операции:
  + сдвинуть циклически текущий *хэш-код* влево на один бит;
  + выполнить операцию XOR для очередного блока и *хэш-кода*.

Это даст эффект "случайности" входа и уничтожит любую регулярность, которая присутствует во входных значениях.

Хотя второй вариант считается более предпочтительным для обеспечения целостности данных и предохранения от случайных сбоев, он не может использоваться для обнаружения преднамеренных модификаций передаваемых сообщений. Зная сообщение, атакующий легко может создать новое сообщение, которое имеет тот же самый *хэш-код*. Для этого следует подготовить альтернативное сообщение и затем присоединить n-битный блок, который является *хэш-кодом* нового сообщения, и блок, который является *хэш-кодом* старого сообщения.

Хотя простого XOR или ротационного XOR (RXOR) недостаточно, если целостность обеспечивается только зашифрованным *хэш-кодом*, а само сообщение не шифруется, подобная простая функция может использоваться, когда все сообщение и присоединенный к нему *хэш-код* шифруются. Но и в этом случае следует помнить о том, что подобная *хэш-функция* не может проследить за тем, чтобы при передаче последовательность блоков не изменилась. Это происходит в силу того, что данная *хэш-функция* определяется следующим образом: для сообщения, состоящего из последовательности 64-битных блоков Х1, Х2,..., ХN, определяется *хэш-код* С как поблочный XOR всех блоков, который присоединяется в качестве последнего блока:

С = ХN+1 = X1  X2  . . .  XN

Затем все сообщение шифруется, включая *хэш-код*, в режиме СВС для создания зашифрованных блоков Y1, Y2, ..., YN+1. По определению СВС имеем:

Х1 = IV  DK [Y1]

Хi = Yi-1  DK [Yi]

ХN+1 = YN  DK [YN+1]

Но XN+1 является *хэш-кодом*:

ХN+1 = X1  X2  ...  XN = (IV  DK [Y1])  (Y1  DK [Y2])  ... 

(YN-1  DK [YN])

Так как сомножители в предыдущем равенстве могут вычисляться в любом порядке, следовательно, *хэш-код* не будет изменен, если зашифрованные блоки будут переставлены.

5.3 Парадокс дня рождения

Предположим, количество выходных значений *хэш-функции* *Н* равно *n*. Каким должно быть число *k*, чтобы для конкретного значения *X* и значений *Y1*, …, *Yk* вероятность того, что хотя бы для одного *Yi* выполнялось равенство

H (X) = H (Y) была бы больше 0,5.

Для одного *Y* вероятность того, что *H* (*X*) = *H* (*Y*), равна 1/*n*.

Соответственно, вероятность того, что *H*(*X*) ≠ *H*(*Y*), равна 1 - 1/*n*.

Если создать *k* значений, то вероятность того, что ни для одного из них не будет совпадений, равна произведению вероятностей, соответствующих одному значению, т.е. (1 - 1/*n*)k.

Следовательно, вероятность, по крайней мере, одного совпадения равна

1 - (1 - 1/n)k

По формуле бинома Ньютона

(1 - a)k = 1 - ka + (k(k-1)/2!)a2 - ... ≈ 1 - ka

1 - (1 - k/n) = k/n = 0,5

k = n/2

Таким образом, мы выяснили, что для m-битового *хэш-кода* достаточно выбрать 2m-1 сообщений, чтобы вероятность совпадения *хэш-кодов* была больше 0,5.

Теперь рассмотрим следующую задачу: обозначим P (n, k) вероятность того, что в множестве из k элементов, каждый из которых может принимать n значений, есть хотя бы два с одинаковыми значениями. Чему должно быть равно k, чтобы P (n, k) была бы больше 0,5?

Число различных способов выбора элементов таким образом, чтобы при этом не было дублей, равноn(n-1) … (n-k+1)n!/(n-k)!

Всего возможных способов выбора элементов равно nk

Вероятность того, что дублей нет, равна n!/(n-k)!nk

Вероятность того, что есть дубли, соответственно равна 1 - n!/(n-k)!nk

P (n, k) = 1 - n! / ((n-k)! × nk) = 1 - (n × (n-1) × … × (n-k-1)) / nk =

1 - [ (n-1)/n × … × (n-k+1)/n] = 1 - [(1- 1/n) × (1 - 2/n) × … × (1 - (k-1)/n)]

Известно, что 1 - x ≤ e-x

P (n, k) > 1 - [e-1/n × e-2/n × … × e-k/n]

P (n, k) > 1 - e-k(k-1)/n

1/2 = 1 - e-k(k-1)/n

2 = ek(k-1)/n

ln 2 = k (k-1) / 2n

k (k-1) ≈ k2

k = (2n × ln 2)1/2 = 1,17 n1/2 ≈ n1/2

Если *хэш-код* имеет длину m бит, т.е. принимает 2m значений, то

k = √2m = 2m/2

Подобный результат называется "парадоксом дня рождения", потому что в соответствии с приведенными выше рассуждениями для того, чтобы вероятность совпадения дней рождения у двух человек была больше 0,5, в группе должно быть всего 23 человека. Этот результат кажется удивительным, возможно, потому, что для каждого отдельного человека в группе вероятность того, что с его днем рождения совпадет день рождения кого-то другого в группе, достаточно мала.

Вернемся к рассмотрению свойств *хэш-функций*. Предположим, что используется 64-битный *хэш-код*. Можно считать, что это вполне достаточная и, следовательно, безопасная длина для *хэш-кода*. Например, если зашифрованный *хэш-код* С передается с соответствующим незашифрованным сообщением М, то противнику необходимо будет найти М' такое, что

Н (М') = Н (М)

для того, чтобы подменить сообщение и обмануть получателя. В среднем противник должен перебрать 263 сообщений для того, чтобы найти такое, у которого *хэш-код* равен перехваченному сообщению.

Тем не менее, возможны различного рода атаки, основанные на "парадоксе дня рождения". Возможна следующая стратегия:

1. Противник создает 2m/2 вариантов сообщения, каждое из которых имеет некоторый определенный смысл. Противник подготавливает такое же количество сообщений, каждое из которых является поддельным и предназначено для замены настоящего сообщения.
2. Два набора сообщений сравниваются в поисках пары сообщений, имеющих одинаковый *хэш-код*. Вероятность успеха в соответствии с "парадоксом дня рождения" больше, чем 0,5. Если соответствующая пара не найдена, то создаются дополнительные исходные и поддельные сообщения до тех пор, пока не будет найдена пара.
3. Атакующий предлагает отправителю исходный вариант сообщения для подписи. Эта подпись может быть затем присоединена к поддельному варианту для передачи получателю. Так как оба варианта имеют один и тот же *хэш-код*, будет создана одинаковая подпись. Противник будет уверен в успехе, даже не зная ключа шифрования.

5.4. Использование цепочки зашифрованных блоков

Существуют различные *хэш-функции*, основанные на создании цепочки зашифрованных блоков, но без использования секретного ключа. Одна из таких *хэш-функций* была предложена Рабином. Сообщение М разбивается на блоки фиксированной длины М1, М2, . . . , МN и используется алгоритм симметричного шифрования, например DES, для вычисления *хэш-кода* G следующим образом:

Н0 = начальное значение

Нi = EMi [Hi-1]

G = HN

Это аналогично использованию шифрования в режиме СВС, но в данном случае секретного ключа нет. Как и в случае любой *простой хэш-функции*, этот алгоритм подвержен "атаке дня рождения", и если шифрующим алгоритмом является DES и создается только 64-битный *хэш-код*, то система считается достаточно уязвимой.

Могут осуществляться другие атаки типа "дня рождения", которые возможны даже в том случае, если противник имеет доступ только к одному сообщению и соответствующему ему зашифрованному *хэш-коду* и не может получить несколько пар сообщений и зашифрованных *хэш-кодов*. Возможен следующий сценарий: предположим, что противник перехватил сообщение с аутентификатором в виде зашифрованного *хэш-кода*, и известно, что незашифрованный *хэш-код* имеет длину m битов. Далее противник должен выполнить следующие действия:

* Используя описанный выше алгоритм, вычислить незашифрованный *хэш-код* G.
* Создать поддельное сообщение в виде Q1, Q2, . . . , QN-2.

Вычислить Нi = EQi[Hi-1] для 1 ≤ i ≤ N-2.

* Создать 2m/2 случайных блока Х и для каждого такого блока Х вычислить ЕХ[HN-2]. Создать дополнительно 2m/2 cлучайных блока Y и для каждого блока Y вычислить DY[G], где D - дешифрующая функция, соответствующая Е. Основываясь на "парадоксе дня рождения" можно сказать, что с высокой степенью вероятности эта последовательность будет содержать блоки Х и Y такие, что ЕХ[HN-2] = DY[Y].
* Создать сообщение Q1, Q2, . . . ,QN-2, X, Y. Это сообщение имеет *хэш-код* G и, следовательно, может быть использовано вместе с зашифрованным аутентификатором.

Эта форма атаки известна как атака "встреча посередине". В различных исследованиях предлагаются более тонкие методы для усиления подхода, основанного на цепочке блоков. Например, Девис и Прайс описали следующий вариант: Hi = EMi [Hi-1] Hi-1

Возможен другой вариант: Hi = EHi-1 [Mi] Mi

Однако обе эти схемы также имеют уязвимости при различных атаках. В более общем случае, можно показать, что некоторая форма "атаки дня рождения" имеет успех при любом хэш-алгоритме, включающем использование цепочки шифрованных блоков без применения секретного ключа.

Дальнейшие исследования были направлены на поиск других подходов к созданию функций хэширования.

5.5. Хэш-функция MD5

Алгоритм MD5 получает на входе сообщение произвольной длины и создает в качестве выхода *дайджест сообщения* длиной 128 бит. Алгоритм состоит из следующих шагов:



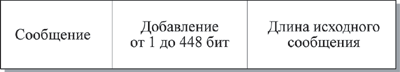
**Рисунок 5.1 – Логика выполнения MD5**

***Шаг 1: добавление недостающих битов***

Сообщение дополняется таким образом, чтобы его длина стала равна 448 по модулю 512 (длина≡448 mod 512).

***Шаг 2: добавление длины***

64-битное представление длины исходного (до добавления) сообщения в битах присоединяется к результату первого шага. Если первоначальная длина больше, чем 264, то используются только последние 64 бита.



**Рисунок 5.2 – Структура расширенного сообщения**

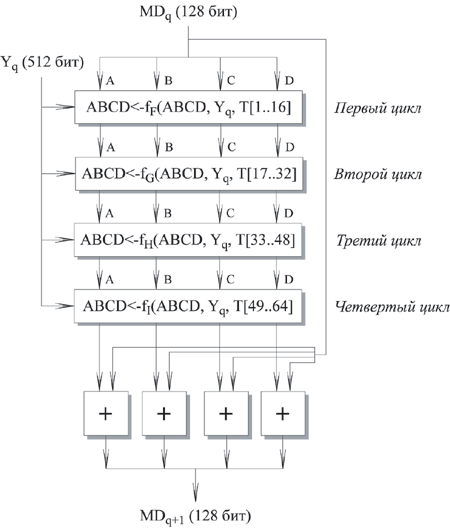
***Шаг 3: инициализация MD-буфера***

Используется 128-битный буфер для хранения промежуточных и окончательных результатов *хэш-функции*. Буфер может быть представлен как четыре 32-битных регистра (A, B, C, D). Эти регистры инициализируются следующими шестнадцатеричными числами:

А = 01234567; В = 89ABCDEF; C = FEDCBA98; D = 76543210

***Шаг 4: обработка последовательности 512-битных (16-словных) блоков***

Основой алгоритма является модуль, состоящий из четырех циклических обработок, обозначенный как HMD5. Четыре цикла имеют похожую структуру, но каждый цикл использует свою элементарную логическую функцию, обозначаемую fF, fG, fH и fI соответственно.



**Рисунок 5.3 – Обработка очередного 512-битного блока**

Каждый цикл принимает в качестве входа текущий 512-битный блок Yq, обрабатывающийся в данный момент, и 128-битное значение буфера ABCD, которое является промежуточным значением *дайджеста*, и изменяет содержимое этого буфера. Каждый цикл также использует четвертую часть 64-элементной таблицы T[1 ... 64], построенной на основе функции sin. i-ый элемент T, обозначаемый T[i], имеет значение, равное целой части от 232 \* abs (sin (i)), i задано в радианах. Так как abs (sin (i)) является числом между 0 и 1, каждый элемент Т является целым, которое может быть представлено 32 битами. Таблица обеспечивает "случайный" набор 32-битных значений, которые должны ликвидировать любую регулярность во входных данных.

Для получения MDq+1 выход четырех циклов складывается по модулю 232 с MDq. Сложение выполняется независимо для каждого из четырех слов в буфере.

***Шаг 5: выход***

После обработки всех *L* 512-битных блоков выходом *L*-ой стадии является 128-битный *дайджест сообщения*.

5.6. Алгоритм MD4

Кратко рассмотрим основные цели MD4:

1. Безопасность.
2. Скорость.
3. Простота и компактность.
4. Желательна little-endian архитектура.

Эти цели преследовались и при разработке *MD5*. *MD5* является более сложным и, следовательно, более медленным при выполнении, чем MD4. Считается, что добавление сложности оправдывается возрастанием уровня безопасности. Главные различия между этими двумя алгоритмами состоят в следующем:

1. MD4 использует три цикла из 16 шагов каждый, в то время как *MD5* использует четыре цикла из 16 шагов каждый.
2. В MD4 дополнительная константа в первом цикле не применяется. Аналогичная дополнительная константа используется для каждого из шагов во втором цикле. Другая дополнительная константа используется для каждого из шагов в третьем цикле. В *MD5* различные дополнительные константы, Т [i], применяются для каждого из 64 шагов.
3. *MD5* использует четыре элементарные логические функции, по одной на каждом цикле, по сравнению с тремя в MD4, по одной на каждом цикле.
4. В *MD5* на каждом шаге текущий результат складывается с результатом предыдущего шага. Например, результатом первого шага является измененное слово А. Результат второго шага хранится в D и образуется добавлением А к циклически сдвинутому влево на определенное число бит результату элементарной функции. Аналогично, результат третьего шага хранится в С и образуется добавлением D к циклически сдвинутому влево результату элементарной функции. MD4 это последнее сложение не включает.

5.7. Хэш-функция SHA-1

Алгоритм получает на входе сообщение максимальной длины 264 бит и создает в качестве выхода дайджест сообщения длиной 160 бит.

Алгоритм состоит из следующих шагов:



**Рисунок 5.4 – Логика выполнения SHA-1**

***Шаг 1: добавление недостающих битов***

Сообщение добавляется таким образом, чтобы его длина была кратна 448 по модулю 512 (длина≡448 mod 512).

***Шаг 2: добавление длины***

К сообщению добавляется блок из 64 битов. Этот блок трактуется как беззнаковое 64-битное целое и содержит длину исходного сообщения до добавления.

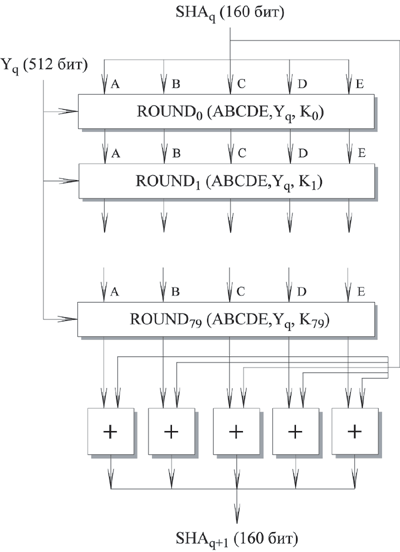
***Шаг 3: инициализация SHA-1 буфера***

Используется 160-битный буфер для хранения промежуточных и окончательных результатов хэш-функции. Буфер может быть представлен как пять 32-битных регистров A, B, C, D и E. Эти регистры инициализируются следующими шестнадцатеричными числами:

A = 67452301; B = EFCDAB89; C = 98BADCFE; D = 10325476; E = C3D2E1F0

***Шаг 4: обработка сообщения в 512-битных (16-словных) блоках***

Основой алгоритма является модуль, состоящий из 80 циклических обработок, обозначенный как HSHA. Все 80 циклических обработок имеют одинаковую структуру.



**Рисунок 5.5 – Обработка очередного 512-битного блока**

Каждый цикл получает на входе текущий 512-битный обрабатываемый блок Yq и 160-битное значение буфера ABCDE, и изменяет содержимое этого буфера.

В каждом цикле используется дополнительная константа Кt, которая принимает только четыре различных значения:

0 ≤ t ≤ 19 Kt = 5A827999 (целая часть числа [230 × 21/2])

20 ≤ t ≤ 39 Kt = 6ED9EBA1 (целая часть числа [230 × 31/2])

40 ≤ t ≤ 59 Kt = 8F1BBCDC (целая часть числа [230 × 51/2])

60 ≤ t ≤ 79 Kt = CA62C1D6 (целая часть числа [230 × 101/2])

Для получения SHAq+1 выход 80-го цикла складывается со значением SHAq. Сложение по модулю 232 выполняется независимо для каждого из пяти слов в буфере с каждым из соответствующих слов в SHAq.

***Шаг 5: выход***

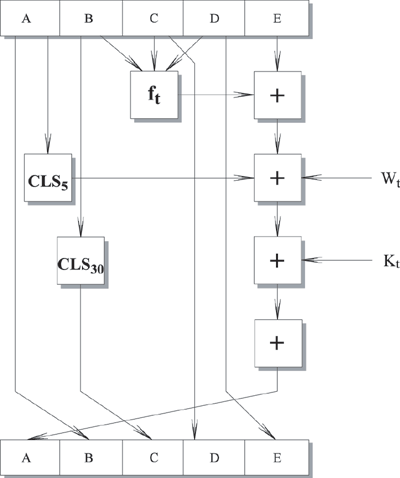
После обработки всех 512-битных блоков выходом L-ой стадии является 160-битный дайджест сообщения.

Рассмотрим более детально логику в каждом из 80 циклов обработки одного 512-битного блока. Каждый цикл можно представить в виде:

A, B, C, D, E (CLS5 (A) + ft (B, C, D) + E + Wt + Kt), A, CLS30 (B), C, D

Где

|  |
| --- |
| A, B, C, D, E - пять слов из буфера. |
| t - номер цикла, 0≤t≤79. |
| ft – элементарная логическая функция. |
| CLSs - циклический левый сдвиг 32-битного аргумента на s битов. |
| Wt - 32-битное слово, полученное из текущего входного 512-битного блока. |
| Kt - дополнительная константа. |
| + - сложение по модулю 232. |

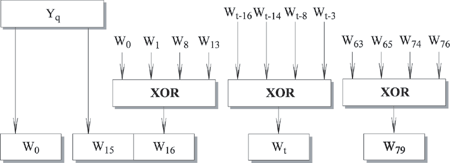


**Рисунок 5.6 – Логика выполнения отдельного цикла**

Каждая элементарная функция получает на входе три 32-битных слова и создает на выходе одно 32-битное слово. Элементарная функция выполняет набор побитных логических операций, т.е. n-ый бит выхода является функцией от n-ых битов трех входов. Функции следующие:

|  |  |
| --- | --- |
| **Номер цикла** | **ft (B, C, D)** |
| (0≤t≤19) | (B C)  (¬ B D) |
| (20≤t≤39) | B C D |
| (40≤t≤59) | (B C)  (B D)  (C D) |
| (60≤t≤79) | B C D |

32-битные слова Wt получаются из очередного 512-битного блока сообщения следующим образом.



**Рисунок 5.7 – Логика выполнения отдельного цикла**

Получение входных значений каждого цикла из очередного блока

Первые 16 значений Wt берутся непосредственно из 16 слов текущего блока. Оставшиеся значения определяются следующим образом:

Wt = Wt-16  Wt-14  Wt-8  Wt-3

В первых 16 циклах вход состоит из 32-битного слова данного блока. Для оставшихся 64 циклов вход состоит из XOR нескольких слов из блока сообщения.

Алгоритм *SHA-1* можно суммировать следующим образом:

SHA0 = IV

SHAq+1 = Σ32 (SHAq , ABCDEq )

SHA = SHAL-1

Где

|  |
| --- |
| IV - начальное значение буфера ABCDE. |
| ABCDEq - результат обработки q-того блока сообщения. |
| L - число блоков в сообщении, включая поля добавления и длины. |
| Σ32 - сумма по модулю 232, выполняемая отдельно для каждого слова буфера. |
| SHA - значение дайджеста сообщения. |

5.8. Хэш-функция SHA-2

В 2001 году NIST принял в качестве стандарта три хэш-функции с существенно большей длиной хэш-кода. Часто эти хэш-функции называют *SHA-2* или SHA-256, SHA-384 и SHA-512 (соответственно, в названии указывается длина создаваемого ими хэш-кода). Эти алгоритмы отличаются не только длиной создаваемого хэш-кода, но и длиной обрабатываемого блока, длиной слова и используемыми внутренними функциями. Сравним характеристики этих хэш-функций.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм | Длина сообщения  (в битах) | Длина блока  (в битах) | Длина слова  (в битах) | Длина дайджеста сообщения  (в битах) | Безопасность  (в битах) |
| SHA-1 | <264 | 512 | 32 | 160 | 80 |
| SHA-256 | <264 | 512 | 32 | 256 | 128 |
| SHA-384 | <2128 | 1024 | 64 | 384 | 192 |
| SHA-512 | <2128 | 1024 | 64 | 512 | 256 |

Под безопасностью здесь понимается стойкость к атакам типа "парадокса дня рождения".

В данных алгоритмах размер блока сообщения равен m бит. Для SHA-256 m = 512, для SHA-384 и SHA-512 m = 1024. Каждый алгоритм оперирует с w-битными словами. Для SHA-256 w = 32, для SHA-384 и SHA-512 w = 64. В алгоритмах используются обычные булевские операции над словами, а также сложение по модулю 2w, правый сдвиг на n бит SHRn (x) , где х - w-битное слово, и циклические (ротационные) правый и левый сдвиги на n бит ROTRn (x) и ROTLn (x), где х - w-битное слово.

SHA-256 использует шесть логических функций, при этом каждая из них выполняется с 32-битными словами, обозначенными как x, y и z. Результатом каждой функции тоже является 32-битное слово.

Ch (x, y, z) = (x  y)  (¬x  z)

Maj (x, y, z) = (x  y)  (x  z)  (y  z)

Σ0{256} (x) = ROTR2 (x)  ROTR13 (x)  ROTR22 (x)

Σ1{256} (x) = ROTR6 (x)  ROTR11 (x)  ROTR25 (x)

σ0{256} (x) = ROTR7 (x)  ROTR18 (x)  SHR3 (x)

σ1{256} (x) = ROTR17 (x)  ROTR19 (x)  SHR10 (x)

SHA-384 и SHA-512 также используют шесть логических функций, каждая из которых выполняется над 64-битными словами, обозначенными как x, y и z. Результатом каждой функции является 64-битное слово.

Ch (x, y, z) = (x  y)  (¬x  z)

Maj (x, y, z) = (x  y)  (x  z)  (y  z)

Σ0{512} (x) = ROTR28 (x)  ROTR34 (x)  ROTR39 (x)

Σ1{512} (x) = ROTR14 (x)  ROTR18 (x)  ROTR41 (x)

σ0{512} (x) = ROTR1 (x)  ROTR8 (x)  SHR7 (x)

σ1{512} (x) = ROTR19 (x)  ROTR61 (x)  SHR6 (x)

Предварительная подготовка сообщения, т.е. добавление определенных битов до целого числа блоков и последующее разбиение на блоки выполняется аналогично тому, как это делалось в *SHA-1* (конечно, с учетом длины блока каждой хэш-функции). После этого каждое сообщение можно представить в виде последовательности N блоков M(1), M(2), … , M(N).

Рассмотрим SHA-256. В этом случае инициализируются восемь 32-битных переменных, которые послужат промежуточным значением хэш-кода:

a, b, c, d, e, f, g, h

Основой алгоритма является модуль, состоящий из 64 циклических обработок каждого блока M(i):

T1 = h + Σ1{256}(e) + Ch(e, f, g) + Kt{256} + Wt

T2 = Σ0{256}(a) + Maj(a, b, c)

h = g

g = f

f = e

e = d + T1

d = c

c = b

b = a

a = T1 + T2

где Ki{256} - шестьдесят четыре 32-битных константы, каждая из которых является первыми 32-мя битами дробной части кубических корней первых 64 простых чисел.

Wt вычисляются из очередного блока сообщения по следующим правилам:

Wt = Mt(i) , 0 ≤ t ≤ 15

Wt = σ1{256}(Wt-2) + Wt-7 + σ0{256}(Wt-15) + Wt-16 , 16 ≤ t ≤ 63

i-ое промежуточное значение хэш-кода H(t) вычисляется следующим образом:

H0(i) = a + H0(i-1)

H1(i) = b + H1(i-1)

H2(i) = c + H2(i-1)

H3(i) = d + H3(i-1)

H4(i) = e + H4(i-1)

H5(i) = f + H5(i-1)

H6(i) = g + H6(i-1)

H7(i) = h + H7(i-1)

Теперь рассмотрим SHA-512. В данном случае инициализируются восемь 64-битных переменных, которые будут являться промежуточным значением хэш-кода:

a, b, c, d, e, f, g, h

Основой алгоритма является модуль, состоящий из 80 циклических обработок каждого блока M(i):

T1 = h + Σ1{512}(e) + Ch(e, f, g) + Kt{512} + Wt

T2 = Σ0{512}(a) + Maj(a, b, c)

h = g

g = f

f = e

e = d + T1

d = c

c = b

b = a

a = T1 + T2

где Ki{512} - восемьдесят 64-битных констант, каждая из которых является первыми 64-мя битами дробной части кубических корней первых восьмидесяти простых чисел.

Wt вычисляются из очередного блока сообщения по следующим правилам:

Wt = Mt(i) , 0 ≤ t ≤ 15

Wt = σ1{512}(Wt-2) + Wt-7 + σ0{512}(Wt-15) + Wt-16 , 16 ≤ t ≤79

i-ое промежуточное значение хэш-кода H(t) вычисляется следующим образом:

H0(i) = a + H0(i-1)

H1(i) = b + H1(i-1)

H2(i) = c + H2(i-1)

H3(i) = d + H3(i-1)

H4(i) = e + H4(i-1)

H5(i) = f + H5(i-1)

H6(i) = g + H6(i-1)

H7(i) = h + H7(i-1)

Рассмотрим SHA-384. Отличия этого алгоритма от SHA-512:

Другой начальный хэш-код H(0).

384-битный дайджест получается из левых 384 битов окончательного хэш-кода H(N): H0(N) || H1(N) || H2(N) || H3(N) || H4(N) || H5(N).

6.9. Хэш-функция ГОСТ 3411

Алгоритм *ГОСТ 3411* является отечественным стандартом для хэш-функций. Длина хэш-кода, создаваемого алгоритмом *ГОСТ 3411*, равна 256 битам. Алгоритм разбивает сообщение на блоки, длина которых также равна 256 битам. Кроме того, параметром алгоритма является стартовый вектор хэширования *Н* - произвольное фиксированное значение длиной также 256 бит.

***Алгоритм обработки одного блока сообщения***

Сообщение обрабатывается блоками по 256 бит справа налево.

Каждый блок сообщения обрабатывается по следующему алгоритму.

1. Генерация четырех ключей длиной 256 бит каждый.
2. Шифрование 64-битных значений промежуточного хэш-кода H на ключах Ki(i = 1, 2, 3, 4) с использованием алгоритма ГОСТ 28147 в режиме простой замены.
3. Перемешивание результата шифрования.

Для генерации ключей используются следующие данные:

* промежуточное значение хэш-кода Н длиной 256 бит;
* текущий обрабатываемый блок сообщения М длиной 256 бит;
* параметры - три значения С2, С3 и С4 длиной 256 бит следующего вида: С2 и С4 состоят из одних нулей, а С3 равно

18 08 116 024 116 08 (08 18)2 18 08 (08 18)4 (18 08)4

где степень обозначает количество повторений 0 или 1.

Используются две формулы, определяющие перестановку и сдвиг.

Перестановка Р битов определяется следующим образом: каждое 256-битное значение рассматривается как последовательность тридцати двух 8-битных значений.

Перестановка Р элементов 256-битной последовательности выполняется по формуле y = φ(x), где x - порядковый номер 8-битного значения в исходной последовательности; y - порядковый номер 8-битного значения в результирующей последовательности.

φ(i + 1 + 4(k - 1)) = 8i + k ; i = 0 ÷ 3, k = 1 ÷ 8

Сдвиг А определяется по формуле

A (x) = (x1  x2) || x4 || x3 || x2

Где

|  |
| --- |
| xi - соответствующие 64 бита 256-битного значения х, |
| || обозначает конкатенацию. |

Присваиваются следующие начальные значения:

i = 1, U = H, V = M.

W = U V, K1 = Р (W)

Ключи K2, K3, K4 вычисляются последовательно по следующему алгоритму:

U = A(U)  Сi, V = A(A(V)), W = U  V, Ki = Р(W)

Далее выполняется шифрование 64-битных элементов текущего значения хэш-кода Н с ключами K1, K2, K3 и K4. При этом хэш-код Н рассматривается как последовательность 64-битных значений:

H = h4 || h3 || h2 || h1

Выполняется шифрование алгоритмом ГОСТ 28147:

si = EKi [hi]    i = 1, 2, 3, 4

S = s1 || s2 || s3 || s4

Наконец на заключительном этапе обработки очередного блока выполняется перемешивание полученной последовательности. 256-битное значение рассматривается как последовательность шестнадцати 16-битных значений. Сдвиг обозначается Ψ и определяется следующим образом:

|  |
| --- |
| η16 || η15 || ... || η1 - исходное значение |
| η1η2η3η4η13η16 || η16 || ... || η2 - результирующее значение |

Результирующее значение хэш-кода определяется следующим образом:

Χ(M, H) = ψ61 (H   ψ (M  ψ 12(S)))

где

|  |
| --- |
| H - предыдущее значение хэш-кода, |
| М - текущий обрабатываемый блок, |
| Ψi - i-ая степень преобразования Ψ. |

Входными параметрами алгоритма являются:

* исходное сообщение М произвольной длины;
* стартовый вектор хэширования Н, длина которого равна 256 битам;
* контрольная сумма Σ, начальное значение которой равно нулю и длина равна 256 битам;
* переменная L, начальное значение которой равно длине сообщения.

Сообщение М делится на блоки длиной 256 бит и обрабатывается справа налево. Очередной блок i обрабатывается следующим образом:

1. H = Χ(Mi, H)
2. Σ = Σ  ' Mi
3. L рассматривается как неотрицательное целое число, к этому числу прибавляется 256 и вычисляется остаток от деления получившегося числа на 2256. Результат присваивается L.

Где ' обозначает следующую операцию: Σ и Mi рассматриваются как неотрицательные целые числа длиной 256 бит. Выполняется обычное сложение этих чисел и находится остаток от деления результата сложения на 2256. Этот остаток и является результатом операции.

Самый левый, т.е. самый последний блок М' обрабатывается так:

1. Блок добавляется слева нулями так, чтобы его длина стала равна 256 битам.
2. Вычисляется Σ = Σ  ' Mi.
3. L рассматривается как неотрицательное целое число, к этому числу прибавляется длина исходного сообщения М и находится остаток от деления результата сложения на 2256.
4. Вычисляется Н = Χ(М', Н).
5. Вычисляется Н = Χ(L, Н).
6. Вычисляется Н = Χ(Σ, Н).

Значением функции хэширования является Н.**ТЕМА 6.** **Коды аутентификации сообщений - МАС**

6.1. Требования к MAC

Один из способов обеспечения целостности - это вычисление *МАС* (Message Authentication Code). Под *МАС* понимается некоторый аутентификатор, являющийся определенным способом вычисленным блоком данных, с помощью которого можно проверить целостность сообщения. В некоторой степени симметричное шифрование всего сообщения может выполнять функцию аутентификации этого сообщения. Но в таком случае сообщение должно содержать достаточную избыточность, которая позволяла бы проверить, что сообщение не было изменено. Избыточность может быть в виде определенным образом отформатированного сообщения, текста на конкретном языке и т.п. Если сообщение допускает произвольную последовательность битов (например, зашифрован ключ сессии), то симметричное шифрование всего сообщения не может обеспечивать его целостность, так как при дешифровании в любом случае получится последовательность битов, правильность которой проверить нельзя. Поэтому гораздо чаще используется критографически созданный небольшой блок данных фиксированного размера, так называемый аутентификатор или имитовставка, с помощью которого проверяется целостность сообщения. Этот блок данных может создаваться с помощью секретного ключа, который разделяют отправитель и получатель. *МАС* вычисляется в тот момент, когда известно, что сообщение корректно. После этого *МАС* присоединяется к сообщению и передается вместе с ним получателю. Получатель вычисляет *МАС*, используя тот же самый секретный ключ, и сравнивает вычисленное значение с полученным. Если эти значения совпадают, то с большой долей вероятности можно считать, что при пересылке изменения сообщения не произошло.

MAC = CK (M)

Предположим, что конфиденциальности сообщения нет, т.е. оппонент имеет доступ к открытому сообщению и соответствующему ему значению *МАС*. Определим усилия, необходимые оппоненту для нахождения ключа *МАС*. Предположим, что k > n, т.е. длина ключа больше длины *МАС*. Тогда, зная М1 и МАС1 = СK (M1), оппонент может вычислить МАС1 = СKi (M1) для всех возможных ключей Ki. При этом, по крайней мере, для одного из ключей будет получено совпадение MACi = MAC1. Оппонент вычислит 2k значений *МАС*, тогда как при длине *МАС* n битов существует всего 2n значений *МАС*. Мы предположили, что k > n, т.е. 2k > 2n. Таким образом, правильное значение *МАС* будет получено для нескольких значений ключей. В среднем совпадение будет иметь место для 2k / 2n = 2(k-n) ключей. Поэтому для вычисления единственного ключа оппоненту требуется знать несколько пар сообщение и соответствующий ему *МАС*.

Таким образом, простой перебор всех ключей требует не меньше, а больше усилий, чем поиск ключа симметричного шифрования той же длины.

Функция вычисления *МАС* должна обладать следующими свойствами:

1. Должно быть вычислительно трудно, зная М и СK (M), найти сообщение М′, такое, что СK(M) = СK(M′).
2. Значения СK(M) должны быть равномерно распределенными в том смысле, что для любых сообщений М и M′ вероятность того, что СK(M) = СK(M′), должна быть равна 2-n, где n - длина значения МАС.

6.2. МАС на основе алгоритма симметричного шифрования

Для вычисления *МАС* может использоваться алгоритм симметричного шифрования (например, DES) в режиме СВС и нулевой инициализационный вектор. В этом случае сообщение представляется в виде последовательности блоков, длина которых равна длине блока алгоритма шифрования. При необходимости последний блок дополняется справа нулями, чтобы получился блок нужной длины. Вычисление *МАС* происходит по следующей схеме:

МАС1 = EK [P1]

МАС2 = EK [P2  MAC1]

. . .

МАСN = EK [PN  MACN-1]

MAC = МАСN

6.3. МАС на основе хэш-функции

Другим способом обеспечения целостности является использование хэш-функции. Хэш-код присоединяется к сообщению в тот момент, когда известно, что сообщение корректно. Получатель проверяет целостность сообщения вычислением хэш-кода полученного сообщения и сравнением его с полученным хэш-кодом, который должен быть передан безопасным способом. Одним из таких безопасных способов может быть шифрование хэш-кода закрытым ключом отправителя, т.е. создание подписи. Возможно также шифрование полученного хэш-кода алгоритмом симметричного шифрования, если отправитель и получатель имеют общий ключ симметричного шифрования.

Еще один вариант использования хэш-функции для получения *МАС* состоит в том, чтобы определенным образом добавить секретное значение к сообщению, которое подается на вход хэш-функции. Такой алгоритм носит название *НМАС*, и он описан в RFC 2104.

При разработке алгоритма НМАС преследовались следующие цели:

* возможность использовать без модификаций уже имеющиеся хэш-функции;
* возможность легкой замены встроенных хэш-функций на более быстрые или более стойкие;
* сохранение скорости работы алгоритма, близкой к скорости работы соответствующей хэш-функции;
* возможность применения ключей и простота работы с ними.

В алгоритме *НМАС* хэш-функция представляет собой "черный ящик". Это, во-первых, позволяет использовать существующие реализации хэш-функций, а во-вторых, обеспечивает легкую замену существующей хэш-функции на новую.

Введем следующие обозначения:

|  |
| --- |
| Н - встроенная хэш-функция. |
| b - длина блока используемой хэш-функции. |
| n - длина хэш-кода. |
| K - секретный ключ. К этому ключу слева добавляют нули, чтобы получить b-битовый ключ K+. |

Вводится два вспомогательных значения:

|  |
| --- |
| Ipad - значение '00110110', повторенное b/8 раз. |
| Opad - значение '01011010', повторенное b/8 раз. |

Далее *НМАС* вычисляется следующим образом:

НМАС = Н ((K+  Opad) || H ((K+  Ipad) || M))

ТЕМА 7. Цифровая подпись

7.1. Требования к цифровой подписи

Аутентификация защищает двух участников, которые обмениваются сообщениями, от воздействия некоторой третьей стороны. Однако простая аутентификация не защищает участников друг от друга, тогда как и между ними тоже могут возникать определенные формы споров.

Например, предположим, что Джон посылает Мери аутентифицированное сообщение, и аутентификация осуществляется на основе общего секрета. Рассмотрим возможные недоразумения, которые могут при этом возникнуть:

* Мери может подделать сообщение и утверждать, что оно пришло от Джона.
* Джон может отрицать, что он посылал сообщение Мери.

В ситуации, когда обе стороны не доверяют друг другу, необходимо нечто большее, чем аутентификация на основе общего секрета. Возможным решением подобной проблемы является использование *цифровой подписи*. *Цифровая подпись* должна обладать следующими свойствами:

1. Должна быть возможность проверить автора, дату и время создания подписи.
2. Должна быть возможность аутентифицировать содержимое во время создания подписи.
3. Подпись должна быть проверяема третьей стороной для разрешения споров.

Таким образом, функция *цифровой подписи* включает функцию аутентификации.

На основании этих свойств можно сформулировать следующие требования к *цифровой подписи*:

1. Подпись должна быть битовым образцом, который зависит от подписываемого сообщения.
2. Подпись должна использовать некоторую уникальную информацию отправителя для предотвращения подделки или отказа.
3. Создавать *цифровую подпись* должно быть относительно легко.
4. Должно быть вычислительно невозможно подделать *цифровую подпись* как созданием нового сообщения для существующей *цифровой подписи*, так и созданием ложной *цифровой подписи* для некоторого сообщения.
5. *Цифровая подпись* должна быть достаточно компактной и не занимать много памяти.

Сильная хэш-функция, зашифрованная закрытым ключом отправителя, удовлетворяет перечисленным требованиям.

Существует несколько подходов к использованию функции *цифровой подписи*. Все они могут быть разделены на две категории: *прямые* и *арбитражные*.

**7.2. Прямая и арбитражная цифровые подписи**

При использовании *прямой цифровой подписи* взаимодействуют только сами участники, т.е. отправитель и получатель. Предполагается, что получатель знает открытый ключ отправителя. *Цифровая подпись* может быть создана шифрованием всего сообщения или его хэш-кода закрытым ключом отправителя.

Конфиденциальность может быть обеспечена дальнейшим шифрованием всего сообщения вместе с подписью открытым ключом получателя (асимметричное шифрование) или разделяемым секретным ключом (симметричное шифрование). Заметим, что обычно функция подписи выполняется первой, и только после этого выполняется функция конфиденциальности. В случае возникновения спора некая третья сторона должна просмотреть сообщение и его подпись. Если функция подписи выполняется над зашифрованным сообщением, то для разрешения споров придется хранить сообщение как в незашифрованном виде (для практического использования), так и в зашифрованном (для проверки подписи). Либо в этом случае необходимо хранить ключ симметричного шифрования, для того чтобы можно было проверить подпись исходного сообщения. Если *цифровая подпись* выполняется над незашифрованным сообщением, получатель может хранить только сообщение в незашифрованном виде и соответствующую подпись к нему.

Все прямые схемы, рассматриваемые далее, имеют общее слабое место. Действенность схемы зависит от безопасности закрытого ключа отправителя. Если отправитель впоследствии не захочет признать факт отправки сообщения, он может утверждать, что закрытый ключ был потерян или украден, и в результате кто-то подделал его подпись. Можно применить административное управление, обеспечивающее безопасность закрытых ключей, для того чтобы, по крайней мере, хоть в какой-то степени ослабить эти угрозы. Один из возможных способов состоит в требовании в каждую подпись сообщения включать отметку времени (дату и время) и сообщать о скомпрометированных ключах в специальный центр.

Другая угроза состоит в том, что закрытый ключ может быть действительно украден у Х в момент времени Т. Нарушитель может затем послать сообщение, подписанное подписью Х и помеченное временной меткой, которая меньше или равна Т.

Проблемы, связанные с *прямой цифровой подписью*, могут быть частично решены с помощью арбитра. Существуют различные схемы с применением *арбитражной подписи*. В общем виде *арбитражная подпись* выполняется следующим образом. Каждое подписанное сообщение от отправителя Х к получателю Y первым делом поступает к арбитру А, который проверяет подпись для данного сообщения. После этого сообщение датируется и посылается к Y с указанием того, что оно было проверено арбитром. Присутствие А решает проблему схем *прямой цифровой подписи*, при которых Х может отказаться от сообщения.

Рассмотрим некоторые возможные технологии *арбитражной цифровой подписи*.

**Симметричное шифрование, арбитр видит сообщение:**

Х→ A: M || EKxa [ IDX || H (M)]

Предполагается, что отправитель Х и арбитр А разделяют секретный ключ KХА и что А и Y разделяют секретный ключ KАY. Х создает сообщение М и вычисляет его хэш-значение Н (М). Затем Х передает сообщение и подпись А. Подпись состоит из идентификатора Х и хэш-значения, все зашифровано с использованием ключа KХА. А дешифрует подпись и проверяет хэш-значение.

A → Y: ЕКay [ IDX || M || EKxa [IDX || H (M)], T ]

Затем А передает сообщение к Y, шифруя его KAY. Сообщение включает IDX, первоначальное сообщение от Х, подпись и отметку времени. Y может дешифровать его для получения сообщения и подписи. Отметка времени информирует Y о том, что данное сообщение не устарело и не является повтором. Y может сохранить М и подпись к нему. В случае спора Y, который утверждает, что получил сообщение М от Х, посылает следующее сообщение к А:

ЕКay [ IDX || M || EKxa [IDX || H (M)] ]

Арбитр использует KAY для получения IDХ, М и подписи, а затем, используя KХА, может дешифровать подпись и проверить хэш-код. По этой схеме Y не может прямо проверить подпись Х; подпись используется исключительно для разрешения споров. Y считает сообщение от Х аутентифицированным, потому что оно прошло через А. В данном сценарии обе стороны должны иметь высокую степень доверия к А:

1. Х должен доверять А в том, что тот не будет раскрывать KХА и создавать фальшивые подписи в форме ЕKка [IDX || H (M)].
2. Y должен доверять А в том, что он будет посылать ЕKay [ IDX || M || EKxa [IDX || H (M)] ] только в том случае, если хэш-значение является корректным и подпись была создана Х.
3. Обе стороны должны доверять А в решении спорных вопросов.

**Симметричное шифрование, арбитр не видит сообщение:**

Если арбитр не является такой доверенной стороной, то Х должен добиться того, чтобы никто не мог подделать его подпись, а Y должен добиться того, чтобы Х не мог отвергнуть свою подпись.

Предыдущий сценарий также предполагает, что А имеет возможность читать сообщения от Х к Y и что возможно любое подсматривание. Рассмотрим сценарий, который, как и прежде, использует арбитраж, но при этом еще обеспечивает конфиденциальность. В таком случае также предполагается, что Х и Y разделяют секретный ключ KXY.

X → A: IDX || EKхy [M] || EKxa [IDX || H (EKxyM]) ]

Х передает А свой идентификатор, сообщение, зашифрованное KXY, и подпись. Подпись состоит из идентификатора и хэш-значения зашифрованного сообщения, которые зашифрованы с использованием ключа KХА. А дешифрует подпись и проверяет хэш-значение. В данном случае А работает только с зашифрованной версией сообщения, что предотвращает его чтение.

A → Y: EKay [ IDX || EKxy] || EKxa [ IDX || H ( EKxyM])], T]

А передает Y все, что он получил от Х плюс отметку времени, все шифруя с использованием ключа KAY.

Хотя арбитр и не может прочитать сообщение, он в состоянии предотвратить подделку любого из участников, Х или Y. Остается проблема, как и в первом сценарии, что арбитр может сговориться с отправителем, отрицающим подписанное сообщение, или с получателем, для подделки подписи отправителя.

**Шифрование открытым ключом, арбитр не видит сообщение:**

Все обсуждаемые проблемы могут быть решены с помощью схемы открытого ключа.

X → A: IDX || EKRх [ IDX || EKUy [EKRx [M] ] ]

В этом случае Х осуществляет двойное шифрование сообщения М, сначала своим закрытым ключом KRX, а затем открытым ключом Y   KUY. Получается подписанная секретная версия сообщения. Теперь это подписанное сообщение вместе с идентификатором Х шифруется KRX и вместе с IDX посылается А. Внутреннее, дважды зашифрованное, сообщение недоступно арбитру (и всем, исключая Y). Однако А может дешифровать внешнюю шифрацию, чтобы убедиться, что сообщение пришло от Х (так как только Х имеет KRX). Проверка дает гарантию, что пара закрытый/открытый ключ законна, и тем самым верифицирует сообщение.

A → Y: EKRa [ IDX || EKUy [EKRx [M] ] || T ]

Затем А передает сообщение Y, шифруя его KRA. Сообщение включает IDX, дважды зашифрованное сообщение и отметку времени.

Эта схема имеет ряд преимуществ по сравнению с предыдущими двумя схемами. Во-первых, никакая информация не разделяется участниками до начала соединения, предотвращая договор об обмане. Во-вторых, некорректные данные не могут быть посланы, даже если KRX скомпрометирован, при условии, что не скомпрометирован KRА. В заключение, содержимое сообщения от Х к Y неизвестно ни А, ни кому бы то ни было еще.

7.3. Стандарт цифровой подписи DSS

Для создания *цифровой подписи* используется алгоритм *DSA* (Digital Signature Algorithm). В качестве хэш-алгоритма стандарт предусматривает использование алгоритма SHA-1 (Secure Hash Algorithm).

*DSS* использует алгоритм, который разрабатывался для использования только в качестве *цифровой подписи*.

*DSS* основан на трудности вычисления дискретных логарифмов и базируется на схеме, первоначально представленной ElGamal и Schnorr.

Существует три параметра, которые являются открытыми и могут быть общими для большой группы пользователей.

160-битное простое число q, т.е. 2159 < q < 2160.

Простое число р длиной между 512 и 1024 битами должно быть таким, чтобы q было делителем (р - 1), т.е. 2L-1 < p < 2L, где 512 < L < 1024 и (p-1)/q является целым.

g = h(p-1)/q mod p, где h является целым между 1 и (р-1) и g должно быть больше, чем 1,10.

Зная эти числа, каждый пользователь выбирает закрытый ключ и создает открытый ключ.

**Закрытый ключ отправителя**

Закрытый ключ х должен быть числом между 1 и (q-1) и должен быть выбран случайно или псевдослучайно.

x - случайное или псевдослучайное целое, 0 < x < q.

**Открытый ключ отправителя**

Открытый ключ вычисляется из закрытого ключа как у = gx mod p. Вычислить у по известному х довольно просто. Однако, имея открытый ключ у, вычислительно невозможно определить х, который является дискретным логарифмом у по основанию g.

y = gx mod p

**Случайное число, уникальное для каждой подписи.**

*k* - случайное или псевдослучайное целое, 0 < k < q, уникальное для каждого подписывания.

**Подписывание**

Для создания подписи отправитель вычисляет две величины, r и s, которые являются функцией от компонент открытого ключа (p, q, g), закрытого ключа пользователя (х), хэш-кода сообщения Н (М) и целого k, которое должно быть создано случайно или псевдослучайно и должно быть уникальным при каждом подписывании.

r = (gk mod p) mod q

s = [ k-1 (H (M) + xr) ] mod q

Подпись = (r, s)

**Проверка подписи**

Получатель выполняет проверку подписи с использованием следующих формул. Он создает величину v, которая является функцией от компонент общего открытого ключа, открытого ключа отправителя и хэш-кода полученного сообщения. Если эта величина равна компоненте r в подписи, то подпись считается действительной.

w = s-1 mod q

u1 = [ H (M) w ] mod q

u2 = r w mod q

v = [ (gu1 yu2) mod p ] mod q

подпись корректна, если v = r.

7.4. Отечественный стандарт цифровой подписи ГОСТ 3410

В стандарте *ГОСТ 3410* используется хэш-функция ГОСТ 3411, которая создает хэш-код длиной 256 бит. Это во многом обуславливает требования к выбираемым простым числам p и q:

р должно быть простым числом в диапазоне

2509 < p < 2512 либо 21020 < p < 21024

q должно быть простым числом в диапазоне

2254 < q < 2256

q также должно быть делителем (р-1).

Аналогично выбирается и параметр g. При этом требуется, чтобы gq (mod p) = 1.

В соответствии с теоремой Ферма это эквивалентно условию в *DSS*, что g = h(p-1)/q mod p.

Закрытым ключом является произвольное число х: 0 < x < q

Открытым ключом является число y: y = gx mod p

Для создания подписи выбирается случайное число k: 0 < k < q

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

r = (gk mod p) mod q

s = (k H(M) + xr) mod q

Обратим внимание на отличия *DSS* и *ГОСТ 3410*.

1. Используются разные хэш-функции: в *ГОСТ 3410* применяется отечественный стандарт на хэш-функции ГОСТ 3411, в *DSS* используется SHA-1, которые имеют разную длину хэш-кода. Отсюда и разные требования на длину простого числа q: в *ГОСТ 3410* длина q должна быть от 254 бит до 256 бит, а в *DSS* длина q должна быть от 159 бит до 160 бит.
2. По-разному вычисляется компонента s подписи. В *ГОСТ 3410* компонента s вычисляется по формуле: s = (k H(M) + xr) mod q

В *DSS* компонента s вычисляется по формуле: s = [k-1 (H(M) + xr)] mod q

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

Получатель вычисляет

w = H(M)-1 mod q

u1 = w s mod q

u2 = (q-r) w mod q

v = [(gu1 yu2) mod p] mod q

Подпись корректна, если v = r.

Структура обоих алгоритмов довольно интересна. Заметим, что значение r совсем не зависит от сообщения. Вместо этого r есть функция от k и трех общих компонент открытого ключа. Мультипликативная инверсия k (mod p) (в случае *DSS*) или само значение k (в случае ГОСТ 3410) подается в функцию, которая, кроме того, в качестве входа имеет хэш-код сообщения и закрытый ключ пользователя. Эта функция такова, что получатель может вычислить r, используя входное сообщение, подпись, открытый ключ пользователя и общий открытый ключ.

В силу сложности вычисления дискретных логарифмов нарушитель не может восстановить k из r или х из s.

Другое важное замечание заключается в том, что экспоненциальные вычисления при создании подписи необходимы только для gk mod p. Так как это значение от подписываемого сообщения не зависит, оно может быть вычислено заранее. Пользователь может заранее просчитать некоторое количество значений r и использовать их по мере необходимости для подписи документов. Еще одна задача состоит в определении мультипликативной инверсии k-1 (в случае *DSS*). Эти значения также могут быть вычислены заранее.

ТЕМА 8. Криптография с использованием эллиптических кривых

8.1. Математические понятия

Преимущество подхода на основе *эллиптических кривых* в сравнении с задачей факторизации числа, используемой в RSA, или задачей целочисленного логарифмирования, применяемой в алгоритме Диффи-Хеллмана и в DSS, заключается в том, что в данном случае обеспечивается эквивалентная защита при меньшей длине ключа.

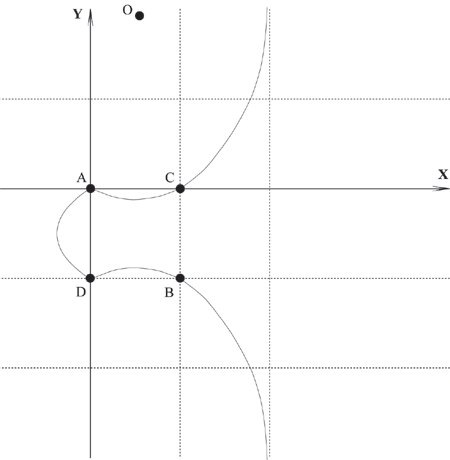
В общем случае уравнение *эллиптической кривой*   Е имеет вид:

y2 + axy + by = x3 + cx2 + dx + e

В качестве примера рассмотрим *эллиптическую кривую*   Е, уравнение которой имеет вид: y2 + y = x3 - x2

На этой кривой лежат только четыре точки, координаты которых являются целыми числами. Это точки

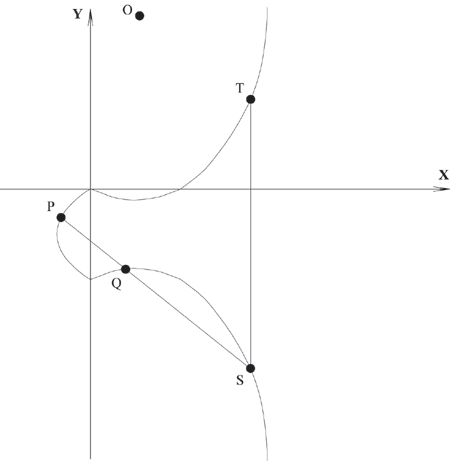
А (0, 0), В (1, -1), С (1, 0) и D (0, -1)



**Рисунок 8.1 – Пример эллиптической кривой с четырьмя точками**

Для определения *операции сложения для точек на эллиптической кривой* сделаем следующие предположения:

* На плоскости существует бесконечно удаленная точка 0Е, в которой сходятся все вертикальные прямые.
* Будем считать, что касательная к кривой пересекает точку касания два раза.
* Если три точки *эллиптической кривой* лежат на прямой линии, то их сумма есть 0.



**Рисунок 8.1 – Сложение точек на эллиптической кривой**

Введем следующие правила сложения точек на *эллиптической кривой*:

* Точка 0 выступает в роли *нулевого элемента*. Так, 0 = -0 и для любой точки Р на *эллиптической кривой* Р + 0 = Р.
* Вертикальная линия пересекает кривую в двух точках с одной и той же координатой х - скажем, S = (x, y) и T = (x, -y). Эта прямая пересекает кривую и в бесконечно удаленной точке. Поэтому Р1 + Р2 + 0 = 0 и Р1 = -Р2.
* Чтобы сложить две точки P и Q (см. рисунок 11.2) с разными координатами х, следует провести через эти точки прямую и найти точку пересечения ее с *эллиптической кривой*. Если прямая не является касательной к кривой в точках P или Q, то существует только одна такая точка, обозначим ее S. Согласно нашему предположению P + Q + S = О

Следовательно, P + Q = -S или P + Q = T.

Если прямая является касательной к кривой в какой-либо из точек P или Q, то в этом случае следует положить S = P или S = Q соответственно.

* Чтобы удвоить точку Q, следует провести касательную в точке Q и найти другую точку пересечения S с *эллиптической кривой*. Тогда Q + Q = 2 × Q = -S.

Введенная таким образом *операция сложения* подчиняется всем обычным правилам сложения, в частности коммутативному и ассоциативному законам. Умножение точки Р *эллиптической кривой* на положительное число k определяется как сумма k точек Р.

В криптографии с использованием *эллиптических кривых* все значения вычисляются по модулю р, где р является простым числом. Элементами данной *эллиптической кривой* являются пары неотрицательных целых чисел, которые меньше р и удовлетворяют частному виду *эллиптической кривой*:

y2 ≡ x3 + ax + b (mod p)

Такую кривую будем обозначать Ep (a,b). При этом числа а и b должны быть меньше р и должны удовлетворять условию 4a3 + 27b2 (mod p) ≠ 0. Множество точек на *эллиптической кривой* вычисляется следующим образом.

Для каждого такого значения х, что 0≤х≤р, вычисляется x3 + ax + b (mod p).

Для каждого из полученных на предыдущем шаге значений выясняется, имеет ли это значение квадратный корень по модулю р. Если нет, то в Ep (a,b) нет точек с этим значением х. Если корень существует, имеется два значения y, соответствующих операции извлечения квадратного корня (исключением является случай, когда единственным значением оказывается y = 0). Эти значения (x,y) и будут точками Ep (a,b).

Множество точек Ep (a,b) обладает следующими свойствами:

1. Р + 0 = Р
2. Если Р = (x,y), то Р + (x,-y) = 0. Точка (x,-y) является отрицательным значением точки Р и обозначается -Р. Заметим, что (x,-y) лежит на *эллиптической кривой* и принадлежит Ep (a,b).
3. Если Р = (x1,y1) и Q = (x2,y2), где P ≠ Q, то P + Q = (x3,y3) определяется по следующим формулам:
4. x3≡ λ2 - x1 - x2 (mod p)
5. y3 ≡ λ (x1 - x3) - y1 (mod p)

где

(y2 - y1)/(x2 - x1) , если P ≠ Q

λ = (3x12 + a)/2y1 , если P = Q

Число λ есть угловой коэффициент секущей, проведенной через точки P = (x1, y1) и Q = (x2, y2). При P = Q секущая превращается в касательную, чем и объясняется наличие двух формул для вычисления λ.

Задача, которую должен решить в этом случае атакующий, есть своего рода задача ***"дискретного логарифмирования на эллиптической кривой"***, и формулируется она следующим образом. Даны точки P и Q на *эллиптической кривой*   Ep (a,b). Необходимо найти коэффициент k < p такой, что

P = k × Q

Относительно легко вычислить P по данным k и Q, но довольно трудно вычислить k, зная P и Q.

**8.2.** **Аналог алгоритма Диффи-Хеллмана обмена ключами**

Обмен ключами с использованием *эллиптических кривых* может быть выполнен следующим образом. Сначала выбирается простое число р ≈ 2180 и параметры a и b для уравнения *эллиптической кривой*. Это задает множество точек Ep (a,b). Затем в Ep (a,b) выбирается генерирующая точка G = (x1,y1). При выборе G важно, чтобы наименьшее значение n, при котором n × G = 0, оказалось очень большим простым числом. Параметры Ep (a,b) и G криптосистемы являются параметрами, известными всем участникам.

Обмен ключами между пользователями А и В производится по следующей схеме.

1. Участник А выбирает целое число nA, меньшее n. Это число является закрытым ключом участника А. Затем участник А вычисляет открытый ключ PA = nA × G, который представляет собой некоторую точку на Ep (a,b).
2. Точно так же участник В выбирает закрытый ключ nB и вычисляет открытый ключ PB.
3. Участники обмениваются открытыми ключами, после чего вычисляют общий секретный ключ K

Участник А: K = nA × PB

Участник В: K = nВ × PА

Следует заметить, что общий секретный ключ представляет собой пару чисел. Если данный ключ предполагается использовать в качестве сеансового ключа для алгоритма симметричного шифрования, то из этой пары необходимо создать одно значение.

8.3. Алгоритм цифровой подписи на основе эллиптических кривых ECDSA

Алгоритм ECDSA (Elliptic Curve Digest Signature Algorithm) принят в качестве стандартов ANSI X9F1 и IEEE P1363.

Создание ключей:

1. Выбирается *эллиптическая кривая*   Ep (a,b). Число точек на ней должно делиться на большое целое n.
2. Выбирается точка РEp (a,b).
3. Выбирается случайное число d  [1, n-1].
4. Вычисляется Q = d × P.
5. Закрытым ключом является d, открытым ключом – (E, P, n, Q).

Создание подписи:

1. Выбирается случайное число k [1, n-1].
2. Вычисляется k × P = (x1, y1) и r = x1 (mod n).

Проверяется, чтобы r не было равно нулю, так как в этом случае подпись не будет зависеть от закрытого ключа. Если r = 0, то выбирается другое случайное число k.

1. Вычисляется k-1 mod n
2. Вычисляется s = k-1 (Н(M) + dr) (mod n)

Проверяется, чтобы s не было равно нулю, так как в этом случае необходимого для проверки подписи числа s-1 mod n не существует. Если s = 0, то выбирается другое случайное число k.

Подписью для сообщения М является пара чисел (r,s).

Проверка подписи:

1. Проверить, что целые числа r и s принадлежат диапазону чисел [0, n-1]. В противном случае результат проверки отрицательный, и подпись отвергается.
2. Вычислить w = s-1 (mod n) и H(M)
3. Вычислить u1 = H(M) w (mod n), u2 = rw (mod n)
4. Вычислить u1P + u2Q = (x0, y0), v = x0 (mod n)
5. Подпись верна в том и только том случае, когда v = r.

8.4. Шифрование/дешифрование с использованием эллиптических кривых

Рассмотрим самый простой подход к шифрованию/дешифрованию с использованием *эллиптических кривых*. Задача состоит в том, чтобы зашифровать сообщение М, которое может быть представлено в виде точки на эллиптической кривой Pm (x,y).

Как и в случае обмена ключом, в системе шифрования/дешифрования в качестве параметров рассматривается *эллиптическая кривая*   Ep (a,b) и точка G на ней. Участник B выбирает закрытый ключ nB и вычисляет открытый ключ PB = nB × G. Чтобы зашифровать сообщение Pm используется открытый ключ получателя B   PB. Участник А выбирает случайное целое положительное число k и вычисляет зашифрованное сообщение Cm, являющееся точкой на *эллиптической кривой*.

Cm = {k × G, Pm + k × PB}

Чтобы дешифровать сообщение, участник В умножает первую координату точки на свой закрытый ключ и вычитает результат из второй координаты:

Pm + k × PB - nB × (k × G) = Pm + k × (nB × G) - nB × (k × G) = Pm

Участник А зашифровал сообщение Pm добавлением к нему kxPB. Никто не знает значения k, поэтому, хотя PB и является открытым ключом, никто не знает k × PB. Противнику для восстановления сообщения придется вычислить k, зная G и k × G. Сделать это будет нелегко.

Получатель также не знает k, но ему в качестве подсказки посылается k × G. Умножив k × G на свой закрытый ключ, получатель получит значение, которое было добавлено отправителем к незашифрованному сообщению. Тем самым получатель, не зная k, но имея свой закрытый ключ, может восстановить незашифрованное сообщение.

**ТЕМА 9. Алгоритмы обмена ключей и протоколы аутентификации**

9.1. Алгоритмы распределения ключей с использованием третьей доверенной стороны

**Понятие мастер-ключа**

При симметричном шифровании два участника, которые хотят обмениваться конфиденциальной информацией, должны иметь один и тот же ключ. Частота изменения ключа должна быть достаточно большой, чтобы у противника не хватило времени для полного перебора ключа. Следовательно, сила любой криптосистемы во многом зависит от технологии распределения ключа. Этот термин означает передачу ключа двум участникам, которые хотят обмениваться данными, таким способом, чтобы никто другой не мог ни подсмотреть, ни изменить этот ключ. Для двух участников *А* и *B* распределение ключа может быть выполнено одним из следующих способов.

1. Ключ может быть создан *А* и физически передан *B*.
2. *Третья сторона* может создать ключ и физически передать его *А* и *B*.
3. *А* и *В* имеют предварительно созданный и недолго используемый ключ, один участник может передать новый ключ другому, применив для шифрования старый ключ.
4. Если А и В каждый имеют безопасное соединение с третьим участником *C*, *C* может передать ключ по этому безопасному каналу *А* и *B*.

Количество требуемых ключей зависит от числа участников, которые должны взаимодействовать. Если выполняется шифрование на сетевом или IP-уровне, то ключ необходим для каждой пары хостов в сети. Таким образом, если есть N хостов, то необходимое число ключей [N (N - 1)]/2. Если шифрование выполняется на прикладном уровне, то ключ нужен для каждой пары прикладных процессов, которых гораздо больше, чем хостов.

Третий способ распределения ключей может применяться на любом уровне стека протоколов, но если атакующий получает возможность доступа к одному ключу, то вся последовательность ключей будет раскрыта. Более того, все равно должно быть проведено первоначальное распространение большого количества ключей.

Поэтому в больших автоматизированных системах широко применяются различные варианты четвертого способа. В этой схеме предполагается существование так называемого центра распределения ключей (Key Destribution Centre - *KDC*), который отвечает за распределение ключей для хостов, процессов и приложений. Каждый участник должен разделять уникальный ключ с *KDC*.

Использование центра распределения ключей основано на использовании иерархии ключей. Как минимум используется два типа ключей: *мастер-ключи* и *ключи сессии*.

9.2. Взаимная аутентификация

Данные протоколы применяются для взаимной *аутентификации* участников и для обмена *ключом сессии*.

Основной задачей таких протоколов является обеспечение конфиденциального распределения *ключа сессии* и гарантирование его своевременности, то есть протокол не должен допускать повторного использования старого *ключа сессии*. Для обеспечения конфиденциальности *ключи сессии* должны передаваться в зашифрованном виде. Вторая задача, обеспечение своевременности, важна, потому что существует угроза перехвата передаваемого сообщения и повторной его пересылки. Такие повторения в худшем случае могут позволять взломщику использовать скомпрометированный *ключ сессии*, при этом успешно подделываясь под другого участника. Успешное повторение может, как минимум, разорвать операцию *аутентификации* участников.

Такие повторы называются *replay-атаками*. Рассмотрим возможные примеры подобных *replay-атак*:

1. Простое повторение: противник просто копирует сообщение и повторяет его позднее.
2. Повторение, которое не может быть определено: противник уничтожает исходное сообщение и посылает скопированное ранее сообщение.

Один из возможных подходов для предотвращения *replay-атак* мог бы состоять в присоединении последовательного номера (sequence number) к каждому сообщению, используемому в *аутентификационном* обмене. Новое сообщение принимается только тогда, когда его последовательный номер правильный. Трудность данного подхода состоит в том, что каждому участнику требуется поддерживать значения sequence number для каждого участника, с которым он взаимодействует в данный момент. Поэтому обычно sequence number не используются для *аутентификации* и обмена ключами. Вместо этого применяется один из следующих способов:

1. *Отметки времени*: участник А принимает сообщение как не устаревшее только в том случае, если оно содержит *отметку времени*, которая, по мнению А, соответствует текущему времени. Этот подход требует, чтобы часы всех участников были синхронизированы.
2. Запрос/ответ: участник А посылает в запросе к В случайное число (***nonce*** - number only once) и проверяет, чтобы ответ от В содержал корректное значение этого *nonce*.

Считается, что подход с *отметкой времени* не следует использовать в приложениях, ориентированных на соединение, потому что это технически трудно, так как таким протоколам, кроме поддержки соединения, необходимо будет поддерживать синхронизацию часов различных процессоров. При этом возможный способ осуществления успешной атаки может возникнуть, если временно будет отсутствовать синхронизация часов одного из участников. В результате различной и непредсказуемой природы сетевых задержек распределенные часы не могут поддерживать точную синхронизацию. Следовательно, процедуры, основанные на любых *отметках времени*, должны допускать окно времени, достаточно большое для приспособления к сетевым задержкам, и достаточно маленькое для минимизации возможности атак.

С другой стороны, подход запрос/ответ не годится для приложений, не устанавливающих соединения, так как он требует предварительного рукопожатия перед началом передач, тем самым отвергая основное свойство транзакции без установления соединения. Для таких приложений доверие к некоторому безопасному серверу часов и постоянные попытки каждой из частей синхронизировать свои часы с этим сервером может быть оптимальным подходом.

***Использование симметричного шифрования***

Для обеспечения *аутентификации* и распределения *ключа сессии* часто используется двухуровневая иерархия ключей симметричного шифрования. В общих чертах эта стратегия включает использование доверенного центра распределения ключей (*KDC*). Каждый участник разделяет секретный ключ, называемый также *мастер-ключом*, с *KDC*. *KDC* отвечает за создание ключей, называемых *ключами сессии*, и за распределение этих ключей с использованием *мастер-ключей*. *Ключи сессии* применяются в течение короткого времени для шифрования только данной сессии между двумя участниками.

Большинство алгоритмов распределения секретного ключа с использованием *KDC*, включает также возможность *аутентификации* участников.

*Протокол Нидхэма и Шредера*

Предполагается, что секретные *мастер-ключи* KA и KB разделяют соответственно А и *KDC* и В и *KDC*. Целью протокола является безопасное распределение *ключа сессии* KS между А и В. Протокол представляет собой следующую последовательность шагов:

1. A → KDC: IDA || IDB || N1

2. KDC → A: EKa [KS || IDB || N1 || EKb [KS || IDA] ]

3. A → B: EKb [KS || IDA]

4. B → A: EKS [N2]

5. A → B: EKS [f (N2)]

А запрашивает у *KDC* *ключ сессии* для установления защищенного соединения с В. Сообщение включает идентификацию А и В и уникальный идентификатор данной транзакции, который обозначен как N1 и называется *nonce*. *Nonce* может быть временной меткой, счетчиком или случайным числом; минимальное требование состоит в том, чтобы он отличался для каждого запроса. Кроме того, для предотвращения подделки желательно, чтобы противнику было трудно предугадать *nonce*. Таким образом, случайное число является лучшим вариантом для *nonce*.

1. *KDC* отвечает сообщением, зашифрованным ключом KА. Таким образом, только А может расшифровать сообщение, и А уверен, что оно получено от *KDC*, так как предполагается, что кроме А и *KDC* этот ключ не знает никто. Это сообщение включает следующие элементы, предназначенные для А:
   * Одноразовый *ключ сессии*.
   * Идентификатор В.
   * *nonce*, который идентифицирует данную сессию .

А должен убедиться, что полученный *nonce* равен значению *nonce* из первого запроса. Это доказывает, что ответ от *KDC* не был модифицирован при пересылке и не является повтором некоторого предыдущего запроса. Кроме того, сообщение включает два элемента, предназначенные для В:

* + Одноразовый *ключ сессии* KS.
  + Идентификатор А IDA.

Эти два последних элемента шифруются *мастер-ключом*, который *KDC* разделяет с В. Они посылаются В при установлении соединения и доказывают идентификацию А.

1. А сохраняет у себя *ключ сессии* и передает В информацию от *KDC*, предназначенную В: ЕKb [KS || IDA]. Так как эта информация зашифрована KВ, она защищена от просмотра. Теперь В знает *ключ сессии* (KS), знает, что другим участником является А, (IDA) и что начальная информация передана от *KDC*, т.к. она зашифрована с использованием KB.

В этой точке *ключ сессии* безопасно передан от А к В, и они могут начать безопасный обмен. Тем не менее, существует еще два дополнительных шага:

1. Используя созданный *ключ сессии*, В пересылает A *nonce* N2.
2. Также используя KS, А отвечает f (N2), где f - функция, выполняющая некоторую модификацию N2.

Эти шаги гарантируют B, что сообщение, которое он получил, не изменено и не является повтором предыдущего сообщения.

Заметим, что реальное распределение ключа включает только шаги 1 - 3, а шаги 4 и 5, как и 3, выполняют функцию *аутентификации*.

А безопасно получает *ключ сессии* на шаге 2. Сообщение на шаге 3 может быть дешифровано только B. Шаг 4 отражает знание В ключа KS, и шаг 5 гарантирует В знание участником А ключа KS и подтверждает, что это не устаревшее сообщение, так как используется *nonce* N2. Шаги 4 и 5 призваны предотвратить общий тип *replay-атак*. В частности, если противник имеет возможность захватить сообщение на шаге 3 и повторить его, то это должно привести к разрыву соединения.

Разрывая рукопожатие на шагах 4 и 5, протокол все еще уязвим для некоторых форм атак повторения. Предположим, что противник Х имеет возможность скомпрометировать старый *ключ сессии*. Маловероятно, чтобы противник мог сделать больше, чем просто копировать сообщение шага 3. Потенциальный риск состоит в том, что Х может заставить взаимодействовать А и B, используя старый *ключ сессии*. Для этого Х просто повторяет сообщение шага 3, которое было перехвачено ранее и содержит скомпрометированный *ключ сессии*. Если В не запоминает идентификацию всех предыдущих *ключей сессий* с А, он не сможет определить, что это повтор. Далее Х должен перехватить сообщение рукопожатия на шаге 4 и представиться А в ответе на шаге 5.

*Протокол Деннинга*

Деннинг предложил преодолеть эту слабость модификацией протокола Нидхэма и Шредера, которая включает дополнительную *отметку времени* на шагах 2 и 3:

1. A → KDC: IDA || IDB

2. KDC → A: EKa [KS || IDB || T || EKb [KS || IDA || T] ]

3. A → B: EKb [KS || IDA || T]

4. B → A: EKS [N1]

5. A → B: EKS [f (N1)]

*Т* – это *отметка времени*, которая гарантирует А и B, что *ключ сессии* является только что созданным. Таким образом, и А, и В знают, что распределенный ключ не является старым. А и В могут верифицировать *временную отметку* проверкой, что |Clock - T| < Δt1 + Δt2

где Δ t1 - оцениваемое нормальное расхождение между часами *KDC* и локальными часами (у А или B) и t2 - ожидаемая сетевая задержка времени. Каждый участник может установить свои часы, ориентируясь на определенный доверенный источник. Поскольку *временная отметка* Т шифруется с использованием секретных *мастер-ключей*, взломщик, даже зная старый *ключ сессии*, не сможет достигнуть цели повторением шага 3 так, чтобы В не заметил искажения времени.

Шаги 4 и 5 не были включены в первоначальное представление, но были добавлены позднее. Эти шаги подтверждают А, что В получил *ключ сессии*.

Протокол Деннинга обеспечивает большую степень безопасности по сравнению с протоколом Нидхэма и Шредера. Однако данная схема требует доверия к часам, которые должны быть синхронизированы в сети. В этом есть определенный риск, который состоит в том, что распределенные часы могут рассинхронизироваться в результате диверсии или повреждений. Проблема возникает, когда часы отправителя спешат по отношению к часам получателя. В этом случае противник может перехватить сообщение от отправителя и повторить его позднее, когда *отметка времени* в сообщении станет равной времени на узле получателя. Это повторение может иметь непредсказуемые последствия.

Один способ вычисления атак повторения состоит в требовании, чтобы участники регулярно сверяли свои часы с часами *KDC*. Другая альтернатива, при которой нет необходимости всем синхронизировать часы, состоит в доверии протоколам рукопожатия, использующим *nonce*.

*Протокол аутентификации с использованием билета*

Данный протокол пытается преодолеть проблемы, возникшие в предыдущих двух протоколах. Он выглядит следующим образом:

1. A → B: IDA || Na

2. B → KDC: IDB || Nb || EKb [IDA || Na || Tb]

3. KDC → A: EKa [IDB || Na || KS || Tb] || EKb [IDA || KS || Tb] || Nb

4. A → B: EKb [IDA || KS || Tb] || EKS [Nb]

А инициализирует *аутентификационный* обмен созданием *nonce* Na и посылкой его и своего идентификатора к В в незашифрованном виде. Этот *nonce* вернется к А в зашифрованном сообщении, включающем *ключ сессии*, гарантируя А, что *ключ сессии* не старый.

1. B сообщает *KDC*, что необходим *ключ сессии*. Это сообщение к KDС включает идентификатор В и *nonce* Nb. Данный *nonce* вернется к В в зашифрованном сообщении, которое включает *ключ сессии*, гарантируя B, что *ключ сессии* не устарел. Сообщение В к *KDC* также включает блок, зашифрованный секретным ключом, разделяемым В и *KDC*. Этот блок используется для указания *KDC*, когда заканчивается время жизни данного *ключа сессии*. Блок также специфицирует намеченного получателя и содержит *nonce*, полученный от А. Этот блок является своего рода "верительной грамотой" или "билетом" для А.
2. *KDC* получил *nonces* от А и В и блок, зашифрованный секретным ключом, который В разделяет с *KDC*. Блок служит билетом, который может быть использован А для последующих *аутентификаций*. *KDC* также посылает А блок, зашифрованный секретным ключом, разделяемым А и *KDC*. Этот блок доказывает, что В получил начальное сообщение А (IDB), что в нем содержится допустимая *отметка времени* и нет повтора (Na). Этот блок обеспечивает А *ключом сессии* (KS) и устанавливает ограничение времени на его использование (Тb).
3. А посылает полученный билет В вместе с *nonce* B, зашифрованным *ключом сессии*. Этот билет обеспечивает В *ключом сессии*, который тот использует для дешифрования и проверки *nonce*. Тот факт, что *nonce* B расшифрован *ключом сессии*, доказывает, что сообщение пришло от А и не является повтором.

Данный протокол аутентифицирует А и В и распределяет *ключ сессии*. Более того, протокол предоставляет в распоряжение А билет, который может использоваться для его последующей *аутентификации*, исключая необходимость повторных контактов с *аутентификационным* сервером. Предположим, что А и В установили сессию с использованием описанного выше протокола и затем завершили эту сессию. Впоследствии, но до истечения лимита времени, установленного протоколом, А может создать новую сессию с B. Используется следующий протокол:

1. A→ B: EKb [IDA || KS || Tb], Na'

2. B → A: Nb', ES [Na']

3. A → B: ES [Nb']

Когда В получает сообщение на шаге 1, он проверяет, что билет не просрочен. Заново созданные *nonces* Na' и Nb' гарантируют каждому участнику, что не было атак повтора. Время Tb является временем относительно часов B. Таким образом, эта временная метка не требует синхронизации, потому что В проверяет только им самим созданные *временные отметки*.

***Использование шифрования с открытым ключом***

*Протокол аутентификации с использованием аутентификационного сервера.*

Рассмотрим протокол, использующий *отметки времени* и *аутентификационный* сервер:

1. A → AS: IDA || IDB

2. AS → A: EKRas [IDA || KUa || T] || EKRas [IDB || KUb || T]

3. A → B: EKRas [IDA || KUa || T] || EKRas [IDB || KUb || T] || EKUb [EKRa [KS || T]]

В данном случае *третья доверенная сторона* является просто *аутентификационным* сервером AS, потому что *третья сторона* не создает и не распределяет секретный ключ. AS просто обеспечивает сертификацию открытых ключей участников. *Ключ сессии* выбирается и шифруется А, следовательно, не существует риска, что AS взломают и заставят распределять скомпрометированные *ключи сессии*. *Отметки времени* защищают от повтора скомпрометированных *ключей сессии*.

Данный протокол компактный, но, как и прежде, требует синхронизации часов.

*Протокол аутентификации с использованием KDC*

Другой подход использует *nonces*. Этот протокол состоит из следующих шагов:

1. A → KDC: IDA || IDB

2. KDC → A: EKRkdc [IDB || KUb]

3. A → B: EKUb [Na || IDA]

4. B → KDC: IDB || IDA || EKUkdc [Na]

5. KDC → B: EKRkdc [IDA || KUa] || EKUb [EKRkdc [Na || KS || IDB]]

6. B → A: EKUa [EKRkdc [Na || KS || IDB] || Nb]

7. A → B: EKS [Nb]

На первом шаге А информирует *KDC*, что хочет установить безопасное соединение с B. *KDC* возвращает А сертификат открытого ключа В (шаг 2). Используя открытый ключ B, А информирует В о создании защищенного соединения и посылает *nonce* Na (шаг 3). На 4-м шаге В спрашивает *KDC* о сертификате открытого ключа А и запрашивает *ключ сессии*. В включает *nonce* A, чтобы *KDC* мог пометить *ключ сессии* этим *nonce*. *Nonce* защищен использованием открытого ключа *KDC*. На 5-м шаге *KDC* возвращает В сертификат открытого ключа А плюс информацию {Na, KS, IDB}. Эта информация означает, что KS является секретным ключом, созданным *KDC* в интересах В и связан с Na. Связывание KS и Na гарантирует А, что KS не устарел. Эта тройка шифруется с использованием закрытого ключа *KDC*, это гарантирует B, что тройка действительно получена от *KDC*. Она также шифруется с использованием открытого ключа B, чтобы никто другой не мог подсмотреть *ключ сессии* и использовать эту тройку для установления соединения с А. На шаге 6 тройка {Na, KS, IDB}, зашифрованная закрытым ключом *KDC*, передается А вместе с *nonce* Nb, созданным B. Все сообщение шифруется открытым ключом А. А восстанавливает *ключ сессии* KS, использует его для шифрования Nb, который возвращает B. Это последнеее сообщение гарантирует B, что А знает *ключ сессии*.

Это достаточно безопасный протокол при различного рода атаках. Однако авторы предложили пересмотренную версию данного алгоритма:

1. A -> KDC: IDA || IDB

2. KDC -> A: EKRauth [IDB || KUb]

3. A -> B: EKUb [Na || IDA]

4. B -> KDC: IDB || IDA || EKUauth [Na]

5. KDC -> B: EKRauth [IDA || KUa] || EKUb [EKRauth [Na || KS || IDA || IDB]]

6. B -> A: EKUa [EKRauth [Na || KS || IDA || IDB] || Nb]

7. A -> B: EKS [Nb]

Добавляется идентификатор А IDA к данным, зашифрованных с использованием закрытого ключа *KDC* на шагах 5 и 6 для идентификации обоих участников сессии. Это включение IDA приводит к тому, что значение *nonce* Na должно быть уникальным только среди всех *nonces*, созданных А, но не среди *nonces*, созданных всеми участниками. Таким образом, пара {IDA, Na} уникально идентифицирует соединение, созданное А.

***Односторонняя аутентификация***

Существует специфическое приложение - электронная почта, для которого шифрование также имеет большое значение. Особенность e-mail состоит в том, что отправителю и получателю нет необходимости быть на связи в одно и то же время. Вместо этого сообщения направляются в почтовый ящик получателя, где они хранятся до тех пор, пока у того не появится возможность получить их.

Заголовок сообщения должен быть незашифрованным, чтобы сообщение могло пересылаться протоколами e-mail, такими как Х.400 или SMTP. Однако желательно, чтобы протоколы управления почтой не имели бы доступа к самому сообщению. Соответственно, e-mail сообщение должно быть зашифровано так, чтобы система управления почтой могла бы не знать ключ шифрования.

Вторым требованием является *аутентификация* сообщения. Обычно получателю нужна определенная гарантия того, что сообщение пришло от законного отправителя.

***Использование симметричного шифрования***

При использовании симметричного шифрования сценарий централизованного распределения ключей в полном объеме непригоден. Эти схемы требуют, чтобы в двух заключительных шагах отправитель посылал запрос получателю, ожидая ответа с созданным *ключом сессии*, и только после этого отправитель может послать сообщение.

С учетом перечисленных ограничений протоколы использования *KDC* являются возможными кандидатами для шифрования электронной почты. Для того чтобы избежать требования к получателю В находиться на связи в то же самое время, когда и отправитель А, шаги 4 и 5 должны быть опущены. Таким образом, остается последовательность шагов:

1. A -> KDC: IDA || IDB || N1

2. KDC -> A: EKa [KS || IDB || N1 || EKb [KS || IDA]]

3. A -> B: EKb [KS , IDA ] || EKS [M]

Данный подход гарантирует, что только требуемый получатель сообщения сможет прочитать его. Это также обеспечивает определенный уровень *аутентификации*, что отправителем является А. Очевидно, что протокол не защищает от атак повтора. Некоторая мера защиты может быть обеспечена включением *отметки времени* в сообщение. Однако поскольку существуют потенциальные задержки в процессе передачи e-mail сообщений, такие *временные отметки* имеют ограниченный срок действия.

***Использование шифрования с открытым ключом***

При использовании шифрования с открытым ключом требуется, чтобы отправитель знал открытый ключ получателя (для обеспечения конфиденциальности), получатель знал открытый ключ отправителя (для обеспечения *аутентификации*), или и то, и другое (для обеспечения конфиденциальности и *аутентификации*).

Если требуется конфиденциальность, то может быть использована следующая схема: A -> B: EKUb [KS] || EKS [M]

В этом случае сообщение шифруется одноразовым секретным ключом. А шифрует этот одноразовый ключ открытым ключом B. Только В имеет соответствующий закрытый ключ для получения одноразового ключа и использования этого ключа для дешифрования сообщения. Эта схема более эффективна, чем простое шифрование всего сообщения открытым ключом B.

Если требуется *аутентификация*, то цифровая подпись может быть создана по такой схеме: A -> B: M || EKRa [H (M)]

Этот метод гарантирует, что А не сможет впоследствии отвергнуть полученное сообщение. Однако данная технология открыта для другого типа подделок. Например, можно получить доступ к почтовой очереди перед доставкой, вырезать подпись отправителя, вставить свою и опять поставить сообщение в очередь на доставку.

Чтобы этого не допустить, сообщение и подпись можно зашифровать ключом симметричного шифрования, который в свою очередь шифруется открытым ключом получателя: A -> B: EKS [M || EKRa [H (M)]] || EKUb [KS]

Следующая схема не требует, чтобы В знал открытый ключ А. В этом случае должен использоваться сертификат открытого ключа:

A -> B: M || EKRa [H (M) ] || EKRas [T || IDA || KUa]

В конце сообщения А посылает В подпись, зашифрованную закрытым ключом А, и сертификат А, зашифрованный закрытым ключом *аутентификационного* сервера. Получатель применяет сертификат для получения открытого ключа отправителя и затем использует открытый ключ отправителя для проверки самого сообщения. Конфиденциальность может быть добавлена аналогично предыдущей схеме.

# Практическийраздел

ИНДИВИДУАЛЬНЫЕ ПРАКТИЧЕСКИЕ РАБОТЫ, ИХ ХАРАКТЕРИСТИКА

### Введение

По результатам работы студентом должен быть представлен и защищен отчет. Содержание отчета включает:

1. Титульный лист
2. Введение, содержащее постановку задачи.
3. Блок – схему алгоритма.
4. Распечатку скриншотов результатов ввода данных и исполнения программы.
5. Распечатку программного кода.
6. Вывод.

### ИПР №1

#### Часть 1.1 Алгоритмы шифрования DES и ГОСТ

**Цель:** реализовать на языке программирования C++ алгоритмы симметричного шифрования DES и ГОСТ 28147**.**

**Результат:** Программа шифрования, осуществляющая криптографическое преобразование введенного текста с помощью алгоритмов симметричного шифрования DES и ГОСТ 28147.

**Общая постановка задачи:**  Создать программу, читающую данные из файла и шифрующие их с помощью алгоритмов DES и ГОСТ 28147.

### ИПР №2

#### Часть 2.1. Алгоритм шифрования RSA

**Цель:** реализовать на языке программирования C++ алгоритм асимметричного шифрования RSA**.**

**Результат:** Программа шифрования, осуществляющая криптографическое преобразование введенного текста с помощью алгоритма симметричного шифрования RSA.

**Общая постановка задачи:**  Создать программу, читающую данные из файла, шифрующие и дешифрующие их с помощью алгоритма RSA.

## КОНТРОЛЬНЫЕ РАБОТЫ

### *Контрольная работа №1*

#### Указания по выбору варианта

Рабочей программой дисциплины «Методы защиты информации» предусмотрено выполнение двух контрольных работ. Контрольная работа № 1 подразумевает изучение и программную реализацию (на языке высокого уровня) алгоритма формирования Хэш-функции ГОСТ 3411. В качестве отчета по контрольной работе высылается листинг программной реализации, представленый в виде теста и исполняемый файл. В контрольной работе № 1 используется **один вариант** (для всех номеров зачетных книжек).

#### Теоретическая часть

1. Изучить алгоритм формирования Хэш-функции ГОСТ 3411.
2. Создать и протестировать алгоритм формирования Хэш-функции ГОСТ 3411 на языке высокого уровня.

### *Контрольная работа №2*

#### Указания по выбору варианта

Рабочей программой дисциплины «Методы защиты информации» предусмотрено выполнение двух контрольных работ. Контрольная работа № 2 подразумевает изучение и программную реализацию (на языке высокого уровня) алгоритма стандарта цифровой подписи ГОСТ 3410. В качестве отчета по контрольной работе высылается листинг программной реализации. представлена в виде теста и исполняемый файл. В контрольной работе № 2 используется **один вариант** (для всех номеров зачетных книжек).

#### Теоретическая часть

1. Изучить алгоритм цифровой подписи ГОСТ 3410.
2. Создать и протестировать алгоритм цифровой подписи ГОСТ 3410 на языке высокого уровня.

## Задание на курсовую работу, ее характеристика.

Курсовая работа учебным планом не предусмотрена.