А.М. ГОЛИКОВ

КОДИРОВАНИЕ И ШИФРОВАНИЕ ИНФОРМАЦИИ В РАДИОЭЛЕКТРОННЫХ СИСТЕМАХ ПЕРЕДАЧИ ИНФОРМАЦИИ. ЧАСТЬ. 2.: ШИФРОВАНИЕ

Учебное пособие

для специалитетета: 11.05.01 - Радиоэлектронные системы и комплексы (Радиоэлектронные системы передачи информации)

Курс лекций, компьютерные лабораторные работы, компьютерный практикум, задание на самостоятельную работу

Томск 2018

Министерство науки и образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
Томский государственный университет систем управления и радиоэлектроники

А.М. ГОЛИКОВ

КОДИРОВАНИЕ И ШИФРОВАНИЕ ИНФОРМАЦИИ В РАДИОЭЛЕКТРОННЫХ СИСТЕМАХ ПЕРЕДАЧИ ИНФОРМАЦИИ. ЧАСТЬ 2.: ШИФРОВАНИЕ

Учебное пособие

для специалитетета: 11.05.01 - Радиоэлектронные системы и комплексы

(Радиоэлектронные системы передачи информации)

Курс лекций, компьютерные лабораторные работы, компьютерный практикум, задание на самостоятельную работу

Томск 2018

УДК 621.39(075.8)
ББК 32.973(я73)
Г 60

Голиков, А. М. Кодирование и шифрование информации в радиоэлектронных системах передачи информации. Часть 2. Шифрование: Курс лекций, компьютерные лабораторные работы, компьютерный практикум, задание на самостоятельную работу [Электронный ресурс] / А. М. Голиков. — Изд. перераб. и доп. — Томск: ТУСУР, 2018. — 377 с. — Режим доступа: https://edu.tusur.ru/publications/

Учебное пособие является учебно-методическим комплексом дисциплины (УМКД) "Кодирование и шифрование информации в радиоэлектронных системах передачи информации" для специалитетета: 11.05.01 - Радиоэлектронные системы и комплексы (Радиоэлектронные системы передачи информации) и включает курс лекций, компьютерные лабораторные работы, компьютерный практикум, задание на самостоятельную работу.

Рассмотрены классические шифры, теория классических шифров, компьютерный практикум для классических шифров и задания на самостоятельную работу по классическим шифрам. Преставлены современные шифры с секретным ключом, теория шифров с секретным ключом, компьютерный практикум для шифров с секретным ключом и задания на самостоятельную работу по шифрам с секретным ключо. Рассмотрены отечественные и зарубежные шифры с открытым ключом, теория шифров с открытым ключом, компьютерный практикум для шифров с открытым ключом и задания на самостоятельную работу по шифрам с открытым ключом. Представлены криптографические протоколы в сетях передачи данных и компьютерный практикум для исследования протоколов SSL и TSL. Рассмотрено шифрование в современных системах связи стандартов GSM и LTE и компьютерный практикум для исследования стандарта LTE в MATLAB

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разные люди понимают под шифрованием разные вещи. Дети играют в игрушечные шифры и секретные языки. Это, однако, не имеет ничего общего с настоящей криптографией. Настоящая криптография (strong cryptography) должна обеспечивать такой уровень секретности, чтобы вы имели возможность надежно защитить критическую информацию от расшифровки крупными организациями — такими как мафия, транснациональные корпорации и крупные государства. Настоящая криптография в прошлом использовалась лишь в военных целях. Однако сейчас, со становлением информационного общества, она становится центральным инструментом для обеспечения конфиденциальности.

По мере образования информационного общества, крупным государствам становятся доступны Технологические средства тотального надзора за миллионами людей. Поэтому криптография становится одним из основных инструментов обеспечивающих конфиденциальность, доверие, авторизацию, электронные платежи, корпоративную безопасность и бесчисленное множество других важных вещей.

Криптография не является более придумкой военных, с которой не стоит связываться. Настала пора снять с криптографии покровы таинственности и использовать все ее возможности на пользу современному обществу. Широкое распространение криптографии является одним из немногих способов защитить человека от ситуации, когда он вдруг обнаруживает, что живет в тоталитарном государстве, которое может контролировать каждый его шаг.

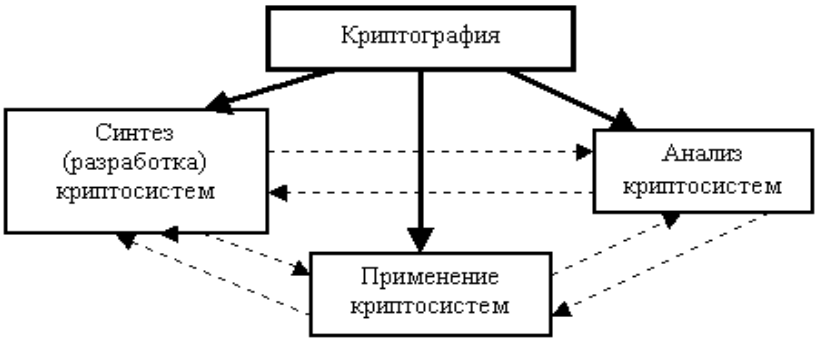
Основные термины криптографии и понятия

Криптография – до 70-х гг. XX в. – область науки и практической деятельности, связанная с разработкой, применением и анализом *шифрсистем*; в настоящее время – область науки, техники и практической деятельности, связанная с разработкой, применением и анализом *криптографических систем* защиты информации. Основными функциями криптографических систем являются обеспечение *конфиденциальности* и *аутентификации* различных аспектов информационного взаимодействия. Источником угроз при решении криптографических задач считаются преднамеренные действия противника или недобросовестного участника информационного взаимодействия, а не случайные искажения информации вследствие помех, отказов и т. п.

Конфиденциальность – защищенность информации от ознакомления с ее содержанием со стороны лиц, не имеющих права доступа к ней.

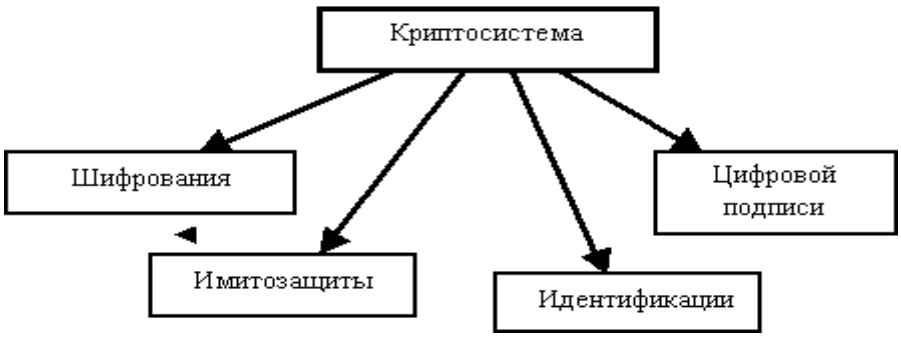
Аутентификация – установление (то есть проверка и подтверждение) подлинности различных аспектов информационного взаимодействия: сеанса связи, *сторон* (идентификация), *содержания* (имитозащита) и *источника* (установление авторства) передаваемых сообщений, времени взаимодействия и т. д. Является важной составной частью проблемы обеспечения достоверности получаемой информации. Особенно остро эта проблема стоит в случае не доверяющих друг другу сторон, когда источником угроз может служить не только третья сторона (противник), но и сторона, с которой осуществляется информационное взаимодействие.

Определение криптографии и показывает основные составляющие ее части. Пунктирные стрелки показывают тесные взаимосвязи между этими тремя составляющими.



Виды криптосистем

Система криптографическая (криптосистема) – система обеспечения безопасности защищенной сети, использующая *криптографические средства*. В качестве подсистем может включать системы *шифрования*, *идентификации*, *имитозащиты*, *цифровой подписи* и др., а также ключевую систему, обеспечивающую работу остальных систем. В основе выбора и построения криптосистемы лежит условие обеспечения *криптографической стойкости*. В зависимости от ключевой системы различают *симметричные* и *асимметричные* криптосистемы.



Средства криптографические – в широком смысле – методы и средства обеспечения безопасности информации, использующие *криптографические преобразования информации*; в узком смысле – средства, реализованные в виде документов, механических, электро-

механических, электронных технических устройств или программ, предназначенные для выполнения функций *криптографической системы*.

Криптографическое преобразование информации – преобразование информации с использованием одного из *криптографических алгоритмов*, определяемое целевым назначением *криптографической системы*.

Симметричные криптосистемы – криптосистемы с симметричными (секретными) ключами. Симметричность означает здесь, что ключи, задающие пару взаимно обратных криптографических преобразований, могут быть получены один из другого с небольшой трудоемкостью. Стойкость симметричной криптосистемы определяется трудоемкостью, с которой противник может вычислить любой из секретных ключей, и оценивается при общепринятом допущении, что противнику известны все элементы криптосистемы, за исключением секретного ключа.

Асимметричные криптосистемы – криптосистемы с асимметричными (секретными и открытыми) ключами. Асимметричность означает здесь, что из двух ключей, задающих пару взаимно обратных криптографических преобразований, один является секретным, а другой открытым. Открытые ключи известны всем участникам защищенной сети и противнику, но каждый участник сети хранит в тайне собственный секретный ключ. Стойкость асимметричной криптосистемы определяется трудоемкостью, с которой противник может вычислить секретный ключ, исходя из знания открытого ключа и другой дополнительной информации о криптосистеме.

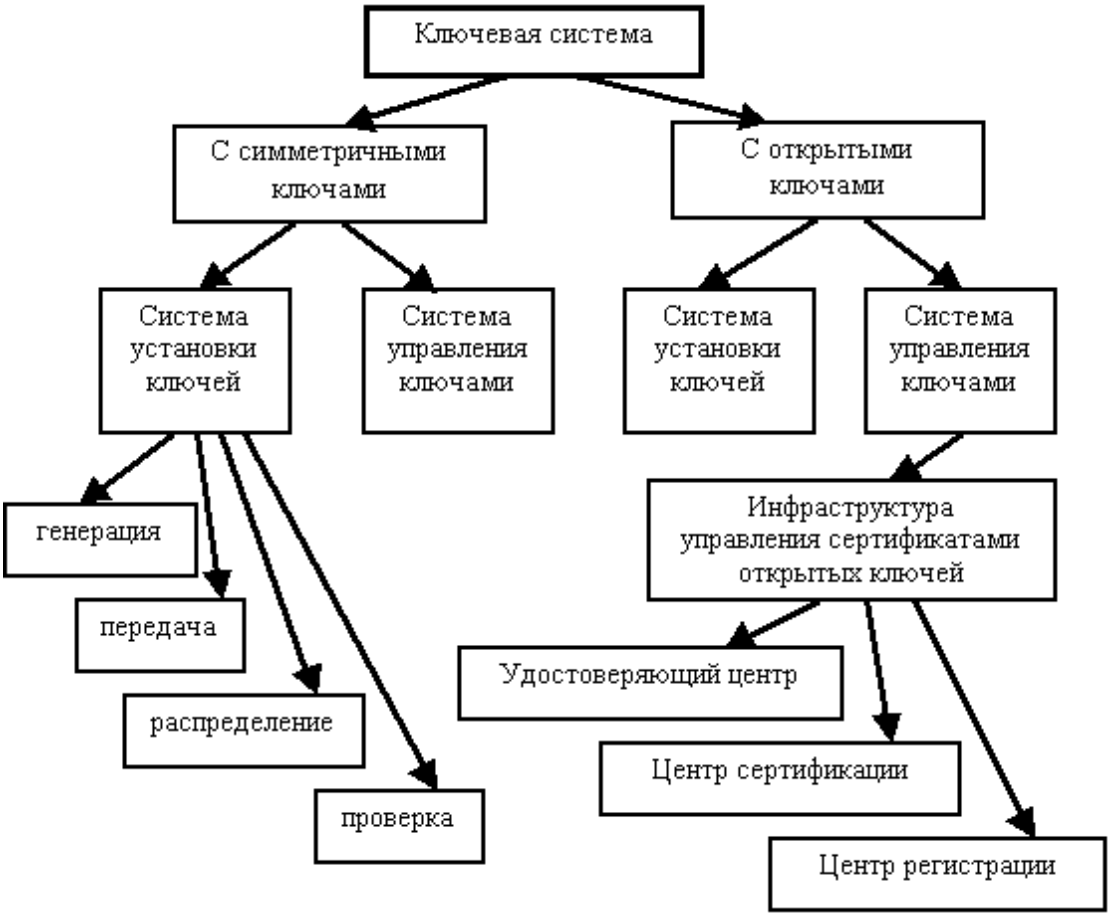
Шифрсистема – *криптографическая система* обеспечения конфиденциальности, предназначенная для защиты информации от ознакомления с ее содержанием лиц, не имеющих права доступа к ней, путем шифрования информации. Математическая модель шифрсистемы включает способ кодирования исходной и выходной информации, *шифр* и *ключевую систему*.

Система имитозащиты (обеспечения целостности) информации – *криптографическая система*, выполняющая функцию *аутентификации* содержания сообщения или документа и предназначенная для защиты от несанкционированного изменения информации или навязывания ложной информации. Математическая модель системы имитозащиты включает криптографический *алгоритм имитозащищенного кодирования информации* (это может быть *алгоритм шифрования*, *код аутентификации*, либо другое преобразование) и алгоритм принятия решения об истинности полученной информации, а также *ключевую систему*.

Система идентификации – *криптографическая система*, выполняющая функцию *аутентификации* сторон в процессе информационного взаимодействия. Математическая модель системы идентификации включает *протокол идентификации* и *ключевую систему*.

Система цифровой подписи – *криптографическая система*, выполняющая функцию *аутентификации* источника сообщения или документа и предназначенная для защиты от отказа субъектов от некоторых из ранее совершенных ими действий. Например, отправитель может отказаться от факта передачи сообщения, утверждая, что его создал сам получатель, а получатель легко может модифицировать, подменить или создать новое сообщение, а затем утверждать, что оно получено от отправителя. Математическая модель системы цифровой подписи включает *схему цифровой подписи* и *ключевую систему*.

Система ключевая – определяет порядок использования *криптографической системы* и включает *системы установки* и *управления ключами*.



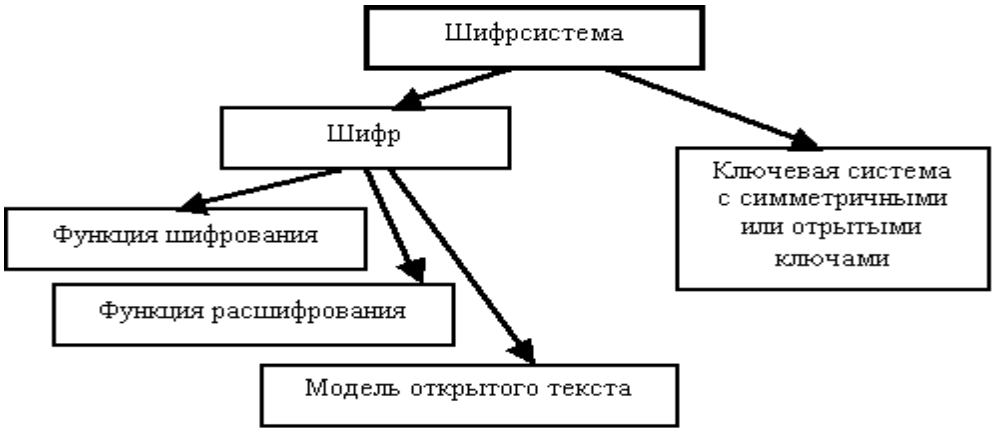
Система установки ключей – определяет алгоритмы и процедуры генерации, распределения, передачи и проверки ключей.

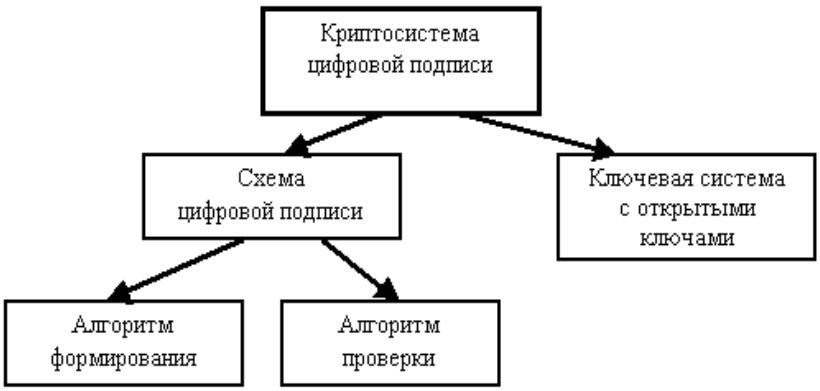
Система управления ключами – определяет порядок использования, смены, хранения и архивирования, резервного копирования и восстановления, замены или изъятия из обращения скомпрометированных, а также уничтожения старых ключей. Целью управления ключами является нейтрализация таких угроз, как: компрометация конфиденциальности секретных ключей, компрометация аутентичности секретных или открытых ключей, несанкционированное использование секретных или открытых ключей, например использование ключа, срок действия которого истек.

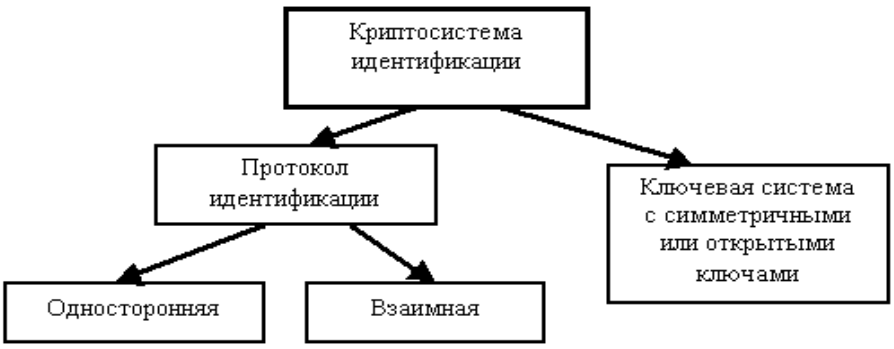
Система ключевая симметричной криптосистемы – основана на использовании симметричных (секретных) ключей. Основными проблемами таких систем являются построение *системы установки ключей* и обеспечение их сохранности для сетей с большим числом абонентов.

Система ключевая асимметричной криптосистемы – основана на использовании асимметричных ключей, состоящих из пары – открытого и секретного (закрытого) ключей. Основными проблемами таких систем являются построение *системы управления ключами*, как правило, представляющей собой инфраструктуру управления сертификатами открытых ключей, включающую центры регистрации и сертификации. Функции обоих центров могут объединяться одном удостоверяющем центре.

Стойкость криптографическая – свойство криптографической системы, характеризующее ее способность противостоять атакам противника, как правило, с целью получить ключ, открытое сообщение или навязать ложное сообщение.







Элементы криптосистем

Алгоритм имитозащищающего кодирования информации – алгоритм преобразования информации (как правило, основан на внесении и использовании избыточности) с целью контроля *целостности*. В отличие от алгоритма формирования цифровой подписи, использует симметричные криптографические системы. В качестве такого преобразования может выступать *код аутентификации*, автоматное и другие преобразования, либо *алгоритм шифрования*.

Алгоритм проверки цифровой подписи – алгоритм, в качестве исходных данных которого используются подписанное сообщение, ключ проверки и параметры *схемы цифровой подписи*, а результатом является заключение о правильности или ошибочности *цифровой подписи*.

Алгоритм расшифрования – алгоритм, реализующий *функцию расшифрования*.

Алгоритм формирования цифровой подписи – алгоритм, в качестве исходных данных которого используются сообщение, ключ подписи и параметры *схемы цифровой подписи*, а в результате формируется *цифровая подпись*.

Алгоритм шифрования – алгоритм, реализующий *функцию шифрования*[1].

Жизненный цикл ключей – последовательность стадий, которые проходят ключи от момента генерации до уничтожения. Включает такие стадии, как: генерация ключей, регистрация пользователей и ключей, инициализация ключей, период действия, хранение ключа, замена ключа, архивирование, уничтожение ключей, восстановление ключей, отмена ключей.

Имитовставка – проверочная комбинация, добавляемая к сообщению для проверки целостности.

Имитостойкость – способность противостоять активным атакам со стороны противника, целью которых является навязывание ложного или подмена передаваемого сообщения или хранимых данных.

Код аутентификации – *алгоритм имитозащищающего кодирования информации* (как правило, вычисляет значение *имитовставки*). К кодам аутентификации предъявляются требования: большая сложность вычисления значения кода аутентификации для заданного сообщения без знания ключа; большая сложность подбора для заданного сообщения с известным значением кода аутентификации другого сообщения с известным значением кода аутентификации без знания ключа. Без знания секретного ключа вероятность успешного навязывания противником искаженной или ложной информации мала.

Открытое распределение ключей (согласование ключа, выработка общего значения ключа) – протокол, позволяющий двум абонентам выработать общий секретный ключ путем обмена сообщениями по открытому каналу связи без передачи какой-либо общей секретной информации, распределяемой заранее. Важным преимуществом открытого распределения является то, что ни один из абонентов заранее не может определить значение ключа, так как ключ зависит от сообщений, передаваемых в процессе обмена.

Помехоустойчивость – способность сохранять устойчивую работу при наличии помех в канале связи.

Протокол – распределенный алгоритм, в котором участвуют две или более стороны, обменивающиеся между собой сообщениями.

Протокол идентификации – протокол *аутентификации* сторон, участвующих во взаимодействии и не доверяющих друг другу. Различают протоколы односторонней и взаимной идентификации. Протоколы идентификации, как правило, основаны на известной обеим сторонам информации (пароли, личные идентификационные номера (PIN), ключи). В дополнение к протоколу идентификации могут использоваться некоторые физические приборы, с помощью которых и проводится идентификация (магнитная или интеллектуальная пластиковая карта, или прибор, генерирующий меняющиеся со временем пароли), а также физические параметры, составляющие неотъемлемую принадлежность доказывающего (подписи, отпечатки пальцев, характеристики голоса, геометрия руки и т. д.).

Протокол криптографический – *протокол*, предназначенный для выполнения функций криптографической системы, в процессе выполнения которого стороны используют *криптографические алгоритмы*.

Протокол распределения ключей – протокол, в результате выполнения которого взаимодействующие стороны (участники, группы участников) получают необходимые для функционирования криптографической системы ключи. Различают следующие типы протоколов распределения ключей: протоколы передачи (уже сгенерированных) ключей; протоколы (совместной) выработки общего ключа (*открытое распределение ключей*); *схемы предварительного распределения ключей*. В зависимости от порядка взаимодействия сторон

выделяют *двусторонние протоколы*, в которых стороны осуществляют передачу ключей при непосредственном взаимодействии, или, иначе, протоколы типа "точка-точка", и *протоколы с централизованным распределением ключей*, предусматривающие наличие третьей стороны, играющей роль доверенного центра.

Схема цифровой подписи состоит из двух алгоритмов, один – для формирования, а второй – для проверки подписи. Надежность схемы цифровой подписи определяется сложностью следующих трех задач для лица, не являющегося владельцем секретного ключа: *подделки подписи*, то есть вычисления значения подписи под заданным документом; *создания подписанного сообщения*, то есть нахождения хотя бы одного сообщения с правильным значением подписи; *подмены сообщения*, то есть подбора двух различных сообщений с одинаковыми значениями подписи.

Схема предварительного распределения ключей – состоит из двух алгоритмов: распределения исходной ключевой информации и формирования ключа. С помощью первого алгоритма осуществляется генерация исходной ключевой информации. Эта информация включает открытую часть, которая будет передана всем сторонам или помещена на общедоступном сервере, а также секретные части каждой стороны. Второй алгоритм предназначен для вычисления действующего значения ключа для взаимодействия между абонентами по имеющейся у них секретной и общей открытой части исходной ключевой информации. Применяется для уменьшения объема хранимой и распределяемой секретной ключевой информации. Схема предварительного распределения ключей должна быть устойчивой, то есть учитывать возможность раскрытия части ключей при компрометации, обмане или сговоре абонентов, и гибкой – допускать возможность быстрого восстановления путем исключения скомпрометированных и подключения новых абонентов.

Функция криптографическая – функция, необходимая для реализации *криптографической системы*, например, генерация ключей и псевдослучайных последовательностей, обратимое преобразование, однонаправленная функция, вычисление и проверка значений имитовставки и цифровой подписи, вычисление значения хэш-функции и т. п., обладают определенными криптографическими свойствами, влияющими на криптографическую стойкость: зависимость от ключа, сложность обращения и др.

Функция расшифрования – осуществляет преобразование множества открытых сообщений в множество шифрованных сообщений, зависящее от ключа, является обратным к преобразованию, осуществляемому *функцией шифрования*.

Функция шифрования – осуществляет преобразование множества открытых сообщений в множество шифрованных сообщений, зависящее от ключа.

Цифровая подпись (сообщения или электронного документа) – представляет собой конечную цифровую последовательность, зависящую от самого сообщения или документа и от секретного ключа, известного только подписывающему субъекту, предназначенная для установления авторства. Предполагается, что цифровая подпись должна быть легко проверяемой без получения доступа к секретному ключу. При возникновении спорной ситуации, связанной с отказом подписывающего от факта подписи некоторого сообщения либо с попыткой подделки подписи, третья сторона должна иметь возможность разрешить спор. Цифровая подпись позволяет решить следующие три задачи: осуществить *аутентификацию* источника данных, установить целостность сообщения или электронного документа, обеспечить невозможность отказа от факта подписи конкретного сообщения.

Шифр – семейство обратимых преобразований множества открытых сообщений в множество шифрованных сообщений и обратно, каждое из которых определяется некоторым параметром, называемым ключом. Математическая модель шифра включает две функции: *шифрования* и *расшифрования*, и модель множества открытых сообщений. В зависимости от способа представления открытых сообщений различают блочные, поточные и другие шифры. Основными требованиями, определяющими качество шифра, являются: *криптографическая стойкость*, *имитостойкость*, *помехоустойчивость* и др.

1. КЛАССИЧЕСКИЕ ШИФРЫ

История криптографии насчитывает не одно тысячелетие. Уже в исторических документах древних цивилизаций – Индии, Египте, Китае, Месопотамии – имеются сведения о системах и способах составления шифрованного письма. Видимо, первые системы шифрования появились одновременно с письменностью в четвертом тысячелетии до нашей эры.

В древнеиндийских рукописях приводится более шестидесяти способов письма, среди которых есть и такие, которые можно рассматривать как криптографические. Имеется описание системы замены гласных букв согласными и наоборот. Один из сохранившихся шифрованных текстов Месопотамии представляет собой табличку, написанную клинописью и содержащую рецепт изготовления глазури для гончарных изделий. В этом тексте использовались редко употребляемые значки, игнорировались некоторые буквы, употреблялись цифры вместо имен. В рукописях Древнего Египта шифровались религиозные тексты и медицинские рецепты. Шифрование использовалось в Библии. Некоторые фрагменты библейских текстов зашифрованы с помощью шифра, который назывался *атбаш.* Правило зашифрования состояло в замене i-й буквы алфавита ($i = \overline{1, n}$) буквой с номером $n - i + 1$, где n – число букв алфавита. Происхождение слова *атбаш* объясняется принципом замены букв. Это слово составлено из букв Алеф, Тае, Бет и Шин, то есть первой и последней, второй и предпоследней букв древнесемитского алфавита.

Развитию криптографии способствовал переход от идеографического письма, основанного на использовании огромного числа иероглифов, к фонетическому письму. В древнем семитском алфавите во втором тысячелетии до нашей эры было уже 30 знаков. Ими обозначались согласные звуки, а также некоторые гласные и слоги. Упрощение письма стимулировало развитие криптографии.

В Древней Греции криптография уже широко использовалась в разных областях деятельности, в особенности в государственной сфере. Плутарх сообщает, что жрецы, например, хранили в форме тайнописи свои прорицания. В Спарте в V – IV веках до н. э. использовалось одно из первых шифровальных приспособлений – *Сцитала.* Это был жезл цилиндрической формы, на который наматывалась лента пергамента. Кроме жезла могли использоваться рукоятки мечей, кинжалов копий и т.д. Вдоль оси цилиндра на пергамент построчно записывался текст, предназначенный для передачи. После записи текста лента сматывалась с жезла и передавалась адресату, который имел точно такую же Сциталу. Ясно, что такой способ шифрования осуществлял перестановку букв сообщения. Ключом шифра

служит диаметр Сциталы. Известен также и метод вскрытия такого шифра, приписываемый Аристотелю. Предлагалось заточить на конус длинный брус и, обернув вокруг него ленту, начать сдвигать ее по конусу от малого диаметра до самого большого. В том месте, где диаметр конуса совпадал с диаметром Сциталы, буквы текста сочетались в слоги и слова. После этого оставалось лишь изготовить цилиндр нужного диаметра.

Другим шифровальным приспособлением времен Спарты была *табличка Энея*. На небольшой табличке горизонтально располагался алфавит, а по ее боковым сторонам имелись выемки для наматывания нити. При зашифровании нить закреплялась у одной из сторон таблички и наматывалась на нее. На нити делались отметки (например, узелки) в местах, которые находились напротив букв данного текста. По алфавиту можно было двигаться лишь в одну сторону, то есть делать по одной отметке на каждом витке. После зашифрования нить сматывалась и передавалась адресату. Этот шифр представляет собой шифр замены букв открытого текста знаками, которые означали расстояния между отметками на нити. Ключом являлись геометрические размеры таблички и порядок расположения букв алфавита. Это был довольно надежный шифр, история не сохранила документов, подтверждающих сведения о методах его вскрытия.

Греческий писатель Полибий использовал систему сигнализации, которая была широко принята как метод шифрования. Он записывал буквы алфавита в квадратную таблицу и заменял их координатами: парами чисел (i, j), где i – номер строки, j – номер столбца. Применительно к латинскому алфавиту *квадрат Полибия* имеет следующий вид.

Таблица 1.1. Квадрат Полибия

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I, J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Пары (i, j) передавались с помощью факелов. Например, для передачи буквы О нужно было взять 3 факела в правую руку и 4 факела – в левую.

Подобные шифровальные приспособления с небольшими изменениями просуществовали до эпохи военных походов Юлия Цезаря. Положение меняется в эпоху расцвета Рима, который первоначально представлял собой лишь небольшую гражданскую общину, со временем разросся, подчинив себе сначала Италию, а затем и все Средиземноморье. Чтобы управлять наместниками в многочисленных провинциях, шифрованная связь для римских органов власти стала жизненно необходимой. Особую роль в сохранении тайны сыграл

способ шифрования, предложенный Юлием Цезарем и изложенный им в "Записках о галльской войне" (I в. до н. э.). Вот что пишет о нем Гай Светоний: "...существуют и его письма к Цицерону и письма к близким о домашних делах: в них, если нужно было сообщить что-нибудь негласно, он пользовался тайнописью, т. е. менял буквы так, чтобы из них не складывалось ни одного слова. Чтобы разобрать и прочитать их, нужно читать всякий раз четвертую букву вместо первой, например. *D* вместо *A* и так далее". Таким образом, Цезарь заменял буквы в соответствии с подстановкой, нижняя строка которой представляет собой алфавит открытого текста, сдвинутый циклически на три буквы влево.

Со времен Цезаря до XV в. шифровальное дело претерпело много изменений, однако нам мало известно о методах и системах шифрования, применяемых в этот период времени. В мрачные годы средневековья практика шифрования сохранялась в строжайшей тайне. Так, в годы крестовых походов шифровальщики, служившие у Папы Римского, после года работы подлежали физическому уничтожению.

В эпоху Возрождения в итальянских городах-государствах параллельно с расцветом культуры и науки активно развивается криптография. Нередко ученые зашифровывали научные гипотезы, чтобы не прослыть еретиком и не подвергнуться преследованиям инквизиции.

Научные методы в криптографии впервые появились, по-видимому, в арабских странах. Арабского происхождения и само слово *шифр.* О тайнописи и ее значении говорится даже в сказках "Тысячи и одной ночи". Первая книга, специально посвященная описанию некоторых шифров, появилась в 855 г., она называлась "Книга о большом стремлении человека разгадать загадки древней письменности". В 1412 г. издается 14-томная энциклопедия, содержавшая систематический обзор всех важнейших областей человеческого знания, – "Шауба аль-Аща". Ее автор – Шехаб аль-Кашканди. В этой энциклопедии есть раздел о криптографии под заголовком "относительно сокрытия в буквах тайных сообщений", в котором приводятся семь способов шифрования. Там же дается перечень букв в порядке частоты их употребления в арабском языке на основе изучения текста Корана, а также приводятся примеры раскрытия шифров *методом частотного анализа* встречаемости букв.

В XIV в. появилась книга о системах тайнописи, написанная сотрудником тайной канцелярии Папы Римского Чикко Симонетти. В этой книге приводятся шифры замены, в которых гласным буквам соответствует несколько значковых выражений. Такие шифры позже стали называться шифрами *многозначной замены* или *омофонами*. Они получили развитие в XV в. Так, в книге "Трактат о шифрах" Габриеля де Лавинды – секретаря папы Климентия XII – приводится описание шифра *пропорциональной замены,* в котором каждой

букве ставится в соответствие несколько эквивалентов, число которых пропорционально частоте встречаемости буквы в открытом тексте. В 1469 г. был предложен подобный же шифр, получивший название "миланский ключ". Появление омофонов свидетельствовало о том, что к тому времени уже хорошо осознавали слабости шифров простой замены. Такая модификация шифра разрушала статистику букв открытого сообщения, что явилось заметным шагом в развитии криптографии.

Еще один значительный шаг вперед криптография сделала благодаря труду Леона Альберти. Известный философ, живописец, архитектор, он в 1466 г. написал труд о шифрах. В этой работе был предложен шифр, основанный на использовании *шифровального диска.* Сам Альберти назвал его шифром, "достойным королей".

Шифровальный диск представлял собой пару соосных дисков разного диаметра (рисунок 1). Больший из них – неподвижный, его окружность разделена на 24 равных сектора, в которые вписаны 20 букв латинского алфавита в их естественном порядке и 4 цифры (от 1 до 4). При этом из 24-буквенного алфавита были удалены 4 буквы, без которых можно было обойтись, подобно тому, как в русском языке обходятся без Ъ, Ё, Й. Меньший диск – подвижный, по его окружности, разбитой также на 24 сектора, были вписаны все буквы смешанного латинского алфавита.

Рис. 1.1. Диск Альберти

Имея два таких прибора, корреспонденты договаривались о первой индексной букве на подвижном диске. При шифровании сообщения отправитель ставил индексную букву против любой буквы большего диска. Он информировал корреспондента о таком положении диска, записывая эту букву внешнего диска в качестве первой буквы шифртекста. Очередная буква открытого текста отыскивалась на неподвижном Диске и стоящая против нее буква

меньшего диска являлась результатом ее зашифрования. После того как были зашифрованы несколько букв текста, положение индексной буквы изменялось, о чем также каким-либо образом передавалось корреспонденту.

Такой шифр имел две особенности, которые делают изобретение Альберти событием в истории криптографии. Во-первых, в отличие от шифров простой замены шифровальный диск использовал не один, а несколько алфавитов для зашифрования. Такие шифры получили название *многоалфавитных*. Во-вторых, шифровальный диск позволял использовать так называемые *коды с перешифрованием*, которые получили широкое распространение лишь в конце XIX века, то есть спустя четыре столетия после изобретения Альберта. Для этой цели на внешнем диске имелись цифры. Альберта составил *код,* состоящий из 336 кодовых групп, занумерованных от 11 до 4444. Каждому кодовому обозначению соответствовала некоторая законченная фраза. Когда такая фраза встречалась в открытом сообщении, она заменялась соответствующим кодовым обозначением, а с помощью диска цифры зашифровывались как обычные знаки открытого текста, превращаясь в буквы.

Богатым на новые идеи в криптографии оказался XVI век. Многоалфавитные шифры получили развитие в вышедшей в 1518 г. первой печатной книге по криптографии под названием "Полиграфия". Автором книги был один из самых знаменитых ученых того времени аббат Иоганнес Тритемий. В этой книге впервые в криптографии появляется квадратная таблица. Шифралфавиты записаны в строки таблицы один под другим, причем каждый из них сдвинут на одну позицию влево по сравнению с предыдущим.

Тритемий предлагал использовать эту таблицу для многоалфавитного зашифрования самым простым из возможных способов: первая буква текста шифруется первым алфавитом, вторая буква – вторым и т. д. В этой таблице не было отдельного алфавита открытого текста, для этой цели служил алфавит первой строки. Таким образом, открытый текст, начинающийся со слов HUNC CAVETO VIRUM..., приобретал вид HXPF GFBMCZ FUEIB....

Преимущество этого метода шифрования по сравнению с методом Альберти состоит в том, что с каждой буквой задействуется новый алфавит. Альберти менял алфавиты лишь после трех или четырех слов. Поэтому его шифртекст состоял из отрезков, каждый из которых обладал закономерностями открытого текста, которые помогали вскрыть криптограмму. Побуквенное зашифрование не дает такого преимущества. *Шифр Тритемия* является также первым нетривиальным примером *периодического шифра*. Так называется многоалфавитный шифр, правило зашифрования которого состоит в использовании периодически повторяющейся последовательности простых замен.

В 1553 г. Джованни Баттиста Белазо предложил использовать для многоалфавитного шифра буквенный, легко запоминаемый ключ, который он назвал *паролем.* Паролем могло служить слово или фраза. Пароль периодически записывался над открытым текстом. Буква пароля, расположенная над буквой текста, указывала на алфавит таблицы, который использовался для зашифрования этой буквы. Например, это мог быть алфавит из таблицы Тритемия, первой буквой которого являлась буква пароля. Однако Белазо, как и Тритемий, использовал в качестве шифралфавитов обычные алфавиты.

Воскресить смешанные алфавиты, которые применял Альберти, и объединить идеи Альберти с идеями Тритемия и Белазо в современную концепцию многоалфавитной замены выпало на долю итальянца Джованни де ла Порта. Ему было 28 лет, когда он в 1563 г. опубликовал книгу "О тайной переписке". По сути, эта книга являлась учебником по криптографии, содержащим криптографические познания того времени. Порта предложил использовать квадратную таблицу с периодически сдвигаемым смешанным алфавитом и паролем. Он советовал выбирать длинный ключ. Впервые им был предложен *шифр простои биграммной замены*, в котором пары букв представлялись одним специальным графическим символом. Они заполняли квадратную таблицу размеров 20x20, строки и столбцы которой занумерованы буквами алфавита

A B C D E F G H I L M N O P Q R S T U Z

Например, биграмма EA заменялась символом Δ, биграмма LF – символом ☉ и т. д. В своей книге Порта ввел многоалфавитный шифр, определяемый таблицей 6.2.

Таблица 1.2. Таблица Порта

A B	a n	b o	c p	d q	e r	f s	g t	h u	i x	k y	l z	m w
C D	a o	b p	c q	d r	e s	f t	g u	h x	i y	k z	l w	m n
E F	a p	b q	c r	d s	e t	f u	g x	h y	i z	k w	l n	m o
G H	a q	b r	c s	d t	e u	f x	g y	h z	i w	k n	l o	m p
I K	a r	b s	c t	d u	e x	f y	g z	h w	i n	k o	l p	m q
L M	a s	b t	c u	d x	e y	f z	g w	h n	i o	k p	l q	m r
N O	a t	b u	c x	d y	e z	f w	g n	h o	i p	k q	l r	m s
P Q	a u	b x	c y	d z	e w	f n	g o	h p	i q	k r	l s	m t
R S	a x	b y	c z	d w	e n	f o	g p	h q	i r	k s	l t	m u

T U	a y	b z	c w	d n	e o	f p	g q	h r	i s	k t	l u	m x
X Y	a z	b w	c n	d o	e p	f q	g r	h s	i t	k u	l x	m y
Z W	a w	b n	c o	d p	e q	f r	g s	h t	i u	k x	l y	m z

Шифрование осуществляется при помощи лозунга, который пишется над открытым текстом. Буква лозунга определяет алфавит (заглавные буквы первого столбца), расположенная под ней буква открытого текста ищется в верхнем или нижнем полуалфавите и заменяется соответствующей ей буквой второго полуалфавита. Например, фраза, начинающаяся словами HUNC CAVETO VIRUM..., будет зашифрована при помощи лозунга DE LA PORTA в XFHP YTMOGA FQEAS.

Еще одно важное усовершенствование многоалфавитных систем, состоящее в идее использования в качестве ключа текста самого сообщения или же шифрованного текста, принадлежит Джероламо Кардано и Блезу де Виженеру. Такой шифр был назван *самоключом*. В книге Виженера "Трактат о шифрах" самоключ представлен следующим образом. В простейшем случае за основу бралась таблица Тритемия с добавленными к ней в качестве первой строки и первого столбца алфавитами в их естественном порядке. Позже такая таблица стала называться *таблицей Виженера.* Подчеркнем, что в общем случае таблица Виженера состоит из циклически сдвигаемых алфавитов, причем первая строка может быть произвольным смешанным алфавитом (см. табл. 1.3).

Таблица 1.3. Таблица Виженера

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**X**	**Y**	**Z**	**W**
A	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W
B	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A
C	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B
D	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C
E	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D
F	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E
G	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F
H	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G
I	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H
K	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I
L	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K
M	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L
N	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M
O	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N
P	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O
Q	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P

R	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q
S	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R
T	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S
U	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T
X	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U
Y	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X
Z	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y
W	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z

Первая строка служит алфавитом открытого текста, а первый столбец – алфавитом ключа. Для зашифрования открытого сообщения ($T_o = t_1t_2...$) Виженер предлагал в качестве *ключевой последовательности* (Г) использовать само сообщение (T_o) с добавленной к нему в качестве первой буквы (t_o), известной отправителю и получателю (этим идея Виженера отличалась от идеи Кардано, у которого не было начальной буквы и система которого не обеспечивала однозначности расшифрования). Последовательности букв подписывались друг под другом:

$Г = t_0t_1t_2... t_{i-1}...$

$T_o = t_1t_2t_3... t_i...$

———————

$T_ш = s_1s_2s_3... s_i...$

При этом пара букв, стоящих друг под другом в Г и T_o, указывала, соответственно, номера строк и столбцов таблицы, на пресечении которых находится знак s_i шифрованного текста ($T_ш$). Например, фраза HUNC CAVETO VIRUM..., использованная в предыдущих примерах, и начальная буква *P* дают шифртекст YCHP ECUWZH IDAMG.

Во втором варианте Виженер предлагал в качестве ключевой последовательности использовать шифрованный текст:

$Г = s_0s_1s_2... s_{i-1}...$

$T_o = t_1t_2t_3... t_i...$

———————

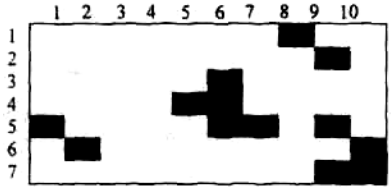
$T_ш = s_1s_2s_3... s_i...$

Самоключ Виженера был незаслуженно забыт на долгое время, а под *шифром Виженера* до сих пор понимают самый простой вариант с коротким ключевым словом и с таблицей, состоящей из обычных алфавитов.

Кардано принадлежит также идея *поворотной решетки* как средства шифрования. Изначально обычная решетка представляла собой лист из твердого материала, в котором через неправильные интервалы сделаны прямоугольные вырезы высотой для одной строчки и различной длины. Накладывая эту решетку на лист писчей бумаги, можно было записывать

в вырезы секретное сообщение. После этого, сняв решетку, нужно было заполнить оставшиеся свободные места на листе бумаги неким текстом, маскирующим секретное сообщение. Подобным стенографическим методом маскировки сообщения пользовались многие известные исторические лица, например кардинал Ришелье во Франции и русский дипломат и писатель А. Грибоедов. Так, Ришелье использовал прямоугольник размера 7×10. Для длинных сообщений прямоугольник использовался несколько раз. Прорези трафарета размещались в позициях:

(1,8), (2,9), (3,6), (4,5), (4,6), (5,1), (5,6), (5,7), (5,9), (6,2), (6,10), (7,9), (7,10).



Следующий текст выглядит как невинное любовное письмо:

	1	2	3	4	5	6	7	8	9	10
1	I		L	O	V	E		Y	O	U
2	I		H	A	V	E		Y	O	U
3	D	E	E	P		U	N	D	E	R
4	M	Y		S	K	I	N		M	Y
5	L	O	V	E		L	A	S	T	S
6	F	O	R	E	V	E	R		I	N
7	H	Y	P	E	R	S	P	A	C	E

Однако, используя трафарет Ришелье, получим зловещую команду:

YOU KILL AT ONES

Кардано использовал квадратную решетку, которая своими вырезами однократно покрывает всю площадь квадрата при ее самосовмещениях. На основе такой решетки он построил шифр перестановки.

Нельзя не упомянуть в историческом обзоре имени Матео Ардженти, работавшего в области криптографии в начале XVII в. Он составил руководство по криптографии на 135 листах, изданное в переплете из телячьей кожи. В этой книге впервые предложено использовать некоторое слово в качестве мнемонического ключа для смешанного алфавита. Началом смешанного алфавита служило ключевое слово (как правило, без повторяющихся букв), за которым следовали остальные буквы в их естественном порядке. Например, ключевое слово PIETRO дает смешанный латинский алфавит

P I E T R O A B C D F G H L M N Q S U Z

Такие смешанные алфавиты часто использовались в качестве алфавитов шифртекста в шифрах простой замены.

С целью усложнения шифра простой замены Ардженти вводил *пустышки,* которые добавлялись в шифрованное сообщение, использовал шифробозначения разной значности, Для некоторых частых сочетаний букв текста вводил отдельные обозначения, придавал

частым буквам несколько обозначений. Позже подобные идеи получили широкое распространение. Приведем пример *шифра Ардженти* (см. табл. 1.4).

Таблица 1.4. Шифр Ардженти

A	B	C	D	E	F	G	H	I	L	M	N	O	P
1	86	02	20	62 82	22	06	60	3	24	26	84	9	66

Q	R	S	T	U	Z	ET	CON	NON	CHE	∅
68	28	42	80	04 40	88	08	64	00	44	5 7

Слово ARGENTI может быть зашифровано многими способами, например так:

5128068285480377

или же так:

172850675628455803

Наибольшим достижением Ардженти считается разработанный им *буквенный код* – один из шифров замены, в котором буквы, слоги, слова и целые фразы заменялись группами букв. Необходимым количеством словарных величин в коде в то время считалось 1200.

В истории криптографии XVII – XVIII века называют эрой *"черных кабинетов"*. В этот период во многих государствах Европы, в первую очередь во Франции, получили развитие дешифровальные подразделения, названные "черными кабинетами". Первый из них образован по инициативе кардинала Ришелье при дворе короля Людовика XIII. Его возглавил первый профессиональный криптограф Франции Антуан Россиньоль. Следует отметить, что некоторые оригинальные идеи, возникшие в криптографии в этот период, связаны с именем самого Ришелье, который использовал, например, для секретной переписки с королем оригинальный шифр перестановки с переменным ключом. Его использование становится понятным из следующего примера:

Шифр Ришелье

Ключ: 2741635; 15243; 671852493; 07; 28615; 943; …

Открытый текст: LETTER SENT TO THE EMPEROR GIVING FULL DETAIL

Ключ: (2741635) (15243) (671852493) (07) (28615) (943) (27 41635)

Шифртекст: TLRTSEE ETOTN EPOEMTHER NI LUGIG VFR TLIE SAD

Известно, что Ришелье пользовался также кодами. Попутно отметим, что свой несложный код был и у знаменитого Наполеона:

Малый шифр Наполеона (Petit Chiffre)

(реконструирован Базери)

A – 15, ar – 25, al – 39

B – 37, bu – 3, bo – 35, bi – 29

C – 6, ca – 32, ce – 20

D – 23, de – 52

E – 53, es – 82, et – 50, en – 68

F – 55, fa – 69, fe – 58, fo – 71

G – 81, ga – 51

H – 85, hi – 77

I – 119, jai – 122

J – 87, jai – 123

K – ?

L – 96, lu – 103, le – 117, la – 106

M – 114, ma – 107

N – 115, ne – 94, ni – 116

O – 90, ot – 153

P – 137, po – 152

Q – 173, que – 136

R – 169, ra – 146, re – 126, ri – 148,

S – 167, sa – 171, se – 177, si – 134, so – 168, su – 174

T – 176, ti – 145, to – 157

U – 138

V – 164, ve – 132, vi – 161, vo – 175

W, X, Y – ?

Z – 166

В то время в Европе получили широкое распространение шифры, называемые *номенклаторами,* объединявшие в себе простую замену и код. В простейших номенклаторах код состоял из нескольких десятков слов или фраз с двухбуквенными кодовыми обозначениями. Со временем списки заменяемых слов в номенклаторах увеличились до двух или трех тысяч эквивалентов слогов и слов. В царской России XVIII в. закодированное открытое сообщение шифровалось далее простой заменой.

Кстати, несколько слов о русской криптографии. Уже с XIV в. в Новгороде существовала техника тайного письма. Использовались в основном шифры простой замены. Благодаря торговым связям Новгорода с Германией в России становятся известными многие западные разработки, в том числе новые системы шифрования. Учреждение постоянной почтовой связи России с Европой дало возможность развития шифрованной переписке. Благодаря

привлечению Петром I для разработки проектов развития образования и государственного устройства России знаменитого Готфрида Вильгельма Лейбница, который известен и как криптограф, в Петербурге появилась цифирная палата, задачами которой было развитие и использование систем шифрования.

Когда Россиньоль начинал свою карьеру, в номенклаторах как элементы открытого текста, так и элементы кода располагались в алфавитном порядке (или в алфавитном и числовом порядке, если код был цифровой). Россиньоль заметил, что такой "параллелизм" открытого текста и кода облегчал восстановление открытого текста. Если, например, он устанавливал, что в английской депеше 137 заменяет FOR, а 168 – IN, то он уже знал, что 21 не может заменять TO, так как цифровые кодовые обозначения для слов, начинающихся с T, должны быть больше, нежели для слов, начинающихся с I. Обнаружив такую слабость, Россиньоль перемешивал кодовые элементы по отношению к открытому тексту. На одном листе он располагал элементы открытого текста в алфавитном порядке, а кодовые элементы – вразброс, на другом листе для облегчения расшифрования кодовые элементы стояли в алфавитном порядке, тогда как их открытые эквиваленты были разбросаны. Это явилось значительным усовершенствованием подобных шифрсистем. Однако составление неалфавитных номенклаторов обходилось очень дорого, и, таким образом, по соображениям экономии и в ущерб надежности многие номенклаторы регрессировали к упрощенному алфавитному типу.

В Англии тоже был свой "черный кабинет". В его работе в XVII в. заметное место занимал Джон Валлис, известный как крупнейший английский математик до Исаака Ньютона. Работы по вскрытию шифров для парламента привели к назначению Валлиса в 1649 г. в Оксфорд профессором геометрии в возрасте 32 лет. В своем труде "Арифметика бесконечного" он сделал выводы, которые послужили Ньютону стартовой площадкой для разработки интегрального исчисления. Валлис ввел знак ∞ для бесконечности и первый путем интерполяции вычислил число π. Кстати, само это обозначение также принадлежит ему.

В Германии начальником первого дешифровального отделения был граф Гронсфельд, создавший один из вариантов усовершенствования шифра Виженера. Он взял числовой, легко запоминаемый лозунг. Вместо таблицы Виженера использовался один несмешанный алфавит. При шифровании знаки открытого текста выписывались под цифрами лозунга. Очередная буква открытого текста заменялась буквой алфавита, отстоящей от нее вправо на количество букв, равное соответствующей цифре лозунга.

Шифр Гронсфельда

Открытый текст: GERMANY

Лозунг: 1 3 5 7 9

Алфавит: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Для удобства выпишем алфавит с порядковыми номерами букв:

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	3	4	5	6	7	8	9	10	11	12	13

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

и лозунг над текстом:

1	3	5	7	9	1	3
G	E	R	M	A	N	Y

Теперь легко получить шифрованный текст: HHWTJOB

Любопытен опыт использования криптографии при составлении *астрономических анаграмм.* Одно из таких применений связано с открытием колец Сатурна.

В годы жизни Галилео Галилея существовал обычай закреплять за собой право на первенство в каком-либо открытии своеобразным способом. Напав на открытие, которое нуждается в дальнейшем подтверждении, ученый из опасения, чтобы его не опередили другие, прибегал к помощи *анаграммы* (перестановке букв); он кратко объявлял о сущности своего открытия в форме анаграммы, истинный смысл которой был известен лишь ему одному. Это давало ученому возможность не спеша проверить свое открытие, а в случае появления другого претендента – доказать свое первенство. Когда же он окончательно убеждался в правильности первоначальной догадки, он раскрывал секрет анаграммы. Заметив в свою несовершенную подзорную трубу, что Сатурн имеет по бокам какие-то придатки, Галилей поспешил сделать заявку на это открытие и опубликовал следующий набор букв:

SMAISMRMIELMEPOETALEUMIBUVNEUGTTAVIRAS

Задача восстановления открытого текста (без какой-либо дополнительной информации об использованном преобразовании) требует перебора $\frac{39!}{3!5!4!4!2!2!5!3!3!2!2!}$ возможных перестановок букв криптограммы (это – число перестановок с повторениями). Приведенное число имеет в своей записи примерно 35 цифр.

Современник итальянского ученого Иоганн Кеплер с присущим ему беспримерным терпением затратил немало труда на то, чтобы проникнуть в сокровенный смысл заявки Галилея, и ему казалось, что он добился этого, когда из опубликованных букв (опустив две из них) составил такую латинскую фразу:

SALVE, UMBISTINEUM GEMINAUM MARTIA PROLES

(Привет вам, близнецы, Марса порождение)

Кеплер был убежден, что Галилей открыл те два спутника Марса, существование которых подозревал он сам (они в действительности и были открыты, но спустя два с половиной века). Однако остроумие Кеплера на этот раз не привело к цели. Когда Галилей раскрыл, наконец, секрет своей заявки, оказалось, что фраза (если двумя буквами пренебречь) такова:

ALTISSIMUM PLANETAM TERGEMINUM OBSERVAVI

(Высочайшую планету тройною наблюдал)

Из-за слабости своей трубы Галилей не мог понять истинного значения этого "тройного" образа Сатурна, а когда спустя несколько лет боковые придатки планеты совершенно исчезли", Галилей решил, что ошибся, и никаких придатков У Сатурна нет. Открыть кольца Сатурна удалось только через полвека Гюйгенсу. Подобно Галилею, он не сразу опубликовал свое открытие, а скрыл догадку под тайнописью:

AAAAAAACCCCCDEEEEGHIIIIIIILLLLMMNNNNNNNNN

OOOOPPQRRSTTTTTUUUUU

Спустя три года, убедившись в правильности своей догадки, Гюйгенс обнародовал смысл заявки:

Annulo cingitur, tenui, piano, nusquam cohaerente,

ad eclipticam inclinato

(Кольцом окружен тонким, плоским, нигде

не прикасающимся к эклиптике)

В целом можно сказать, что XVII и XVIII века не дали новых идей в криптографии. Эра "черных кабинетов" закончилась в 40-х годах XIX в. в период революционного подъема.

Много новых идей в криптографии принес XIX в. Изобретение в середине XIX в. телеграфа и других технических видов связи дало новый толчок развитию криптографии. Информация передавалась в виде токовых и бестоковых посылок, то есть представлялась в двоичном виде. Поэтому возникла проблема "рационального" представления информации, которая решалась с помощью кодов. Коды позволяли передать длинное слово или целую фразу двумя-тремя знаками. Появилась потребность в высокоскоростных способах шифрования и в корректирующих кодах, необходимых в связи с неизбежными ошибками при передаче сообщений.

Однако еще до изобретения телеграфа появился ряд интересных шифровальных устройств. Приблизительно в 1800 г. была создана одна шифровальная система, занимающая особое место в истории криптографии. Речь идет о "дисковом шифре" Т.Джефферсона – первого государственного секретаря США, ставшего позже третьим президентом.

Дисковый шифратор Т.Джефферсона состоял из 25 – 36 деревянных дисков одинакового размера, насаженных на общую ось (рисунок 1.2).

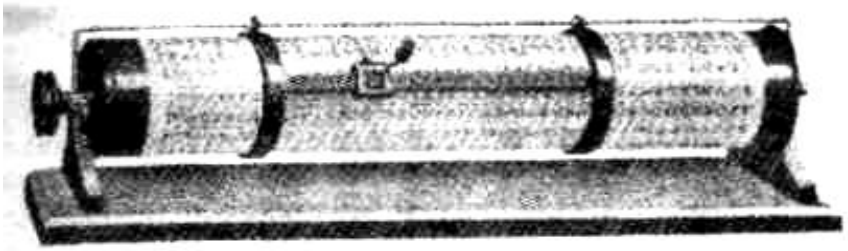


Рис.1.2. Дисковый шифратор Т. Джефферсона

На одном конце оси имелась неподвижная головка, на другом – резьба и гайка, с помощью которой все диски фиксировались в любом нужном угловом положении. Имелась также прямолинейная рейка, способная вращаться на оси и позволяющая выделить строку букв на дисках, параллельную оси. На боковой поверхности каждого диска, разделенной на 26 равных частей, наносились буквы смешанных английских алфавитов. Для зашифрования части сообщения (длина которой равнялась числу дисков на оси) под рейку, находящуюся в фиксированном угловом положении, подводилась первая буква сообщения, найденная на первом диске, затем – вторая буква сообщения, найденная на втором диске, и т.д., так чтобы все подобранные буквы оказались в одной строке. Положение дисков фиксировалось гайкой, после чего рейка подводилась под любую другую строку цилиндра, буквы которой составляли шифрованный текст. При расшифровании буквы шифрованного текста, набранные на последовательных дисках, подводились аналогичным образом под рейку, положение дисков фиксировалось гайкой, после чего с помощью рейки просматривались образовавшиеся строки цилиндра, среди которых несложно было найти открытое сообщение.

Кажущаяся некорректность, связанная с возможностью неоднозначности расшифрования, устраняется достаточно большим числом используемых дисков. Это замечание относится, конечно, лишь к осмысленным текстам. При зашифровании неосмысленных текстов требовалась дополнительная информация о величине сдвига рейки, без чего однозначное расшифрование невозможно.

Такая шифрсистема имеет огромное количество ключевых элементов. К ним относятся: расположение букв алфавита на дисках, расстановка дисков на оси, выбор набора дисков из имеющегося запаса. Дисковый шифр можно отнести по типу к многоалфавитной замене. Его особенностью является поблочный характер зашифрования, при котором каждый участок

текста (блок) шифруется независимо от других. Позже такие шифры стали называться *блочными шифрами.*

Вместо того чтобы (пользуясь служебным положением) внедрить свое замечательное изобретение в практику, Джефферсон, по-видимому, отложил его в архив и предал забвению. Шифр был обнаружен в его бумагах в библиотеке конгресса лишь в 1922 г., по иронии судьбы именно в том году, когда в армии США начали применять почти аналогичную систему, изобретенную независимо от Джефферсона.

В 1817г. другой американец Десиус Уодсворт сконструировал шифровальное устройство, которое также внесло новый принцип в криптографию. Его нововведение состояло в том, что он сделал алфавиты открытого и шифрованного текстов различных длин. Устройство, с помощью которого он это осуществил, представляло собой диск, на котором были расположены два подвижных кольца с алфавитами. Внешний алфавит состоял из 26 букв и 7 цифр (от 2 до 8). Внутренний алфавит состоял лишь из 26 букв. Диск имел подобие неподвижной часовой стрелки, в двух прорезях которой появлялись расположенные друг под другом буквы алфавитов. На внутреннем кольце указывалась буква открытого текста, на внешнем кольце – соответствующая буква шифртекста. Оба кольца могли вращаться и были связаны друг с другом с помощью двух шестерен, одна из которых имела 33 зубца, а другая – 26. Буквы и цифры внешнего кольца были съемными и могли быть собраны в любом порядке. Перед зашифрованием корреспонденты договаривались относительно взаимного начального положения обоих колец. Для установки дисков в такое положение шестерни можно было разъединить. Проследим на примере слова "введение" процесс зашифрования.

Сначала внутреннее кольцо поворачивалось до тех пор, пока в прорези стрелки не показывалась буква "в". Стоящая в другой прорези буква внешнего кольца записывалась в качестве первой буквы шифртекста. Затем внутреннее кольцо вращалось до тех пор, пока буква "в" вновь не показывалась в прорези. Это вращение посредством шестерен передавалось на внешнее кольцо, но из-за различия в числе букв алфавитов оно совершало лишь 26/33 полного оборота, в то время как

внутреннее кольцо совершало полный оборот. Значит, второй знак шифртекста располагался во внешнем алфавите на расстоянии семи мест вперед от первого знака, несмотря на то, что оба знака представляли одну и ту же букву открытого текста. Если этот процесс зашифрования осуществлять дальше, то эквиваленты шифртекста для буквы "в" начнут повторяться лишь после того, как будут использованы все 33 буквы и цифры внешнего алфавита. Это объясняется тем, что числа 26 и 33 не имеют общих делителей, благодаря которым такое повторение могло бы произойти раньше. Следующие буквы открытого текста шифровались аналогично.

Такая шифрсистема реализует периодическую многоалфавитную замену. Различие чисел букв алфавитов открытого и шифрованного текстов приводит к существенным отличиям этой системы от предыдущих многоалфавитных систем. Так, в устройстве Уодсворда используется 33 шифралфавита, а не 24 или 26, как в системах Тритемия или Виженера. Важнее то, что эти алфавиты используются не непосредственно один за Другим, а в произвольном порядке, который зависит от букв открытого текста. Этот произвольный порядок служит гораздо более надежной защитой шифра, чем правильная последовательность использования алфавитов, как в системе Тритемия.

Идея Уодсворда была незаслуженно забыта. Славу открытия приписывают английскому ученому Чарлзу Уитстону, который значительно позже и независимо изобрел свое устройство на том же принципе. Основное отличие заключалось в том, что в устройстве Уитстона алфавиты были неподвижными, но зато имелась пара подвижных стрелок, соединенных шестеренками.

Уитстон более известен как ученый, предложивший идею электрического телеграфа, изобретатель концертино, автор первых стереоскопических рисунков. Он высказал гипотезу о создании говорящих машин, разработал метод точного измерения электрического сопротивления, который называется "мостик Уитстона".

Впервые свое устройство Уитстон продемонстрировал на Всемирной выставке в Париже в 1876 г. На внешнем кольце находился алфавит открытого текста, состоящий из 27 элементов: 26 букв, расположенных в обычном порядке, и знака пробела между словами. Внутренний алфавит состоял из 26 букв, расположенных в произвольном порядке.

Уитстон изобрел шифр, который позже стали называть *шифром Плейфера.* Дело в том, что Лион Плейфер, заместитель председателя Палаты общин, министр почт, председатель Британской ассоциации развития науки, был другом Уитстона, был похож на него, так что их часто путали. В 1854 г. Плейфер продемонстрировал систему шифрования, которую он назвал "недавно открытый симметричный шифр Уитстона". Это был первый из известных биграммых буквенных шифров (напомним, что биграммный шифр Порта был значковым). То обстоятельство, что Плейфер популяризировал изобретение Уитстона, сохранило его имя в названии шифра. Этот шифр использовался англичанами в период I мировой войны.

Во второй половине XIX в. появился весьма устойчивый способ усложнения числовых кодов – *гаммирование.* Он заключался в перешифровании закодированного сообщения с помощью некоторого ключевого числа, которое и называлось *гаммой.* Шифрование с помощью гаммы состояло в сложении всех кодированных групп сообщения с одним и тем же ключевым числом. Эту операцию стали называть *"наложением гаммы"* Например,

результатом наложения гаммы 6413 на кодированный текст 3425 7102 8139 являлась числовая последовательность 9838 3515 4552:

3425 7102 8139

+ 6413 6413 6413

9838 3515 4552

Единицы переноса, появляющиеся при сложении между кодовыми группами, опускались. *"Снятие гаммы"* являлось обратной операцией:

9838 3515 4552

– 6413 6413 6413

3425 7102 8139

В 1888 г. француз маркиз де Виари в одной из своих научных статей, посвященных криптографии, обозначил греческой буквой Х любую букву шифрованного текста, греческой буквой Г любую букву гаммы и строчной буквой *с* любую букву открытого текста. Он, по сути, *доказал,* что алгебраическая формула

$$X = (c + Г) \bmod 26$$

воспроизводит зашифрование по Виженеру при замене букв алфавита числами согласно следующей таблице:

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Тем самым была заложена алгебраическая основа для исследования шифров замены типа шифра Виженера. Используя уравнение шифрования, можно было отказаться от громоздкой таблицы Виженера.

Позже лозунговая гамма стала произвольной последовательностью, а шифр с уравнением шифрования (1) стал называться *шифром гаммирования*.

Еще одним известным криптографом того времени был голландец Керкгоффс. Его полным именем было Жан-Вильгельм-Губерт-Виктор-Франсуа-Александр-Огюст Керкгоффс ван Ньювенгоф. Разносторонний ученый, преподававший 6 иностранных языков, историю и математику, он в возрасте 47 лет написал книгу "Военная криптография" В ней сформулированы 6 конкретных требований к шифрам, два из которых относятся к стойкости шифрования, а остальные – к эксплуатационным качествам. Одно из них ("компрометация системы не должна причинять неудобств корреспондентам") стало называться *"правилом Керкгоффса"*. Суть этого правила состоит в том, что *стойкость* (или надежность) шифра

определяется лишь секретностью ключа. Другими словами, оценка качества шифра (на основе некоторого шифрованного текста) должна проводиться при условии, что о данном шифре известно все, кроме использованного ключа.

XX век "прославился" двумя мировыми войнами. Эти войны оставили свой отпечаток на всех процессах, происходивших в человеческом обществе. Они не могли не сказаться и на развитии криптографии.

В период первой мировой войны в качестве полевых шифров широко использовались ручные шифры, в первую очередьшифры перестановки с различными усложнениями. Это были *вертикальные перестановки*, усложненные перекодировкой исходного алфавита, а также *двойные вертикальные перестановки.*

Первая мировая война явилась поворотным пунктом в истории криптографии: если до войны криптография представляла собой достаточно узкую область, то после войны она стала широким полем деятельности. Причина этого состояла в необычайном росте объема шифрпереписки, передаваемой по различным каналам связи. Криптоанализ стал важнейшим элементом разведки.

Прогресс этой области криптографии характеризовался и изменениями в самом криптоанализе. Эта наука переросла методы индивидуальной работы криптоаналитика над криптограммой. Системы секретной связи перестали быть настолько малочисленными и однородными, что один специалист мог овладеть всеми специализациями. Характер используемых шифров потребовал для их вскрытия скрупулезного анализа переписки, поиска ситуаций, благоприятствующих успешному криптоанализу, знания соответствующей обстановки. Кроме того, криптоанализ обогатился большим опытом использования в годы войны ошибок неопытных или ленивых шифровальщиков. Еще Ф. Бэкон писал, что "в результате неловкости и неискусности тех рук, через которые проходят величайшие секреты, эти секреты во многих случаях оказывались обеспеченными слабейшими шифрами" [Kah67| Этот печальный опыт привел к необходимости введения строгой дисциплины среди шифровальщиков.

Несмотря на указанные последствия, первая мировая война не породила никаких новых научных идей в криптографии. Наоборот, полностью исчерпали свои возможности ручное шифрование, с одной стороны, и техническая сторона криптоанализа, состоявшая в подсчете частот встречаемости знаков, с другой.

В тот период проявились таланты целого ряда ставших впоследствии известными криптографов. В их числе был ГО. Ярдли, который вскоре после объявления США войны в 1917 г. *убедил* военное министерство в необходимости создания криптографической службы. В 27 лет он был назначен начальником криптографического отдела (MI-8) разведки военного

министерства. При отделе было создано учебное отделение по подготовке криптоаналитиков для американской армии. Отдел MI-8 добился больших успехов в дешифровании дипломатической переписки многих развитых стран. В 1919 г. отдел был преобразован в "черный кабинет" с совместным финансированием от военного министерства и госдепартамента в объеме 100 тыс. долларов в год Одной из главных задач "черного кабинета" было раскрытие японских кодов, некоторые из которых содержали до 25 тысяч кодовых величин. В период с 1917 г. по 1929 г. специалистам "черного кабинета" удалось дешифровать более 45 тысяч криптограмм различных стран, в том числе и Япония.

Ярдли, желая упрочить успехи, подготовил докладную записку Президенту США о мерах по укреплению своей службы. Однако ставший в то время Государственным секретарем Г. Стимсон был шокирован, узнав о существовании "черного кабинета", и полностью осудил его деятельность Ему принадлежит знаменитая фраза: "Джентльмены не читают писем друг друга". Финансирование "черного кабинета" было прекращено, и Ярдли лишился работы Он написал книгу "Американский черный кабинет", в которой рассказал о многих успехах по дешифрованию. Книга была издана большими тиражами в ряде стран и произвела эффект разорвавшейся бомбы. Позже он написал книгу "Японские дипломатические секреты", в которой приводились многие японские телеграммы. Рукопись этой книги была конфискована по решению суда. Последние годы жизни Ярдли не занимался криптографией. Он умер в 1958 г. и был похоронен с воинскими почестями на Арлингтонском национальном кладбище. В некрологе он был назван "отцом американской криптографии".

Значительный успех в криптографии связан с еще одним американцем – Г. Вернамом. В 1917 г. он, будучи сотрудником телеграфной компании, предложил идею автоматического шифрования телеграфных сообщений. Речь шла о своеобразном наложении гаммы на знаки алфавита, представленные в соответствии с телетайпным кодом Бодо пятизначными "импульсными комбинациями". Например, буква *а* представлялась комбинацией (+ + – – –), а комбинация (+ + – + +) представляла символ перехода от букв к цифрам. На бумажной ленте, используемой при работе телетайпа, знаку "+" отвечало наличие отверстая, а знаку "–" – его отсутствие. При считывании с ленты металлические щупы проходили через отверстия, замыкали электрическую цепь и тем самым посылали в линию импульс тока.

Вернам предложил электромеханически покоординатно складывать "импульсы" знаков открытого текста с импульсами" гаммы, предварительно нанесенными на ленту. Сложение проводилось "по модулю 2 ". Имеется в виду, что если "+" отождествить с 1, а "–" с 0, то сложение определяется двоичной арифметикой:

+	0	1
0	0	1

1	1	0

Например, наложение на знак открытого текста (11001) знака гаммы (01111) давало знак шифртекста (10110). При расшифровании нужно было произвести ту же операцию со знаком шифртекста: (10110) ⊕ (01111) = (11001).

Вернам сконструировал и устройство для такого сложения. Замечательно то, что процесс шифрования оказывался полностью автоматизированным, в предложенной схеме исключался шифровальщик. Кроме того, оказывались слитыми воедино процессы шифрования-расшифрования и передачи по каналу связи. Тем самым наряду с традиционной схемой *предварительного шифрования,* когда по каналу передается предварительно зашифрованное сообщение, положено начало *линейному шифрованию.*

В 1918 г. два комплекта соответствующей аппаратуры были изготовлены и испытаны. Испытания дали положительные результаты. Единственное неудовлетворение специалистов-криптографов было связано с гаммой. Дело в том, что первоначально гамма была нанесена на ленту, склеенную в кольцо. Несмотря на то, что знаки гаммы на ленте выбирались случайно, при зашифровании длинных сообщений гамма регулярно повторялась. Этот недостаток так же отчетливо осознавался, как и для шифра Внженера. Уже тогда хорошо понимали, что повторное использование гаммы недопустимо даже в пределах одного сообщения. Хотя сам Вернам не был математиком, он, может и неосознанно, предлагал однократное использование гаммы. Попытки удлинить гамму приводили к неудобствам в работе с длинным кольцом. Тогда был предложен вариант с двумя лентами, одна из которых шифровала другую, в результате чего получалась гамма, имеющая длину периода, равную произведению длин исходных периодов.

Несмотря на то, что *шифр Вернама* обладал целым рядом достоинств, он не получил широкого распространения. Трудности, связанные с изготовлением, рассылкой и учетом использованной гаммы, особенно в условиях военной связи, при передаче больших объемов сообщений, стали непреодолимыми. Вспомнили о шифре Вернама лишь накануне II мировой войны.

Почти половина XX в. была связана с использованием *колесных шифраторов.* Различные их конструкции были запатентованы примерно в одно и то же время (в период 1917 – 1919 гг.) в разных странах: американцем Э. Х. Хеберном, голландцем Х. Ф. Кохом, немцем А. Шербиусом и шведом А. Г. Даммом.

Чертежи своей схемы на основе *шифрующего диска* Хеберн представил в 1917 г., и уже в следующем году был построен первый дисковый аппарат, получивший одобрение ВМС США. В 1921 г. Хеберн основал первую в США компанию по производству шифрмашин, которую через десять лет ждал бесславный конец, связанный с финансовыми трудностями.

Что представлял собой шифрующий диск? Корпус диска (имевшего размеры хоккейной шайбы) состоял из изоляционного материала, например твердой резины. По окружностям каждой из его сторон были вмонтированы на равном расстоянии друг от друга 26 электрических контактов (см. рис. 1.3.). Каждый контакт был соединен внутри корпуса с некоторым контактом на другой стороне. Контакты на *входной стороне* представляли буквы открытого текста, контакты на *выходной стороне* – буквы шифртекста.

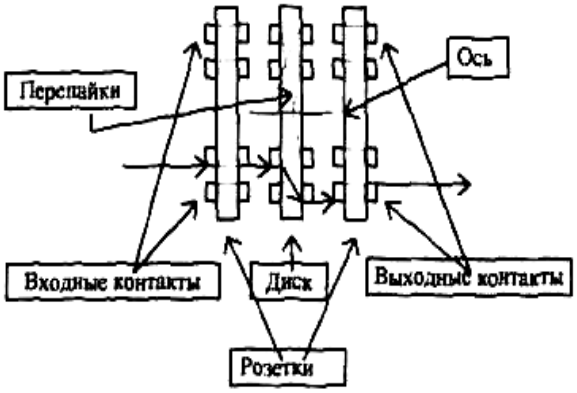


Рис. 1.3. Шифрующий диск

Диск устанавливался на оси между двумя неподвижными пластинами *(розетками),* каждая из которых также была изготовлена из изолятора и имела 26 контактов, соответствующих расположению контактов на диске. Контакты входной розетки соединялись с клавиатурой пишущей машинки, печатающей буквы открытого текста. Контакты выходной розетки соединялись с выходным устройством, указывающим буквы шифртекста, например, с помощью лампочек. При фиксированном угловом положении диска электрические цепи, соединяющие входные и выходные контакты, реализовывали одно-алфавитную замену. При повороте же диска (на углы $2\pi k/26$) схема реализовывала многоалфавитную замену (с 26 простыми заменами).

Рядом с одним диском можно было установить и другие диски. Тем самым схема токопрохождения удлинялась и число возможных простых замен, реализуемых многодисковой схемой значительно возрастало. При движении k дисков по простейшей схеме одометра получался период, равный 26^k, который можно было сделать астрономическим числом.

Подобные шифрмашины обслуживали значительную часть линий связи высшего командования ВМС США, начиная с 20-х годов.

Х. Ф. Кох предлагал конструкцию шифрующего диска, в котором роль электричества выполняла пневматика. Речь идет о каналах, соединяющих входные и выходные контакты,

по которым может проходить поток воздуха, водная или масляная струя и т.п. Любопытно, что подобные дисковые системы на основе пневматики были реально изготовлены и использовались на практике.

Принцип шифрующего диска использовали и шифрмашины, разработанные А. Шербиусом. Самой знаменитой из них была *"Энигма"*, которая в двух отношениях отличалась от других дисковых машин. Во-первых, после блока дисков была расположена неподвижная *обратимая розетка,* контакты которой были попарно соединены друг с другом. Импульс тока, приходивший на этот контакт, заворачивался и вновь проходил через блок дисков в противоположном направлении. Это давало двойное шифрование каждой буквы. Другая особенность "Энигмы" заключалась в неравномерном движении дисков, которое управлялось зубчатыми колесами.

В 1923 г. "Энигма" выставлялась на конгрессе международного почтового союза, однако это не способствовало ее коммерческому успеху: она не раскупалась. За десять лет фирма Шербиуса, производившая "Энигму", не получила прибыли и в 1934 г. была ликвидирована и передала свои активы другой фирме. После прихода к власти в Германии Гитлера началось серьезное перевооружение армии, и немецкие специалисты сочли "Энигму" достаточно удобной и надежной шифрмашиной. В довоенный период и во время II мировой войны "Энигма" широко использовалась в германской армии, ВМС и ВВС. Она была портативной (размером с пишущую машинку), работала от батареи, имела деревянный футляр. Ее серьезный недостаток состоял в том, что она не печатала шифртекст (а имела лишь загорающиеся лампочки, отвечающие буквам), и для быстрой работы требовалось три или даже четыре человека – для чтения и набора на клавиатуре текста сообщения, диктовки высвечивающихся букв шифртекста и их записи.

С "Энигмой" теснейшим образом связан ход многих событий периода II мировой войны. Дело в том, что она являлась источником ценнейших сведений для английских спецслужб, читавших переписку "Энигмы" (в рамках *операции "Ультра"*). Эта информация стоила так дорого, что У. Черчиль пожертвовал городом Ковентри, когда ему стал известен план германской бомбардировки этого английского города. С "Энигмой" связано также появление первой в истории вычислительной машины, сконструированной в 1942 г. для перебора ключевых элементов группой специалистов-криптографов под руководством известного математика А. Тьюринга.

Еще один патент на дисковую машину был выдан А. Г. Дамму в 1919 г. Устройство этой машины было настолько сложным, что никогда не было реализовано. Но его автор основал компанию по производству шифрмашин. которая впоследствии стала прибыльной. Среди вкладчиков капитала были Э. Нобель, племянник знаменитого А. Нобеля, и Ц. Хагелин,

управляющий нефтедобывающей компанией братьев Нобелей в России и некоторое время бывший генеральным консулом Швеции в Санкт-Петербурге. До 1927 г. эта компания не имела больших успехов. Их появление было связано с именем сына Ц. Хагелина – Б. Хагелина, родившегося на Кавказе, проучившегося несколько лет в Петербургском университете и получившего позже диплом инженера-механика в Швеции.

В 1925 г. Б. Хагелину удалось модернизировать одну из машин Дамма, снабдив ее клавиатурой и индикаторными лампочками, как у "Энигмы". Это была также колесная машина, работающая, однако, по иному, чем дисковые машины, принципу. Это была машина В-21. Ее работа была основана на матричном *коммутаторе,* в котором электрически изменялось соединение строк и столбцов для преобразования буквы открытого текста в букву шифртекста. Эти изменения определялись группой ключевых колес, каждое из которых имело по ободу выдвинутые или вдвинутые *штифты*. Колеса имели различные числа штифтов, так что период многоалфавитного шифра, реализуемого машиной, был равен произведению чисел штифтов на всех колесах. В 1926 г. Б. Хагелин предложил. В-21 шведской армии, которая сделала на нее большой заказ.

В 1927 г. Б. Хагелин возглавил фирму, выкупленную семьей Хагелин. Свою следующую машину В-211 он снабдил печатающим устройством, работавшим со скоростью около 200 знаков в минуту. Она была самой портативной печатающей шифрмашиной в 1934 г.

В том же году французский генштаб заказал Б. Хагелину карманную печатающую машину, которая могла бы обслуживаться одним человеком. Через некоторое время такая машина была изготовлена. Она реализовывала шифр гаммирования, причем для выработки гаммы была использована идея суммирующего устройства, состоящего из комбинационных линеек, расположенных в цилиндрическом барабане. На линейках рядами были расположены так называемые *рейтеры*. При повороте барабана на 360° рейтеры, вступая во взаимодействие с другими элементами схемы, могли выдвигать некоторые линейки влево, причем число выдвинутых линеек и определяло значение знака гаммы (от 0 до 25) в данный такт шифрования. Во взаимодействие с рейтерами вступали штифты, расположенные на колесах блока дисков, составляющего вторую основную часть машины. Размеры и схема движения дисков обеспечивали период, приблизительно равный $1{,}01 \cdot 10^8$. Как расположение рейтеров, так и расположение штифтов могло легко меняться, они являлись ключевыми элементами. Это была машина С-36, ставшая впоследствии знаменитой. По размерам она была меньше телефонного аппарата, весила вместе с футляром около двух с половиной килограммов. Французы сразу же сделали заказ на 5000 машин. Позднее машина была существенно усовершенствована, ею заинтересовались в США. В 1939 г. она была взята на вооружение армии США. Под военным наименованием М-209 она использовалась в качестве

полевого шифра на протяжении всей II мировой войны. Всего было произведено около 140000 таких машин. Позже фирма Хагелин стала производить широко известные машины С-48, С-52, Т-55 и многие другие. Среди заметных фигур в криптографии первой половины XX в. выделяется У. Фридман, получивший серьезные теоретические результаты в криптоанализе и ставший известным благодаря своим заслугам по вскрытию военных шифров Японии и Германии.

У. Фридман родился в 1891 г. в Кишиневе, в семье переводчика, работавшего в русском почтовом ведомстве. В 1892 г. его семья эмигрировала в США, где отец стал заниматься швейными машинами. У.Фридман в 1914 г. окончил Корнельский университет по специальности генетика. В городе Итака, где проживала семья Фридмана, крупный бизнесмен Д. Фабиан имел собственные лаборатории по акустике, генетике и криптографии. Любопытно, что криптографией Д. Фабиан увлекся, пытаясь доказать, что автором пьес У. Шекспира являлся Ф. Бэкон.

В 1915 г. Д. Фабиан нанял на работу в свое поместье Ривербэнк специалиста по генетике. Им стал У. Фридман. Вскоре он увлекся криптографией и проявил себя в этом деле. Через некоторое время У. Фридман уже возглавлял в Ривербэнкских лабораториях два отдела – генетики и шифров.

Помимо криптоаналитической работы У. Фридман занимался преподаванием в классе, состоявшем из армейских офицеров, присланных в Ривербэнк для изучения криптографии. До 1918 г. им был подготовлен цикл из семи лекций, восьмую он написал после возвращения со службы в качестве дешифровальщика в американских экспедиционных силах (шла I мировая война). Известные все вместе как *Ривербэнкские публикации,* эти работы являются серьезным вкладом в теоретическую криптографию.

Наибольший интерес с точки зрения современной криптографии представляют лекции "Методы раскрытия шифров с длинной связной гаммой" и "Индекс совпадения и его применения в криптографии" ([Fri20]). В первой из них предлагается бесключевой метод чтения при использовании неравновероятной гаммы. Во второй излагается так называемый *k-тест,* позволяющий выяснить, можно ли подписать друг под другом две (или более) криптограммы (или отрезки криптограмм) так, чтобы буквы в каждой колонке оказались бы зашифрованы одинаковыми знаками гаммы.

Поступив в 1921 г. на службу в войска связи, У. Фридман успешно применял свои методы для вскрытия машинных шифров. Когда была создана служба радиоразведки, У. Фридман стал ее главой и продолжил свои разработки, самой значимой из которых было вскрытие японской *пурпурной шифрмашины*. В 1929 г. он стал широко известен как один из ведущих криптографов мира, когда "Британская энциклопедия" поместила его статью "О

кодах и шифрах". С основными результатами У. Фридмана можно познакомиться в четырехтомнике "Военная криптография".

Выдающиеся результаты в применении математических методов в криптографии принадлежат Клоду Шеннону. К. Шеннон получил образование по электронике и математике в Мичиганском университете, где и начал проявлять интерес к теории связи и теории шифров. В 1940 г. он получил степень доктора по математике, в течение года обучался в Принстонском институте усовершенствования, после чего был принят на службу в лабораторию компании "Bell Telephone".

К 1944 г. К. Шеннон завершил разработку теории секретной связи. В 1945 г. им был подготовлен секретный доклад "Математическая теория криптографии", который был рассекречен в 1949 г.

В данной работе излагается теория так называемых *секретных систем,* служащих фактически математической моделью шифров. Помимо основных алгебраических (или функциональных) свойств шифров, постулируемых в модели, множества сообщений и ключей наделяются соответствующими априорными вероятностными свойствами, что позволяет формализовать многие постановки задач синтеза и анализа шифров. Так, и сегодня при разработке новых классов шифров широко используется принцип Шеннона *рассеивания и перемешивания,* состоящий в использовании при шифровании многих итераций "рассеивающих" и "перемешивающих" преобразований.

Разработанные К. Шенноном концепции *теоретической* и *практической секретности* (или стойкости) позволяют количественно оценивать криптографические качества шифров и пытаться строить в некотором смысле *идеальные* или *совершенные шифры.* Моделируется также и язык открытых сообщений. А именно, предлагается рассматривать язык как *вероятностный процесс,* который создает дискретную последовательность символов в соответствии с некоторой вероятностной схемой.

Центральной в работах К.Шеннона является концепция *избыточной информации,* содержащейся в текстовых сообщениях. Избыточность означает, что в сообщении содержится больше символов, чем в действительности требуется для передачи содержащейся в нем информации. Например, всего лишь десять английских слов – the, of, and, to, a, in, that, it, is, i – составляют более 25% любого (английского) текста. Легко понять, что их можно изъять из текста без потери информации, так как их легко восстановить по смыслу (или по контексту). Фактически К. Шеннон показал, что успех криптоанализа определяется тем, насколько избыточность, имеющаяся в сообщении, "переносится" в шифрованный текст. Если шифрование "стирает" избыточность, то восстановить текст сообщения по криптограмме становится принципиально невозможно.

Задачу дешифрования К Шеннон рассматривает как задачу вычисления апостериорных знаний противника о шифре после перехвата криптограммы. Дело в том, что вероятности сообщений и ключей составляют априорные знания противника, которыми он располагает в соответствии с правилом Керкгоффса. После перехвата криптограммы он может (по крайней мере, в принципе, поскольку множества сообщений и ключей *конечны*) вычислить апостериорные вероятности возможных ключей и сообщений, которые могли быть использованы при составлении данной криптограммы. Вот эти вероятности и составляют апостериорные знания противника. С этой точки зрения показателен следующий пример.

Пусть для зашифрования нормативного английского языка применяется шифр простой замены, в котором каждый из 26! ключей может быть выбран с равной вероятностью. Пусть противник знает об источнике сообщений лишь то, что он создает английский текст. Тогда априорными вероятностями различных сообщений из N букв являются их относительные частоты в нормативном тексте. Если же противник перехватил криптограмму из N букв, то он может вычислить условные вероятности открытых текстов и ключей, которые могут создать такую криптограмму. Если N достаточно велико, скажем $N = 50$, то *обычно* имеется единственное сообщение (и единственный ключ) с условной вероятностью, близкой к единице (это – само сообщение, подвергнутое шифрованию), в то время как все другие сообщения имеют суммарную вероятность, близкую к нулю. Таким образом, имеется, по существу, единственное "решение" такой криптограммы. Для меньших значений N, скажем $N = 10$, *обычно* найдется несколько пар сообщений и ключей, вероятности которых сравнимы друг с другом, то есть нет ни одного сообщения (и ключа) с вероятностью, близкой к единице. В этом случае "решение" криптограммы неоднозначно.

Понятие *совершенной секретности* К. Шеннон определяет требованием, чтобы апостериорные знания противника в точности совпадали бы с априорными знаниями. Он приводит пример *совершенного шифра,* которым является шифр Вернама (со случайной равновероятной гаммой). Следует подчеркнуть, что все рассуждения о стойкости шифров К. Шеннон проводит лишь для одной постановки задачи криптоанализа: когда противник располагает лишь *одной криптограммой* и требуется найти текст сообщения. Для других постановок задач требуются отдельные исследования.

Теоретической мерой секретности (или стойкости) по К. Шеннону является энтропийная характеристика – *неопределенность шифра по открытому сообщению,* которая измеряет (в статистическом смысле), насколько "близка" *средняя* криптограмма из N букв к единственному "решению". Он выводит формулу для приближенного вычисления минимального N, при котором находится единственное "решение". Такая величина получила название *расстояния единственности*. Формула для расстояния единственности связывает между

собой неопределенность шифра по открытому тексту и избыточность текста. Чем большим оказывается расстояние единственности, тем более шифр приближается к совершенному шифру, для которого формально расстояние единственности равно ∞.

Наконец, К. Шеннон вводит понятие *рабочей характеристики* шифра, подходя к практической оценке стойкости. Он формулирует также основные критерии оценки качества секретных систем с позиций практики их использования.

Как видим, К. Шеннону удалось решить фундаментальные проблемы в теоретической криптографии. Его работы стимулировали бурный рост научных исследований по теории информации и криптографии.

В работах К. Шеннона по исследованию свойств языка важную роль играет величина удельной энтропии *H* на букву текста, другими словами, среднее количество информации, передаваемой буквой открытого текста. Предложенный им метод экспериментов с угадыванием очередной буквы английского текста по предыдущим буквам оказался неэффективным при получении оценок величины *H* для других языков. Метод "отгадывания" развил в своих работах А. Н. Колмогоров. Достаточно точные приближения параметра *H* для русского и французского языков получил Б. Б. Пиотровский. Он указал на существенную разницу между значениями *H* для текстов различного характера (литературных, деловых, разговорной речи).

Понятие "количества информации", содержащейся в тексте, базировалось, по К. Шеннону, лишь на частотных характеристиках. В своих фундаментальных работах 60-х годов А.Н. Колмогоров подошел к определению количества информации с учетом *смыслового содержания* текста, что позволило уточнить приближение величины *H* для литературных текстов. Необходимо также отметить, что еще задолго до К. Шеннона частотные характеристики языка изучал выдающийся русский ученый А. А. Марков. Сегодня часто используются так называемые *марковские модели* открытых текстов, учитывающие зависимости букв текста от предыдущих букв.

Следующая страница в истории криптографии XX в. посвящена телефонным шифраторам, которые были разработаны в 30-х годах и стали широко использоваться во время II мировой войны. В России разработка телефонного шифратора велась под руководством В. А. Котельникова, ставшего впоследствии академиком, ученым с мировым именем. Ему принадлежит знаменитая *теорема дискретизации* (или *теорема отсчетов*), лежащая в основе теории *цифровой обработки сигналов.*

Идея телефонного шифратора была запатентована Д.Х.Роджерсом еще в 1881 г., спустя пять лет после изобретения Беллом телефона. Идея состояла в передаче телефонного сообщения по нескольким (в простейшем случае – по двум) цепям поочередными

импульсами в некоторой быстро изменяющейся последовательности. Предлагалось разнести такие линии на значительное расстояние друг от друга с тем, чтобы устранить возможность подключения сразу ко всем одновременно. Подключение же к одной из них позволяло бы слышать лишь отдельные неразборчивые сигналы.

В более поздних разработках предлагались различные преобразования непосредственно самой речи. Звуки речи преобразуются телефоном в непрерывный электрический сигнал, который с помощью соответствующих устройств изменяется шифратором по законам электричества. К числу возможных изменений относятся: *инверсия, смещение,* или *деление диапазона* частот, *шумовые маскировки, временные перестановки* частей сигнала, а также различные комбинации перечисленных преобразований. Естественно, каждое из указанных преобразований производится под управлением ключа, который имеется у отправителя и получателя. Наиболее просто реализуемым являлось преобразование инверсии. Сложнее реализовались временные перестановки. Для их осуществления речевой сигнал в некоторый промежуток времени предварительно записывался на магнитофонной ленте. Запись делилась на отрезки длительностью в доли секунд. Отрезки с помощью нескольких магнитных головок разносились и перемешивались, в результате чего в канале слышалась хаотическая последовательность звуков. Использовалась также движущаяся магнитная головка, которая в зависимости от направления движения считывала сигналы быстрее или медленнее, чем они были записаны на ленте. В результате тон сигналов становился выше или ниже обычного, в канале быстро чередовались высокие и низкие звуки, не воспринимаемые ухом. Следует отметить, что одной из самых сложных проблем, которые возникали при разработке телефонных шифраторов, была *проблема узнавания* восстановленной после расшифрования речи.

В США первый *телефонный шифратор*, под названием А3, был принят в эксплуатацию в 1937 г. Именно он доставил президенту Рузвельту известие о начале II мировой войны утром 1 сентября 1939 г. по вызову американского посла в Париже. А3 осуществлял инверсию и перестановку 5 поддиапазонов частот. Из 3840 возможных комбинаций ($5!\cdot 2^5$) фактически использовались лишь 6, которые менялись 36 раз за каждые 20 секунд. Слабость используемой криптографии компенсировалась регулярным изменением частот передачи.

В настоящее время аналоговая телефония уступает место цифровой телефонии. Тем самым и многие технические проблемы, связанные с криптографическими преобразованиями аналоговых сигналов, отпадают за ненадобностью. Дело в том, что *оцифрованный* сигнал является дискретным и, следовательно, к нему можно применить хорошо разработанную надежную "дискретную криптографию".

Во второй половине XX в.. вслед за развитием элементной базы вычислительной техники, появились *электронные шифраторы*, разработка которых потребовала серьезных теоретических исследований во многих областях прикладной и фундаментальной математики, в первую очередь алгебре, теории вероятностей и математической статистике. Сегодня именно электронные шифраторы составляют подавляющую долю средств шифрования. Они удовлетворяют всё возрастающим требованиям по надежности и скорости шифрования. Прогресс в развитии вычислительной техники сделал возможными *программные реализации* криптографических алгоритмов, которые все увереннее вытесняют во многих сферах традиционные аппаратные средства.

В семидесятых годах произошло два события, серьезно повлиявших на дальнейшее развитие криптографии. Во-первых, был принят (и опубликован!) первый *стандарт шифрования данных* (DES), "легализовавший" принцип Керкгоффса в криптографии. Во-вторых, после работы американских математиков У. Диффи и М. Хеллмана [36] родилась "новая криптография" – *криптография с открытым ключом.* Оба этих события были рождены потребностями бурно развивающихся средств коммуникаций, в том числе локальных и глобальных компьютерных сетей, для защиты которых потребовались легко доступные и достаточно надежные криптографические средства. Криптография стала широко востребоваться не только в военной, дипломатической, государственной сферах, но также в коммерческой, банковской и других сферах.

Вслед за идеей Диффи и Хеллмана, связанной с гипотетическим понятием *однонаправленной (или односторонней) функции с секретом,* появились "кандидат" на такую функцию и реально осуществленная шифрсистема RSA с открытым ключом. Такая система была предложена в 1978 г Райвестом, Шамиром и Адлеманом. Парадоксальным казалось то, что в RSA для зашифрования и расшифрования используются разные ключи, причем ключ зашифрования может быть *открытым*, то есть всем известным. Вслед за RSA появился целый ряд других систем. В связи с несимметричным использованием ключей стал использоваться термин *асимметричная шифрсистема,* в то время как традиционные шифрсистемы стали называться *симметричными*.

Наряду с идеей открытого шифрования Диффи и Хеллман предложили идею *открытого распределения ключей,* позволяющую избавиться от защищенного канала связи при рассылке криптографических ключей. Их идея основывалась на сложности решения задачи *дискретного логарифмирования*, то есть задачи, являющейся обратной для задачи возведения в степень в конечном поле большого порядка.

Позволим себе остановиться в нашем кратком историческом обзоре на этом этапе развития криптографии, поскольку рассказ о ее современных проблемах и приложениях заставил бы нас приступить к изложению основного содержания книги.

.Теория классических шифров

Основные характеристики открытого текста

Криптография занимается защитой сообщений, содержащихся на некотором материальном носителе. При этом сами сообщения представляют собой последовательности знаков (или *слова*) некоторого *алфавита.* Различают *естественные* алфавиты, например русский или английский, и *специальные* алфавиты (цифровые, буквенно-цифровые), например двоичный алфавит, состоящий из символов 0 и 1. В свою очередь, естественные алфавиты также могут отличаться друг от друга даже для данного языка.

Алфавиты открытых сообщений

Наиболее привычны буквенные алфавиты, например русский, английский и т. д. Приведем сведения об алфавитах некоторых естественных европейских языков.

Полный русский алфавит состоит из 33 букв:

А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я

Вместе с тем используются и сокращенные русские алфавиты, содержащие 32, 31 или 30 букв. Можно отождествить буквы Е и Ё, И и Й, Ь и Ъ. часто бывает удобно включить в алфавит знак пробела между словами, в качестве которого можно взять, например, символ “– ”.

Английский алфавит состоит из 26 букв:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Иногда используется сокращенный 25-буквенный алфавит, в котором отождествлены буквы I и J.

Любопытно отметить, что полинезийский язык Самоа имеет алфавит, содержащий всего 16 букв, из которых около 60% – гласных. В арабском языке и иврите согласные вообще не используются. Они опускаются в письменном тексте и восстанавливаются читателем по смыслу.

В вычислительной технике распространены 128-битовые и 256-битовые алфавиты, использующие представление знаков алфавита в виде 7- или 8-значных двоичных комбинаций.

Наиболее известен код ASCII (American Standart Code for Information Interchange) – американский стандартный код информационного обмена.

Буквенный алфавит, в котором буквы расположены в их естественном порядке, обычно

называют *нормальным алфавитом*. В противном случае говорят о *смешанных алфавитах.* В свою очередь, смешанные алфавиты делят на *систематически перемешанные алфавиты* и *случайные алфавиты*. К первым относят алфавиты, полученные из нормального на основе некоторого правила, ко вторым – алфавиты, буквы которых следуют друг за другом в хаотическом (или случайном) порядке.

Смешанные алфавиты обычно используются в качестве нижней строки подстановки, представляющей собой ключ шифра простой замены. Для запоминания ключа (это надежнее, чем хранение ключа на некотором носителе) применяется несложная процедура перемешивания алфавита, например, основанная на *ключевом слове*.

Частотные характеристики текстовых сообщений

Криптоанализ любого шифра невозможен без учета особенностей текстов сообщений, подлежащих шифрованию. Глубинные закономерности текстовых сообщений исследуются в теории информации. Наиболее важной для криптографии характеристикой текстов является избыточность текста, введенная К. Шенноном. Именно избыточность открытого текста, проникающая в шифртекст, является основной слабостью шифра.

Более простыми характеристиками текстов, используемыми в криптоанализе, являются такие характеристики, как *повторяемость* букв, пар букв *(биграмм)* и вообще *m-ок* (*m-грамм), сочетаемость* букв друг с другом, чередование гласных и согласных и некоторые другие. Такие характеристики изучаются на основе эмпирических наблюдений текстов достаточно большой длины.

Для установления статистических закономерностей проводилась большая серия экспериментов по оценке вероятностей появления в открытом тексте фиксированных *m*-грамм (для небольших значений *m*).

Суть экспериментов состоит в подсчете чисел вхождений каждой из n^m возможных *m*-грамм в достаточно длинных открытых текстах $T = t_1\, t_2 \ldots t_l$, составленных из букв алфавита $\{a_1, a_2, \ldots, a_n\}$. При этом просматриваются подряд идущие *m*-граммы текста:

$t_1\, t_2 \ldots t_m$, $t_2\, t_3 \ldots t_{m+1}$, $\ldots\, t_{l-m+1}\, t_{l-m+2} \ldots t_l$.

Если $\vartheta\,(a_{i1}\, a_{i2} \ldots a_{im})$ – число появлений *m*-граммы $a_{i1}\, a_{i2} \ldots a_{im}$ в тексте *T,* а *L* – общее число подсчитанных *m*-грамм, то опыт показывает, что при достаточно больших *L* частоты

$$\frac{\vartheta(a_{i1}a_{i2}\ldots a_{im})}{L}$$

для данной *m*-граммы мало отличаются друг от друга. В силу этого относительную частоту считают приближением вероятности P $(a_{i1}\, a_{i2} \ldots a_{im})$ появления данной *m*-граммы в случайно выбранном месте текста (такой подход принят при статистическом определении вероятности). Например, при $m = 1$ хорошее приближение вероятностей появления букв

достигается на текстах длины в несколько тысяч букв.

Некоторая разница значений частот в приводимых в различных источниках таблицах объясняется тем обстоятельством, что частоты существенно зависят не только от длины текста, но и от его характера. Так, в технических текстах редкая буква Ф может стать довольно частой в связи с частым использованием таких слов, как *функция*, *дифференциал*, *диффузия*, *коэффициент* и т. п.

Еще большие отклонения от нормы в частоте употребления отдельных букв наблюдаются в некоторых художественных произведениях, особенно в стихах. Поэтому для надежного определения средней частоты буквы желательно иметь набор различных текстов, заимствованных из различных источников. Вместе с тем, как правило, подобные отклонения незначительны, и в первом приближении ими можно пренебречь.

В связи с этим подобные таблицы, используемые в криптографии, должны составляться с учетом *характера переписки*.

Для русского языка частоты (в порядке убывания) знаков алфавита, в котором отождествлены Е с Ё, Ь с Ъ, а также имеется знак пробела (–) между словами.

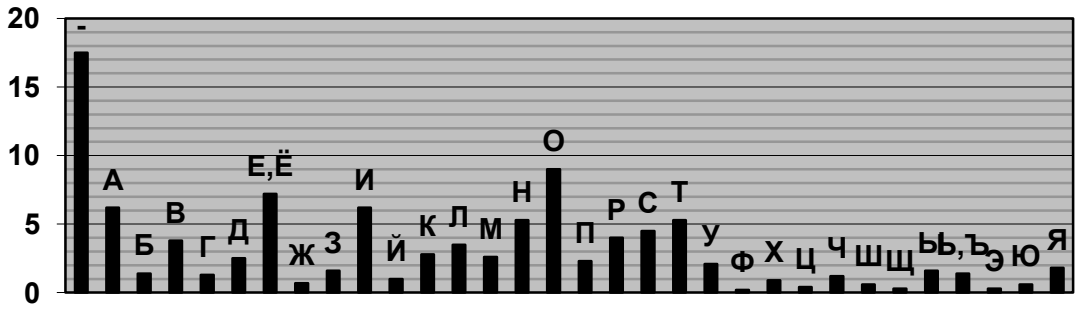


Рис. 1.4. Диаграмма частот букв русского языка

Имеется мнемоническое правило запоминания десяти наиболее частых букв русского алфавита. Эти буквы. СОС1ЗВляют нелепое слово СЕНОВАЛИТР. Можно также предложить аналогичный способ запоминания частых букв английского языка, например, с помощью слова TETRIS-HONDA.

Устойчивыми являются также частотные характеристики биграмм, триграмм и четырехграмм осмысленных текстов.

Хорошие таблицы *k*-грамм легко получить, используя тексты электронных версий многих книг, содержащихся на CD-дисках.

Для получения более точных сведений об открытых текстах можно строить и анализировать таблицы *k*-грамм при $k > 2$, однако для учебных целей вполне достаточно ограничиться биграммами. Неравновероятность *k*-грамм (и даже слов) тесно связана с характерной особенностью открытого текста – наличием в нем большого числа повторений

отдельных фрагментов текста: корней, окончаний, суффиксов, слов и фраз. Так, для русского языка такими привычными фрагментами являются наиболее частые биграммы и триграммы:

СТ, НО, ЕН, ТО, НА, ОВ, НИ, РА, ВО, КО, СТО, ЕНО, НОВ, ТОВ, ОВО, ОВА

Полезной является информация о сочетаемости букв, то есть о предпочтительных связях букв друг с другом, которую легко извлечь из таблиц частот биграмм.

Имеется в виду таблица, в которой слева и справа от каждой буквы расположены наиболее предпочтительные “соседи” (в порядке убывания частоты соответствующих биграмм). В таких таблицах обычно указывается также доля гласных и согласных букв (в процентах), предшествующих (или следующих за) данной букве.

При анализе сочетаемости букв друг с другом следует иметь в виду зависимость появления букв в открытом тексте от значительного числа предшествующих букв. Для анализа этих закономерностей используют понятие *условной вероятности*.

Наблюдения над открытыми текстами показывают, что для условных вероятностей выполняются неравенства

$p(a_{i1}) \neq p(a_{i1} / a_{i2})$, $p(a_{i1} / a_{i2}) \neq p(a_{i1} / a_{i2}\, a_{i3})$,… .

Систематически вопрос о зависимости букв алфавита в открытом тексте от предыдущих букв исследовался известным русским математиком А. А. Марковым (1856 – 1922). Он доказал, что появления букв в открытом тексте нельзя считать независимыми друг от друга. В связи с этим А. А. Марковым отмечена еще одна устойчивая закономерность открытых текстов, связанная с чередованием гласных и согласных букв. Им были подсчитаны частоты встречаемости биграмм вида гласная-гласная (*г*, *г*), гласная-согласная (*г*, *с*), согласная-гласная (*с*, *г*), согласная-согласная (*с*, *с*) в русском тексте длиной в 10^5 знаков.

Из этой таблицы видно, что для русского языка характерно чередование гласных и согласных, причем относительные частоты могут служить приближениями соответствующих условных и безусловных вероятностей:

p (г/с) ≈ 0,663, p (с/г) ≈ 0,872

p (г) ≈ 0,432, p (с) ≈ 0,568.

После А. А. Маркова зависимость появления букв текста вслед за несколькими предыдущими исследовал методами теории информации К. Шеннон. Фактически им было показано, в частности, что такая зависимость ощутима на глубину приблизительно в 30 знаков, после чего она практически отсутствует.

Приведенные выше закономерности имеют место для обычных “читаемых” открытых текстов, используемых при общении людей. Как уже отмечалось ранее, эти закономерности играют большую роль в криптоанализе. В частности, они используются при построении формализованных критериев на открытый текст, позволяющих применять методы матема-

тической статистики в задаче распознавания открытого текста в потоке сообщений. При использовании же специальных алфавитов требуются аналогичные исследования частотных характеристик “открытых текстов”, возникающих, например, при межмашинном обмене информацией или в системах передачи данных. В этих случаях построение формализованных критериев на “открытый текст” – задача значительно более сложная.

Помимо криптографии частотные характеристики открытых сообщений существенно используются и в других сферах. Например, клавиатура компьютера, пишущей машинки или линотипа – это замечательное воплощение идеи ускорения набора текста, связанное с оптимизацией расположения букв алфавита относительно друг друга в зависимости от частоты их применения.

Классификация шифров

В качестве первичного признака, по которому производится классификация шифров, используется тип преобразования, осуществляемого с открытым текстом при шифровании. Если фрагменты открытого текста (отдельные буквы или группы букв) заменяются некоторыми их эквивалентами в шифртексте, то соответствующий шифр относится к классу *шифров замены*. Если буквы открытого текста при шифровании лишь меняются местами друг с другом, то мы имеем дело с *шифрам перестановки.* С целью повышения надежности шифрования шифрованный текст, полученный применением некоторого шифра, может быть еще раз зашифрован с помощью другого шифра. Всевозможные такие композиции различных шифров приводят к третьему классу шифров, которые обычно называют *композиционными шифрами*. Заметим, что композиционный шифр может не входит ни в класс шифров замены, ни в класс шифров перестановки. В результате получаем первый уровень классификации шифров.

Классификация шифров замены

Если ключ зашифрования совпадает с ключом расшифрования $k_з = k_р$, то такие шифры называют *симметричными* если же $k_з \neq k_р$, – *асимметричными*.

Исторически известный шифр – *пропорциональной замены* представляет собой пример шифра многозначной замены, *шифр гаммирования* – пример шифра однозначной замены.

Также шифры замены делятся на шифры однозначной и многозначной замены, поточные и блочные, одноалфавитные (шифры простой замены) и многоалфавитные.

Классификация шифров в виде схемы приведена на рисунке 1.5.

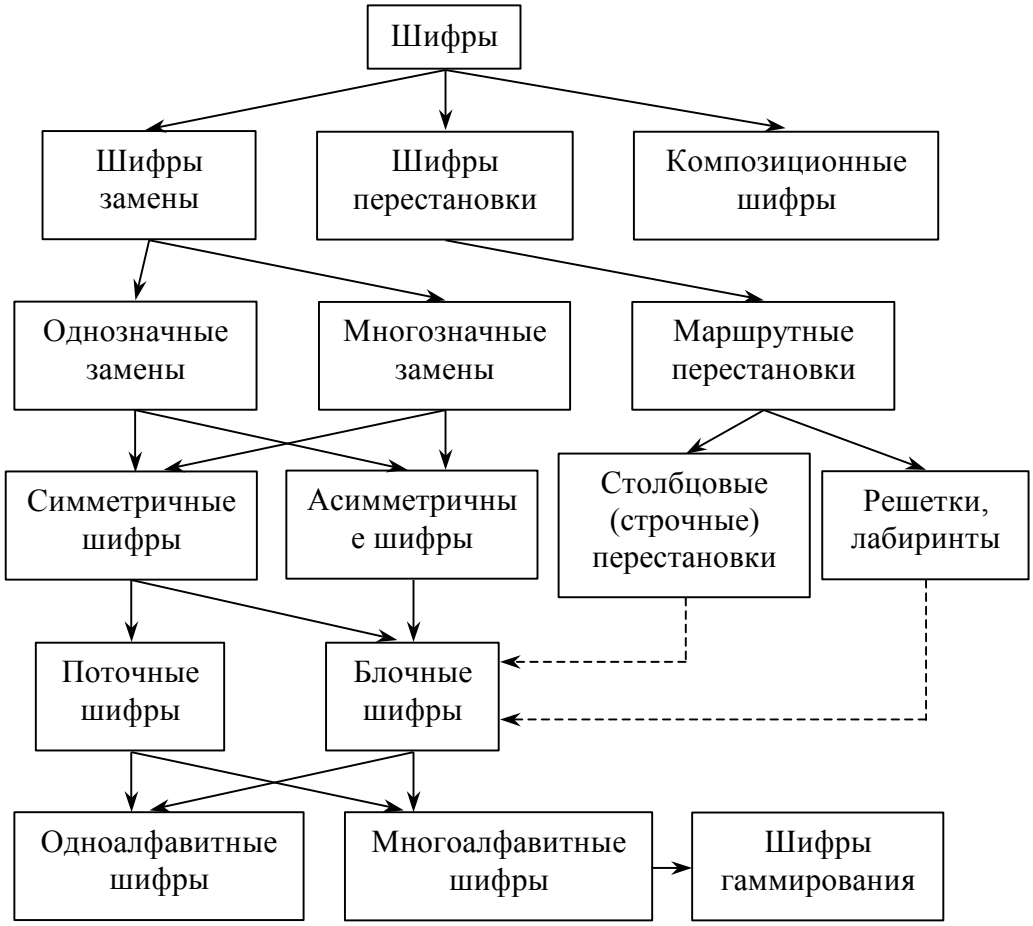


Рис. 1.5. Классификация шифров

Прокомментируем приведенную схему. Подчеркнем, что стрелки, выходящие из любого прямоугольника схемы, указывают лишь на наиболее значимые частные подклассы шифров. Пунктирные стрелки, ведущие из подклассов шифров перестановки, означают, что эти шифры можно рассматривать и как блочные шифры замены в соответствии с тем, что открытый текст делится при шифровании на блоки фиксированной длины, в каждом из которых производится некоторая перестановка букв. Одноалфавитные и многоалфавитные шифры могут быть как поточными, так и блочными. В то же время шифры гаммирования, образующие подкласс многоалфавитных шифров, относится к поточным, а не к блочным шифрам. Кроме того, они являются симметричными, а не асимметричными шифрами.

Шифры перестановки

При шифровании перестановкой символы шифруемого текста переставляются по определенному правилу в пределах блока этого текста. Шифры перестановки являются самыми простыми и, вероятно, самыми древними шифрами.

Широкое применение получили так называемые *маршрутные перестановки,* основанные на некоторой геометрической фигуре. Отрезок открытого текста записывается в такую фигуру по некоторой траектории.

В качестве ключа в шифрующих таблицах используются:

1. размер таблицы;
2. слово или фраза, задающие перестановку;

Широкое распространение получила разновидность маршрутной перестановки, называемая *вертикальной перестановкой*: сообщение записывается в таблицу по строкам слева направо. Например, зашифруем фразу

ВОТ ПРИМЕР ШИФРА ВЕРТИКАЛЬНОЙ ПЕРЕСТАНОВКИ

Запишем эту фразу в таблицу (таблица 1.5).

Таблица 1.5.

В	О	Т	П	Р	И	М
Е	Р	Ш	И	Ф	Р	А
В	Е	Р	Т	И	К	А
Л	Ь	Н	О	Й	П	Е
Р	Е	С	Т	А	Н	О
В	К	И				

После заполнения таблицы текстом сообщения по строкам для формирования шифртекста считывают содержимое таблицы по столбцам. Получается такое шифрованное сообщение:

ВЕВЛРВОРЕЬЕКТШРНСИПИТОТРФИЙАИРКПНМААЕО

Естественно, отправитель и получатель сообщения должны заранее условиться об общем ключе в виде размера таблицы.

При расшифровании, в первую очередь, надо определить число длинных столбцов, то есть число букв в последней строке прямоугольника. Для этого нужно разделить число букв в сообщении на число столбцов таблицы. Ясно, что остаток от деления и будет искомым числом. В нашем примере $38 = 7 \times 5 + 3$, поэтому в заполненной таблице имеется 3 длинных и 4 коротких столбца. Когда это число определено, буквы криптограммы можно водворить на их собственные места, и сообщение будет прочитано естественным образом.

Несколько большей стойкостью к раскрытию обладает метод шифрования, называемый *одиночной перестановкой по ключу*. Этот метод отличается от предыдущего тем, что

столбцы таблицы переставляются по ключевому слову, фразе или набору чисел длиной в строку таблицы.

Применим в качестве ключа, например, слово

ПЕЛИКАН,

а текст сообщения возьмем из предыдущего примера (таблица 6.6).

Таблица 1.6

Ключ →	П	Е	Л	И	К	А	Н
	7	2	5	3	4	1	6
	В	О	Т	П	Р	И	М
	Е	Р	Ш	И	Ф	Р	А
	В	Е	Р	Т	И	К	А
	Л	Ь	Н	О	Й	П	Е
	Р	Е	С	Т	А	Н	О
	В	К	И				

В верхней строке таблицы записан ключ, а номера под буквами ключа определены в соответствии с естественным порядком соответствующих букв ключа в алфавите. Если бы в ключе встретились одинаковые буквы, они бы были понумерованы слева направо.

Отметим, что нецелесообразно заполнять последнюю строку прямоугольника "нерабочими" буквами, так как это дало бы противнику, получившему в свое распоряжение данную криптограмму, сведения о длине числового ключа. В самом деле, в этом случае длину ключа следовало бы искать среди делителей длины сообщения.

При считывании содержимого таблицы по столбцам в порядке, указанном числовым ключом, получим шифрованное сообщение:

ИРКПН ОРЕЬЕК ПИТОТ РФИЙА ТШРНСИ МААЕО ВЕВЛРВ

При расшифровании, аналогично предыдущему примеру, в первую очередь, надо определить число длинных столбцов. В нашем примере $38 = 7 \times 5 + 3$, поэтому в заполненной таблице имеется 3 длинных и 4 коротких столбца. Согласно числовому ключу, начальные буквы криптограммы берутся из шестого (по счету слева) столбца, он – короткий (так как первые три столбца – длинные), поэтому первые пять букв образуют шестой столбец. Следующие шесть букв образуют второй столбец (он – длинный). И так далее.

Более сложные маршрутные перестановки могут использовать другие геометрические фигуры и более "хитрые" маршруты, как, например, при обходе шахматной доски "ходом коня", пути в некотором лабиринте и т.п. Возможные варианты зависят от фантазии составителя системы и, конечно, естественного требования простоты ее использования.

Для обеспечения дополнительной скрытности можно повторно зашифровать сообщение, которое уже прошло шифрование. Такой метод шифрования называется *двойной перестановкой*. В случае двойной перестановки столбцов и строк таблицы перестановки определяются отдельно для столбцов и отдельно для строк. Сначала в таблицу записывается текст сообщения, а потом поочередно переставляются столбцы, а затем строки. При расшифровании порядок перестановок должен быть обратным.

В средние века для шифрования перестановкой применялись и магические квадраты.

Магическими квадратами называют квадратные таблицы с вписанными в их клетки последовательными натуральными числами, начиная от 1, которые дают в сумме по каждому столбцу, каждой строке и каждой диагонали одно и то же число.

Шифруемый текст вписывали в магические квадраты в соответствии с нумерацией их клеток. Если затем выписать содержимое такой таблицы по строкам, то получится шифртекст, сформированный благодаря перестановке букв исходного сообщения. В те времена считалось, что созданные с помощью магических квадратов шифртексты охраняет не только ключ, но и магическая сила.

Пример магического квадрата и его заполнения сообщением

ПРИЛЕТАЮ ВОСЬМОГО

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

О	И	Р	М
Е	О	С	Ю
В	Т	А	Ь
Л	Г	О	П

Рис. 1.6. Пример магического квадрата 4×4 и его заполнения сообщением ПРИЛЕТАЮ ВОСЬМОГО

Шифртекст, получаемый при считывании содержимого правой таблицы по строкам, имеет вполне загадочный вид:

ОИРМ ЕОСЮ ВТАЬ ЛГОП

Число магических квадратов быстро возрастает с увеличением размера квадрата. Существует только один магический квадрат размером 3×3 (если не учитывать его повороты). Количество магических квадратов 4×4 составляет уже 880, а количество магических квадратов 5×5 – около 250000.

Магические квадраты средних и больших размеров могли служить хорошей базой для обеспечения нужд шифрования того времени, поскольку практически нереально выполнить вручную перебор всех вариантов для такого шифра.

Шифры простой замены

При шифровании заменой (подстановкой) символы шифруемого текста заменяются символами того же или другого алфавита с заранее установленным правилом замены.

Будем рассматривать только одноалфавтные однозначные замены, т.е. каждый символ исходного текста заменяется символами того же алфавита одинаково на всем протяжении текста. Такие замены называют *шифрами простой замены*.

Система шифрования Цезаря

Свое название шифр Цезаря получил по имени римского императора Гая Юлия Цезаря, который использовал этот шифр при переписке с Цицероном (около 50 г.до н.э.).

При шифровании исходного текста каждая буква заменялась на другую букву того же алфавита по следующему правилу. Заменяющая буква определялась путем смещения по алфавиту от исходной буквы на k букв. При достижении конца алфавита выполнялся циклический переход к его началу. Цезарь использовал шифр замены при смещении $k = 3$. Такой шифр замены можно задать таблицей подстановок, содержащей соответствующие пары букв открытого текста и шифртекста. Совокупность возможных подстановок для $k = 3$ показана в таблице 1.7.

Таблица 1.7. Одноалфавитные подстановки ($k = 3$, $m = 26$)

A	→	D	J	→	M	S	→	V
B	→	E	K	→	N	T	→	W
C	→	F	L	→	O	U	→	X
D	→	G	M	→	P	V	→	Y
E	→	H	N	→	Q	W	→	Z
F	→	I	O	→	R	X	→	A
G	→	J	P	→	S	Y	→	B
H	→	K	Q	→	T	Z	→	C

Например, послание Цезаря

VENI VIDI VICI

(в переводе на русский означает "Пришел, Увидел, Победил"), направленное его другу Аминтию после победы над понтийским царем Фарнаком, сыном Митридата, выглядело бы в зашифрованном виде так:

YHQL YLGL YLFL

Формализуем шифр Цезаря. Введем следующие обозначения:

m – число букв в алфавите (мощность алфавита);

$(x_0, x_1, \ldots)$ – открытый текст;

$(y_0, y_1, \ldots)$ – шифртекст (закрытый текст);

k – ключ, $0 \le k < m$.

Тогда процесс шифрования можно представить в виде

$y_i = E_k(x_i) = (x_i + k) \bmod m$ – зашифрование,

$x_i = D_k(y_i) = (y_i + 26 – k) \bmod m$ – расшифрование.

Для шифра Цезаря $k = 3$.

Достоинством системы шифрования Цезаря является простота шифрования и расшифрования. К недостаткам системы Цезаря следует отнести следующие:

- подстановки, выполняемые в соответствии с системой Цезаря, не маскируют частот появления различных букв исходного открытого текста. Шифр Цезаря легко вскрывается на основе анализа частот появления букв в шифртексте
- сохраняется алфавитный порядок в последовательности заменяющих букв; при изменении значения k изменяются только начальные позиции такой последовательности;
- число возможных ключей k мало;

Криптоаналитическая атака против системы одноалфавитной замены начинается с подсчета частот появления символов: определяется число появлений каждой буквы в шифртексте. Затем полученное распределение частот букв в шифртексте сравнивается с распределением частот букв в алфавите исходных сообщений, например в английском. Буква с наивысшей частотой появления в шифртексте заменяется на букву с наивысшей частотой появления в английском языке и т.д. Вероятность успешного вскрытия системы шифрования повышается с увеличением длины шифртекста.

Концепция, заложенная в систему шифрования Цезаря, оказалась весьма плодотворной, о чем свидетельствуют ее многочисленные модификации. Несколько таких модификаций будут рассмотрены ниже.

Аффинная система подстановок Цезаря

В системе шифрования Цезаря использовалось только сложение по модулю m. Применяя одновременно операции сложения и умножения по модулю m над элементами множества $Z_m = \{0, 1, 2, \ldots, m–1\}$, можно получить систему подстановок, которую называют аффинной системой подстановок Цезаря.

Введем следующие обозначения:

m – число букв в алфавите (мощность алфавита);

$(x_0, x_1, \ldots)$ – открытый текст;

$(y_0, y_1, \ldots)$ – шифртекст (закрытый текст);

В качестве ключа используется пара чисел $k = (a, b)$, где a, b – целые числа, $0 < a < m$, $0 \le b < m$. Числа a и m должны быть взаимно простыми, т.е. наибольший общий делитель НОД $(a, m) = 1$.

Тогда процесс шифрования представляется в виде

$y_i = E_{a,b}(x_i) = (ax_i + b) \bmod m$ – зашифрование,

$x_i = D_{a,b}(y_i) = a^{-1} (y_i + 26 - b) \bmod m$ – расшифрование,

где a^{-1} – обратный a элемент.

Если числа a и m взаимно просты, то существует число a^{-1}, удовлетворяющее сравнению $a \cdot a^{-1} \equiv 1 \bmod m$.

Число a^{-1} называют обратным к a по модулю m. Наприме, $3^{-1} = 9$, т.е. $3 \cdot 9 \equiv 27 \bmod m \equiv 1 \bmod m$.

Зашифруем слово HOPE с помощью аффинного шифра, полагая $k = (3, 5)$.

Для удобства вычисления заменим буквы латинского алфавита на их порядковые номера (Таблица 1.8):

Таблица 1.8

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Используя аффинный шифр с ключом $k = (3,5)$, получаем следующее соответствие между числовыми кодами букв:

t	0	1	2	3	4	5	6	7	8	9	10	11	12
$3t + 5$	5	8	11	14	17	20	23	0	3	6	9	12	15

t	13	14	15	16	17	18	19	20	21	22	23	24	25
$3t + 5$	18	21	24	1	4	7	10	13	16	19	22	25	2

Преобразуя числа в буквы английского языка, получаем следующее соответствие для букв открытого текста и шифртекста:

A	B	C	D	E	F	G	H	I	J	K	L	M
F	I	L	O	R	U	X	A	D	G	J	M	P

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
S	V	Y	B	E	H	K	N	Q	T	W	Z	C

Исходное сообщение HOPE преобразуется в шифртекст AVYR.

Достоинством аффинной системы является удобное управление ключами – ключи шифрования и расшифрования представляются в компактной форме в виде пары чисел (*a, b*). Недостатки аффинной системы аналогичны недостаткам системы шифрования Цезаря.

Аффинная система использовалась на практике несколько веков назад, а сегодня ее применение ограничивается большей частью иллюстрациями основных криптологических положений.

Система Цезаря с ключевым словом

Система шифрования Цезаря с ключевым словом является одноалфавитной системой подстановки. Особенностью этой системы является использование ключевого слова для смещения и изменения порядка символов в алфавите подстановки.

Выберем некоторое число k, $0 \le k < 26$, и слово или короткую фразу в качестве *ключевого слова.* Желательно, чтобы все буквы ключевого слова были различными. Пусть выбраны слово DIPLOMAT в качестве ключевого слова и число $k = 5$.

Ключевое слово записывается под буквами алфавита, начиная с буквы, числовой код которой совпадает с выбранным числом k:

0	1	2	3	4	5					10					15					20					25
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
					D	I	P	L	O	M	A	T													

Оставшиеся буквы алфавита подстановки записываются после ключевого слова в алфавитном порядке:

					5																				
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
V	W	X	Y	Z	**D**	**I**	**P**	**L**	**O**	**M**	**A**	**T**	B	C	E	F	G	H	J	K	N	Q	R	S	U

Теперь мы имеем подстановку для каждой буквы произвольного сообщения.

Исходное сообщение SEND MORE MONEY

шифруется как HZBY TCGZ TCBZS

Следует отметить, что требование о различии всех букв ключевого слова не обязательно. Можно просто записать ключевое слово (или фразу) без повторения одинаковых букв. Например, ключевая фраза

КАК ДЫМ ОТЕЧЕСТВА НАМ СЛАДОК И ПРИЯТЕН

и число $k = 3$ порождают следующую таблицу подстановок:

0			3																											
А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ы	Ъ	Э	Ю	Я
Ъ	Э	Ю	**К**	**А**	**Д**	**Ы**	**М**	**О**	**Т**	**Е**	**Ч**	**С**	**В**	**Н**	**Л**	**И**	**П**	**Р**	**Я**	Б	Г	Ж	З	Й	У	Ф	Х	Ц	Ш	Щ	Ь

Несомненным достоинством системы Цезаря с ключевым словом является то, что количество возможных ключевых слов практически неисчерпаемо. Недостатком этой системы является возможность взлома шифртекста на основе анализа частот появления букв.

Биграммный шифр Плейфейра

Биграммными называются шифры, позволяющие шифровать сразу по 2 буквы.

Шифр Плейфейра, изобретенный в 1854 г., является наиболее известным биграммным шифром замены. Он применялся Великобританией во время первой мировой войны. Основой шифра Плейфейра является шифрующая таблица со случайно расположенными буквами алфавита исходных сообщений.

Для удобства запоминания шифрующей таблицы отправителем и получателем сообщений можно использовать ключевое слово (или фразу) при заполнении начальных строк таблицы. В таблицу сначала вписывается по строкам ключевое слово, причем повторяющиеся буквы отбрасываются. Затем эта таблица дополняется не вошедшими в нее буквами алфавита по порядку. Поскольку ключевое слово или фразу легко хранить в памяти, то такой подход упрощает процессы шифрования и расшифрования.

Поясним этот метод шифрования на примере. Для русского алфавита шифрующая таблица может иметь размер 4×8. Выберем в качестве ключа слово БАНДЕРОЛЬ (таблица 1.9).

Таблица 1.9. Шифрующая таблица с ключевым словом БАНДЕРОЛЬ

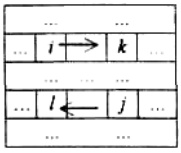
Б	А	Н	Д	Е	Р	О	Л
Ь	В	Г	Ж	З	И	Й	К
М	П	С	Т	У	Ф	Х	Ц
Ч	Ш	Щ	Ы	Ъ	Э	Ю	Я

Процедура шифрования включает следующие шаги:

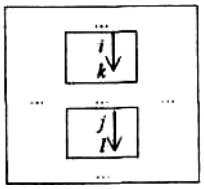
1. Открытый текст исходного сообщения разбивается на пары букв (биграммы). Текст должен иметь четное количество букв и в нем не должно быть биграмм, содержащих две одинаковые буквы. Если эти требования не выполнены, то в текст добавляются “пустышки”. “Пустышкой” является некоторая редкая для данного типа текста буква (или знак), которая вставляется между одинакоывми буквами биграммы или добавляется в текст для того, чтобы его длина стала четной.

2. Последовательность биграмм открытого текста преобразуется с помощью шифрующей таблицы в последовательность биграмм шифртекста по следующим правилам:

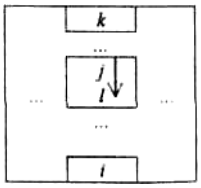
a) Если обе буквы биграммы открытого текста не попадают на одну строку или столбец (как, например, буквы А и Й в таблице 3.5), тогда находят буквы в углах прямоугольника, определяемого данной парой букв. (В нашем примере пара букв АЙ отображается в пару ОВ.)



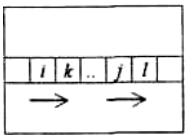
b) Если обе буквы биграммы открытого текста принадлежат одному столбцу таблицы, то буквами шифртекста считаются буквы, которые лежат под ними. (Например, биграмма НС дает биграмму шифртекста ГЩ.)



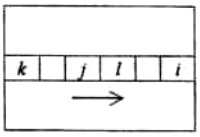
Если при этом буква открытого текста находится в нижней строке, то для шифртекста берется соответствующая буква из верхней строки того же столбца. (Например, биграмма ВШ дает биграмму шифртекста ПА.)



c) Если обе буквы биграммы открытого текста принадлежат одной строке таблицы, то буквами шифртекста считаются буквы, которые лежат справа от них. (Например, биграмма НО дает биграмму шифртекста ДЛ.)



Если при этом буква открытого текста находится в крайнем правом столбце, то для шифра берут соответствующую букву из левого столбца в той же строке. (Например, биграмма ФЦ дает биграмму шифртекста ХМ.)



Зашифруем текст

ВСЕ ТАЙНОЕ СТАНЕТ ЯВНЫМ

Разбиение этого текста на биграммы дает

ВС ЕТ АЙ НО ЕС ТА НЕ ТЯ ВН ЫМ

Данная последовательность биграмм открытого текста преобразуется с помощью шифрующей таблицы (таблица 3.5) в следующую последовательность биграмм шифртекста:

ГП ДУ ОВ ДЛ НУ ПД ДР ЦЫ ГА ЧТ

При расшифровании применяется обратный порядок действий.

Криптоанализ шифра Плейфера опирается на частотный анализ биграмм, триграмм и четырехграмм шифртекста и особенности замены шифрвеличин на шифробозначения, связанные с расположением алфавита в прямоугольнике.

Шифр Хилла

Шифр был назван по имени своего создателя Лестора Хилла.

Шифрвеличинами здесь являются n-граммы открытого текста ($n \geq 2$), представленного некоторым числовым кодом.

Правило зашифрования:

Имеется алфавит открытого текста, состоящий из m букв.

Ключ K – некоторая обратимая матрица размером $[n \times n]$ над множеством $Z_m = \{0, 1, 2, \ldots, m-1\}$.

Исходный текст разбивается на блоки по n букв (n-граммы), каждую из которых можно представить в виде вектора $\vec{x}_j = (x_{1j}, x_{2j}, \ldots, x_{nj})$, где j – номер блока.

$\vec{y}_j = (y_{1j}, y_{2j}, \ldots, y_{nj})$ – n-грамма шифртекста.

Процесс шифрования представляется в виде:

$\vec{y}_j = K \cdot \vec{x}_j \pmod m$ – зашифрование,

$\vec{x}_j = K^{-1} \cdot \vec{y}_j \pmod m$ – расшифрование,

Рассмотрим пример.

Алфавит: латинский, $m = 26$.

Открытый текст: PAY MORE MONEY.

Ключ: $K = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$.

$n = 2$, т.е. разбиение открытого текста производится на биграммы.

Разобьем открытый текст на биграммы и заменим каждую букву ее числовым эквивалентом.

PA	→	15 0
YM	→	24 12

OR	$\rightarrow$	14 17
EM	$\rightarrow$	4 12
ON	$\rightarrow$	14 13
EY	$\rightarrow$	4 24

Преобразуем биграммы открытого текста $\vec{x}_j$ в биграммы шифртекста $\vec{y}_j$.

$$\vec{y}_1 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 15 \\ 0 \end{bmatrix} (\text{mod } 26) = \begin{bmatrix} 45 \\ 30 \end{bmatrix} (\text{mod } 26) = \begin{bmatrix} 19 \\ 4 \end{bmatrix},$$

$$\vec{y}_2 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 24 \\ 12 \end{bmatrix} (\text{mod } 26) = \begin{bmatrix} 4 \\ 4 \end{bmatrix},$$

$$\vec{y}_3 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 14 \\ 17 \end{bmatrix} (\text{mod } 26) = \begin{bmatrix} 15 \\ 9 \end{bmatrix},$$

$$\vec{y}_4 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 12 \end{bmatrix} (\text{mod } 26) = \begin{bmatrix} 22 \\ 16 \end{bmatrix},$$

$$\vec{y}_5 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 14 \\ 13 \end{bmatrix} (\text{mod } 26) = \begin{bmatrix} 3 \\ 15 \end{bmatrix},$$

$$\vec{y} = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 24 \end{bmatrix} (\text{mod } 26) = \begin{bmatrix} 6 \\ 24 \end{bmatrix}.$$

Заменяя в биграммах шифртекста числа на соответствующие буквы согласно таблице 3.4, получаем 12-грамму шифртекста

TE EE PJ WQ DP GY

Для расшифрования биграмм $\vec{y}_j$, шифртекста и восстановления биграмм $\vec{x}_j$ открытого текста необходимо выполнить обратное преобразование K^{-1} согласно уравнению

$$\vec{x}_j = K^{-1} \cdot \vec{y}_j.$$

В рассмотренном примере матрицы преобразования имели размер 2×2 и шифровались биграммы (пары) букв. Хотя буква Е может быть зашифрована по-разному в различных парах исходного сообщения, одна и та же пара, например ЕМ, будет шифроваться всегда одинаково на протяжении всего исходного текста.

Увеличение значности шифрвеличин (n) резко усложняет попытки вскрытия открытого текста по известному тексту криптограммы.

Шифры сложной замены

Шифры сложной замены называют многоалфавитными, так как для шифрования каждого символа исходного сообщения применяют свой шифр простой замены. Многоалфавитная подстановка последовательно и циклически меняет используемые алфавиты.

Правила замены при r-алфавитной подстановке:

символ x_0 исходного сообщения заменяется символом y_0 из алфавита B_0,

символ x_1 заменяется символом y_1 из алфавита B_1, и так далее,

символ x_{r-1} заменяется символом y_{r-1} из алфавита B_{r-1},

символ x_r заменяется символом y_r снова из алфавита B_0, и т.д.

Общая схема многоалфавитной подстановки для случая r = 4 показана на в таблице 6.10.

Таблица 1.10. Схема r-алфавитной подстановки для случая r = 4

Входной символ:	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
Алфавит подстановки:	B_0	B_1	B_2	B_3	B_0	B_1	B_2	B_3	B_0	B_1

Эффект использования многоалфавитной подстановки заключается в том, что обеспечивается маскировка естественной статистики исходного языка, так как конкретный символ из исходного алфавита А может быть преобразован в несколько различных символов шифровальных алфавитов B_j. Степень обеспечиваемой защиты теоретически пропорциональна длине периода r в последовательности используемых алфавитов B_j.

Одной из старейших и наиболее известных многоалфавитных систем является система шифрования Виженера.

Шифр "двойной квадрат" Уитстона

В 1854 г. англичанин Чарльз Уитстон разработал новый метод шифрования биграммами, который называют "двойным квадратом". Свое название этот шифр получил по аналогии с полибианским квадратом. Шифр Уитстона открыл новый этап в истории развития криптографии. В отличие от полибианского шифр "двойной квадрат" использует сразу две таблицы, размещенные по одной горизонтали, а шифрование идет биграммами, как в шифре Плейфейра. Эти не столь сложные модификации привели к появлению на свет качественно новой криптографической системы ручного шифрования. Шифр "двойной квадрат" оказался очень надежным и удобным и применялся Германией даже в годы второй мировой войны.

Поясним процедуру шифрования этим шифром на примере. Пусть имеются две таблицы со случайно расположенными в них русскими алфавитами (рисунок 1.6).

Ж	Щ	Н	Ю	Р
И	Т	Ь	Ц	Б

И	Ч	Г	Я	Т
,	Ж	Ь	М	О

Я	М	Е	.	С
В	Ы	П	Ч	
:	Д	У	О	К
З	Э	Ф	Г	Ш
Х	А	,	Л	Ъ

З	Ю	Р	В	Щ
Ц	:	П	Е	Л
Ъ	А	Н	.	Х
Э	К	С	Ш	Д
Б	Ф	У	Ы	

Рис. 1.6. Две таблицы со случайно расположенными символами русского алфавита для шифра "двойной квадрат"

Перед шифрованием исходное сообщение разбивают на биграммы. Каждая биграмма шифруется отдельно. Первую букву биграммы находят в левой таблице, а вторую букву – в правой таблице. Затем мысленно строят прямоугольник так, чтобы буквы биграммы лежали в его противоположных вершинах. Другие две вершины этого прямоугольника дают буквы биграммы шифртекста.

Предположим, что шифруется биграмма исходного текста ИЛ. Буква И находится в столбце 1 и строке 2 левой таблицы. Буква Л находится в столбце 5 и строке 4 правой таблицы. Это означает, что прямоугольник образован строками 2 и 4, а также столбцами 1 левой таблицы и 5 правой таблицы. Следовательно, в биграмму шифртекста входят буква О, расположенная в столбце 5 и строке 2 правой таблицы, и буква В, расположенная в столбце 1 и строке 4 левой таблицы, т.е. получаем биграмму шифртекста ОВ.

Ж	Щ	Н	Ю	Р
И	Т	Ь	Ц	Б
Я	М	Е	.	С
В	Ы	П	Ч	
:	Д	У	О	К
З	Э	Ф	Г	Ш
Х	А	,	Л	Ъ

И	Ч	Г	Я	Т
,	Ж	Ь	М	О
З	Ю	Р	В	Щ
Ц	:	П	Е	Л
Ъ	А	Н	.	Х
Э	К	С	Ш	Д
Б	Ф	У	Ы	

Если обе буквы биграммы сообщения лежат в одной строке, то и буквы шифртекста берут из этой же строки. Первую букву биграммы шифртекста берут из левой таблицы в столбце, соответствующем второй букве биграммы сообщения. Вторая же буква биграммы шифртекста берется из правой таблицы в столбце, соответствующем первой букве биграммы сообщения. Поэтому биграмма сообщения ТО превращается в биграмму шифртекста ЖБ. Аналогичным образом шифруются все биграммы сообщения:

Сообщение ПР ИЛ ЕТ АЮ _Ш ЕС ТО ГО

Шифртекст ПЕ ОВ ЩН ФМ ЕШ РФ БЖ ДЦ

Шифрование методом "двойного квадрата" дает весьма устойчивый к вскрытию и простой в применении шифр. Взламывание шифртекста "двойного квадрата" требует больших усилий.

Одноразовая система шифрования

Почти все применяемые на практике шифры характеризуются как условно надежные, поскольку они могут быть в принципе раскрыты при наличии неограниченных вычислительных возможностей. Абсолютно надежные шифры нельзя разрушить даже при использовании неограниченных вычислительных возможностей. Существует единственный такой шифр, применяемый на практике, -одноразовая система шифрования. Характерной особенностью одноразовой системы шифрования является одноразовое использование ключевой последовательности.

Одноразовая система шифрует исходный открытый текст (x_0, x_1, …, x_{n-1}) в шифртекст (y_0, y_1, …, y_{n-1}) посредством подстановки Цезаря

$y_i = (x_i + k_i) \bmod m$, $0 \le i < n$,

где k_i – i-й элемент случайной ключевой последовательности.

Ключевое пространство одноразовой системы представляет собой набор дискретных случайных величин из Z_m и содержит m^n значений.

Процедура расшифрования описывается соотношением

$x_i = (y_i + k_i) \bmod m$, $0 \le i < n$,

где k_i – i-й элемент той же самой случайной ключевой последовательности.

Одноразовая система изобретена в 1917 г. американцами Дж. Моборном и Г. Вернамом. Для реализации этой системы подстановки иногда используют одноразовый блокнот. Этот блокнот составлен из отрывных страниц, на каждой из которых напечатана таблица со случайными числами (ключами) k_i. Блокнот выполняется в двух экземплярах: один используется отправителем, а другой - получателем. Для каждого символа x_i сообщения используется свой ключ k_i из таблицы только один раз. После того как таблица использована, она должна быть удалена из блокнота и уничтожена. Шифрование нового сообщения начинается с новой страницы.

Этот шифр абсолютно надежен, если набор ключей k_i действительно случаен и непредсказуем. Если криптоаналитик попытается использовать для заданного шифртекста все возможные наборы ключей и восстановить все возможные варианты исходного текста, то они все окажутся равновероятными. Не существует способа выбрать исходный текст, который был действительно послан. Теоретически доказано, что одноразовые системы являются нераскрываемыми системами, поскольку их шифртекст не содержит достаточной информации для восстановления открытого текста.

Казалось бы, что благодаря данному достоинству одноразовые системы следует применять во всех случаях, требующих абсолютной информационной безопасности. Однако возможности применения одноразовой системы ограничены чисто практическими аспектами. Существенным моментом является требование одноразового использования случайной ключевой последовательности. Ключевая последовательность с длиной, не меньшей длины сообщения, должна передаваться получателю сообщения заранее или отдельно по некоторому секретному каналу. Это требование не будет слишком обременительным для передачи действительно важных одноразовых сообщений, например, по горячей линии Вашингтон-Москва. Однако такое требование практически неосуществимо для современных систем обработки информации, где требуется шифровать многие миллионы символов.

В некоторых вариантах одноразового блокнота прибегают к более простому управлению ключевой последовательностью, но это приводит к некоторому снижению надежности шифра. Например, ключ определяется указанием места в книге, известной отправителю и получателю сообщения. Ключевая последовательность начинается с указанного места этой книги и используется таким же образом, как в системе Вижинера. Иногда такой шифр называют шифром с бегущим ключом. Управление ключевой последовательностью в таком варианте шифра намного проще, так как длинная ключевая последовательность может быть представлена в компактной форме. Но с другой стороны, эти ключи не будут случайными. Поэтому у криптоаналитика появляется возможность использовать информацию о частотах букв исходного естественного языка.

Шифрование методом гаммирования

Под *гаммированием* понимают процесс наложения по определенному закону гаммы шифра на открытые данные. *Гамма шифра* – это псевдослучайная последовательность, выработанная по заданному алгоритму для зашифрования открытых данных и расшифрования зашифрованных данных.

Процесс зашифрования заключается в генерации гаммы шифра и наложении полученной гаммы на исходный открытый текст обратимым образом, например с использованием операции сложения по модулю 2.

Следует отметить, что перед зашифрованием открытые данные разбивают на блоки T_i^o одинаковой длины, обычно по 64 бита. Гамма шифра вырабатывается в виде последовательности блоков $\Gamma_i^{ш}$ аналогичной длины.

Уравнение зашифрования можно записать в виде

$T_i^{ш} = \Gamma_i^{ш} \oplus T_i^{o}$, $i = 1 \ldots M$,

где $T_i^{ш}$ – i-й блок шифртекста;

$\Gamma_i^{ш}$ – i-й блок гаммы шифра;

T_i^{o} – i-й блок открытого текста;

M – количество блоков открытого текста.

Процесс расшифрования сводится к повторной генерации гаммы шифра и наложению этой гаммы на зашифрованные данные. Уравнение расшифрования имеет вид

$T_i^{o} = \Gamma_i^{ш} \oplus T_i^{ш}$.

Получаемый этим методом шифртекст достаточно труден для раскрытия, поскольку теперь ключ является переменным. По сути дела гамма шифра должна изменяться случайным образом для каждого шифруемого блока. Если период гаммы превышает длину всего шифруемого текста и злоумышленнику неизвестна никакая часть исходного текста, то такой шифр можно раскрыть только прямым перебором всех вариантов ключа. В этом случае криптостойкость шифра определяется длиной ключа.

Методы генерации псевдослучайных последовательностей чисел

При шифровании методом гаммирования в качестве ключа используется случайная строка битов, которая объединяется с открытым текстом, также представленным в двоичном виде (например, А= 00000, В = 00001, С = 00010 и т.д.), с помощью побитового сложения по модулю 2, и в результате получается шифрованный текст. Генерирование непредсказуемых двоичных последовательностей большой длины является одной из важных проблем классической криптографии. Для решения этой проблемы широко используются генераторы двоичных псевдослучайных последовательностей.

Генерируемые псевдослучайные ряды чисел часто называют гаммой шифра или просто гаммой (по названию буквы γ греческого алфавита, часто используемой в математических формулах для обозначения случайных величин).

Обычно для генерации последовательности псевдослучайных чисел применяют компьютерные программы, которые, хотя и называются генераторами случайных чисел, на самом деле выдают детерминированные числовые последовательности, которые по своим свойствам очень похожи на случайные.

К криптографически стойкому генератору псевдослучайной последовательности чисел (гаммы шифра) предъявляются три основных требования:

- период гаммы должен быть достаточно большим для шифрования сообщений различной длины;

- гамма должна быть практически непредсказуемой, что означает невозможность предсказать следующий бит гаммы, даже если известны тип генератора и предшествующий кусок гаммы;
- генерирование гаммы не должно вызывать больших технических сложностей.

Длина периода гаммы является самой важной характеристикой генератора псевдослучайных чисел. По окончании периода числа начнут повторяться, и их можно будет предсказать. Требуемая длина периода гаммы определяется степенью закрытости данных. Чем длиннее ключ, тем труднее его подобрать. Длина периода гаммы зависит от выбранного алгоритма получения псевдослучайных чисел.

Второе требование связано со следующей проблемой: как можно достоверно убедиться, что псевдослучайная гамма конкретного генератора является действительно непредсказуемой? Пока не существуют такие универсальные и практически проверяемые критерии и методики. Чтобы гамма считалась непредсказуемой, т.е. истинно случайной, необходимо, чтобы ее период был очень большим, а различные комбинации битов определенной длины были равномерно распределены по всей ее длине.

Третье требование обусловливает возможность практической реализации генератора программным или аппаратным путем с обеспечением необходимого быстродействия.

Один из первых способов генерации псевдослучайных чисел на ЭВМ предложил в 1946 г. Джон фон Нейман. Суть этого способа состоит в том, что каждое последующее случайное число образуется возведением в квадрат предыдущего числа с отбрасыванием цифр младших и старших разрядов. Однако этот способ оказался ненадежным и от него вскоре отказались.

Линейный конгруэнтный генератор

Из известных процедур генерации последовательности псевдослучайных целых чисел наиболее часто применяется так называемый линейный конгруэнтный генератор. Этот генератор вырабатывает последовательность псевдослучайных чисел y_1, y_2, …, y_{i-1}, y_i, …, используя соотношение

$y_i = (a \cdot y_{i-1} + b) \bmod m$,

где y_i – i-е (текущее) число последовательности;

y_{i-1} – предыдущее число последовательности;

a – множитель (коэффициент);

b – приращение;

m – модуль;

y_0 – порождающее число (исходное значение).

Данное уравнение генерирует псевдослучайные числа с периодом повторения, который зависит от выбираемых значений параметров a, b и m и может достигать значения m.

Например, при $y_0 = a = b = 7$, $m = 10$ последовательность выглядит так: 7, 6, 9, 0, 7, 6, 9, 0, 7, …. Длина периода полученной последовательности равна 4.

Значение модуля m должно быть велико, обычно m берется равным 2^n. Приращение b должно быть взаимно простым с m, т.е. НОД (b, m) = 1. Коэффициент a должен быть нечетным числом.

Конгруэнтные генераторы, работающие по алгоритму, предложенному Национальным бюро стандартов США, используются, в частности, в системах программирования. Эти генераторы имеют длину периода 2^{24} и обладают хорошими статистическими свойствами. Однако такая длина периода мала для криптографических применений. Кроме того, доказано, что последовательности, генерируемые конгруэнтными генераторами, не являются криптографически стойкими.

Регистр сдвига с линейной обратной связью

Существует способ генерации последовательностей псевдослучайных чисел на основе линейных рекуррентных соотношений.

Рассмотрим рекуррентные соотношения и их разностные уравнения:

$$\sum_{j=0}^{k} h_j a_{i+j} = 0,$$

$$a_{i+k} = -\sum_{j=0}^{k-1} h_j a_{i+j}, \qquad (3)$$

где $h_0 \neq 0$, $h_k = 1$ и каждое h_i принадлежит полю GF(q).

Полем называется множество F, на котором определены операции сложения и умножения, удовлетворяющие ассоциативному, коммутативному и дистрибутивному законам, причём имеются как аддитивная (0), так и мультипликативная (1) единицы, каждый элемент имеет обратный элемент по сложению, кроме того, каждый элемент, кроме аддитивной единицы 0 имеет и обратный элемент по умножению.

Конечное поле $F(p)$ – это поле с конечным числом p элементов. В общем случае число элементов $p = q^n$, где q – некоторое простое число и $n \geq 1$. Такие конечные поля называют *полями Галуа* и обозначают GF(q^n) или GF(q) при $n = 1$.

Решением этих уравнений является последовательность элементов a_0, a_1, a_2, … поля GF(q). Соотношение (3.1) определяет правило вычисления a_k по известным значениям величин a_0, a_1, a_2, …, a_{k-1}. Затем по известным значениям a_0, a_1, a_2, …, a_k находят a_{k+1} и т.д. В результате по начальным значениям a_0, a_1, a_2, …, a_{k-1} можно построить бесконечную последовательность, причем каждый ее последующий член определяется из k предыдущих.

Последовательности такого вида легко реализуются на компьютере, при этом реализация получается особенно простой, если все h_i и a_i принимают значения 0 и 1 из поля GF(2).

На рисунке 1.6 показана линейная последовательная переключательная схема, которая может быть использована для вычисления суммы (3) и, следовательно, для вычисления значения a_k по значениям k предыдущих членов последовательности.

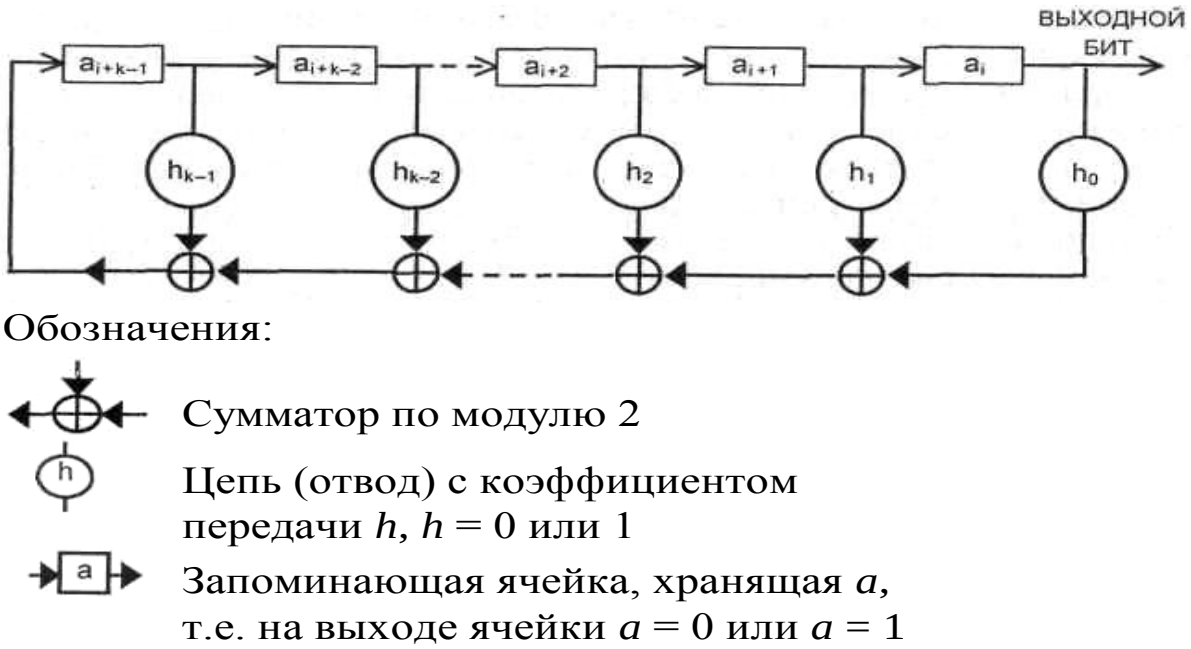


Рис. 1.6. Генератор с регистром сдвига

Исходные величины a_0, a_1, a_2, …, a_{k-1} помещаются в разряды сдвигового регистра, последовательные сдвиги содержимого которого соответствуют вычислению последовательных символов, при этом выход после i-го сдвига равен a_i. Данное устройство называют генератором последовательности чисел, построенным на базе сдвигового регистра с линейной обратной связью.

Конфигурацию обратных связей (отводов) h_i в генераторе со сдвиговым регистром определяет вид многочлена

$$h(x) = \sum_{j=0}^{k} h_j x^j ,$$

где x – формальная переменная;

h_j – коэффициент при x^j, принимающий значение 0 или 1;

Другими словами, если у многочлена $h(x)$ коэффициент $h_j = 1$, это означает, что отвод h_j в схеме генератора присутствует, если же у многочлена $h(x)$ коэффициент $h_j = 0$, то отвод h_j в схеме генератора отсутствует.

В качестве $h(x)$ необходимо выбирать неприводимый примитивный многочлен

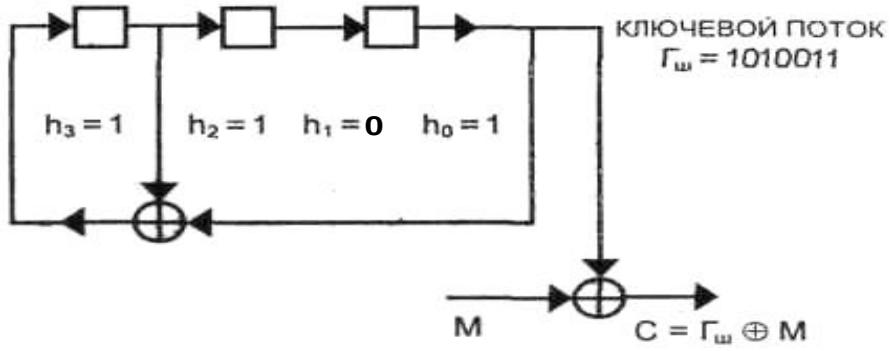
Многочлен $h(x)$ называется *неприводимым многочленом* степени m, если $h(x)$ нельзя разложить на сомножители – многочлены степени меньше m.

При таком выборе многочлена $h(x)$ со старшей степенью m генератор обеспечивает выдачу псевдослучайной последовательности двоичных чисел с максимально возможным периодом $2^m - 1$.

Рассмотрим в качестве примера трехразрядный сдвиговый регистр с линейной обратной связью (рисунок 1.7), построенный в соответствии с неприводимым примитивным многочленом

$h(x) = x^3 + x^2 + 1$,

где коэффициенты $h_3 = 1$, $h_2 = 1$, $h_1 = 0$, $h_0 = 1$.



Состояния регистра:		
1	0	1
0	1	0
0	0	1
1	0	0
1	1	0
1	1	1
0	1	1
1	0	1
...	...	...

Рис. 1.7. Трехразрядный регистр сдвига с обратными связями
(генератор гаммы шифра $\Gamma_ш$)

Пусть ключом является 101. Регистр начинает работать с этого состояния; последовательность состояний регистра приведена на рисунке 3.5. Регистр проходит через все семь ненулевых состояний и снова возвращается в свое исходное состояние 101. Это – наиболее длинный период данного регистра с линейной обратной связью. Такая последовательность называется *последовательностью максимальной длины* для сдвигового регистра (Maximal Lenght Shift Register Sequence – MLSRS). Питерсон и Уэлдон показали, что при любом целом m существует m-битовая последовательность MLSRS с периодом 2^m–1. В частности, при $m = 100$ последовательность будет иметь период 2^{100}-1 и не повторится 10^{16} лет при передаче ее по линии связи со скоростью 1 Мбит/с.

В нашем примере выходной последовательностью (гаммой шифра) $\Gamma_ш$ сдвигового регистра с обратной связью является последовательность 1010011, которая циклически повторяется. В этой последовательности имеется четыре единицы и три нуля, и их

распределение настолько близко к равномерному, насколько это возможно в последовательности, имеющей длину 7. Если рассмотреть пары последовательных битов, то пары 10 и 01 появляются по два раза, а пары 00 и 11-один раз, что опять оказывается настолько близким к равномерному распределению, насколько это возможно. В случае последовательности максимальной длины для *m*-разрядного регистра это свойство равнораспределенности распространяется на тройки, четверки и т.д. битов, вплоть до *m*-битовых групп. Благодаря такой близости к равномерному распределению последовательности максимальной длины часто используются в качестве псевдослучайных последовательностей в криптографических системах, которые имитируют работу криптостойкой системы одноразового шифрования.

Хотя такая криптографическая система осуществляет имитацию заведомо криптостойкой системы одноразового шифрования, сама она не отличается стойкостью и может быть раскрыта за несколько секунд работы компьютера при условии наличия известного открытого текста.

Если отводы регистра с обратной связью зафиксированы, то для нахождения начального состояния регистра достаточно знать *m* битов открытого текста. Чтобы найти *m* битов ключевого потока, *m* битов известного открытого текста складывают по модулю 2 с соответствующими *m* битами шифртекста. Полученные *m* битов дают состояние сдвигового регистра с обратной связью в обратном направлении на некоторый момент времени. Затем, моделируя работу регистра назад, можно определить его исходное состояние.

Компьютерный практикум по классическим шифрам

Шифры замены

1. Основы шифрования.

2. Шифры однозначной замены.

3. Полиграммные шифры.

4. Омофонические шифры.

5. Полиалфавитные шифры.

6. Нерегулярные шифры.

Основы шифрования

Сущность шифрования методом замены заключается в следующем [9]. Пусть шифруются сообщения на русском языке и замене подлежит каждая буква этих сообщений. Тогда, букве **А** исходного алфавита сопоставляется некоторое множество символов (шифрозамен) $\mathbf{M_А}$, **Б –** $\mathbf{M_Б}$, **…, Я –** $\mathbf{M_Я}$. Шифрозамены выбираются таким образом, чтобы любые два множества ($\mathbf{M_I}$ и $\mathbf{M_J}$, $\mathbf{i \neq j}$) не содержали одинаковых элементов ($\mathbf{M_I \cap M_J = Ø}$).

Таблица, приведенная на рис.1.8, является ключом шифра замены. Зная ее, можно осуществить как шифрование, так и расшифрование.

А	Б	…	Я
$M_А$	$M_Б$	…	$M_Я$

Рис.1.8. Таблица шифрозамен

При шифровании каждая буква **А** открытого сообщения заменяется любым символом из множества $\mathbf{M_А}$. Если в сообщении содержится несколько букв **А**, то каждая из них заменяется на любой символ из $\mathbf{M_А}$. За счет этого с помощью одного ключа можно получить различные варианты шифрограммы для одного и того же открытого сообщения.

Так как множества $\mathbf{M_А}$, $\mathbf{M_Б}$, ..., $\mathbf{M_Я}$ попарно не пересекаются, то по каждому символу шифрограммы можно однозначно определить, какому множеству он принадлежит, и, следовательно, какую букву открытого сообщения он заменяет. Поэтому расшифрование возможно и открытое сообщение определяется единственным образом.

Приведенное выше описание сущности шифров замены относится ко всем их разновидностям за исключением полиалфавитных шифров, в которых для зашифрования разных символов исходного алфавита могут использоваться одинаковые шифрозамены (т.е. $\mathbf{M_I \cap M_J \neq Ø}$, $\mathbf{i \neq j}$).

Метод замены часто реализуется многими пользователями при работе на компьютере. Если по забывчивости не переключить на клавиатуре набор символов с латиницы на кириллицу, то вместо букв русского алфавита при вводе текста будут печататься буквы латинского алфавита («шифрозамены»).

Шифры замены можно разделить на следующие **подклассы** (разновидности):

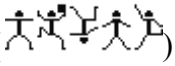
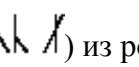
- шифры однозначной замены (моноалфавитные, простые подстановочные). Количество шифрозамен для каждого символа исходного алфавита равно 1 ($|\mathbf{M_i}| = 1$ для одного символа);

- полиграммные шифры. Аналогичен предыдущему за исключением того, что шифрозамене соответствует сразу блок символов исходного сообщения ($|\mathbf{M_i}| = 1$ для блока символов);

- омофонические шифры (однозвучные, многозначной замены). Количество шифрозамен для отдельных символов исходного алфавита больше 1 ($|\mathbf{M_i}| \geq 1$ для одного символа);

- полиалфавитные шифры (многоалфавитные). Состоит из нескольких шифров однозначной замены. Выбор варианта алфавита для зашифрования одного символа зависит от особенностей метода шифрования ($|\mathbf{M_i}| > 1$ для одного символа);

- нерегулярные шифры. Шифрозамены состоят из разного количество символов.

Для записи исходных и зашифрованных сообщений используются строго определенные алфавиты. Алфавиты для записи исходных и зашифрованных сообщений могут отличаться. Символы обоих алфавитов могут быть представлены буквами, их сочетаниями, числами, рисунками, звуками, жестами и т.п. В качестве примера можно привести пляшущих человечков из рассказа А. Конан Дойла () и рукопись рунического письма () из романа Ж. Верна «Путешествие к центру Земли».

2. Шифры однозначной замены

Максимальное количество ключей для любого шифра этого вида не превышает **n!,** где **n** – количество символов в алфавите. С увеличением числа **n** значение **n!** растет очень быстро (1! = 1, 5! = 120, 10! = 3628800, 15! = 1307674368000). При больших **n** для приближенного вычисления **n!** можно воспользоваться формулой Стирлинга

$$n! \approx \sqrt{2\pi n} * \left(\frac{n}{e}\right)^n.$$

Шифр Цезаря. Данный шифр был придуман Гаем Юлием Цезарем и использовался им в своей переписке (1 век до н.э.). Применительно к русскому языку суть его состоит в

следующем. Выписывается исходный алфавит (**А, Б, ..., Я**), затем под ним выписывается тот же алфавит, но с циклическим сдвигом на 3 буквы влево.

А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю	Я
Г	Д	Е	Ё	Ж	З	И	И	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю	Я	А	Б	В

Рис.1.9. Таблица шифрозамен для шифра Цезаря

При зашифровке буква **А** заменяется буквой **Г**, **Б** - на **Д** и т. д. Так, например, исходное сообщение «АБРАМОВ» после шифрования будет выглядеть «ГДУГПСЕ». Получатель сообщения «ГДУГПСЕ» ищет эти буквы в нижней строке и по буквам над ними восстанавливает исходное сообщение «АБРАМОВ».

Ключом в шифре Цезаря является величина сдвига нижней строки алфавита. Количество ключей для всех модификаций данного шифра применительно к алфавиту русского языка равно 33. Возможны различные модификации шифра Цезаря, в частности лозунговый шифр.

Лозунговый шифр. Для данного шифра построение таблицы шифрозамен основано на лозунге (ключе) – легко запоминаемом слове. Вторая строка таблицы шифрозамен заполняется сначала словом-лозунгом (причем повторяющиеся буквы отбрасываются), а затем остальными буквами, не вошедшие в слово-лозунг, в алфавитном порядке. Например, если выбрано слово-лозунг «ДЯДИНА», то таблица имеет следующий вид.

А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю	Я
Д	Я	И	Н	А	Б	В	Г	Е	Ё	Ж	З	Й	К	Л	М	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю

Рис. 1.9. Таблица шифрозамен для лозунгового шифра

При шифровании исходного сообщения «АБРАМОВ» по приведенному выше ключу шифрограмма будет выглядеть «ДЯПДКМИ».

В качестве лозунга рекомендуется выбирать фразу, в которой содержаться конечные буквы алфавита. В общем случае, количество вариантов нижней строки (применительно к русскому языку) составляет 33! ($\geq 10^{35}$).

Полибианский квадрат. Шифр изобретен греческим государственным деятелем, полководцем и историком Полибием (III век до н.э.). Применительно к русскому алфавиту суть шифрования заключалась в следующем. В квадрат 6х6 выписываются буквы (необязательно в алфавитном порядке).

	1	2	3	4	5	6
1	А	Б	В	Г	Д	Е
2	Ё	Ж	З	И	Й	К
3	Л	М	Н	О	П	Р
4	С	Т	У	Ф	Х	Ц
5	Ч	Ш	Щ	Ъ	Ы	Ь
6	Э	Ю	Я	-	-	-

Рис. 1.10. Таблица шифрозамен для полибианского квадрата

Шифруемая буква заменяется на координаты квадрата (строка-столбец), в котором она записана. Например, если исходное сообщение «АБРАМОВ», то шифрограмма – «11 12 36 11 32 34 13». В Древней Греции сообщения передавались с помощью оптического телеграфа (с помощью факелов). Для каждой буквы сообщения вначале поднималось количество факелов, соответствующее номеру строки буквы, а затем номеру столбца.

Тюремный шифр. Эта звуковая разновидность полибианского квадрата была разработана заключенными. Система состояла из нескольких ударов, обозначающих строки и столбцы в таблице с буквами алфавита. Один удар, а потом еще два соответствовали строке 1 и столбцу 2, т.е. букве **Б**. Пауза служила разделителем между строками и столбцами. Таким образом, зашифровать исходное сообщение «АБРАМОВ» можно следующим образом.

А	тук ___ тук
Б	тук ___ тук, тук
Р	тук, тук, тук ___ тук, тук, тук, тук, тук, тук
А	тук ___ тук
М	тук, тук, тук ___ тук, тук
О	тук, тук, тук ___ тук, тук, тук, тук
В	тук ___ тук, тук, тук

Рис. 6.11. Пример использования тюремного шифра

Шифрующая система Трисемуса (Тритемия). В 1508 г. аббат из Германии Иоганн Трисемус написал печатную работу по криптологии под названием «Полиграфия». В этой книге он впервые систематически описал применение шифрующих таблиц, заполненных алфавитом в случайном порядке. Для получения такого шифра замены обычно использовались таблица для записи букв алфавита и ключевое слово (или фраза). В таблицу сначала вписывалось по строкам ключевое слово, причем повторяющиеся буквы

отбрасывались. Затем эта таблица дополнялась не вошедшими в нее буквами алфавита по порядку. На рис. 6.12 изображена таблица с ключевым словом «ДЯДИНА».

Д	Я	И	Н	А	Б
В	Г	Е	Ё	Ж	З
Й	К	Л	М	О	П
Р	С	Т	У	Ф	Х
Ц	Ч	Ш	Щ	Ы	Ь
Ъ	Э	Ю	-	-	-

Рис.1.12. Таблица шифрозамен для шифра Трисемуса

Каждая буква открытого сообщения заменяется буквой, расположенной под ней в том же столбце. Если буква находится в последней строке таблицы, то для ее шифрования берут самую верхнюю букву столбца. Например, исходное сообщение «АБРАМОВ», зашифрованное – «ЖЗЦЖУФЙ».

Шифр масонов. В XVIII в. масоны создали шифр, чтобы скрыть от общественности свои коммерческие сделки. Как поведали те, кто прежде состоял в рядах этого общества, масоны пользовались способом засекречивания, весьма похожим на шифр розенкрейцеров. В «решетке» и в углах находятся точки, которыми заменяются буквы:

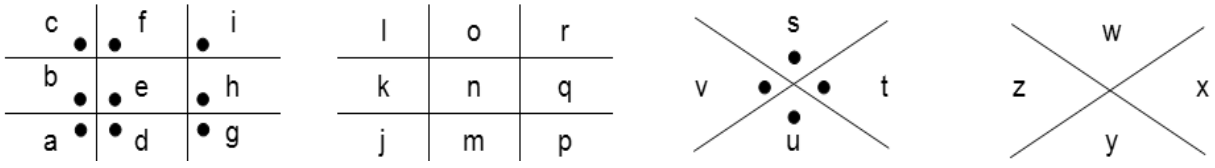


Рис. 1.13. Шифр масонов

Так как клятвы хранить тайну нарушались не раз, большинство Великих лож масонов в США больше не пользуются письменными шифрами, предпочитая передавать устные инструкции во время закрытых ритуалов.

С помощью шифра масонов можно легко расшифровать следующую фразу.

Рис. 1.14. Пример использования шифра масонов

Это первый уровень, на котором находятся все впервые вступившие в общество члены: Blue Lodge (рус. «Голубая (Синяя) ложа»).

Одним из существенных **недостатков шифров однозначной замены** является их легкая вскрываемость. При вскрытии шифрограмм используются различные приемы, которые даже при отсутствии мощных вычислительных средств позволяют добиться положительного результата. Один из таких приемов базируется на том, что в шифрограммах остается информация о частоте встречаемости букв исходного текста. Если в открытом сообщении часто встречается какая-либо буква, то в шифрованном сообщении также часто будет встречаться соответствующий ей символ. Еще в 1412 году Шихаба ал-Калкашанди в своем труде «Субх ал-Ааша» привел таблицу частоты появления арабских букв в тексте на основе анализа текста Корана. Для разных языков мира существуют подобные таблицы. Так, например, для русского языка такая таблица выглядит следующим образом.

Таблица 1.11. Вероятности появления букв русского языка в текстах*

Буква (символ)	Вероят-ность	Буква	Вероят-ность	Буква	Вероят-ность	Буква	Вероят-ность
Пробел	0.146	Р	0.042	Я	0.017	Ж	0.007
О	0.094	Л	0.039	З	0.016	Ш	0.006
Е	0.071	В	0.038	Ы	0.015	Ц, Ю	0.005
А	0.069	К	0.029	Г	0.014	Щ	0.004
И	0.064	М	0.027	Ь, Б	0.013	Ф	0.003
Н	0.057	П	0.026	Ч	0.012	Э	0.002
Т	0.054	Д	0.024	Й	0.010	Ъ	0.001
С	0.046	У	0.023	Х	0.008		

*) В таблице приведены оценки вероятностей появления букв русского языка и пробела, полученные на основе анализа научно-технических и художественных текстов общим объемом более 1000000 символов.

Существуют подобные таблицы для пар букв (биграмм). Например, часто встречаемыми биграммами являются «то», «но», «ст», «по», «ен» и т.д. Другой прием вскрытия шифрограмм основан на исключении возможных сочетаний букв. Например, в текстах (если они написаны без орфографических ошибок) нельзя встретить сочетания «чя», «щы», «ьъ» и т.п.

Для усложнения задачи вскрытия шифров однозначной замены еще в древности перед шифрованием из исходных сообщений исключали пробелы и/или гласные буквы. Другим способом, затрудняющим вскрытие, является шифрование **биграммами** (парами букв).

Полиграммные шифры

Полиграммные шифры замены - это шифры, в которых одна шифрозамена соответствует сразу нескольким символам исходного текста.

Биграммный шифр Порты. Шифр Порты, представленный им в виде таблицы, является первым известным биграммным шифром. Размер его таблицы составлял 20 х 20 ячеек; наверху горизонтально и слева вертикально записывался стандартный алфавит (в нем не было букв J, K, U, W, X и Z). В ячейках таблицы могли быть записаны любые числа, буквы или символы - сам Джованни Порта пользовался символами - при условии, что содержимое ни одной из ячеек не повторялось. Применительно к русскому языку таблица шифрозамен может выглядеть следующим образом.

	А	Б	В	Г	Д	Е (Ё)	Ж	З	И (Й)	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
А	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019	020	021	022	023	024	025	026	027	028	029	030	031
Б	032	033	034	035	036	037	038	039	040	041	042	043	044	045	046	047	048	049	050	051	052	053	054	055	056	057	058	059	060	061	062
В	063	064	065	066	067	068	069	070	071	072	073	074	075	076	077	078	079	080	081	082	083	084	085	086	087	088	089	090	091	092	093
Г	094	095	096	097	098	099	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124
Д	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155
Е (Ё)	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186
Ж	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217
З	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248
И (Й)	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279
К	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310
Л	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341
М	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372
Н	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403
О	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434
П	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465
Р	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496
С	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527
Т	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558
У	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589
Ф	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620
Х	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651
Ц	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682
Ч	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713
Ш	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744
Щ	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775
Ъ	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806
Ы	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837
Ь	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868
Э	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899
Ю	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930
Я	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961

Рис. 1.15. Таблица шифрозамен для шифра Порты

Шифрование выполняется парами букв исходного сообщения. Первая буква пары указывает на строку шифрозамены, вторая - на столбец. В случае нечетного количества букв в исходном сообщении к нему добавляется вспомогательный символ («пустой знак»). Например, исходное сообщение «АБ РА МО В», зашифрованное – «002 466 355 093». В качестве вспомогательного символа использована буква «Я».

Шифр Playfair (англ. «Честная игра»). В начале 1850-х гг. Чарлз Уитстон придумал так называемый «прямоугольный шифр». Леон Плейфер, близкий друг Уитстона, рассказал об этом шифре во время официального обеда в 1854 г. министру внутренних дел лорду Пальмерстону и принцу Альберту. А поскольку Плейфер был хорошо известен в военных и дипломатических кругах, то за творением Уитстона навечно закрепилось название «шифр Плейфера».

Данный шифр стал первым буквенным биграммным шифром (в биграммной таблице Порты использовались символы, а не буквы). Он был предназначен для обеспечения секретности телеграфной связи и применялся британскими войсками в Англо-бурской и Первой мировой войнах. Им пользовалась также австралийская служба береговой охраны островов во время Второй мировой войны.

Шифр предусматривает шифрование пар символов (биграмм). Таким образом, этот шифр более устойчив к взлому по сравнению с шифром простой замены, так как затрудняется частотный анализ. Он может быть проведен, но не для 26 возможных символов (латинский алфавит), а для 26 х 26 = 676 возможных биграмм. Анализ частоты биграмм возможен, но является значительно более трудным и требует намного большего объема зашифрованного текста.

Для шифрования сообщения необходимо разбить его на биграммы (группы из двух символов), при этом, если в биграмме встретятся два одинаковых символа, то между ними добавляется заранее оговоренный вспомогательный символ (в оригинале – **X**, для русского алфавита - **Я**). Например, «зашифрованное сообщение» становится «за ши фр ов ан но ес о**Я** об ще ни е**Я**». Для формирования ключевой таблицы выбирается лозунг и далее она заполняется по правилам шифрующей системы Трисемуса. Например, лозунг «ДЯДИНА»

Д	Я	И	Н	А	Б
В	Г	Е	Ё	Ж	З
Й	К	Л	М	О	П
Р	С	Т	У	Ф	Х
Ц	Ч	Ш	Щ	Ы	Ь
Ъ	Э	Ю	-	1	2

Рис.1.16. Ключевая таблица для шифра Playfair

Затем, руководствуясь следующими правилами, выполняется зашифровывание пар символов исходного текста:

1. Если символы биграммы исходного текста встречаются в одной строке, то эти символы замещаются на символы, расположенные в ближайших столбцах справа от соответствующих символов. Если символ является последним в строке, то он заменяется на первый символ этой же строки.

2. Если символы биграммы исходного текста встречаются в одном столбце, то они преобразуются в символы того же столбца, находящимися непосредственно под ними. Если символ является нижним в столбце, то он заменяется на первый символ этого же столбца.

3. Если символы биграммы исходного текста находятся в разных столбцах и разных строках, то они заменяются на символы, находящиеся в тех же строках, но соответствующие другим углам прямоугольника.

Пример шифрования.

- биграмма «за» формирует прямоугольник – заменяется на «жб»;

- биграмма «ши» находятся в одном столбце – заменяется на «юе»;

- биграмма «фр» находятся в одной строке – заменяется на «хс»;

- биграмма «ов» формирует прямоугольник – заменяется на «йж»;

- биграмма «ан» находятся в одной строке – заменяется на «ба»;

- биграмма «но» формирует прямоугольник – заменяется на «ам»;

- биграмма «ес» формирует прямоугольник – заменяется на «гт»;

- биграмма «оя» формирует прямоугольник – заменяется на «ка»;

- биграмма «об» формирует прямоугольник – заменяется на «па»;

- биграмма «ще» формирует прямоугольник – заменяется на «шё»;

- биграмма «ни» формирует прямоугольник – заменяется на «ан»;

- биграмма «ея» формирует прямоугольник – заменяется на «ги».

Шифрограмма – «жб юе хс йж ба ам гт ка па шё ан ги».

Для расшифровки необходимо использовать инверсию этих правил, откидывая символы **Я** (или **Х**), если они не несут смысла в исходном сообщении.

Шифр Хилла. Первый практически реализуемый способ шифрования с использованием алгебры был придуман в 1929 г. математиком Лестером Хиллом - профессором из Хантер-колледжа в Нью-Йорке, статья которого «Cryptography in an Algebraic Alphabet» была опубликована в журнале «The American Mathematical Monthly».

Каждой букве алфавита сопоставляется число. Для русского алфавита можно использовать простейшую схему: А = 0, Б = 1, ..., Я = 32. Для зашифрования блок исходного сообщения из **n** букв рассматривается как n-мерный вектор чисел и умножается на матрицу размером **n** x **n** по модулю 33. Данная матрица, совместно с кодовой таблицей сопоставления букв алфавита с числами, является ключом зашифрования. Для расшифрования применяется обратная матрица[1] по модулю.

Например, для триграммных замен могут использоваться следующие матрицы зашифрования / расшифрования.

6	27	1
13	16	32
28	17	15

матрица зашифрования

2	26	17
26	20	4
13	30	21

матрица расшифрования

Рис. 1.17. Матрицы зашифрования / расшифрования

Исходное сообщение «АБРАМОВ», дополненное двумя вспомогательными буквами «яя» (для кратности трем), после сопоставления букв с числами будет выглядеть следующим образом «0 1 17 0 13 15 2 32 32». После перемножения троек чисел на матрицу зашифрования шифрограмма примет следующий вид «11 32 8 3 28 17 17 11 24» (или в буквенном эквиваленте «КЯЗ ГЬР РКЧ»).

АБР - 0 1 17

(6 * 0 + 27 * 1 + 1 * 17) mod 33 = 11 (К)

(13 * 0 + 16 * 1 + 32 * 17) mod 33 = 32 (Я)

(18 * 0 + 17 * 1 + 15 * 17) mod 33 = 8 (З)

АМО - 0 13 15

(6 * 0 + 27 * 13 + 1 * 15) mod 33 = 3 (Г)

(13 * 0 + 16 * 13 + 32 * 15) mod 33 = 28 (Ь)

(28 * 0 + 17 * 13 + 15 * 15) mod 33 = 17 (Р)

Вяя - 2 32 32

(6 * 2 + 27 * 32 + 1 * 32) mod 33 = 17 (Р)

(13 * 2 + 16 * 32 + 32 * 32) mod 33 = 11 (К)

(28 * 2 + 17 * 32 + 15 * 32) mod 33 = 24 (Ч)

Для расшифрования тройки чисел шифрограммы необходимо умножить на матрицу расшифрования.

КЯЗ - 11 32 8

(2 * 11 + 26 * 32 + 17 * 8) mod 33 =

0 (А)

(26 * 11 + 20 * 32 + 4 * 8) mod 33 =
1 (Б)

(13 * 11 + 30 * 32 + 21 * 8) mod 33 =
17 (Р)

ГЬР - 3 28 17

(2 * 3 + 26 * 28 + 17 * 17) mod 33 =
0 (А)

(26 * 3 + 20 * 28 + 4 * 17) mod 33 =
13 (М)

(13 * 3 + 30 * 28 + 21 * 17) mod 33 =
15 (О)

РКЧ - 17 11 24

(2 * 17 + 26 * 11 + 17 * 24) mod 33 =
2 (В)

(26 * 17 + 20 * 11 + 4 * 24) mod 33 =
32 (я)

(13 * 17 + 30 * 11 + 21 * 24) mod 33
= 32 (я)

В результате будет получен набор чисел «0 1 17 0 13 15 2 32 32», соответствующий исходному сообщению со вспомогательными символами «АБРАМОВяя».

---

[1]**Обратная матрица** - матрица A^{-1}, при умножении на которую, исходная матрица А дает в результате единичную матрицу Е.

Омофонические шифры

Другое направление повышения стойкости шифров замены состоит в том, чтобы каждое множество шифрообозначений $\mathbf{M_i}$ содержало более одного элемента. При использовании такого шифра одну и ту же букву (если она встречается несколько раз в сообщении) заменяют на разные шифрозамены из $\mathbf{M_i}$. Это позволяет скрыть истинную частоту встречаемости букв открытого сообщения.

Система омофонов. В 1401 г. Симеоне де Крема стал использовать таблицы омофонов для сокрытия частоты появления гласных букв в тексте при помощи более чем одной шифрозамены. Такие шифры позже стали называться **шифрами многозначной замены** или **омофонами**[2]. Они получили развитие в XV веке. В книге «Трактат о шифрах» Леона Баттисты Альберти (итальянский ученый, архитектор, теоретик искусства, секретарь папы Климентия XII), опубликованной в 1466 г., приводится описание шифра замены, в котором каждой букве ставится в соответствие несколько эквивалентов, число которых пропорционально частоте встречаемости буквы в открытом тексте. Так, если ориентироваться на то число шифрозамен для буквы **О** должно составлять 94, для буквы **Е** – 71 и т.д. При этом каждая шифрозамена должна состоять из 3 цифр и их общее количество равно 1000. На рис.1.18 представлен фрагмент таблицы шифрозамен.

№ п/п	Пробел	А	Б	В	…	М	…	О	…	Р	…	Я
1	012	311	128	175	…	037	…	248	…	064	…	266
2	042	357	950	194	…	149	…	267	…	189	…	333
…	…	…	…	…	…	…	…	…	…	…	…	…
13	278	495	990	199	…	349	…	303	…	374	…	749
…	…	…		…	…	…	…	…	…	…	…	…
17	342	519		427	…	760	…	306	…	469	…	845
…	…	…		…	…	…	…	…	…	…		
27	437	637		524	…	777	…	432	…	554		
…	…	…		…				…	…	…		
38	457	678		644				824	…	721		
…	…	…						…	…	…		
42	628	776						828	…	954		
…	…	…						…				
69	681	901						886				
…	…							…				
94	974							903				
…	…											
146	976											

Рис.1.18. Фрагмент таблицы шифрозамен для системы омофонов

При шифровании символ исходного сообщения заменяется на любую шифрозамену из своего столбца. Если символ встречается повторно, то, как правило, используют разные шифрозамены. Например, исходное сообщение «АБРАМОВ» после шифрования может выглядеть «357 990 374 678 037 828 175».

Книжный шифр. Заметным вкладом греческого ученого Энея Тактика в криптографию является предложенный им так называемый книжный шифр, описанный в сочинении «Об обороне укреплённых мест». Эней предложил прокалывать малозаметные дырки в книге или в другом документе над буквами секретного сообщения. Интересно отметить, что в первой мировой войне германские шпионы использовали аналогичный шифр, заменив дырки на точки, наносимые симпатическими чернилами[3] на буквы газетного текста.

После первой мировой войны книжный шифр приобрел иной вид. Шифрозамена для каждой буквы определялась набором цифр, которые указывали на номер страницы, строки и позиции в строке. Количество книг, изданных за всю историю человечества, является величиной ограниченной (по крайней мере, явно меньше, чем 15!). Однако отсутствие полной электронной базы по изданиям делает процедуру вскрытия шифрограмм почти не выполнимой. В связи с этим книжный шифр относят к категории совершенных.

Вариантные шифры . Вариантные шифры напоминают полибианский квадрат, но для каждой строки и столбца используется по два буквенных идентификатора. В квадрат (прямоугольник) шифрозамен вначале записывается ключевое слово без повторяющихся букв, а затем дополняется не вошедшими в него буквами по порядку следования в алфавите. Каждой строке и столбцу квадрата ставится в соответствие по две буквы алфавита. Буквы для идентификации строк и столбцов не должны повторяться.

	Й Ц	У К	Е Н	Г Ш	Щ З	Х Ъ
Ф Ы	Д	Я	И	Н	А	Б
В А	В	Г	Е	Ё	Ж	З
П Р	Й	К	Л	М	О	П
О Л	Р	С	Т	У	Ф	Х
Д Ж	Ц	Ч	Ш	Щ	Ы	Ь
Э Я	Ъ	Э	Ю	-	-	-

Рис. 1.19. Пример таблицы шифрозамен вариантного шифра с ключевым словом «ДЯДИНА»

Комбинации букв-идентификаторов строки и столбца дают по восемь шифрозамен для каждой буквы исходного текста. Например, для буквы **Д** возможны шифрозамены: **ФЙ**,**ЙФ**, **ФЦ**, **ЦФ**, **ЫЙ**, **ЙЫ**, **ЫЦ** и **ЦЫ**. Для таблицы шифрозамен, приведенной исходное сообщение «АБРАМОВ» может быть зашифровано как «ЫЗ ЫХ ОЦ ЗФ ГР РЩ АЙ».

[2]**Омофоны** (греч. homos - одинаковый и phone - звук) - слова, которые звучат одинаково, но пишутся по-разному и имеют разное значение.

[3]**Симпатические (невидимые) чернила** — чернила, записи которыми являются изначально невидимыми и становятся видимыми только при определенных условиях (нагрев, освещение, химический проявитель и т. д.).

Полиалфавитные шифры

Напомним, что полиалфавитные шифры состоят из нескольких шифров однозначной замены и отличаются друг от друга способом выбор варианта алфавита для зашифрования одного символа.

Диск Альберти. В «Трактате о шифрах» Альберти приводит первое точное описание многоалфавитного шифра на основе шифровального диска.

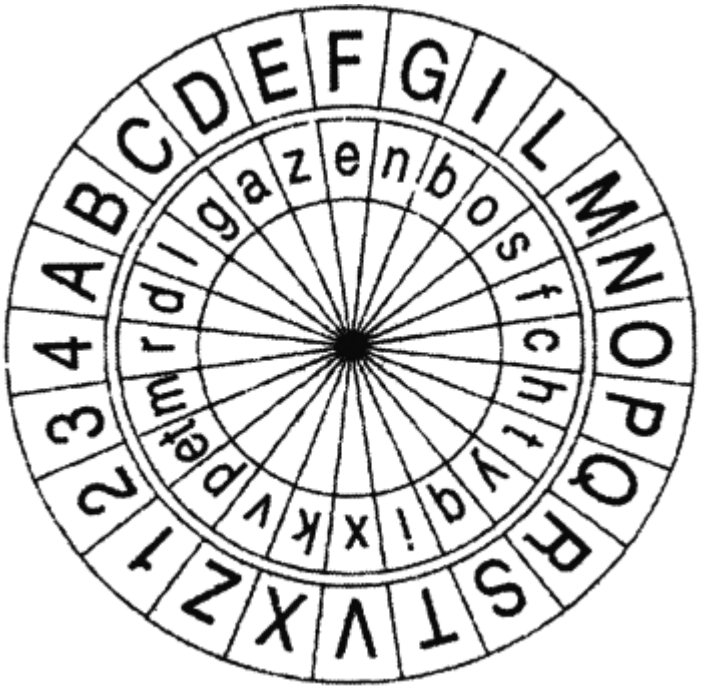


Рис.1.20. Диск Альберти

Он состоял из двух дисков – внешнего неподвижного (на нем были нанесены буквы в алфавитном порядке и цифры 1, 2, 3, 4) и подвижного внутреннего диска на котором буквы были переставлены. Процесс шифрования заключался в нахождении буквы открытого текста на внешнем диске и замене ее на букву с внутреннего диска, стоящую под ней. После этого внутренний диск сдвигался на одну позицию и шифрование второй буквы производилось уже по новому шифралфавиту. Ключом данного шифра являлся порядок расположения букв на внутреннем диске и его начальное положение относительно внешнего диска.

Таблица Трисемуса. Одним из шифров, придуманных немецким аббатом Трисемусом, стал многоалфавитный шифр, основанный на так называемой «таблице Трисемуса» - таблице со стороной равной **n**, где **n** – количество символов в алфавите. В первой строке матрицы записываются буквы в порядке их очередности в алфавите, во второй – та же последовательность букв, но с циклическим сдвигом на одну позицию влево, в третьей – с циклическим сдвигом на две позиции влево и т.д.

А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А
В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б
Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В
Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г
Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д
Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е
З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж
И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З
Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И
К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й
Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К
М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л
Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М
О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н
П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р
Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С
У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т
Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У
Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф
Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х
Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц
Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч
Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш
Ъ	Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ
Ы	Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ
Ь	Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы
Э	Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь
Ю	Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э
Я	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю

Рис. 1.21. Таблица Трисемуса

Здесь первая строка является одновременно и строкой букв открытого текста. Первая буква текста шифруется по первой строке, вторая буква по второй и так далее после использования последней строки вновь возвращаются к первой. Так сообщение «АБРАМОВ» приобретет вид «АВТГРУИ».

Система шифрования Виженера. В 1586 г. французский дипломат Блез Виженер представил перед комиссией Генриха III описание простого, но довольно стойкого шифра, в основе которого лежит таблица Трисемуса.

Перед шифрованием выбирается ключ из символов алфавита. Сама процедура шифрования заключается в следующем. По i-ому символу открытого сообщения в первой строке определяется столбец, а по i-ому символу ключа в крайнем левом столбце – строка. На пересечении строки и столбца будет находиться i-ый символ, помещаемый в шифрограмму. Если длина ключа меньше сообщения, то он используется повторно. Например, исходное сообщение «АБРАМОВ», ключ – «ДЯДИНА», шифрограмма – «ДАФИЩОЖ».

Справедливости ради, следует отметить, что авторство данного шифра принадлежит итальянцу Джованни Батиста Беллазо, который описал его в 1553 г. История «проигнорировала важный факт и назвала шифр именем Виженера, несмотря на то, что он ничего не сделал для его создания». Беллазо предложил называть секретное слово или фразу **паролем** (ит. password; фр. parole - слово).

В 1863 г. Фридрих Касиски опубликовал алгоритм атаки на этот шифр, хотя известны случаи его взлома шифра некоторыми опытными криптоаналитиками ещё в XVI веке. Несмотря на это шифр Виженера имел репутацию исключительно стойкого к «ручному» взлому еще долгое время. Так, известный писатель и математик Чарльз Лютвидж Доджсон (Льюис Кэрролл) в своей статье «Алфавитный шифр», опубликованной в детском журнале в 1868 г., назвал шифр Виженера невзламываемым. В 1917 году научно-популярный журнал «Scientific American» также отозвался о шифре Виженера, как о неподдающемся взлому.

Роторные машины. Идеи Альберти и Беллазо использовались при создании электромеханических роторных машин первой половины XX века. Некоторые из них использовались в разных странах вплоть до 1980-х годов. Большинство использовало понятие ротора - механического колеса, используемого для выполнения подстановки. Наиболее известной из роторных машин является немецкая машина времен Второй мировой войны «Энигма».

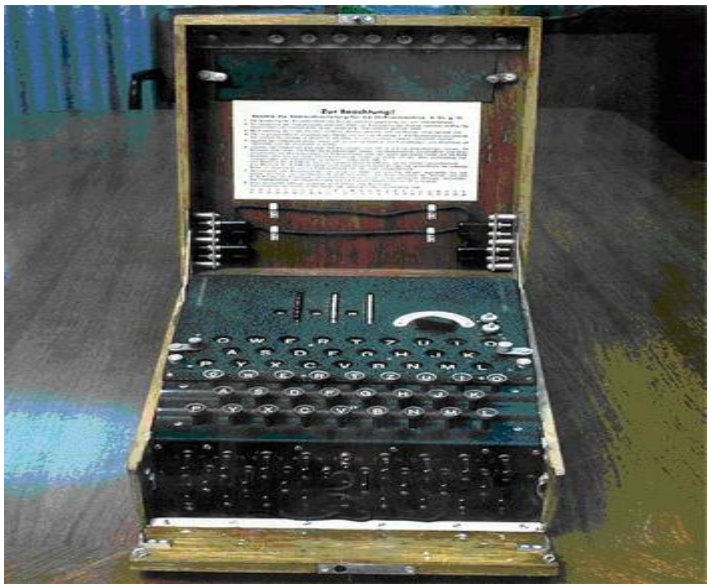


Рис. 1.22. Шифровальная машина Энигма

Роторная машина, включающая клавиатуру и набор роторов, реализует вариант шифра Виженера. Каждый ротор представляет собой произвольное размещение алфавита, имеет 26 позиций (применительно к латинскому алфавиту) и выполняет простую подстановку. Например, ротор может быть использован для замены **A** на **F**, **B** на **U**, **C** на **I** и так далее.

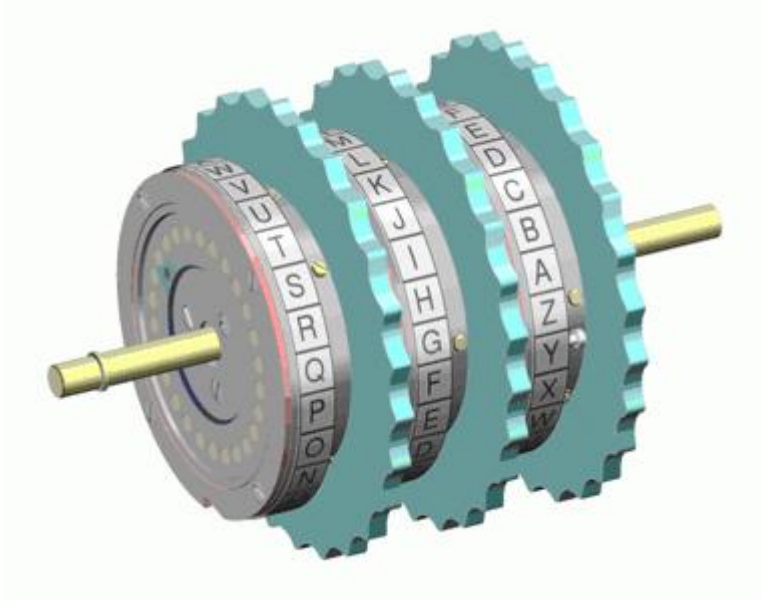


Рис. 1.23. Три последовательно соединённых ротора

Выходные штыри одного ротора соединены с входными штырями следующего ротора и при нажатии символа исходного сообщения на клавиатуре замыкали электрическую цепь, в результате чего загоралась лампочка с символом шифрозамены.

Шифрующее действие «Энигмы» показано для двух последовательно нажатых клавиш - ток течёт через роторы, «отражается» от рефлектора, затем снова через роторы.

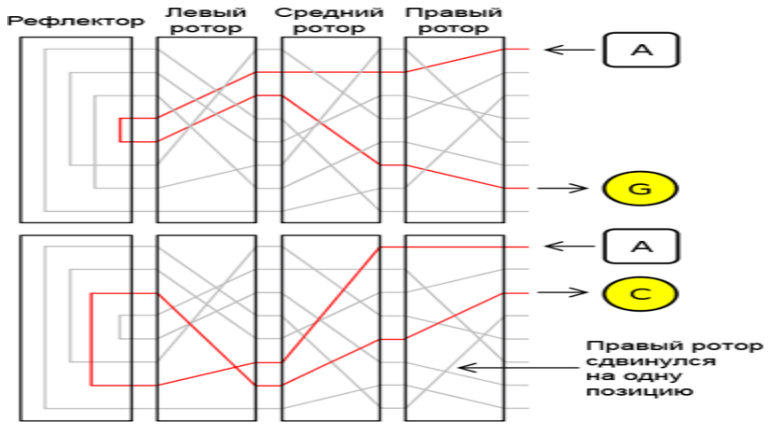


Рис. 1.24. Схема шифрования

Примечание. Серыми линиями показаны другие возможные электрические цепи внутри каждого ротора. Буква **A** шифруется по-разному при последовательных нажатиях одной клавиши, сначала в **G**, затем в **C**. Сигнал идет по другому маршруту за счёт поворота ротора после нажатия предыдущей буквы исходного сообщения.

В некоторых шифрах в самой шифрограмме могут содержаться символы, предписывающие использование того или иного алфавита.

Шифры Тени. Главными развлечениями для американцев тридцатых годов XX века были бульварное чтиво и радио. Для раскрутки своих книжек издательство Street & Smith проспонсировало радиопередачу, ведущим в которой был Тень (англ. Shadow), загадочный рассказчик со зловещим голосом, который в начале каждого выпуска заявлял: «Кто знает, что за зло прячется в сердцах людей? Тень знает!». Успех радиопередачи подтолкнул издательство к решению начать выпускать серию книг, в которой главным героем был бы Тень. Свои услуги предложил Уолтер Гибсон, большой любитель фокусов и головоломок. Под псевдонимом Максвелл Грант он принялся писать роман за романом, да с такой скоростью, что за свою жизнь написал почти 300 книжек о грозе тех, кто нечист помыслами. В новелле «Цепочка смерти» супергерой воспользовался так называемым кодом направления, хотя на самом деле он действует скорее как шифр, чем как код:

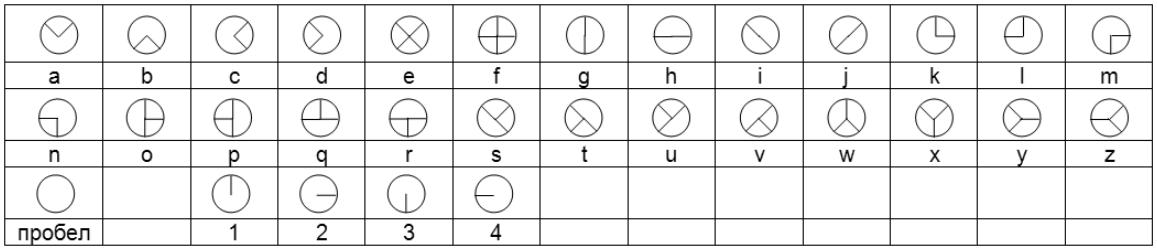


Рис. 1.25. Таблица шифрозамен и управляющих символов

Управляющие символы в последней строке таблицы служат для изменения кода (выбора шифралфавита) для зашифрования/дешифрования. Линии внутри каждого кружка фактически являются стрелками, подсказывающими адресату, как держать лист бумаги. Символ 1 означает, что лист надо держать как обычно: верх и низ расположены на своих местах, а сообщение читается слева направо. Символ 2 требует поворота на 90° вправо, а символ 3 указывает, что лист бумаги следует перевернуть вверх ногами. Символ 4 обозначает поворот на 90° влево.

Эти дополнительные символы могут появляться перед любой строчкой текста, а также в ее середине.

Из нижеприведенного примера можно узнать настоящие имя и фамилию супергероя.

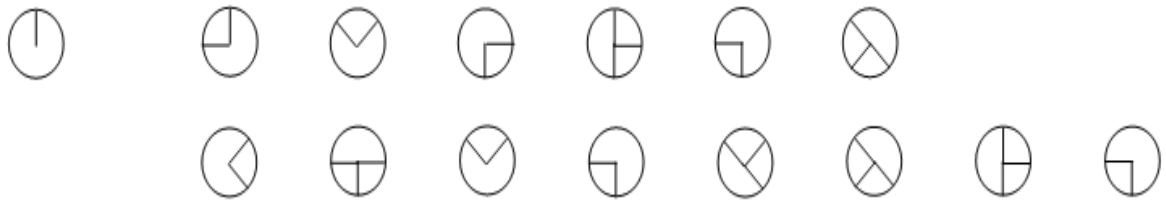


Рис. 1.26. Настоящие имя и фамилия Тени

Согласно первому управляющему символу, лист следует держать обычным образом, не поворачивая, и после замены буквы образуют «Lamont Cranston» (Ламонт Крэнстон).

Нерегулярные шифры

Еще одним направлением повышения стойкости шифров замены заключается в использовании нерегулярных шифров. В приведенных выше шифрах (**регулярных**) шифрозамены состоят из строго определенного количества символов (букв, цифр, графических элементов и т.д.) или в шифрограмме они отделяются друг от друга специальными символами (пробелом, точкой, запятой и т.д.). В нерегулярных шифрах

шифрозамены состоят из разного количества символов и записываются в шифрограмме в подряд (без выделения друг от друга), что значительно затрудняет криптоанализ.

Совмещенный шифр (совмещенная таблица). Данный шифр применялся еще семейством Ардженти - криптологами, разрабатывавшими шифры для Папы Римского в XVI в. В XX столетии этим способом пользовались коммунисты в ходе гражданской войны в Испании. В начале войны противники фашизма в Испании контролировали большинство крупных городов и защищали свою связь, включая радиопередачи, с помощью различных методов шифрования, в том числе совмещенных шифров.

Вариант коммунистов получил название «совмещенный» из-за необычного использования одно- и двухцифровых шифрозамен, благодаря чему сообщение приобретало дополнительную защиту от потенциального дешифровальщика. Некоторые буквы зашифровывались одной цифрой, другие же - парой цифр. При этом криптоаналитик противника совершенно не представлял, где в перехваченных сообщениях находятся одноцифровые, а где двухцифровые шифрозамены.

Таблица шифрозамен состоит из 10 столбцов с нумерацией 0, 9, 8, 7, 6, 5, 4, 3, 2 и 1. В начальную строку вписывается ключевое слово без повторяющихся букв. В последующие строки вписываются по десять не вошедших в него букв по порядку следования в алфавите. Строки, за исключением начальной, нумеруются по порядку, начиная с 1.

	0	9	8	7	6	5	4	3	2	1
	Д	Я	И	Н	А					
1	Б	В	Г	Е	Ё	Ж	З	Й	К	Л
2	М	О	П	Р	С	Т	У	Ф	Х	Ц
3	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю	-	-

Рис. 1.27. Пример таблицы шифрозамен совмещенного шифра с ключевым словом «ДЯДИНА»

При шифровании буквы исходного сообщения, входящие в ключевое слово, заменяются на одну цифру (номер столбца), остальные – двумя (номера строки и столбца). Например, для приведенной выше таблицы шифрозамен исходное сообщение «АБРАМОВ» будет зашифровано как «610276202919».

При получении шифрограммы адресат знает, что когда появляются цифры 1, 2 или 3, с ними обязательно связана еще одна цифра, поскольку они представляют собой цифровую пару. Так что 35 - это, несомненно, пара, а 53 - нет, ведь в таблице нет строки с номером 5. Перехват такого сообщения третьей стороной даст ей всего лишь ряд цифр, потому что криптоаналитик противника не имеет ни малейшего представления, какие цифры одиночные, а какие входят в состав пар.

Компьютерный практикум

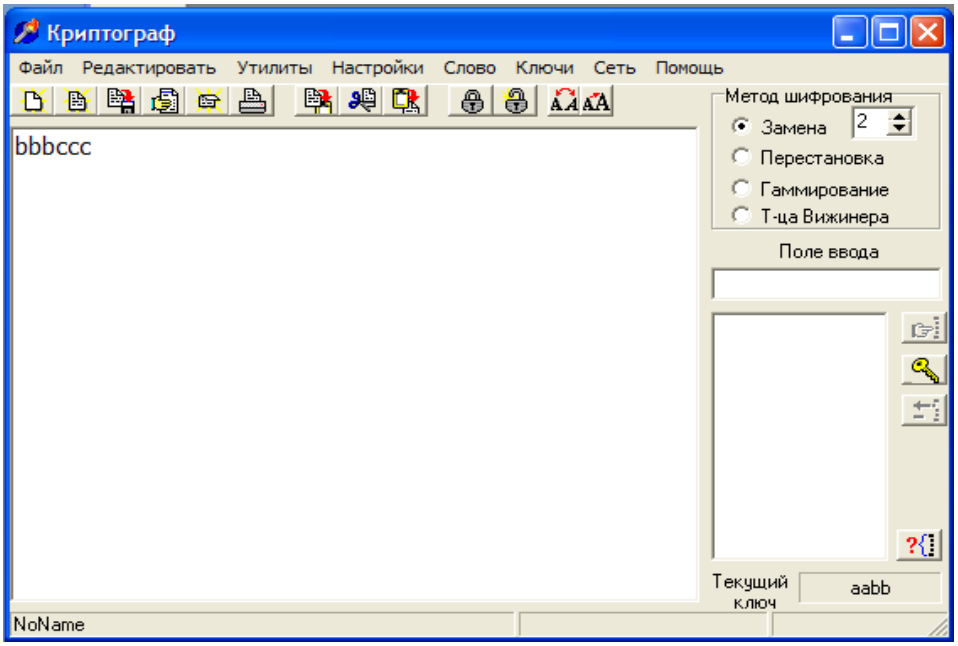
Задание 1

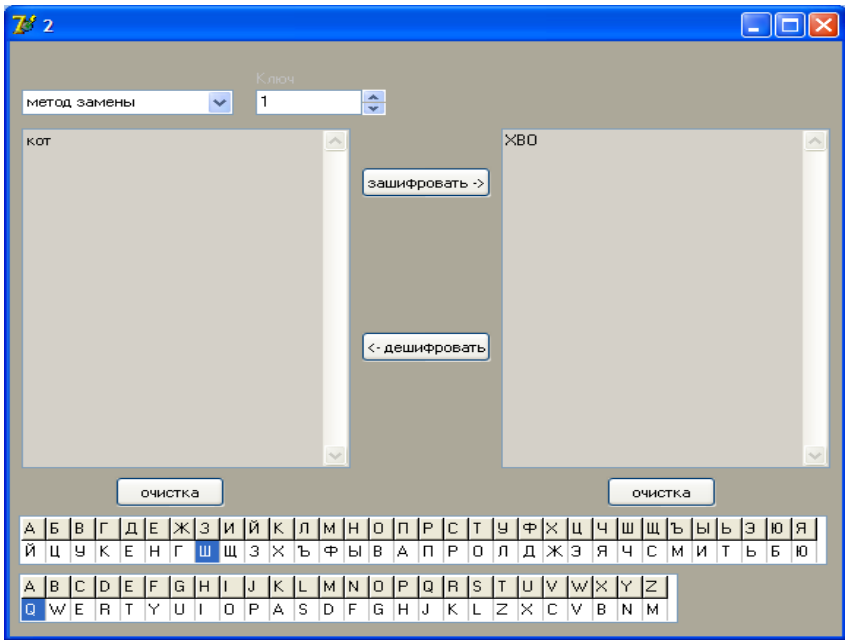
В данной работе изучаются три программы:

- Криптограф
- Шифр Цезаря

Криптограф

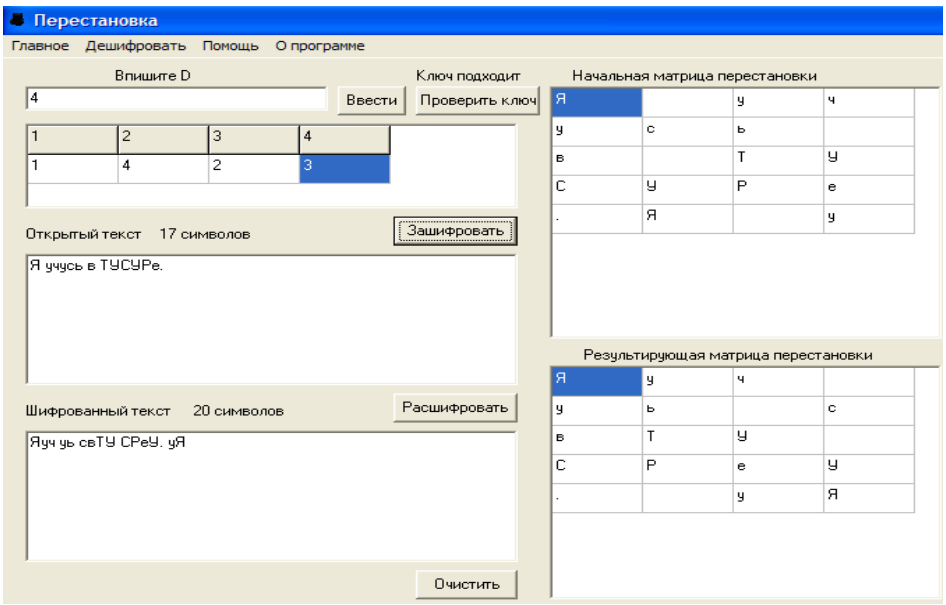
В программе реализуется шифр замены, перестановки, гаммирования и шифрование на основе таблицы Вижинера. В поле метод шифрования можно выбрать ключ, на который сдвигается шифруемое сообщение. Специальными кнопками можно выбрать шифрование или дешифрование.

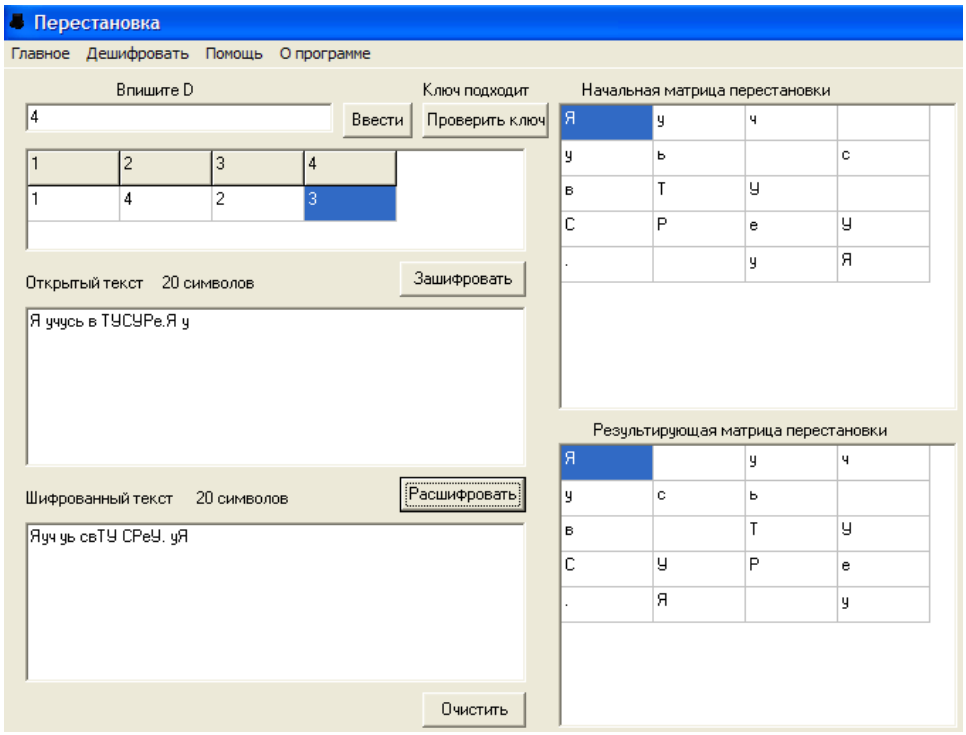




В данной программе шифр замены имеет специальную таблицу для шифрования, поэтому ключом является таблица.

Задание 2





Задания на самостоятельную работу по классическим шифрам

Задания на криптоанализ классических шифров

Шифр столбцовой перестановки

При решении заданий на криптоанализ шифров перестановки необходимо восстановить начальный порядок следования букв текста. Для этого используется анализ совместимости символов, в чем может помочь таблица сочетаемости.

Таблица 1. Сочетаемость букв русского языка

Г	С	Слева		Справа	Г	С
3	97	л, д, к, т, в, р, н	**А**	л, н, с, т, р, в, к, м	12	88
80	20	я, е, у, и, а, о	**Б**	о, ы, е, а, р, у	81	19
68	32	я, т, а, е, и, о	**В**	о, а, и, ы, с, н, л, р	60	40
78	22	р, у, а, и, е, о	**Г**	о, а, р, л, и, в	69	31
72	28	р, я, у, а, и, е, о	**Д**	е, а, и, о, н, у, р, в	68	32
19	81	м, и, л, д, т, р, н	**Е**	н, т, р, с, л, в, м, и	12	88
83	17	р, е, и, а, у, о	**Ж**	е, и, д, а, н	71	29
89	11	о, е, а, и	**З**	а, н, в, о, м, д	51	49

27	73	р, т, м, и, о, л, н	**И**	с, н, в, и, е, м, к, з	25	75
55	45	ь, в, е, о, а, и, с	**К**	о, а, и, р, у, т, л, е	73	27
77	23	г, в, ы, и, е, о, а	**Л**	и, е, о, а, ь, я, ю, у	75	25
80	20	я, ы, а, и, е, о	**М**	и, е, о, у, а, н, п, ы	73	27
55	45	д, ь, н, о, а, и, е	**Н**	о, а, и, е, ы, н, у	80	20
11	89	р, п, к, в, т, н	**О**	в, с, т, р, и, д, н, м	15	85
65	35	в, с, у, а, и, е, о	**П**	о, р, е, а, у, и, л	68	32
55	45	и, к, т, а, п, о, е	**Р**	а, е, о, и, у, я ,ы, н	80	20
69	31	с, т, в, а, е, и, о	**С**	т, к, о, я, е, ь, с, н	32	68
57	43	ч, у, и, а, е, о, с	**Т**	о, а, е, и, ь, в, р, с	63	37
15	85	п, т, к, д, н, м, р	**У**	т, п, с, д, н, ю, ж	16	84
70	30	н, а, е, о, и	**Ф**	и, е, о, а, е, о, а	81	19
90	10	у, е, о, а, ы, и	**Х**	о, и, с, н, в, п, р	43	57
69	31	е, ю, н, а, и	**Ц**	и, е, а, ы	93	7
82	18	е, а, у, и, о	**Ч**	е, и, т, н	66	34
67	33	ь, у, ы, е, о, а, и, в	**Ш**	е, и, н, а, о, л	68	32
84	16	е, б, а, я, ю	**Щ**	е, и, а	97	3
0	100	м, р, т, с, б, в, н	**Ы**	л, х, е, м, и, в, с, н	56	44
0	100	н, с, т, л	**Ь**	н, к, в, п, с, е, о, и	24	76
14	86	с, ы, м, л, д, т, р, н	**Э**	н, т, р, с, к	0	100
58	42	ь, о, а, и, л, у	**Ю**	д, т, щ, ц, н, п	11	89
43	57	о, н, р, л, а, и,	**Я**	в, с, т, п, д, к, м, л	16	84

		c				

Таблица . Сочетаемость букв английского языка

Г	С	Слева		Справа	Г	С
19	81	l,c,d,m,n,s,w,t,r,e,h	**A**	n,t,s,r,l,d,c,m	6	94
55	45	y,b,n,t,u,d,o,s,a,e	**B**	e,l,u,o,a,y,b,r	70	30
61	39	u,o,s,n,a,i,l,e	**C**	h,o,e,a,i,t,r,l,k	59	41
52	48	r,i,l,a,n,e	**D**	e,i,t,a,o,u	54	46
8	92	c,b,e,m,v,d,s,l,n,t,r,h	**E**	r,d,s,n,a,t,m,e,c,o	21	79
69	31	s,n,f,d,a,i,e,o	**F**	t,o,e,i,a,r,f,u	52	48
36	64	o,d,u,r,i,e,a,n	**G**	e.h.o.r.a.t.f.w.i.s	42	58
7	93	g,e,w,s,c,t	**H**	e,a,i,o	90	10
13	87	f,m,w,e,n,l,d,s,r,h,t	**I**	n,t,s,o,c,r,e,m,a,l	17	83
28	72	y,w,t,s,n,e,c,b,a,c	**J**	u,o,a,e,m,w	88	12
53	47	y,u,i,n,a,r,o,c	**Ќ**	e,i,n,a,t,s	68	32
52	48	m,p,t,i,b,u,o,e,l,a	**L**	e,i,y,o,a,d,u	65	35
69	31	s,d,m,r,i,a,o,e	**M**	e,a,o,i,p,m	71	29
89	11	u,e,o,a,i	**N**	d,t,g,e,a,s,o,i,c	32	68
21	79	o,d,l,p,h,n,e,c,f,s,i,r,t	**O**	n,f,r,u,t,m,l,s,w,o	18	82
47	53	r,l,t,n,i,p,m,a,o,u,e,s	**P**	o,e,a,r,l,u,p,t,i,s	59	41
20	80	o,n,l,e,d,r,s	**Q**	u	10 0	0
70	30	p,i,u,t,a,o,e	**R**	e,o,a,t,i,s,y	61	39
48	52	d,t,o,u,r,n,s,i,a,e	**S**	t,e,o,i,s,a,h,p,u	41	59
43	57	u,o,d,t,f,e,i,n,s.a	**T**	h,i,o,e,a,t,r	38	62
35	65	p,f,t,l,b,d,s,o	**И**	n,s,t,r,l,p,b,c	8	92
88	12	r,u,o,a,i,e	**V**	e,i,o,a	99	1
48	52	g,d,y,n,s,t,o,e	**W**	a,h,i,e,o,n	80	20
95	5	u,n,i,e	**X**	p,t,i,a,u,c,k,o	38	62
24	76	b,n,a,t,e,r,l	**Y**	a,o,s,t,w,h,i,e,d,m	38	62
88	12	o,n,a,i	**Z**	e,i,w	86	14

При анализе сочетаемости букв друг с другом следует иметь в виду зависимость появления букв в открытом тексте от значительного числа предшествующих букв. Для анализа этих закономерностей используют понятие условной вероятности.

Систематически вопрос о зависимости букв алфавита в открытом тексте от предыдущих букв исследовался известным русским математиком А.А.Марковым (1856 — 1922). Он доказал, что появления букв в открытом тексте нельзя считать независимыми друг от друга. В связи с этим А. А. Марковым отмечена еще одна устойчивая закономерность открытых текстов, связанная с чередованием гласных и согласных букв. Им были подсчитаны частоты встречаемости биграмм вида гласная-гласная (*г,г*), гласная-согласная (*г,с*), согласная-гласная (*с,г*), согласная-согласная (*с,с*) в русском тексте длиной в 10^5 знаков. Результаты подсчета отражены в следующей таблице:

Таблица 1.12. Чередование гласных и согласных

	Г	С	Всего
Г	6588	38310	44898
С	38296	16806	55102

Пример решения:

Дан шифр-текст: СВПООЗЛУЙЬСТЬ_ЕДПСОКОКАЙЗО

Текст содержит 25 символов, что позволяет записать его в квадратную матрицу 5х5. Известно, что шифрование производилось по столбцам, следовательно, расшифрование следует проводить, меняя порядок столбцов.

С	В	П	О	О
З	Л	У	Й	Ь
С	Т	Ь	_	Е
Д	П	С	О	К
К	А	Й	З	О

Необходимо произвести анализ совместимости символов (Таблица сочетаемости букв русского и английского алфавита, а также таблицы частот биграмм представлена выше). В первом и третьем столбце сочетание СП является крайне маловероятным для русского языка, следовательно, такая последовательность столбцов быть не может. Рассмотрим другие

запрещенные и маловероятные сочетания букв: ВП (2,3 столбцы), ПС (3,1 столбцы), ПВ (3,2 столбцы). Перебрав их все, получаем наиболее вероятные сочетания биграмм по столбцам:

В	О	С	П	О
Л	Ь	З	У	Й
Т	Е	С	Ь	_
П	О	Д	С	К
А	З	К	О	Й

Получаем осмысленный текст: ВОСПОЛЬЗУЙТЕСЬ_ПОДСКАЗКОЙ

Задание: Расшифровать фразу, зашифрованную столбцовой перестановкой.

1. ОКЕСНВРП_ЫРЕАДЕЫН_В_РСИКО
2. ДСЛИЕЗТЕА_Ь_ЛЬЮВМИ_ _АОЧХК
3. НМВИАИ_НЕВЕ_СМСТУОРДИАНКМ
4. ЕДСЗЬНДЕ_МУБД_УЭ_КРЗЕМНАЫ
5. СОНРЧОУО_ХДТ_ИЕИ_ВЗКАТРРИ
6. _ОНКА_БНЫЕЦВЛЕ_К_ТГОАНЕИР
7. НЗМАЕЕАА_Г_НОТВОССОТЬЯАЛС
8. РППОЕААДТВЛ_ЕБЬЛНЫЕ_ПА_ВР
9. ОПЗДЕП_ИХРДОТ_И_ВРИТЧ_САА
10. ВКЫОСИРЙУ_ОЬВНЕ_СОАПНИОТС
11. ПКТИРАОЛНАОИЧ_З_ЕСЬНЕЛНЖО
12. ИПКСОЕ_ТСМНАЧИ_ОЕН_ГДЕЛА_
13. АМВИННЬТЛЕАНЕ_ЙОВ_ОПХАРТО
14. АРЫКЗЫ_КЙТНЛ_ААЫ_ОЛБКЫТРТ
15. _ПАРИИВИАРЗ_БРА_ИСТЬЛТОЕК
16. П_ЛНАЭУВКАА_ЦИИВР_ОКЧЕДРО
17. ЖВНОАН_АТЗОЬСН_ЫО_ФВИИКИЗ
18. ОТВГОСЕЬЬТАДВ_С_ЬЗАТТЕЫАЧ
19. ЯАМРИТ_ДЖЕХ_СВЕД_ТСУВЕТНО
20. УЬБДТ_ОЕГТВ_ОЫКЭА_ВКАИУЦИ
21. ЛТБЕЧЛЖЫЕ_ _ОАПТЖРДУ_ЛМНОА
22. ИТПРКРФАГО_АВЯИА_ЯНЖУАКАН
23. ПКЕЕРРПО_ЙУСТ_ИТПСУТЛЯЕИН
24. ИЬЖЗНСД_ТДН_ЕТ_НУВЕУРЫГОЫ

25. ЕОУРВА_НЬРИАДИЦЕПИ_РНШВЫЕ

Шифр двойной перестановки

Пример решения:

Дан шифр-текст: ЫОЕЧТТОУ_СНСОРЧТРНАИДЬН_Е

Текст содержит 25 символов, что позволяет записать его в квадратную матрицу 5х5. известно, что шифрование производилось сначала по столбцам, а затем по строкам, следовательно, расшифрование следует проводить тем же способом.

Ы	О	Е	Ч	Т
Т	О	У	_	С
Н	С	О	Р	Ч
Т	Р	Н	А	И
Д	Ь	Н	_	Е

Производим анализ совместимости символов. Если в примере столбцовой перестановки можно было легко подобрать нужную комбинацию путем перебора, то здесь лучше воспользоваться таблицей частот букв русского языка (см. приложение). Для оптимизации скорости выполнения задания можно проверить все комбинации букв только в первой строке. Получаем ОЕ-15, ОЧ-12, ЕТ-33, ТЕ-31, ЧО-х, ЕО-7, ЧЫ-х, ОЫ-х, ТЫ-11, ТЧ-1, ЧЕ-23 (где х-запрещенная комбинация).

Из полученных результатов можно предположить следующую комбинацию замены столбцов **2 4 3 5 1:**

О	Ч	Е	Т	Ы
О	_	У	С	Т
С	Р	О	Ч	Н
Р	А	Н	И	Т
Ь	_	Н	Е	Д

Теперь необходимо переставить строки в нужном порядке. **3 2 4 5 1:**

С	Р	О	Ч	Н
О	_	У	С	Т

Р	А	Н	И	Т
Ь	_	Н	Е	Д
О	Ч	Е	Т	Ы

Получаем осмысленный текст: СРОЧНО_УСТРАНИТЬ_НЕДОЧЕТЫ

Задание: Расшифровать фразу, зашифрованную двойной перестановкой (сначала были переставлены столбцы, затем строки)

1. СЯСЕ_ _ЛУНЫИАККННОГЯДУЧАТН
2. МСЕЫ_ЛЫВЕНТОСАНТУЕИ_РЛПОБ
3. АМНРИД_УЕБСЫ_ЕЙРСООКОТНВ_
4. ОПЧУЛС_БООНЕВ_ОЖАЕОНЕЩЕИН
5. ЕШИАНИРЛПГЕЧАВРВ_СЕЫНА_ЛО
6. АРАВНРСВЕЕОАВ_ЗАНЯА_КМРЕИ
7. А_ЛТАВЙООЛСО_ТВ_ШЕЕНЕСТ_Ь
8. ФИ_ЗИММУЫНУУБК_Е_ДЬШЫИВЧУ
9. ВР_ЕСДЕИ_ТПХРОИ_ЗБУАДНУА_
10. ЦТААЙПЕЕ_ТБГУРРСВЬЕ_ОРЗВВ
11. АВАРНСЧАА_НЕДВЕДЕРПЕОЙ_ИС
12. ДОПК_СОПАЛЕЧНЛ_ГИНЙОИЖЕ_Т
13. ЛУАЗИЯНСА_ДТДЕАИ_ШРФЕОНГ_
14. С_ОЯНВ_СЬСЛААВРЧЕАРТОГДЕС
15. ЗШАФИПРАЛОЕНЖ_ОЬН_ДАРВОНА
16. КЭЕ_ТДУМБ_ЬСЗЕДНЕЗМАОР_ТУ
17. _ЕАЛЯРАНВЯАЧДА_ЕРПЕСАНВ_Ч
18. _И_ЕНТРЗИ_ОКЕВНОДЛЕША_ИМП
19. РОБДОЕВПС_МСХЬА_ _ИВПСНИОТ
20. ЕСДНОГТЕАНН_НЕОВМР_ЕУНПТЕ
21. _ЙЕСТОВО_НИИНЛАЕТИЖДСОПВ_
22. НДИАЕОЫЛППНЕ_ _НВЕАНГТ_ИЗЛА
23. П_БИРДЛЬНЕВ_ОП_ОПЗДЕВЫГЕА
24. МДООИТЕЬ_СМТ_НАДТЕСУБЕХНО
25. АИНАЛЖНОЛЕШФ_ЗИ_УАРОЬСНЕ_

Пример решения самостоятельной работы

15 вариант

Скользящая перестановка

Текст для расшифровки: _ПАРИИВИАРЗ_БРА_ИСТЬЛТОЕК

Текст содержит 25 символов, т.е. записываем его в таблицу 5*5.

_	П	А	Р	И
И	В	И	А	Р
З	_	Б	***Р***	***А***
_	И	С	***Т***	***Ь***
Л	Т	О	***Е***	***К***

Расшифровку следует проводить меня порядок столбцов.

Воспользуемся таблицей сочетаемости букв. 4 и 5 столбцы идут друг за другом, т.к. биграммы РА, ТЬ и ЕК наиболее распространенные. 2 и 4 столбец идут друг за другом, т.к. биграммы ИТ и ТЕ тоже распространены. По выше перечисленным признакам не трудно догадаться, что столбцы будут располагаться в следующем порядке: 2,4,5,3,1. И зашифрованной фразой будет: ***При_аварии_разбить_стекло.***

В данном случае дешифровать текст можно было обычным методом перебора, не обращаясь к таблице сочетаемости букв.

Шифр двойной перестановки

Текст для расшифровки: ЗШАФИПРАЛОЕНЖ_ОЬН_ДАРВОНА.

Текст содержит 25 символов, т.е. записываем его в таблицу 5*5.

З	Ш	А	Ф	И
П	Р	А	Л	О
Е	Н	Ж	_	О
Ь	Н	_	Д	А
Р	В	О	Н	А

Расшифровку следует проводить меня порядок столбцов и строк.

Глядя на зашифрованный текст по первым пяти символам можно сразу предположить, что столбцы меняются следующим образом: 1,3,2,5,4.

З	А	Ш	И	Ф
П	А	Р	О	Л
Е	Ж	Н	О	_
Ь	_	Н	А	Д
Р	О	В	А	Н

Глядя на вторую таблицу можно, также, без труда определить порядок строк: 2,4,3,1,5.

Расшифрованный текст: Пароль_надежно_зашифрован.

Шифр простой замены

Расшифрованный текст:

ЧЕМ ДАЛЬШЕ, ТЕМ СИЛЬНЕЕ ОН ЧУВСТВОВАЛ НЕШУТОЧНОЕ РАЗДРАЖЕНИЕ, ПОРОЮ ПЕРЕХОДИВШЕЕ В ПРИЛИВЫ ЗЛОСТИ-ОТТОГО, ЧТО ОНИ ЧЕТВЕРО СУТОК, ОБРАТИВШИСЬ В ЗРЕНИЕ И СЛУХ, ТОРЧАЛИ В ЧАЩОБЕ, КАК ДИКИЕ ОБЕЗЬЯНЫ ИЗ БРАЗИЛИИ, ОТТОГО, ЧТО ПОДВЕРНУЛСЯ ТУПОЙ КАЙМАН, С ОДИНАКОВЫМ УСЕРДИЕМ НАПАДАВШИЙ И НА ЛЕСНУЮ СВИНЬЮ, И НА ОТЛИЧНОГО ПАРНЯ С ДРУГОГО КОНТИНЕНТА А В ЭТО ВРЕМЯ ТЕ, НА БАЗЕ, ЖИЛИ В СВОЕ УДОВОЛЬСТВИЕ, СПАЛИ НА ЧИСТЕНЗЫЗИХ ПРОСТЫНКАХ В КОНДИЦИОНИРОВАННОЙ ПРОХЛАДЕ, ПРИНИМАЛИ ДУШ, ЖРАЛИ НА ЗАВТРАК ФРУКТЫ, ДЖЕМ И БИФШТЕКСЫ В ТРИ ПАЛЬЦА ТОЛЩИНОЙ, И ОКНА ТАК УЮТНО СВЕТИЛИСЬ, И МУЗЫКА ИГРАЛА, И ФУТБОЛ ПО ТЕЛЕВИЗОРУ

НИЧЕГО В ЭТОЙ ЗЛОСТИ НЕ БЫЛО ПЛОХОГО, НАОБОРОТ - ТАКОЙ НАСТРОЙ КАК РАЗ И ПРИДАЕТ БОЕВОГО КУРАЖА…

А ПОТОМ ПРИШЕЛ КОНЕЦ И ПОСТОРОННИМ МЫСЛЯМ И БЕЗДЕЛЬЮ. МОРСКОЙ ЗМЕЙ НАКОНЕЦ-ТО ПОДАЛ ЗНАК, КОТОРОГО ОНИ ЖДАЛИ ЧЕТВЕРО СУТОК, И ЭТО БЫЛО СЛОВНО МЕДНЫЙ РЕВ БОЕВОЙ ТРУБЫ, ЭТО ОЗНАЧАЛО, ЧТО НАЧАЛИСЬ РАБОТЫ, И НИЧЕГО УЖЕ НЕ ИЗМЕНИТЬ, НЕ ОСТАНОВИТЬ, НЕ ПЕРЕИГРАТЬ…

2. ШИФРОВАНИЕ С СЕКРЕТНЫМ КЛЮЧОМ

Теория шифров с секретным ключом

Блочные и поточные системы шифрования

Используемые в настоящее время системы шифрования делятся на два класса: блочные и поточные системы. Основной критерий такого разделения – мощность алфавита, над знаками которого производится операция шифрования. Если открытый текст перед шифрованием разбивается на блоки, состоящие из нескольких знаков, то есть исходное сообщение обрабатывается блоками, то мы имеем дело с блочным шифром. Если каждый знак сообщения шифруется отдельно, то такой шифр – поточный.

Разделение шифров на поточные и блочные связано с алгоритмическими и техническими особенностями реализации шифрующих преобразований, использующими возможности существующей элементной базы (разрядность процессоров, быстродействие микросхем, объем памяти компьютера). При увеличении мощности алфавита необходимо исследовать, прежде всего, вопросы о выборе преобразований, реализуемых криптосхемой, и способе их практической реализации, влияющем на эффективность функционирования криптосхемы с точки зрения эксплуатационных характеристик.

Естественно, что точное значение мощности алфавита, начиная с которого шифр следует считать уже не поточным, а блочным, назвать нельзя. Более того, с развитием техники эта характеристика меняется в сторону увеличения. Например, в настоящее время используются 16- и 32-разрядные процессоры, а перспективная шифровальная техника проектируется уже на 64-разрядных процессорах. Поэтому при построении поточных шифров могут быть использованы алфавиты мощности 2^{32} и 2^{64}.

Принципы построения блочных шифров

Как правило, алфавитом, на котором действует блочный шифр, является множество двоичных векторов-блоков открытого текста одинаковой длины (64, 128 и т. д.).

К.Шеннон сформулировал общий принцип построения шифрующих преобразований – принцип "*перемешивания*". Суть его состоит в требовании, чтобы применение шифрующего преобразования к наборам аргументов, отличающихся в незначительном числе позиций, приводило к существенному изменению результата.

Блочные шифры реализуются путем многократного применения к блокам открытого текста некоторых базовых преобразований. Базовые преобразования должны удовлетворять ряду требований, обусловленных тем, что они, во-первых, должны быть просто реализуемым, в том числе, программным способом на ЭВМ, и, во-вторых, при небольшом числе итераций давать аналитически сложные преобразования.

Обычно используются базовые преобразования двух типов – сложные в криптографическом отношении локальные преобразования над отдельными частями шифруемых блоков и простые преобразования, переставляющие между собой части шифруемых блоков. В криптографической литературе первые преобразования получили название "*перемешивающих*", а вторые – "*рассеивающих*". Качественно можно сказать, что перемешивание усложняет восстановление взаимосвязи статистических и аналитических свойств открытого и шифрованного текстов, а рассеивание распространяет влияние одного знака открытого текста на большое число знаков шифртекста, что позволяет сгладить влияние статистических свойств открытого текста на свойства шифртекста.

Алгоритм шифрования выполняет некоторое число циклов (итераций). Каждый цикл состоит в применении преобразований первого и второго типов. Такой принцип построения дает возможность реализовать каждый цикл шифрования с использованием однотипных узлов, а также выполнять расшифрование путем обработки данных в обратном направлении.

Удобной моделью для реализации базовых преобразований служат *регистры сдвига.* При этом рассеивающие преобразования определяются функциями обратной связи, а перемешивающие – сдвигами информации в регистре.

Получили распространение алгоритмы, в которых осуществляются преобразования над векторами, представляющими собой левую и правую половины содержимого регистра сдвига. Для построения таких алгоритмов часто используется конструкция, называемая *сетью Фейстеля* (Feistel Network) (рисунок 2.1).

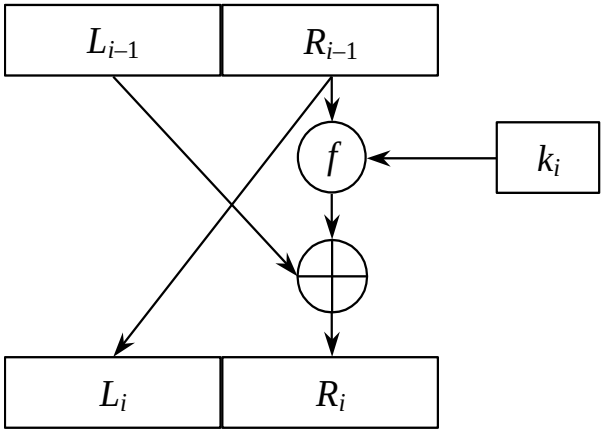


Рис. 2.1. Сеть Фейстеля

Преобразование, реализуемое сетью Фейстеля в i-м цикле шифрования, имеет вид

$$\begin{cases} L_i = R_{i-1}, \\ R_i = L_{i-1} \oplus f_i(R_{i-1}, k_i), \end{cases} \qquad (5)$$

где L_{i-1} и R_{i-1} – левая и правая части входного блокаL_i и R_i – левая и правая части блока, являющегося результатом зашифрования входного блока на ключе k_i с помощью функции f_i.

Алгоритм шифрования реализуется несколькими итерациями преобразования сети Фейстеля с использованием ключа k. При этом очередная (i-я) итерация использует в качестве входного блока результат предыдущей итерации и ключ k_i, вычисляемый определенным образом по ключу k. Функция f_i может зависеть или не зависеть от номера итерации.

Ценность преобразований подобного вида заключается в том, что даже если f_i не является обратимой функцией, преобразование сети Фейстеля обратимо. В самом деле, из (4.1) сразу следует, что

$$\begin{cases} L_{i-1} = R_i \oplus f_i(L_i, k_i), \\ R_{i-1} = L_i. \end{cases}$$

Стандарт шифрования данных ГОСТ 28147-89

В России установлен единый алгоритм криптографического преобразования данных – ГОСТ 28147-89. Этот алгоритм предназначен для аппаратной и программной реализации, удовлетворяет необходимым криптографическим требованиям и не накладывает ограничений на степень секретности защищаемой информации. Алгоритм реализует шифрование 64-битовых блоков данных с помощью 256-битового ключа.

ГОСТ 28147-89 содержит описание алгоритмов нескольких уровней. На самом верхнем находятся практические алгоритмы, предназначенные для шифрования массивов данных и выработки для них имитовставки. Все они опираются на три алгоритма низшего уровня, называемые в тексте ГОСТа ***циклами***. Будем называть эти фундаментальные алгоритмы ***базовыми циклами***, чтобы отличать их от всех прочих циклов. Они имеют следующие названия и обозначения, последние приведены в скобках и смысл их будет объяснен позже:

- цикл зашифрования (32-З);
- цикл расшифрования (32-Р);
- цикл выработки имитовставки (16-З).

В свою очередь, каждый из базовых циклов представляет собой многократное повторение одной единственной процедуры, которую для определенности будем называть ***основным шагом криптопреобразования***.

Таким образом, чтобы разобраться в ГОСТе, надо понять три следующие вещи:

1. что такое ***основной шаг*** криптопреобразования;
2. как из ***основных шагов*** складываются ***базовые циклы***;
3. как из трех ***базовых циклов*** складываются все практические алгоритмы ГОСТа.

Прежде чем перейти к изучению этих вопросов, следует поговорить о ключевой информации, используемой алгоритмами ГОСТа. В соответствии с принципом Кирхгофа, которому удовлетворяют все современные известные широкой общественности шифры, именно ее секретность обеспечивает секретность зашифрованного сообщения. В ГОСТе ключевая информация состоит из двух структур данных. Помимо собственно ***ключа,*** необходимого для всех шифров, она содержит еще и ***таблицу замен***:

1. ***Ключ*** является массивом из восьми 32-битных элементов кода и обозначается символом K: $K = \{K_i\}_{0 \le i \le 7}$. В ГОСТе элементы ключа используются

как 32-разрядные целые числа без знака: $0 \le K_i < 2^{32}$. Таким образом, размер ключа составляет 32×8=256 бит или 32 байта.

2. ***Таблица замен*** является матрицей 8×16, содержащей 4-битовые элементы, которые можно представить в виде целых чисел от 0 до 15. Строки ***таблицы замен*** называются ***узлами замен***, они должны содержать различные значения, то есть каждый ***узел замен*** должен содержать 16 различных чисел от 0 до 15 в произвольном порядке. Таблица замен обозначается символом H: $H = \{H_{i,j}\}_{0 \le j \le 15}^{0 \le i \le 7}$, $0 \le H_{i,j} \le 15$. Таким образом, общий объем таблицы замен равен: 8 узлов × 16 элементов/узел × 4 бита/элемент = 512 бит или 64 байта.

Основной шаг криптопреобразования.

Основной шаг криптопреобразования по своей сути является оператором, определяющим преобразование 64-битового блока данных. Дополнительным параметром этого оператора является 32-битовый блок, в качестве которого используется какой-либо элемент ключа. Схема алгоритма основного шага приведена на рисунке 2.2.

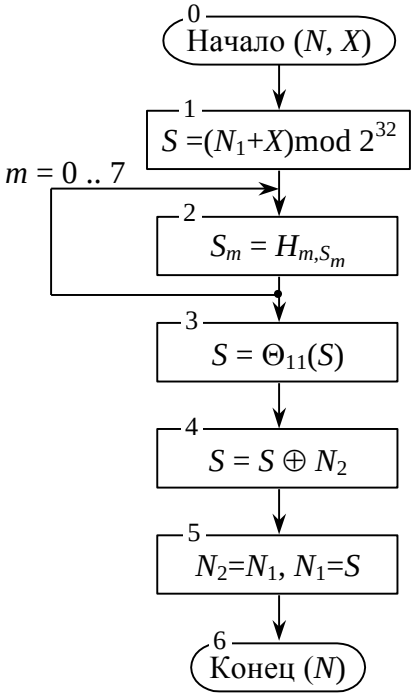


Рис. 2.2. Схема основного шага криптопреобразования алгоритма ГОСТ 28147-89

Ниже даны пояснения к алгоритму основного шага:

0. Определяет исходные данные для основного шага криптопреобразования:

- N – преобразуемый 64-битовый блок данных, в ходе выполнения шага его младшая (N_1) и старшая (N_2) части обрабатываются как отдельные 32-битовые целые числа без знака. Таким образом, можно записать $N=(N_1,N_2)$.
- X – 32-битовый элемент ключа;

1. Сложение с ключом. Младшая половина преобразуемого блока складывается по модулю 2^{32} с используемым на шаге элементом ключа, результат передается на следующий шаг;
2. Поблочная замена. 32-битовое значение, полученное на предыдущем шаге, интерпретируется как массив из восьми 4-битовых блоков кода:

$S = (S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7)$.

Далее значение каждого из восьми блоков заменяется на новое, которое выбирается по таблице замен следующим образом: значение блока S_i заменяется на S_i -тый по порядку элемент (нумерация с нуля) i-того узла замен (т.е. i-той строки таблицы замен, нумерация также с нуля). Другими словами, в качестве замены для значения блока выбирается элемент из таблицы замен с номером строки, равным номеру заменяемого блока, и номером столбца, равным значению заменяемого блока как 4-битового целого неотрицательного числа. Теперь становится понятным размер таблицы замен: число строк в ней равно числу 4-битных элементов в 32-битном блоке данных, то есть восьми, а число столбцов равно числу различных значений 4-битного блока данных, равному как известно 2^4, шестнадцати.

3. Циклический сдвиг на 11 бит влево. Результат предыдущего шага сдвигается циклически на 11 бит в сторону старших разрядов и передается на следующий шаг. На схеме алгоритма символом Θ_{11} обозначена функция циклического сдвига своего аргумента на 11 бит в сторону старших разрядов.
4. Побитовое сложение: значение, полученное на шаге 3, побитно складывается по модулю 2 со старшей половиной преобразуемого блока.
5. Сдвиг по цепочке: младшая часть преобразуемого блока сдвигается на место старшей, а на ее место помещается результат выполнения предыдущего шага.
6. Полученное значение преобразуемого блока возвращается как результат выполнения алгоритма основного шага криптопреобразования.

Базовые циклы криптографических преобразований.

ГОСТ относится к классу блочных шифров, то есть единицей обработки информации в нем является блок данных. Следовательно, вполне логично ожидать, что в нем будут

определены алгоритмы для криптографических преобразований, то есть для зашифрования, расшифрования и «учета» в контрольной комбинации одного блока данных. Именно эти алгоритмы и называются ***базовыми циклами*** ГОСТа, что подчеркивает их фундаментальное значение для построения этого шифра.

Базовые циклы построены из ***основных шагов*** криптографического преобразования, рассмотренного в предыдущем разделе. В процессе выполнения основного шага используется только один элемент ключа, в то время как ключ ГОСТ содержит восемь таких элементов. Следовательно, чтобы ключ был использован полностью, каждый из базовых циклов должен многократно выполнять основной шаг с различными его элементами. Вместе с тем кажется вполне естественным, что в каждом базовом цикле все элементы ключа должны быть использованы одинаковое число раз, по соображениям стойкости шифра это число должно быть больше одного.

Все сделанные выше предположения, опирающиеся просто на здравый смысл, оказались верными. Базовые циклы заключаются в многократном выполнении *основного шага* с использованием разных элементов ключа и отличаются друг от друга только числом повторения шага и порядком использования ключевых элементов. Ниже приведен этот порядок для различных циклов.

1. Цикл зашифрования 32-З:

$K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0$.

2. Цикл расшифрования 32-Р:

$K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0$.

3. Цикл выработки имитовставки 16-З:

$K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7$.

Каждый из циклов имеет собственное буквенно-цифровое обозначение, соответствующее шаблону «*n*-*X*», где первый элемент обозначения (*n*), задает число повторений основного шага в цикле, а второй элемент обозначения (*X*), буква, задает порядок зашифрования («З») или расшифрования («Р») в использовании ключевых элементов. Этот порядок нуждается в дополнительном пояснении:

Цикл расшифрования должен быть обратным циклу зашифрования, то есть последовательное применение этих двух циклов к произвольному блоку должно дать в итоге исходный блок, что отражается следующим соотношением:

$Ц_{32\text{-}Р}(Ц_{32\text{-}З}(T))=T$,

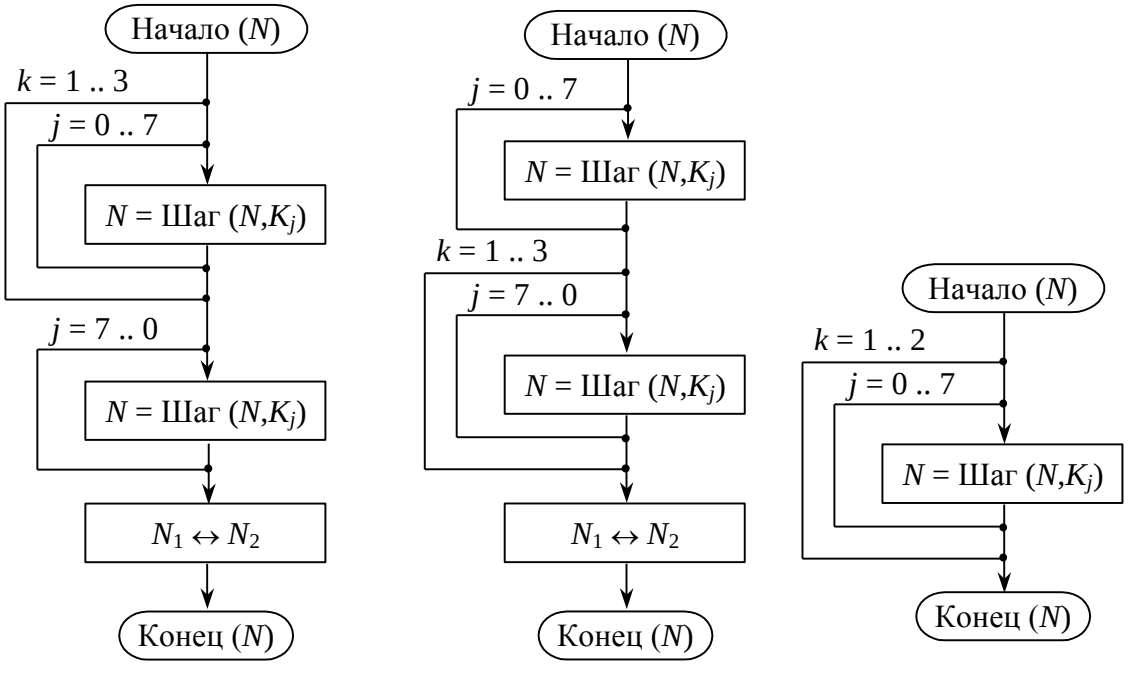
где T – произвольный 64-битный блок данных,

$Ц_X(T)$ – результат выполнения цикла X над блоком данных T.

Для выполнения этого условия для алгоритмов, подобных ГОСТу, необходимо и достаточно, чтобы порядок использования ключевых элементов соответствующими циклами был взаимно обратным. В справедливости записанного условия для рассматриваемого случая легко убедиться, сравнив приведенные выше последовательности для циклов 32-З и 32-Р. Из сказанного вытекает одно интересное следствие: свойство цикла быть обратным другому циклу является взаимным, то есть цикл 32-З является обратным по отношению к циклу 32-Р. Другими словами, зашифрование блока данных теоретически может быть выполнено с помощью цикла расшифрования, в этом случае расшифрование блока данных должно быть выполнено циклом зашифрования. Из двух взаимно обратных циклов любой может быть использован для зашифрования, тогда второй должен быть использован для расшифрования данных, однако стандарт ГОСТ28147-89 закрепляет роли за циклами и не предоставляет пользователю права выбора в этом вопросе.

Цикл выработки имитовставки вдвое короче циклов шифрования, порядок использования ключевых элементов в нем такой же, как в первых 16 шагах цикла зашифрования, в чем нетрудно убедиться, рассмотрев приведенные выше последовательности, поэтому этот порядок в обозначении цикла кодируется той же самой буквой «З».

Схемы базовых циклов приведены на рисунках 4.3 а-в. Каждый из них принимает в качестве аргумента и возвращает в качестве результата 64-битный блок данных, обозначенный на схемах N.



а) б) в)

Рис. 2.3. Схемы цикла зашифрования 32-З (а), расшифрования 32-Р (б) и выработки имитовставки 16-З (в)

Символ Шаг(N,X) обозначает выполнение основного шага криптопреобразования для блока N с использованием ключевого элемента X. Между циклами шифрования и вычисления имитовставки есть еще одно отличие, не упомянутое выше: в конце базовых циклов шифрования старшая и младшая часть блока результата меняются местами, это необходимо для их взаимной обратимости.

Основные режимы шифрования.

Для решения разнообразных криптографических задач блочные шифры используют в нескольких режимах работы. ГОСТ 28147-89 предусматривает три следующих режима шифрования данных:

- простая замена,
- гаммирование,
- гаммирование с обратной связью,

и один дополнительный режим выработки имитовставки.

В любом из этих режимов данные обрабатываются блоками по 64 бита, на которые разбивается массив, подвергаемый криптографическому преобразованию, именно поэтому ГОСТ относится к блочным шифрам. Однако в двух режимах гаммирования есть возможность обработки неполного блока данных размером меньше 8 байт, что существенно при шифровании массивов данных с произвольным размером, который может быть не кратным 8 байтам.

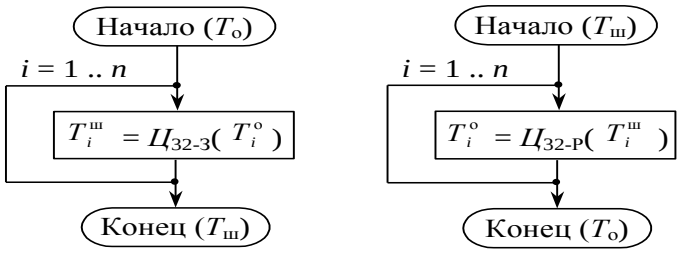
Введем некоторые обозначения:

- $T_{\text{о}}$, $T_{\text{ш}}$ – массивы соответственно открытых и зашифрованных данных;
- $T_i^{\text{о}}$, $T_i^{\text{ш}}$ – i-тые по порядку 64-битные блоки соответственно открытых и зашифрованных данных: $T_{\text{о}} = (T_1^{\text{о}}, T_2^{\text{о}}, \ldots, T_n^{\text{о}})$, $T_{\text{ш}} = (T_1^{\text{ш}}, T_2^{\text{ш}}, \ldots, T_n^{\text{ш}})$, $1 \le i \le n$. Последний блок может быть неполным: $|T_i^{\text{о}}| = |T_i^{\text{ш}}| = 64$ при $1 \le i < n$, $1 \le |T_n^{\text{о}}| = |T_n^{\text{ш}}| \le 64$;
- n – число 64-битных блоков в массиве данных;
- $Ц_{\text{X}}$ – функция преобразования 64-битного блока данных по алгоритму базового цикла «X»;

Теперь опишем основные режимы шифрования.

Простая замена

Зашифрование в данном режиме заключается в применении цикла 32-З к блокам открытых данных, расшифрование – цикла 32-Р к блокам зашифрованных данных. Это наиболее простой из режимов, 64-битовые блоки данных обрабатываются в нем независимо друг от друга. Схемы алгоритмов зашифрования и расшифрования в режиме простой замены приведены на рисунках 2.4 а и 2.4 б соответственно.



а) б)

Рис. 2.4. Алгоритмы зашифрования (а) и расшифрования (б) данных в режиме простой замены

Размер массива открытых или зашифрованных данных, подвергающийся соответственно зашифрованию или расшифрованию, должен быть кратен 64 битам: $|T_о| = |T_ш| = 64 \times n$, после выполнения операции размер полученного массива данных не изменяется.

Режим шифрования простой заменой имеет следующие особенности:

1. Так как блоки данных шифруются независимо друг от друга и от их позиции в массиве, при зашифровании двух одинаковых блоков открытого текста получаются одинаковые блоки шифротекста и наоборот. Отмеченное свойство позволит криптоаналитику сделать заключение о тождественности блоков исходных данных, если в массиве зашифрованных данных ему встретились идентичные блоки, что является недопустимым для серьезного шифра.

2. Если длина шифруемого массива данных не кратна 8 байтам или 64 битам, возникает проблема, чем и как дополнять последний неполный блок данных массива до полных 64 бит. Эта задача не так проста, как кажется на первый взгляд, поскольку очевидные решения типа «дополнить неполный блок нулевыми битами» или, более обще, «дополнить неполный блок фиксированной комбинацией нулевых и единичных битов» могут при определенных условиях дать в руки криптоаналитика возможность методами перебора определить содержимое этого самого неполного блока, и этот факт означает снижение стойкости шифра. Кроме

того, длина шифротекста при этом изменится, увеличившись до ближайшего целого, кратного 64 битам, что часто бывает нежелательным.

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

Гаммирование

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

64 бита, использоваться может и часть такого блока с размером, равным размеру шифруемого блока.

Теперь перейдем непосредственно к описанию режима гаммирования. Гамма для этого режима получается следующим образом: с помощью некоторого алгоритмического рекуррентного генератора последовательности чисел (РГПЧ) вырабатываются 64-битные блоки данных, которые далее подвергаются преобразованию по циклу 32-З, то есть зашифрованию в режиме простой замены, в результате получаются блоки гаммы. Благодаря тому, что наложение и снятие гаммы осуществляется при помощи одной и той же операции побитового исключающего или, алгоритмы зашифрования и расшифрования в режиме гаммирования идентичны, их общая схема приведена на рисунке 2.5.

РГПЧ, используемый для выработки гаммы, является рекуррентной функцией:

$\Omega_{i+1} = f(\Omega_i)$,

где Ω_i – элементы рекуррентной последовательности,

f – функция преобразования.

Следовательно, неизбежно возникает вопрос о его инициализации, то есть об элементе Ω_0. В действительности, этот элемент данных является параметром алгоритма для режимов гаммирования, на схемах он обозначен как S, и называется в криптографии ***синхропосылкой***, а в нашем ГОСТе – ***начальным заполнением*** одного из регистров шифрователя. По определенным соображениям разработчики ГОСТа решили использовать для инициализации РГПЧ не непосредственно синхропосылку, а результат ее преобразования по циклу 32-З:

$\Omega_0 = Ц_{32\text{-}З}(S)$.

Последовательность элементов, вырабатываемых РГПЧ, целиком зависит от его начального заполнения, то есть элементы этой последовательности являются функцией своего номера и начального заполнения РГПЧ:

$\Omega_i = f_i(\Omega_0)$,

где $f_i(X) = f(f_{i-1}(X))$, $f_0(X) = X$.

С учетом преобразования по алгоритму простой замены добавляется еще и зависимость от ключа:

$\Gamma_i = Ц_{32\text{-}З}(\Omega_i) = Ц_{32\text{-}З}(f_i(\Omega_0)) = Ц_{32\text{-}З}(f_i(Ц_{32\text{-}З}(S))) = \varphi_i(S,K)$,

где Γ_i – i-тый элемент гаммы,

K – ключ.

Таким образом, последовательность элементов гаммы для использования в режиме гаммирования однозначно определяется ключевыми данными и синхропосылкой. Естественно, для обратимости процедуры шифрования в процессах за- и расшифрования

должна использоваться одна и та же синхропосылка. Из требования уникальности гаммы, невыполнение которого приводит к катастрофическому снижению стойкости шифра, следует, что для шифрования двух различных массивов данных на одном ключе необходимо обеспечить использование различных синхропосылок. Это приводит к необходимости хранить или передавать синхропосылку по каналам связи вместе с зашифрованными данными, хотя в отдельных особых случаях она может быть предопределена или вычисляться особым образом, если исключается шифрование двух массивов на одном ключе.

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

- период повторения последовательности чисел, вырабатываемой РГПЧ, не должен сильно (в процентном отношении) отличаться от максимально возможного при заданном размере блока значения 2^{64};
- соседние значения, вырабатываемые РГПЧ, должны отличаться друг от друга в каждом байте, иначе задача криптоаналитика будет упрощена;
- РГПЧ должен быть достаточно просто реализуем как аппаратно, так и программно на наиболее распространенных типах процессоров, большинство из которых, как известно, имеют разрядность 32 бита.

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

- в 64-битовом блоке старшая (Ω_i^1) и младшая (Ω_i^0) части обрабатываются независимо друг от друга: $\Omega_i=(\Omega_i^0,\Omega_i^1)$, $|\Omega_i^0| = |\Omega_i^1| = 32$, $\Omega_{i+1}^0 = f_{РГПЧ1}(\Omega_i^0)$, $\Omega_{i+1}^1 = f_{РГПЧ2}(\Omega_i^1)$, фактически, существуют два независимых РГПЧ для старшей и младшей частей блока.
- рекуррентные соотношения для старшей и младшей частей следующие:

$\Omega_{i+1}^0 = (\Omega_i^0 + C_1) \bmod 2^{32}$, где $C_1 = 1010101_{16}$;

$\Omega_{i+1}^1 = (\Omega_i^1 + C_2 - 1) \bmod (2^{32} - 1) + 1$, где $C_2 = 1010104_{16}$;

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

- период повторения последовательности для младшей части составляет 2^{32}, для старшей части 2^{32}–1, для всей последовательности период составляет $2^{32}\cdot(2^{32}–1)$, доказательство этого факта, весьма несложное, получите сами. Первая формула из двух реализуется за одну команду, вторая, несмотря на ее кажущуюся громоздкость, за две команды на всех современных 32-разрядных процессорах.

Схема алгоритма шифрования в режиме гаммирования приведена на рисунке 2.5.

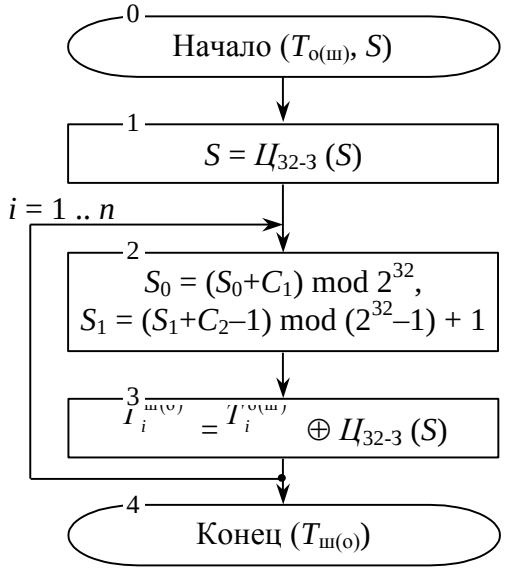


Рис. 2.5. Алгоритм зашифрования (расшифрования) данных в режиме гаммирования

Ниже изложены пояснения к схеме:

0. Определяет исходные данные для основного шага криптопреобразования:

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

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

2. Один шаг работы РГПЧ, реализующий его рекуррентный алгоритм. В ходе данного шага старшая (S_1) и младшая (S_0) части последовательности данных вырабатываются независимо друг от друга;

3. Гаммирование. Очередной 64-битный элемент, выработанный РГПЧ, подвергается процедуре зашифрования по циклу 32–З, результат используется как

элемент гаммы для зашифрования (расшифрования) очередного блока открытых (зашифрованных) данных того же размера.

4. Результат работы алгоритма – зашифрованный (расшифрованный) массив данных.

Ниже перечислены особенности гаммирования как режима шифрования.

1. Одинаковые блоки в открытом массиве данных дадут при зашифровании различные блоки шифротекста, что позволит скрыть факт их идентичности.

2. Поскольку наложение гаммы выполняется побитно, шифрование неполного блока данных легко выполнимо как шифрование битов этого неполного блока, для чего используется соответствующие биты блока гаммы . Так, для зашифрования неполного блока в 1 бит можно использовать любой бит из блока гаммы.

3. Синхропосылка, использованная при зашифровании, каким-то образом должна быть передана для использования при расшифровании. Это может быть достигнуто следующими путями:

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

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

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

Режим гаммирования имеет еще одну интересную особенность. В этом режиме биты массива данных шифруются независимо друг от друга. Таким образом, каждый бит шифротекста зависит от соответствующего бита открытого текста и, естественно, порядкового номера бита в массиве:

$$t_i^{\text{ш}} = t_i^{\text{о}} \oplus \gamma_i = f\left(t_i^{\text{о}}, i\right).$$

Из этого вытекает, что изменение бита шифротекста на противоположное значение приведет к аналогичному изменению бита открытого текста на противоположный:

$$\overline{t_i^{\text{ш}}} = t_i^{\text{ш}} \oplus 1 = \left(t_i^{\text{о}} \oplus \gamma_i\right) \oplus 1 = \left(t_i^{\text{о}} \oplus 1\right) \oplus \gamma_i = \overline{t_i^{\text{о}}} \oplus \gamma_i,$$

где $\bar{t}$ обозначает инвертированное по отношению к t значение бита ($\bar{0} = 1$, $\bar{1} = 0$).

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

Гаммирование с обратной связью

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

Схема алгоритмов за- и расшифрования в режиме гаммирования с обратной связью приведена на рисунке 2.6.

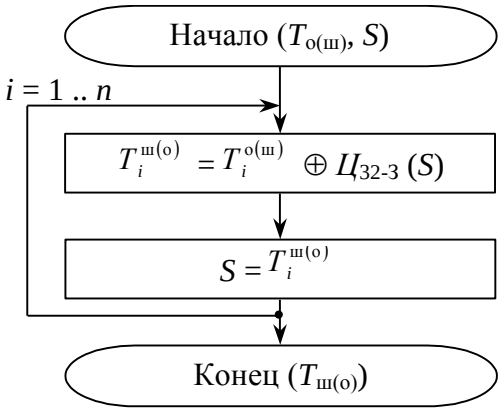


Рис. 2.6. Алгоритм зашифрования (расшифрования) данных в режиме гаммирования с обратной связью

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

$T_i^{о} = T_i^{ш} \oplus Г_i$ – гаммирование;

$T_i^{о} = T_i^{ш} \oplus Ц_{32-З}\left(T_{i-1}^{ш}\right)$ – гаммирование с обратной связью.

Если в режиме обычного гаммирования изменения в определенных битах шифротекста влияют только на соответствующие биты открытого текста, то в режиме гаммирования с обратной связью картина несколько сложнее. Как видно из соответствующего уравнения, при расшифровании блока данных в режиме гаммирования с обратной связью, блок открытых данных зависит от соответствующего и предыдущего блоков зашифрованных данных. Поэтому, если внести искажения в зашифрованный блок, то после расшифрования

искаженными окажутся два блока открытых данных – соответствующий и следующий за ним, причем искажения в первом случае будут носить тот же характер, что и в режиме гаммирования, а во втором случае – как в режиме простой замены. Другими словами, в соответствующем блоке открытых данных искаженными окажутся те же самые биты, что и в блоке шифрованных данных, а в следующем блоке открытых данных все биты независимо друг от друга с вероятностью 1/2 изменят свои значения.

Выработка имитовставки к массиву данных.

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

Для решения задачи обнаружения искажений в зашифрованном массиве данных с заданной вероятностью в ГОСТе предусмотрен дополнительный режим криптографического преобразования – выработка имитовставки. Имитовставка – это контрольная комбинация, зависящая от открытых данных и секретной ключевой информации. Целью использования имитовставки является обнаружение всех случайных или преднамеренных изменений в массиве информации. Проблема, изложенная в предыдущем пункте, может

быть успешно решена с помощью добавления к шифрованным данным имитовставки. Для потенциального злоумышленника две следующие задачи практически неразрешимы, если он не владеет ключевой информацией:

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

Схема алгоритма выработки имитовставки приведена на рисунке 2.7.

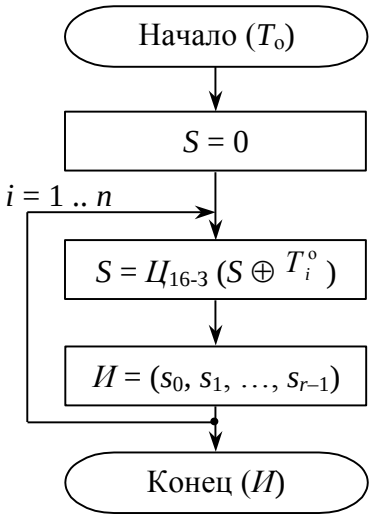


Рис. 2.7. Алгоритм выработки имитовставки для массива данных

В качестве имитовставки берется часть блока, полученного на выходе, обычно 32 его младших бита. При выборе размера имитовставки надо принимать во внимание, что вероятность успешного навязывания ложных данных равна величине $2^{-|И|}$ на одну попытку подбора. При использовании имитовставки размером 32 бита эта вероятность равна $2^{-32} \approx 0{,}23 \times 10^{-9}$.

Американский стандарт шифрования данных DES

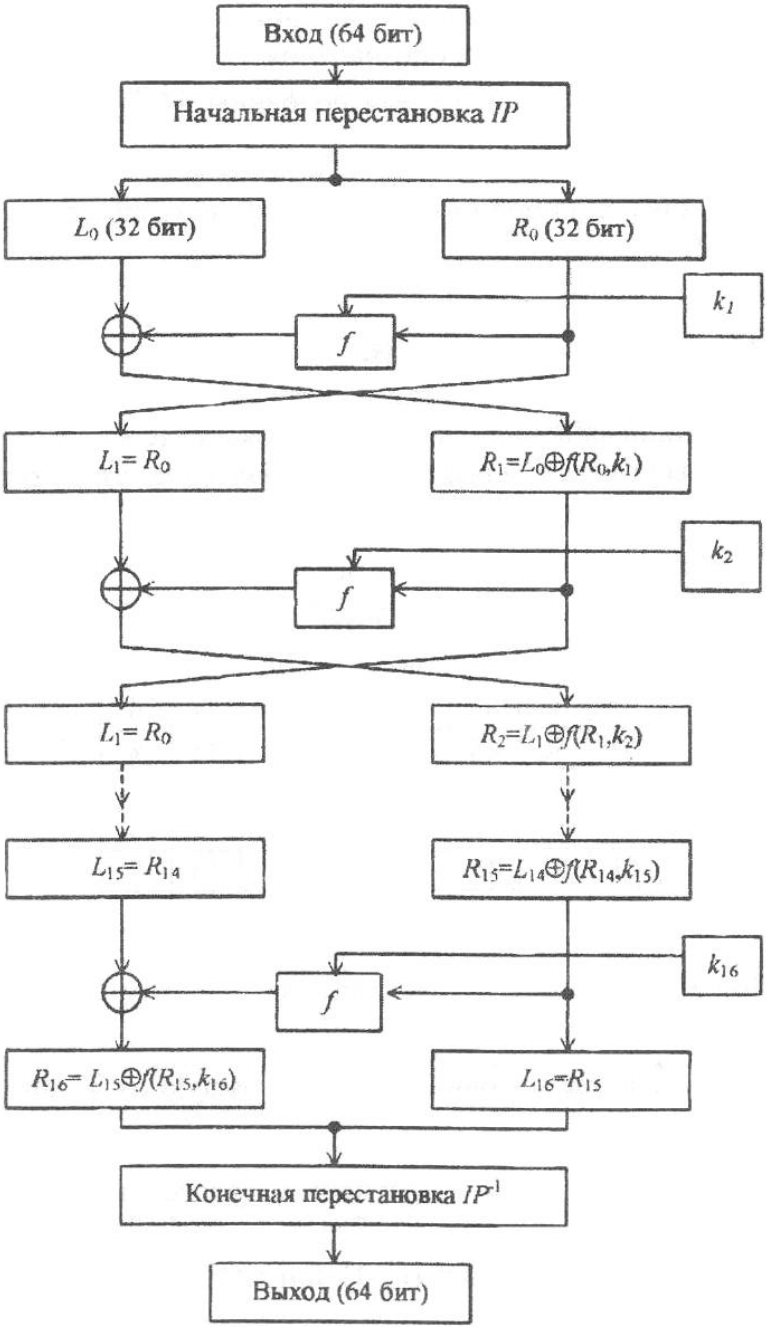
Стандарт шифрования данных DES (*Data Encryption Standard*) опубликован Национальным бюро стандартов США в 1977 г. В 1980 г. этот алгоритм был принят Национальным институтом стандартов и технологий США (НИСТ) в качестве стандарта шифрования данных для защиты от несанкционированного доступа к важной, но несекретной информации в государственных и коммерческих организациях США.

Основные достоинства алгоритма DES:

- простота ключевой системы (используется только один ключ длиной 56 бит);

- относительная простота алгоритма обеспечивает высокую скорость аппаратной и программной реализации;
- достаточно высокая криптографическая стойкость алгоритма шифрования.

Алгоритм DES использует комбинацию подстановок и перестановок. DES осуществляет шифрование 64-битовых блоков данных с помощью 56-битового ключа k, в котором значащими являются 56 бит (остальные 8 бит – проверочные биты для контроля на четность). Дешифрование в DES является операцией, обратной шифрованию, и выполняется путем повторения операций шифрования в обратной последовательности. Структура алгоритма DES изображена на рисунке 2.8.



L_i, R_i – левая и правая половины 64-битового блока;

$\oplus$ – операция побитового сложения блоков по модулю 2 (XOR);

k_i – 48-битовые ключи, получаемые из исходного ключа k;

f – функция шифрования;

IP – начальная перестановка степени 64.

Рис. 2.8. Структура алгоритма DES

Процесс шифрования состоит в начальной перестановке битов 64-битового входного блока, шестнадцати циклах шифрования и, наконец, конечной перестановке битов.

Отметим, что все приводимые ниже таблицы являются стандартными и должны использоваться при реализации алгоритма DES в неизменном виде. Все перестановки и коды в таблицах подобраны разработчиками таким образом, чтобы максимально затруднить процесс вскрытия шифра путем подбора ключа.

При зашифровании очередного блока T его биты подвергаются *начальной перестановке IP* согласно приводимой таблице 2.1.

Таблица 2.1. Начальная перестановка *IP*

8	0	2	4	6	8	0	
0	2	4	6	8	0	2	
2	4	6	8	0	2	4	
4	6	8	0	2	4	6	
7	9	1	3	5	7		
9	1	3	5	7	9	1	
1	3	5	7	9	1	3	
3	5	7	9	1	3	5	

При этом бит 58 блока T становится битом 1, бит 50 – битом 2 и т. д. Полученный после перестановки блок $IP(T)$ разделяется на две половины: L_0, состоящую из 32 старших бит, и R_0, состоящую из 32 младших бит.

Затем выполняется итеративный процесс шифрования, состоящий из 16 циклов преобразования Фейстеля. Пусть $T_{i-1} = L_{i-1}R_{i-1}$ – результат $(i-1)$-й итерации.

Тогда результат i-й итерации $T_i = L_iR_i$ определяется формулами

$$\begin{cases} L_i = R_{i-1}, \\ R_i = L_{i-1} \oplus f_i(R_{i-1}, k_i), i = \overline{1,16}. \end{cases}$$

Функция f называется *функцией шифрования.* Ее аргументами являются 32-битовый вектор R_{i-1} и 48-битовый ключ k_i, который является результатом преобразования 56-битового ключа шифра k. Результатом последней итерации является блок $T_{16} = L_{16}R_{16}$. По окончании шифрования осуществляется восстановление позиций битов применением к T_{16} обратной перестановки IP^{-1}.

При расшифровании данных все действия выполняются в обратном порядке, при этом вместо соотношений (4.2) следует пользоваться соотношениями

$$\begin{cases} R_{i-1} = L_i, \\ L_{i-1} = R_i \oplus f_i(L_i, k_i), i = \overline{16,1}, \end{cases}$$

пользуясь которыми можно "спуститься" от L_{16} и R_{16} к R_0 и L_0.

Схема вычисления значения функции шифрования $f(R_{i-1}, k_i)$ изображена на рисунке 2.9.

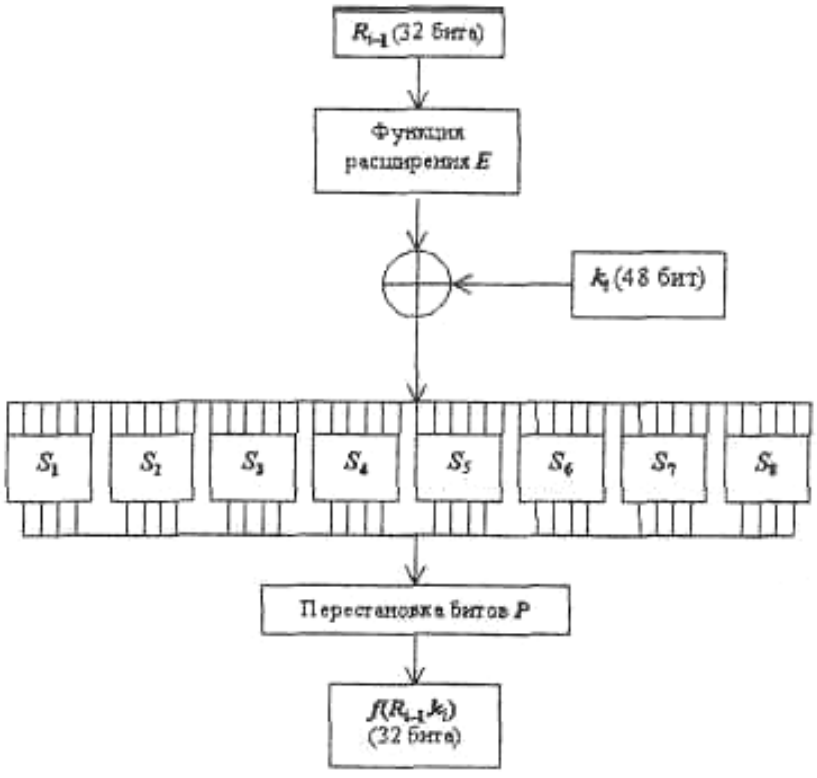


Рис. 2.9. Схема вычисления функции шифрования *f*

Для вычисления значения функции *f* используются:

- *функция расширения E*;
- преобразование *S*, составленное из восьми преобразований *S-блоков* S_1, S_2, …, S_8;
- перестановка *P*.

Аргументами функции *f* являются вектор R_{i-1} (32 бита) и вектор k_i (48 бит). Функция *E* "расширяет" 32-битный вектор R_{i-1} до 48-битного вектора $E(R_{i-1})$ путем дублирования некоторых битов вектора R_{i-1}, при этом порядок следования битов в $E(R_{i-1})$ указан в таблице 2.2.

Таблица 2.2. Функция расширения *E*

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

Первые три бита в $E(R_{i-1})$ – это соответственно биты 32, 1 и 2 вектора R_{i-1}, а последние три бита – это соответственно биты 31, 32, 1 вектора R_{i-1}.

Полученный результат складывается побитно по модулю 2 с текущим значением ключа k_i и затем представляется в виде восьми последовательных 6-битовых блоков B_1, B_2, …, B_8:

$E(R_{i-1}) \oplus k_i = B_1 \ldots B_8$.

Далее каждый из блоков B_j трансформируется в 4-битовый блок B'_j с помощью подходящей таблицы *S*-блока S_j, список которых приведен в таблице 3.

Таблица 2.3. Функция преобразования *S*

		НОМЕР СТОЛБЦА																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
НОМЕР СТРОКИ	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
	2	4	1	4	8	13	6	2	11	15	12	9	7	3	10	5	0	
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S_4
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S_5
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S_6
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	1	6	
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S_7
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S_8
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Преобразование блока B_j и B'_j осуществляется следующим образом. Пусть, например, B_2 равен 111010. Первый и последний разряды B_2 являются двоичной записью числа a, $0 \le a \le 3$.

Аналогично средние 4 разряда представляют число b, $0 \le b \le 15$. В нашем примере $a = 2$, $b = 13$.

Строки и столбцы таблицы S_2 пронумерованы числами a и b. Таким образом, пара (a, b) однозначно определяет число, находящееся на пересечении строки с номером a и столбца с номером b. В данном случае это число равно 3. Записывая его в двоичной форме, получаем B'_2, равный 0011.

Значение $f(R_{i-1}, k_i)$ теперь получается применением перестановки битов P, заданной таблицей к результирующему 32-битовому блоку $B'_1, B'_2, \ldots, B'_8$.

Таблица 2.4. Функция P перестановки битов

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

На каждой итерации используется текущее значение ключа k_i (48 бит), получаемое из исходного ключа k следующим образом.

Сначала пользователи выбирают сам ключ k, содержащий 56 случайных значащих битов. Восемь битов, находящихся в позициях 8,16,...,64, добавляются в ключ таким образом, чтобы каждый байт содержал нечетное число единиц. Это используется для обнаружения ошибок при обмене и хранении ключей. Значащие 56 бит ключа подвергаются перестановке, приведенной в таблице 2.5.

Таблица 2.5. Функция первоначальной подготовки ключа

57	49	41	33	25	17	9	
1	58	50	42	34	26	18	
10	2	59	51	43	35	27	

19	1		3 0	6 2	5 4	4	36
63	5	7	4 9	3 1	3 3	2	15
7	2	4	5 6	4 8	3 0	3	22
14		1	6 3	5 5	4 7	3	29
21	3		5 8	2 0	2 2	1	4

Эта перестановка определяется двумя блоками C_0 и D_0 по 28 бит в каждом (они занимают соответственно верхнюю и нижнюю половины таблицы). Так, первые три бита в C_0 есть соответственно 57, 49, 41 биты ключа. Затем индуктивно определяются блоки C_i и $D_i = \overline{1{,}16}$.

Если уже определены C_{i-1} и D_{i-1}, то C_i и D_i получаются из них одним или двумя левыми циклическими сдвигами согласно таблице 2.6.

Таблица 2. 6. Таблица сдвигов для вычисления ключа

i										0	1	2	3	4	5	6
Число сдвигов																

Теперь определим ключи k_i, $1 \le i \le 16$. Ключ k_i состоит из 48 битов, выбираемых из битов блока C_iD_i согласно таблице 7. Первыми тремя битами в k_i являются биты 14, 17, 11 из блока C_iD_i. Отметим, что 8 из 56 бит (с номерами 9, 18, 22, 25, 35, 38, 43, 54) из C_iD_i отсутствуют в k_i.

Таблица 2.7. Функция завершающей обработки ключа

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Официально DES являлся стандартом шифрования данных до 31 декабря 1998 г. В 1997 г. был объявлен конкурс на новый стандарт, который был назван AES (Advanced Encryption Standard). 2 октября 2000 г. Национальный институт стандартов и технологий (НИСТ) США объявил победителя конкурса AES. Однако для того, чтобы этот алгоритм завоевал мировое признание, необходимы серьезные исследования его свойств специалистами различных стран.

Основные режимы шифрования

Алгоритм DES вполне подходит как для шифрования, так и для аутентификации данных. Он позволяет непосредственно преобразовывать 64-битовый входной открытый текст в 64-битовый выходной шифрованный текст, однако данные редко ограничиваются 64 разрядами.

Чтобы использовать алгоритм DES для решения разнообразных криптографических задач, разработаны четыре режимах:

- *режим электронной кодовой книги* (ECB – *Electronic Code Book*);
- *режим сцепления блоков шифра* (CBC – *Cipher Block Chaining*);
- *режим обратной связи по шифртексту* (CFB – *Cipher Feed Back*);
- *режим обратной связи по выходу* (OFB – *Output Feed Back*).

Изучение стандарта криптографической защиты AES (Advanced Encryption Standart)

Международный стандарт криптографической защиты AES (Advanced Encryption Standart)

В 1997 г. Национальный институт стандартов и технологий США (NIST) объявил о начале программы по принятию нового стандарта криптографической защиты - стандарта XXI в. для закрытия важной информации правительственного уровня на замену существующему с 1974 г. алгоритму DES, самому распространенному криптоалгоритму в мире.

Требования к кандидатам были следующие:

- криптоалгоритм должен быть открыто опубликован;
- криптоалгоритм должен быть симметричным блочным шифром, допускающим размеры ключей в 128, 192 и 256 бит;
- криптоалгоритм должен быть предназначен как для аппаратной, так и для

программной реализации;

- криптоалгоритм должен быть доступен для открытого использования в любых продуктах, а значит, не может быть запатентован, в противном случае патентные права должны быть аннулированы;
- криптоалгоритм подвергается изучению по следующим па раметрам: стойкости, стоимости, гибкости, реализуемости в smart-картах.

В финал конкурса вышли следующие алгоритмы: MARS, TWOFISH и RC6 (США), RUNDAEL (Бельгия), SERPENT (Великобритания, Израиль, Норвегия). По своей структуре TWOFISH является классическим шифром Фейстеля; MARS и RC6 можно отнести к модифицированным шифрам Фейстеля, в них используется новая малоизученная операция циклического "прокручивания" битов слова на число позиций, изменяющихся в зависимости от шифруемых данных и секретного ключа; RUNDAEL и SERPENT являются классическими SP-сетями. MARS и TWOFISH имеют самую сложную конструкцию, RUNDAEL и RC6 - самую простую.

В октябре 2000 г. конкурс завершился. Победителем был признан бельгийский шифр RIJNDAEL, как имеющий наилучшее сочетание стойкости, производительности, эффективности реализации и гибкости. Его низкие требования к объему памяти делают его идеально подходящим для встроенных систем. Авторами шифра являются Йон Дэмен (Joan Daemen) и Винсент Рюмен (Vincent Rijmen), начальные буквы фамилий которых и образуют название алгоритма - RIJNDAEL.

После этого NIST начал подготовку предварительной версии Федерального Стандарта Обработки Информации (Federal Information Processing Standart - FIPS) и в феврале 2001 г. опубликовал его на сайте http://csrc.nist.gov/encryption/aes/. В течение 90-дневного периода открытого обсуждения предварительная версия FIPS пересматривалась с учетом комментариев, после чего начался процесс исправлений и утверждения. Наконец 26 ноября 2001 г. была опубликована окончательная версия стандарта FIPS-197, описывающего новый американский стандарт шифрования AES. Согласно этому документу стандарт вступил в силу с 26 мая 2002 г.

Блочный криптоалгоритм RIJNDAEL и стандарт AES

Высочайшую надежность AES NIST подтверждает астрономическими числами. 128битный ключ обеспечивает 340 андециллионов (340*1036) возможных комбинаций, а

256битный ключ увеличивает это число до 11*1076. Для сравнения, старый алгоритм DES, дает общее число комбинаций в 72*1015 На их перебор у специально построенной машины "DES Cracker" уходит несколько часов. Но даже если бы она делала это всего за одну секунду, то на перебор 128битного ключа машина потратила бы 149 триллионов лет. Межу тем, возраст всей Вселенной ученые оценивают в менее, чем 20 миллиардов лет.

Описание криптоалгоритма

Формат блоков данных и число раундов

RIJNDAEL - это итерационный блочный шифр, имеющий архитектуру "Квадрат". Шифр имеет переменную длину блоков и различные длины ключей. Длина ключа и длина блока могут быть равны независимо друг от друга 128, 192 или 256 битам. В стандарте AES определена длина блока данных, равная 128 битам.

Введем следующие обозначения:

- Nb - число 32-битных слов содержащихся во входном блоке, Nb=4;
- Nk - число 32-битных слов содержащихся в ключе шифрования, Nk=4,6,8;
- Nr - число раундов шифрования, как функция от Nb и Nk, Nr=10,12,14.

Входные (input), промежуточные (state) и выходные (output) результаты преобразований, выполняемых в рамках криптоалгоритма, называются состояниями (State). Состояние можно представить в виде прямоугольного массива байтов (рис. 1). При размере блока, равном 128 битам, этот 16-байтовый массив (рис. 2, а) имеет 4 строки и 4 столбца (каждая строка и каждый столбец в этом случае могут рассматриваться как 32-разрядные слова). Входные данные для шифра обозначаются как байты состояния в порядке S00, S10, S20, S30, S01, S11, S21, S31

После завершения действия шифра выходные данные получаются из байтов состояния в том же порядке. В общем случае число столбцов Nb равно длине блока, деленной на 32.

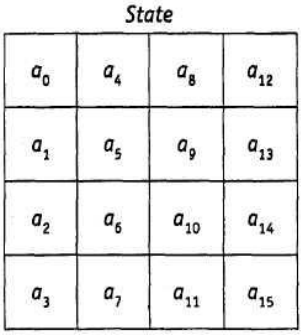


Рис. 2.8. Пример представления 128-разрядного блока данных в виде массива State, где a,- - байт блока данных, а каждый столбец - одно 32-разрядное слово

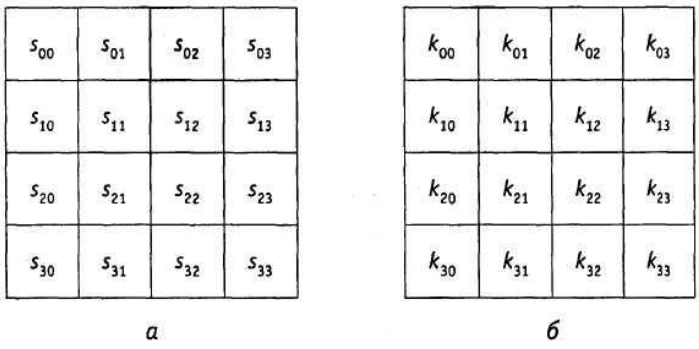


Рис. 2.9. Форматы данных: а - пример представления блока (Nb, = 4); 6 - ключа шифрования (Nk = 4), где s и k - соответственно байты массива State и ключа, находящиеся на пересечении i-й строки и j-то столбца

Рисунок 2.10 демонстрирует такое представление, носящее название архитектуры «Квадрат».

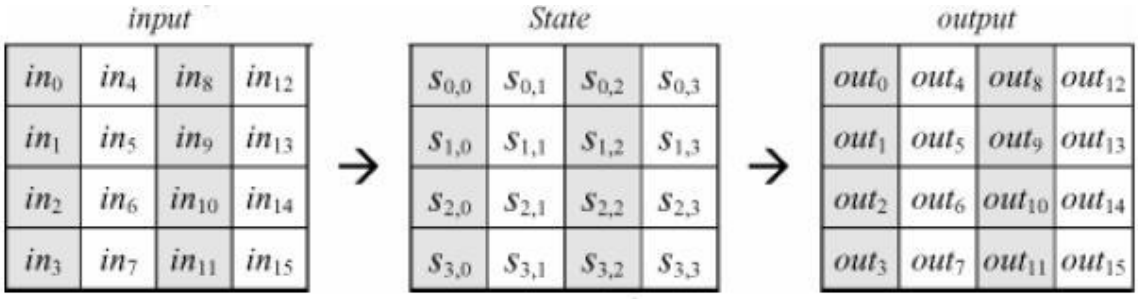


Рис. 2.10. Пример представления блока в виде матрицы 4Nb

Ключ шифрования также представлен в виде прямоугольного массива с четырьмя строками (рис. 7.9, б). Число столбцов этого массива равно длине ключа, деленной на 32. В стандарте определены ключи всех трех размеров - 128 бит, 192 бита и 256 бит, то есть

соответственно 4, 6 и 8 32-разрядных слова (или столбца – в табличной форме представления). В некоторых случаях ключ шифрования рассматривается как линейный массив 4-байтовых слов. Слова состоят из 4 байтов, которые находятся в одном столбце (при представлении в виде прямоугольного массива).

Число раундов N_r в алгоритме RIJNDAEL зависит от значений Nb и Nk, как показано в табл. 2.8. В стандарте AES определено соответствие между размером ключа, размером блока данных и числом раундов шифрования, как показано в табл.

Таблица 2.8. Число раундов N_r как функция от длины ключа Nk и длины блока Nb

N_r	$N_b = 4$	$N_b = 6$	$N_b = 8$
$N_k = 4$	10	12	14
$N_k = 6$	12	12	14
$N_k = 8$	14	14	14

Таблица 2.9. Соответствие между длиной ключа, размером блока данных и числом раундов в стандарте AES

Стандарт	Длина ключа (N_k слов)	Размер блока данных (N_b слов)	Число раундов (N_r)
AES-128	10	12	14
AES-192	12	12	14
AES-256	14	14	14

Раундовое преобразование

Раунд состоит из четырех различных преобразований:

- замены байтов SubBytes() - побайтовой подстановки в S-блоках с фиксированной таблицей замен размерностью 8 x 256;
- сдвига строк ShiftRows() - побайтового сдвига строк массива State на различное количество байт;

- перемешивания столбцов MixColumns() — умножения столбцов состояния, рассматриваемых как многочлены над $GF(2^8)$, на многочлен третьей степени g(x) по модулю $x^4 + 1$;
- сложения с раундовым ключом AddRoundKey() - поразрядного XOR с текущим фрагментом развернутого ключа.

Замена байтов (SubBytes). Преобразование SubBytes() представляет собой нелинейную замену байтов, выполняемую независимо с каждым байтом состояния. Таблицы замены S-блока являются инвертируемыми и построены из композиции следующих двух преобразований входного байта:

1. получение обратного элемента относительно умножения в поле $GF(2^8)$, нулевой элемент {00} переходит сам в себя;
2. применение преобразования над $GF(2^8)$, определенного следующим образом:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Другими словами суть преобразования может быть описана уравнениями:

$$b'_I = b_i \oplus b_{(I+4) \bmod 8} \oplus b_{(I+5) \bmod 8} \oplus b_{(I+6) \bmod 8} \oplus b_{(I+7) \bmod 8} \oplus c_i$$

где $c_0=c_1=c_5=c_6= 0$, $c_2=c_3=c_4=c_7= 1$, b'_i и b_i - соответственно преобразованное и исходное значение i-го бита, i = 0,1,2….7.

Применение описанного S-блока ко всем байтам состояния обозначается как SubBytes(State). Рис. 4. иллюстрирует применение преобразования SubBytes() к состоянию. Логика работы S-блока при преобразовании байта {xy} отражена в табл. 3. Например, результат {ed} преобразования байта {53} находится на пересечении 5-й строки и 3-го столбца.

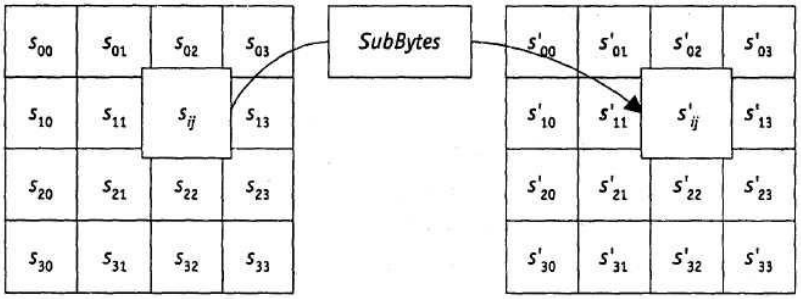


Рис. 2.10. SubBytes() действует на каждый байт состояния

Таблица 2.11. Таблица замен S-блока

x	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Преобразование сдвига строк (ShiftRows). Последние 3 строки состояния циклически сдвигаются влево на различное число байтов. Строка 1 сдвигается на C1 байт, строка 2 - на C2 байт, и строка 3 - на C3 байт. Значения сдвигов C1, C2 и C3 в RIJNDAEL зависят от длины блока N_b. Их величины приведены в табл. 2.12.

Таблица 2.12. Величина сдвига для разной длины блоков

Nb	C1	C2	C3

4	1	2	3
6	1	2	3
8	1	3	3

В стандарте AES, где определен единственный размер блока, равный 128 битам, C1 = 1, C2 = 2 и C3 = 3.

Операция сдвига последних трех строк состояния обозначена как ShiftRows(State). Рис. 2.11 показывает влияние преобразования на состояние.

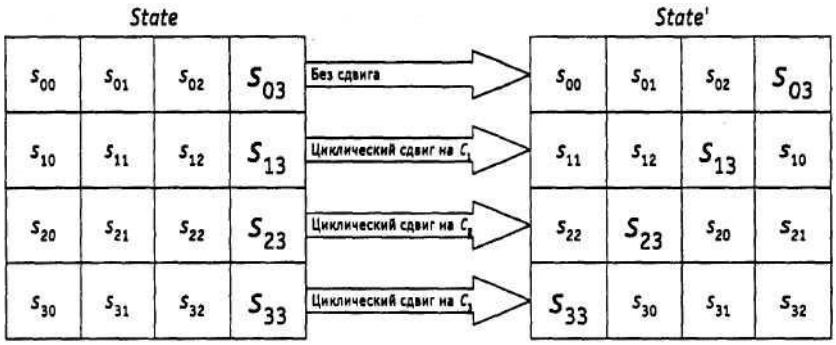


Рис. 2.11. ShiftRows() действует на строки состояния

Преобразование перемешивания столбцов (MixColumns).В этом преобразовании столбцы состояния рассматриваются как многочлены над $GF(2^8)$ и умножаются по модулю $x^4 + 1$ на многочлен g(x), выглядящий следующим образом:

$g(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

Это может быть представлено в матричном виде следующим образом:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}, 0 \le c \le 3,$$

где c - номер столбца массива State

В результате такого умножения байты столбца S0c, S1c, S2c, S3c заменяются соответственно на байты

$S'0c = (\{02\} \bullet S0c) \oplus (\{03\} \bullet S1c) \oplus S2c \oplus S3c,$

S'1c = S0c ⊕ ({02} • S1c) ⊕ ({03} • S2c) ⊕ S3c,

S'2c = S0c ⊕ S1c ⊕ ({02} • S2c) ⊕ ({03} • S3c),

S'3c = ({03} • S0c) ⊕ S1c © S2c ⊕ ({02} • S3c).

Применение этой операции ко всем четырем столбцам состояния обозначено как MixColumns(State). Рис. 2.12 демонстрирует применение преобразования MixColumnsQ к столбцу состояния.

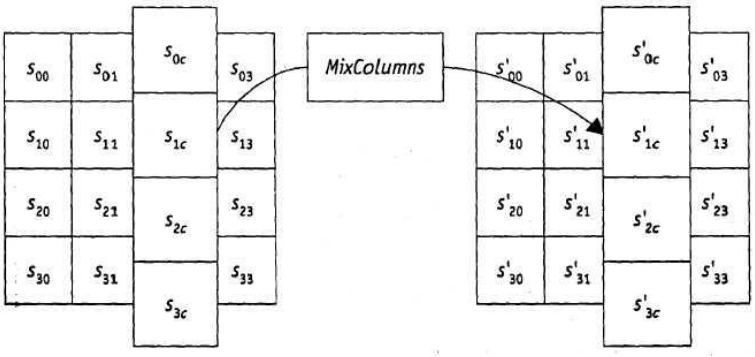


Рис. 2.12. MixColumnsQ действует на столбцы состояния

Добавление раундового ключа (AddRoundKey). В данной операции раундовый ключ добавляется к состоянию посредством простого поразрядного XOR. Раундовый ключ вырабатывается из ключа шифрования посредством алгоритма выработки ключей (key schedule). Длина раундового ключа (в 32-разрядных словах) равна длине блока Nb. Преобразование, содержащее добавление посредством XOR раундового ключа к состоянию (рис. 2.13), обозначено как AddRoundKey(State, RoundKey).

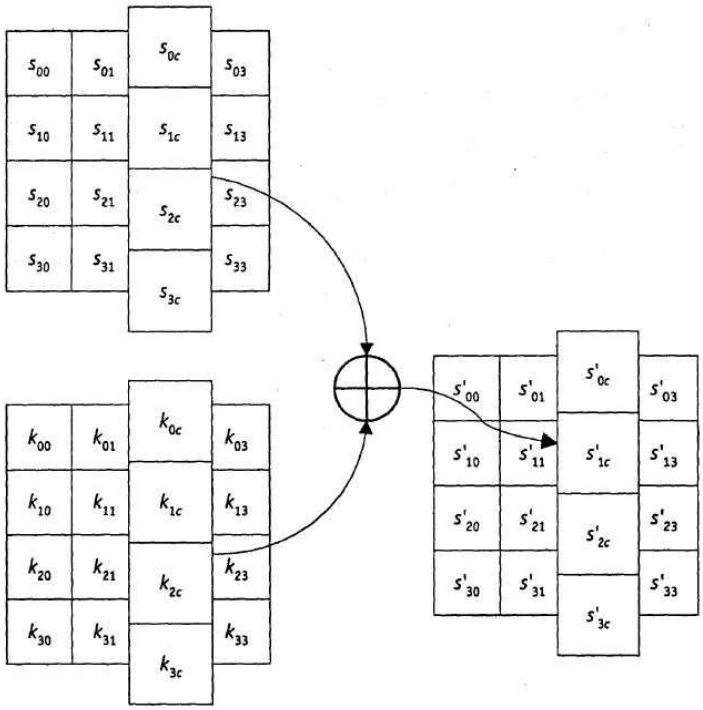


Рис. 2.13. При добавлении ключа раундовый ключ складывается посредством операции X0R с состоянием

Алгоритм выработки ключей (Key Schedule)

Раундовые ключи получаются из ключа шифрования посредством алгоритма выработки ключей. Он содержит два компонента: расширение ключа (Key Expansion) и выбор раундового ключа (Round Key Selection). Основополагающие принципы алгоритма выглядят следующим образом:

- общее число битов раундовых ключей равно длине блока, умноженной на число раундов, плюс 1 (например, для длины блока 128 бит и 10 раундов требуется 1408 бит раундовых ключей);
- ключ шифрования расширяется в расширенный ключ (Expanded Key);
- раундовые ключи берутся из расширенного ключа следующим образом: первый раундовый ключ содержит первые Nb слов, второй - следующие Nb слов и т. д.

Расширение ключа (Key Expansion). Расширенный ключ в RIJNDAEL представляет собой линейный массив w[i] из $Nb(N_r+ 1)$ 4-байтовых слов, $i = 0,1…Nb(N_r + 1)$. В AES массив w[i] состоит из $4(N_r +1)$ 4-байтовых слов, $i = 0,1…4(N_r + 1)$.

Примечание. Необходимо учитывать, что с целью полноты описания здесь приводится алгоритм для всех возможных длин ключей, на практике же полная его реализация нужна не всегда.

Первые Nk слов содержат ключ шифрования. Все остальные слова определяются рекурсивно из слов с меньшими индексами. Алгоритм выработки ключей зависит от величины Nk.

Как можно заметить (рис. 8, а), первые Nk слов заполняются ключом шифрования. Каждое последующее слово w[i] получается посредством XOR предыдущего слова w[i-l] и слова на Nk позиций ранее w[i - Nk]'.

$w[i] = w[i - 1] \oplus w[i - N_k]$.

Для слов, позиция которых кратна N_k, перед XOR применяется преобразование к w[i-1], а затем еще прибавляется раундовая константа Rcon. Преобразование реализуется с помощью двух дополнительных функций: RotWord(), осуществляющей побайтовый сдвиг 32-разрядного слова по формуле {a0 a1 a2 a3} → {a1 a2 a3 a0}, и SubWord() осуществляющей побайтовую замену с использованием 5-блока функции SubBytes(). Значение Rcon[j] равно 2^{j-1}. Значение w[i] в этом случае равно

$w[i] - SubWord(RotWord(w[i - 1])) \oplus Rcon[i/N_k] \oplus w[i - N_k]$.

Рис. 2.14. Процедуры:

а - расширения ключа (светло-серым цветом выделены слова расширенного ключа, которые формируются без использования функций SubWord() и RotWordQ; темно-серым

цветом выделены слова расширенного ключа, при вычислении которых используются преобразования SubWordQ и RotWordQ);

б - выбора раундового ключа для Nk - 4

Выбор раундового ключа (Round Key Selection). Раундовый ключ i получается из слов массива раундового ключа от W[Nb i] и до W[N_b (i + 1)], как показано на рис. 8.

Примечание. Алгоритм выработки ключей можно осуществлять и без использования массива w[i]. Для реализаций, в которых существенно требование к занимаемой памяти, раундовые ключи могут вычисляться "на лету" посредством использования буфера из Nk слов. Расширенный ключ должен всегда получаться из ключа шифрования и никогда не указывается напрямую. Нет никаких ограничений на выбор ключа шифрования.

Функция зашифрования

Шифр RIJNDAEL состоит (рис. 2.15):

- из начального добавления раундового ключа;
- Nr - 1 раундов;
- заключительного раунда, в котором отсутствует операция MixColumns().

На вход алгоритма подаются блоки данных State, в ходе преобразований содержимое блока изменяется и на выходе образуется шифротекст, организованный опять же в виде блоков State, как показано на рис. 2.6, где Nb - 4, in_m и out_m - m-е байты соответственно входного и выходного блоков, m =0,1…15, s_{ij} - байт, находящийся на пересечении i-й строки и j-го столбца массива State, i = j = 0,1..3.

Перед началом первого раунда происходит суммирование по модулю 2 с начальным ключом шифрования, затем - преобразование массива байтов State в течение 10, 12 или 14 раундов в зависимости от длины ключа. Последний раунд несколько отличается от предыдущих тем, что не задействует функцию перемешивания байт в столбцах MixColumns().

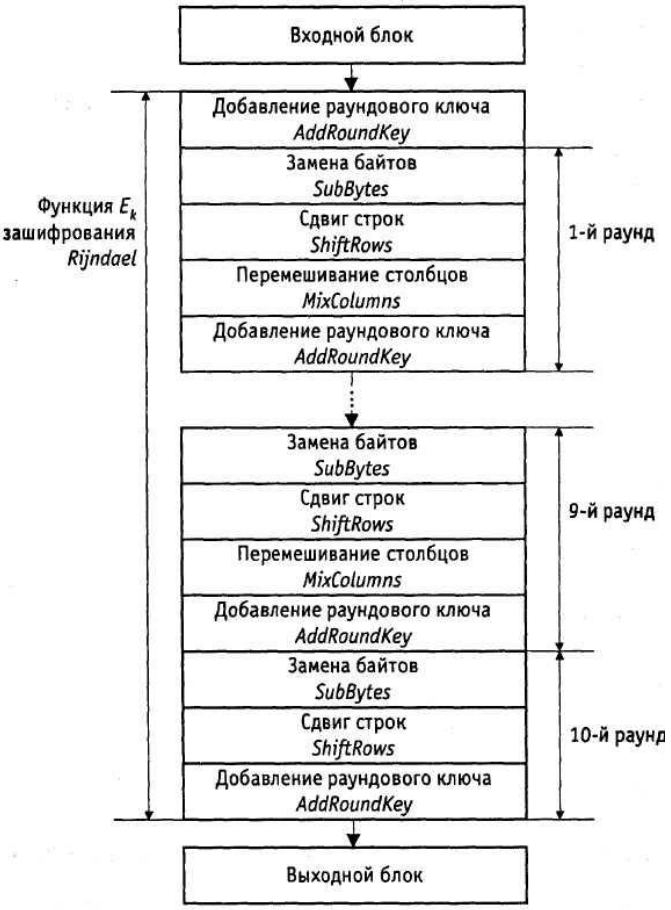


Рис. 2.15. Схема функции $E_к$ зашифрования криптоалгоритма RIJNDAEL при Nk = Nb = 4

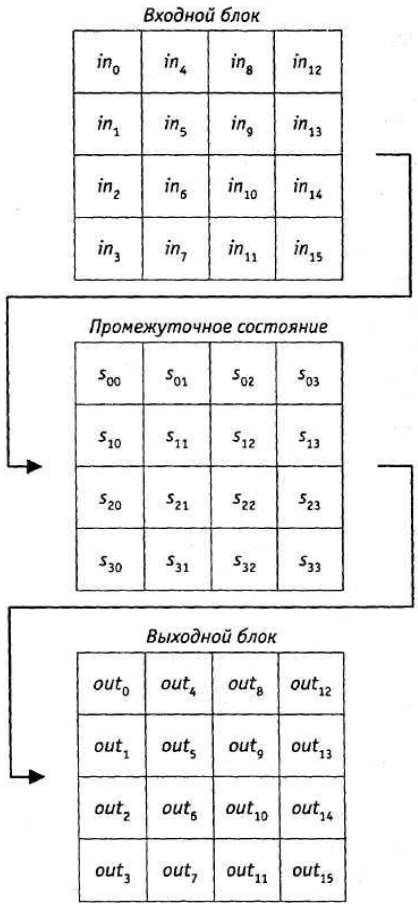


Рис. 2.16. Ход преобразования данных, организованных в виде блоков State

Рис. 2.17 демонстрирует и рассеивающие и перемешивающие свойства шифра. Видно, что два раунда обеспечивают полное рассеивание и перемешивание.

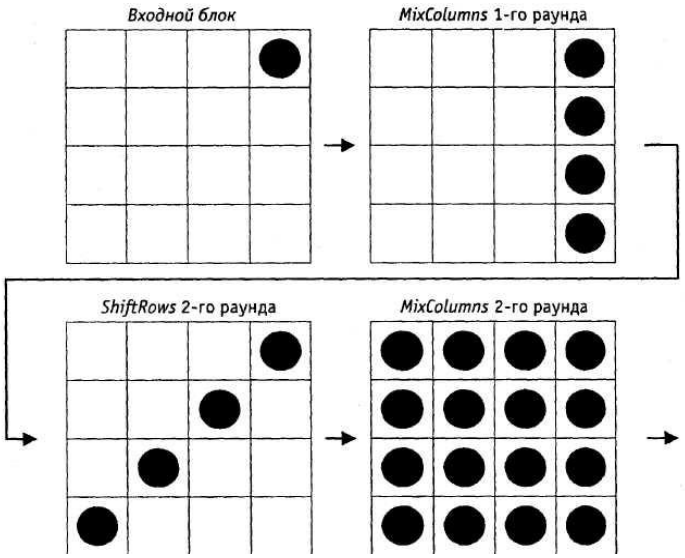


Рис. 2.17. Принцип действия криптоалгоритма RIJNDAEL; ⊡ - измененный байт

Функция обратного расшифрования

Если вместо SubBytes(), ShiftRows(), MixColumns() и AddRoundKey() в обратной последовательности выполнить инверсные им преобразования, можно построить функцию обратного расшифрования. При этом порядок использования раундовых ключей является обратным по отношению к тому, который используется при зашифровании.

Далее приводится описание функций, обратных используемым при прямом зашифровании.

Функция AddRoundKey() обратна сама себе, учитывая свойства используемой в ней операции XOR.

Преобразование InvSubBytes. Логика работы инверсного S-блока при преобразовании байта {xy} отражена в табл. 2.13.

Таблица 2.13. Таблица замен инверсного S-блока

x	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Преобразование InvShiftRows. Последние 3 строки состояния циклически сдвигаются вправо на различное число байтов. Строка 1 сдвигается на C1 байт, строка 2 - на C2 байт, и строка 3 - на C3 байт. Значения сдвигов C1, C2, C3 зависят от длины блока Nb. ***Преобразование InvMixColumns***. В этом преобразовании столбцы состояния рассматриваются как многочлены над GF(2^8) и умножаются по модулю $x^4 + 1$ на многочлен $g^{-1}(x)$, выглядящий следующим образом:

$g^{-1}(x) = \{0b\}ч^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

Это может быть представлено в матричном виде следующим образом:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}, 0 \le c \le 3.$$

В результате на выходе получаются следующие байты

$$\begin{aligned} s'_{0c} &= (\{0e\} \bullet s_{0c}) \oplus (\{0b\} \bullet s_{1c}) \oplus (\{0d\} \bullet s_{2c}) \oplus (\{09\} \bullet s_{3c}), \\ s'_{1c} &= (\{09\} \bullet s_{0c}) \oplus (\{0e\} \bullet s_{1c}) \oplus (\{0b\} \bullet s_{2c}) \oplus (\{0d\} \bullet s_{3c}), \\ s'_{2c} &= (\{0d\} \bullet s_{0c}) \oplus (\{09\} \bullet s_{1c}) \oplus (\{0e\} \bullet s_{2c}) \oplus (\{0b\} \bullet s_{3c}), \\ s'_{3c} &= (\{0b\} \bullet s_{0c}) \oplus (\{0d\} \bullet s_{1c}) \oplus (\{09\} \bullet s_{2c}) \oplus (\{0e\} \bullet s_{3c}). \end{aligned}$$

Функция прямого расшифрования

Алгоритм обратного расшифрования, описанный выше, имеет порядок приложения операций-функций, обратный порядку операций в алгоритме прямого шифрования, но использует те же параметры (развернутый ключ). Однако некоторые свойства алгоритма шифрования RIJNDAEL позволяют применить для расшифрования тот же порядок приложения функций (обратных используемым для зашифрования) за счет изменения некоторых параметров, а именно - развернутого ключа.

Два следующих свойства позволяют применить алгоритм прямого расшифрования.

Порядок приложения функций SubBytes() и ShiftRows() не играет роли. То же самое верно и для операций InvSubBytes() и InvShiftRows(). Это происходит потому, что функции SubBytes() и InvSubBytes() работают с байтами, а операции ShiftRows() и InvShifiRows() сдвигают целые байты, не затрагивая их значений.

Операция MixColumns() является линейной относительно входных данных, что означает

InvMixColumns(State XOR RoundKey) =

= lnvMixColumns(State) XOR InvMixColumns(RoundKey)

Эти свойства функций алгоритма шифрования позволяют изменить порядок применения функций InvSubBytes() и InvShiftRows(). Функции AddRounKey() и InvMixColwnns() также могут быть применены в обратном порядке, но при условии, что столбцы (32-битные слова) развернутого ключа расшифрования предварительно пропущены через функцию invMixColumns().

Таким образом, можно реализовать более эффективный способ расшифрования с тем же порядком приложения функций как и в алгоритме зашифрования.

При формировании развернутого ключа шифрования в процедуру развертывания ключа необходимо добавить следующий код

Примечание. В последнем операторе (в функции InvMixColumn()) происходит преобразование типа, так как развернутый ключ хранится в виде линейного массива 32-разрядных слов, в то время как входной параметр функции - двумерный массив байтов.

В табл. 6 приведена процедура зашифрования, а также два эквивалентных варианта процедуры расшифрования при использовании двухраундового варианта Rijndael. Первый вариант функции расшифрования суть обычная инверсия функции зашифрования. Второй вариант функции зашифрования получен из первого после изменения порядка следования операций в трех парах преобразований:

invShiftRows — InvSubBytes (дважды)

и AddRoundKey — InvMixColumns.

Таблица. 2.14. Последовательность преобразований в двухраундовом варианте RIJNDAEL

Функция зашифрования двухраундового варианта *RIJNDAEL*	**Функция обратного расшифрования двухраундового варианта *RIJNDAEL***	**Эквивалентная функция прямого расшифрования двухраундового варианта *RIJNDAEL***
AddRoundKey	AddRoundKey	AddRoundKey
SubBytes	InvShiftRows	InvSubBytes
ShiftRows	InvSubBytes	InvShiftRows
MixColumns	AddRoundKey	InvMixColumns
AddRoundKey	InvMixColumns	AddRoundKey
SubBytes	InvShiftRows	InvSubBytes
ShiftRows	InvSubBytes	InvShiftRows
AddRoundKey	AddRoundKey	AddRoundKey

Очевидно, что результат преобразования при переходе от исходной к обратной последовательности выполнения операций в указанных парах не изменится.

Видно, что процедура зашифрования и второй вариант процедуры расшифрования совпадают с точностью до порядка использования раундовых ключей (при выполнении операций AddRoundKey), таблиц замен (при выполнении операций SubBytes и InvSubBytes) и матриц преобразования (при выполнении операций MixColumns и invMixColumns). Данный результат легко обобщить и на любое другое число раундов.

Атака “Квадрат”

Атака "Квадрат" была специально разработана для одноименного шифра SQUARE (авторы J. Daemen, L. Knudsen, V. Rijmen). Атака использует при своем проведении байт-ориентированную структуру шифра. Учитывая, что RIJNDAEL унаследовал многие свойства шифра SQUARE, эта атака применима и к нему. Далее приведено описание атаки "Квадрат" применительно к RIJNDAEL.

Атака "Квадрат" основана на возможности свободного подбора атакующим некоторого набора открытых текстов для последующего их зашифрования. Она независима от таблиц замен блоков, многочлена функции MixColumns() и способа разворачивания ключа. Эта атака для 6-раундового шифра RIJNDAEL, состоящего из 6 раундов, эффективнее, чем полный перебор по всему ключевому пространству. После описания базовой атаки на 4-раундовый RUNDAEL, будет показано, как эту атаку можно продлить на 5 и даже 6 раундов. Но уже для 7 раундов “Квадрат” становится менее эффективным, чем полный перебор.

Предпосылки

Пусть L-набор - такой набор из 256 входных блоков (масивов State), каждый из которых имеет байты (назовем их активными), значения которых различны для всех 256 блоков. Остальные байты (будем называть их пассивными) остаются одинаковыми для всех 256 блоков из L-набора. То есть:

$$\forall x, y \in L \begin{cases} x_{ij} \neq y_{ij} & \text{, если байт с номером ij активный;} \\ x_{ij} = y_{ij} & \text{, в противном случае.} \end{cases}$$

Будучи подвергнутыми обработке функциями SubBytes() и AddRoundKey() блоки L-набора дадут в результате другой L-набор с активными байтами в тех же позициях, что и у исходного. Функция ShiftRows() сместит эти байты соответственно заданным в ней смещениям в строках массивов State. После функции MixColumns() L-набор в общем случае необязательно останется L-набором (т. е. результат преобразования может перестать удовлетворять определению L-набора). Но поскольку каждый байт результата функции MixColumns() является линейной комбинацией (с обратимыми коэффициентами) четырех входных байт того же столбца:

$$b_{ij} = 2a_{ij} \oplus 3a_{(i+1)j} \oplus a_{(i+2)j} \oplus a_{(i+3)j}$$

столбец с единственным активным байтом на входе даст в результате на выходе столбец со всеми четырьмя байтами - активными.

Базовая атака “Квадрат” на 4 раунда

Рассмотрим L-набор, во всех блоках которого активен только один байт. Иначе говоря, значение этого байта различно во всех 256 блоках, а остальные байты одинаковы (скажем, равны нулю). Проследим эволюцию этого байта на протяжении трех раундов. В первом раунде функция MixColumns() преобразует один активный байт в столбец из 4 активных байт. Во втором раунде эти 4 байта разойдутся по 4 различным столбцам в результате преобразования функцией ShiftRows(). Функция же MixColumns() следующего, третьего раунда преобразует эти байты в 4 столбца, содержащие активные байты. Этот набор все еще остается L-набором до того момента, когда он поступает на вход функции MixColumns() третьего раунда.

Основное свойство L-набора, используемое здесь, то, что поразрядная сумма по модулю 2 всех блоков такого набора всегда равна нулю. Действительно, поразрядная сумма неактивных (с одинаковыми значениями) байт равна нулю по определению операции поразрядного XOR, а активные байты, пробегая все 256 значений, также при поразрядном суммировании дадут нуль. Рассмотрим теперь результат преобразования функцией MixColumns() в третьем раунде байтов входною массива данных a в байты выходного

массива данных b. Покажем, что и в этом случае поразрядная сумма всех блоков выходного набора будет равна нулю, то есть:

$$\bigoplus_{b=mixcolumns(a),a\in L} b_{ij} = \bigoplus_{a\in L}(2a_{ij} \oplus 3a_{(i+1)j} \oplus a_{(i+2)j} \oplus a_{(i+3)j}) =$$

$$= 2\bigoplus_{a\in L} a_{ij} \oplus 3\bigoplus_{a\in L} a_{(i+1)j} \bigoplus_{a\in L} a_{(i+2)j} \bigoplus_{a\in L} a_{(i+3)j} = 2\cdot 0 \oplus 3\cdot 0 \oplus 1\cdot 0 \oplus 1\cdot 0 = 0$$

Таким образом, вес данные на входе четвертого раунда сбалансированы (то есть их полная сумма равна нулю). Этот баланс в общем случае нарушается последующим преобразованием данных функцией SubBytes().

Мы предполагаем далее, что четвертый раунд является последним, то есть в нем нет функции MixColumns(). Тогда каждый байт выходных данных этого раунда зависит только от одного байта входных данных. Если обозначить через a байт выходных данных четвертого раунда, через b байт входных данных и через k - соответствующий байт раундового ключа, то можно записать:

$$a_{ij} = SubBytes(b_{ij}) \oplus k_{ij} \qquad (8)$$

Отсюда, предполагая значение k_{ij} можно по известному a_{ij} вычислить b_{ij}, а затем проверить правильность догадки о значении k_{ij}; если значения байта b_{ij}, полученные при данном k_{ij} не будут сбалансированы по всем блокам (то есть не дадут при поразрядном суммировании нулевой результат), значит догадка неверна. Перебрав максимум 2^8 вариантов байта раундового ключа, мы найдем его истинное значение.

По такому же принципу могут быть определены и другие байты раундового ключа. За счет того, что поиск может производиться отдельно (читай - параллельно) для каждого байта ключа, скорость подбора всего значения раундового ключа весьма велика. А по значению полного раундового ключа, при известном алгоритме его развертывания, не составляет труда восстановить сам начальный ключ шифрования.

Добавление пятого раунда в конец базовой атаки “Квадрат”

Если будет добавлен пятый раунд, то значения b_{ij} придется вычислять уже на основании выходных данных не четвертого, а пятого раунда. И дополнительно кроме байта раундового ключа четвертого раунда перебирать еще значения четырех байт столбца раундового ключа для пятого раунда. Только так мы сможем выйти на значения сбалансированных байт b_{ij} входных данных четвертого раунда.

Таким образом, теперь нам нужно перебрать 2^{40} значений - 2^{32} вариантов для 4 байт столбца раундового ключа пятого раунда и для каждого из них 2^8 вариантов для одного байта четвертого раунда. Эту процедуру нужно будет повторить для всех четырех столбцов пятого раунда. Поскольку при подборе "верного" значения байта раундового ключа четвертого раунда количество "неверных" ключей уменьшается в 2^8 раз, то работая одновре-

менно с пятью L-наборами, можно с большой степенью вероятности правильно подобрать все 2^{40} бита. Поиск может производиться отдельно (т. е. параллельно) для каждого столбца каждого из пяти L-наборов, что опять же гораздо быстрее полного перебора всех возможных значений ключа.

Добавление шестого раунда в начало базовой атаки "Квадрат"

Основная идея заключается в том, чтобы подобрать такой набор блоков открытого текста, который на выходе после первого раунда давал бы L-набор с одним активным байтом. Это требует предположения о значении четырех байт ключа, используемых функцией AddRoundKey() перед первым раундом.

Для того чтобы на входе второго раунда был только один активный байт достаточно, чтобы в первом раунде один активный байт оставался на выходе функции MixColumns(). Это означает, что на входе MixColumns() первого раунда должен быть такой столбец, байты а которого для набора из 256 блоков в результате линейного преобразования:

$$b_{ij} = 2a_i \oplus 3a_{i+1} \oplus a_{i+2} \oplus a_{i+3}, 0 \le i \le 3,$$

где i – номер строки, для одного определенного i давали 256 различных значений, в то время как для каждого из остальных трех значений i результат этого преобразования должен оставаться постоянным. Следуя обратно по порядку приложения функций преобразования в первом раунде, к ShiftRows() данное условие нужно применить к соответственно разнесенным по столбцам 4 байтам. С учетом применения функции SubBytes() и сложения с предполагаемым значением 4-байтового раундового ключа можно смело составлять уравнения и подбирать нужные значения байт открытого текста, подаваемых на зашифрование для последующего анализа результата:

$$b_{ij} = 2SubBytes(a_{ij} \oplus k_{ij}) \oplus 3SubBytes(a_{(i+1)(j+1)} \oplus k_{(i+1)(j+1)}) \oplus$$

$$\oplus SubBytes(a_{(i+2)(j+2)} \oplus k_{(i+2)(j+2)}) \oplus SubBytes(a_{(i+3)(j+3)} \oplus k_{(i+3)(j+3)}), 0 \le i, j \le 3$$

Таким образом, получаем следующий алгоритм взлома. Имеем всего 2^{32} различных значений а для определенных i и j. Остальные байты для всех блоков одинаковы (пассивные байты). Предположив значения четырех байт k ключа первого раунда, подбираем (исходя из вышеописанного условия) набор из 256 блоков. Эти 256 блоков станут L-набором после первого раунда. К этому L-наборе применима базовая атака для 4 раундов. Подобранный с ее помощью один байт ключа последнего раунда фиксируется. Теперь подбирается новый набор из 256 блоков для того же значения 4 байт k ключа первого раунда. Опять осуществляется базовая атака, дающая один байт ключа последнего раунда. Если после нескольких попыток значение этого байта не меняется, значит мы на верном пути. В противном случае нужно менять предположение о значении 4 байт k ключа первого раунда.

Такой алгоритм действий достаточно быстро приведет к полному восстановлению всех байт ключа последнего раунда.

Таким образом, атака "Квадрат" может быть применена к 6 раундам шифра RIJNDAEL, являясь при этом более эффективной, чем полный перебор по всему ключевому пространству. Любое известное продолжение атаки "Квадрат" на 7 и более раундов становится более трудоемким, чем даже обычный полный перебор значений ключа.

Поточные системы шифрования

Поточные шифры обычно делят на синхронные и самосинхронизирующиеся.

В синхронном поточном шифре шифрующая последовательность (гамма, ключевой поток, keystream) генерируется независимо от потока открытого текста и потока шифруемого текста [6].

Генератор гаммы в синхронной поточной криптосистеме может быть описан следующими уравнениями

$$s_{i+1} = F(s_i),$$
$$k_i = f(s_i),$$

где s_i – значение внутреннего состояния (state), F – функция переходов, f – выходная функция, k_i – очередной элемент гаммы.

Начальное состояние (initial state) s_0 может быть определено из ключа k и вектора инициализации IV (initial vector).

Цель генератора гаммы – развернуть короткий случайный ключ k в длинную псевдослучайную последовательность (ПСП) $k_1, k_2, \ldots, k_n$.

Работа синхронной киптосистемы схематически изображена на рисунке 2.17.

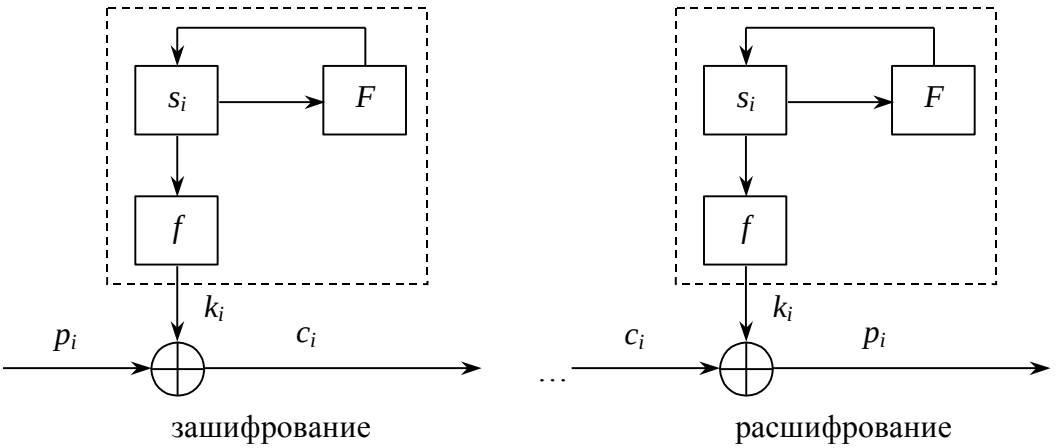


Рис. 2.17 Синхронный поточный шифр

В процессе зашифрования элементы гаммы и открытого текста подвергаются операции

XOR (exclusive OR, исключающее ИЛИ), в результате чего создается шифртекст:

$c_i = p_i \oplus k_i$.

При расшифровании, для восстановления открытого текста, шифртекст и гамма также подвергаются операции XOR:

$p_i = c_i \oplus k_i$.

В самосинхронизирующемся поточном шифре гамма зависит не только от ключа, но еще и от определенного количества элементов ранее вычисленного шифртекста.

Генератор гаммы в самосинхронизирующейся поточной криптосистеме описывается следующим образом:

$s_{i+1} = F(c_{i-1}, c_{i-2}, \ldots, c_{i-N})$,

$k_i = f(k, s_i)$.

Работа самосинхронизирующейся криптосистемы схематически изображена на рисунке 7.18

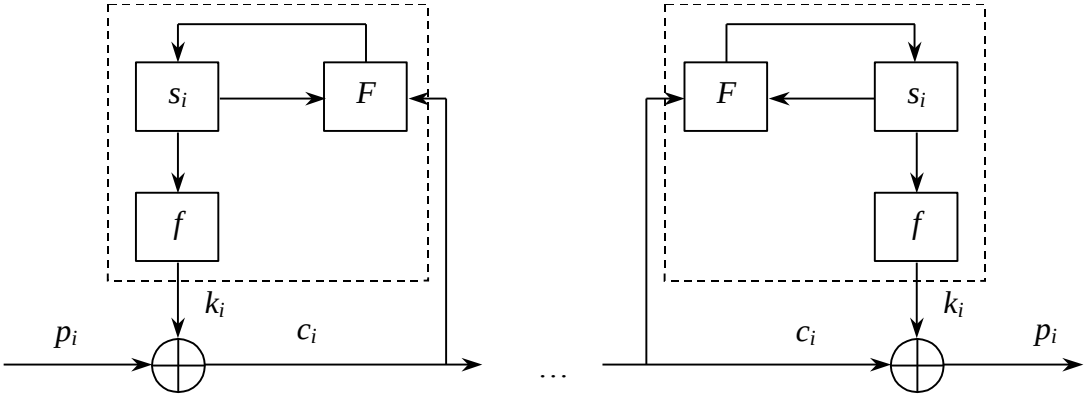


Рис. 2.18. Самосинхронизирующийся поточный шифр

Синхронные и самосинхронизирующиеся поточные шифры обладают как достоинствами так и недостатками. При использовании синхронного поточного шифра устройство зашифрования отправителя и устройство расшифрования получателя должны работать синхронно. Если в процессе передачи был потерян хотя бы один элемент шифртекста, то получатель обнаружит лишь бессмысленные данные, начиная с того места, где сбилась синхронизация. Обычно синхронизация достигается вставкой в передаваемое сообщение специальных “маркеров”. В результате этого элемент шифртекста, пропущенный в процессе передачи, приводит к неправильному расшифрованию лишь до тех пор, пока не будет принят один из маркеров. Другое решение – реинициализация состояний как устройства зашифрования, так и устройства расшифрования при некотором предварительно согласованном условии.

Самосинхронизирующийся поточный шифр обладает свойством, как свидетельствует его

название, автоматически синхронизировать себя. Внутреннее состояние такого шифра зависит от N предыдущих элементов шифртекста. Если в процессе передачи был потерян или изменен элемент шифртекста, то после приема N правильных последовательных элементов шифртекста внутреннее состояние устройства расшифрования становится идентичным внутреннему состоянию устройства зашифрования, т.е. синхронизация восстановлена. Недостатком самосинхронизирующегося поточного шифра является распространение ошибки. Для каждого элемента шифртекста, искаженного при передаче, устройство расшифрования произведет N некорректных элементов гаммы. Следовательно, пока испорченный элемент влияет на внутреннее состояние, каждой ошибке шифртекста будет соответствовать N ошибок в открытом тексте. Синхронный поточный шифр имеет свойство не распространять ошибки. Расшифрование искаженного элемента шифртекста влияет только на соответствующий элемент открытого текста. Хотя такое свойство может показаться желательным, у него есть и другая сторона. В этом случае ограничивается возможность обнаружения ошибки при расшифровании, но что еще более важно, противник имеет возможность произвести управляемые изменения в части шифртекста, совершенно точно зная, какие это вызовет изменения в соответствующем открытом тексте.

Поточные режимы блочных шифров

Для различных ситуаций, встречающихся на практике, разработано значительное количество режимов работы блочных шифров. Используя некоторый из них, блочные шифры можно реализовать как синхронные или самосинхронизирующиеся поточные шифры.

Для зарубежных стандартов блочного шифрования DES (Data Encryption Standard) и AES (Advanced Encryption Standard) существуют следующие основные режимы:

- Режим электронной кодовой книги, ECB (Electronic Code Book).
- Режим сцепления блоков шифртекста, CBC (Ciphertext Block Chaining).
- Режим обратной связи по шифртексту, CFB (Ciphertext Feedback).
- Режим обратной связи по выходу, OFB (Output Feedback).
- Режим счетчика, CTR (Counter mode).

В режимах OFB и CTR блочный шифр работает как синхронный поточный шифр, а в режиме CFB – как самосинхронизирующийся.

Отечественный стандарт блочного шифрования ГОСТ 28147-89 может работать в следующих режимах:

- Режим простой замены.
- Режим гаммирования.

- Режим гаммирования с обратной связью.
- Режим выработки имитовставки.

В режиме гаммирования алгоритм шифрования ГОСТ 28147-89 работает как синхронный поточный шифр, а в режиме гаммирования с обратной связью – как самосинхронизирующийся.

Строительные блоки поточных шифров

Рассмотрим основные блоки, используемые для построения поточных шифров.

Регистры сдвига с обратной связью

Большинство предложенных до настоящего времени алгоритмов поточного шифрования так или иначе основаны на использовании регистров сдвига с обратной связью.

Регистр сдвига с обратной связью (feedback shift register, FSR) состоит из двух частей: регистра сдвига и функции обратной связи (рисунок 3). Регистр сдвига представляет собой последовательность битов. Длина регистра сдвига выражается числом битов. Если длина регистра равна *n* битам, регистр называют *n*-битовым регистром сдвига. При каждом извлечении бита все биты регистра сдвига сдвигаются вправо на 1 позицию. Новый старший бит рассчитывается как функция от всех остальных битов регистра. На выходе регистра сдвига оказывается 1 бит.

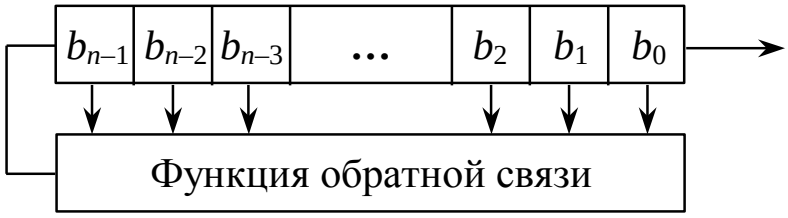


Рис. 2.19. Регистр сдвига с обратной связью

Регистры сдвига с линейной обратной связью

К простейшему типу FSR относится регистр сдвига с линейной обратной связью (linear feedback shift register, LFSR). Подавляющее большинство предложенных до настоящего времени генераторов поточного шифрования так или иначе основано на LFSR.

На это существует несколько причин:

- LFSR хорошо подходят для аппаратной реализации;
- LFST могут производить последовательности большого периода;
- LFSR могут производить последовательности с хорошими статистическими свойствами;
- LFSR могут быть легко проанализированы с помощью алгебраических техник.

- LFSR длины n состоит из n элементов задержки (ячеек) b_{n-1}, b_{n-2}, …, b_1, b_0, каждый из которых может хранить один бит и имеет по одному входу и выходу.

Исходной информацией для построения LFSR является образующий многочлен. Степень этого многочлена определяет разрядность регистра сдвига, а ненулевые коэффициенты – характер обратных связей (номера отводов сигналов обратной связи). В общем случае двоичный образующий многочлен степени n имеет вид:

$$c(x) = \sum_{i=0}^{n} c_i x^i = 1 + c_1 x + c_2 x_2 + \ldots + c_{n-1} x_{n-1} + c_n x_n,$$

где $c_n = c_0 = 1$, $c_j \in \{0, 1\}$ для $j = 1, \ldots, (n-1)$.

В общем случае двоичному образующему многочлену $c(x)$ соответствует две схемы: Фибоначчи (рисунок 2.20) и Галуа (рисунок 2.21).

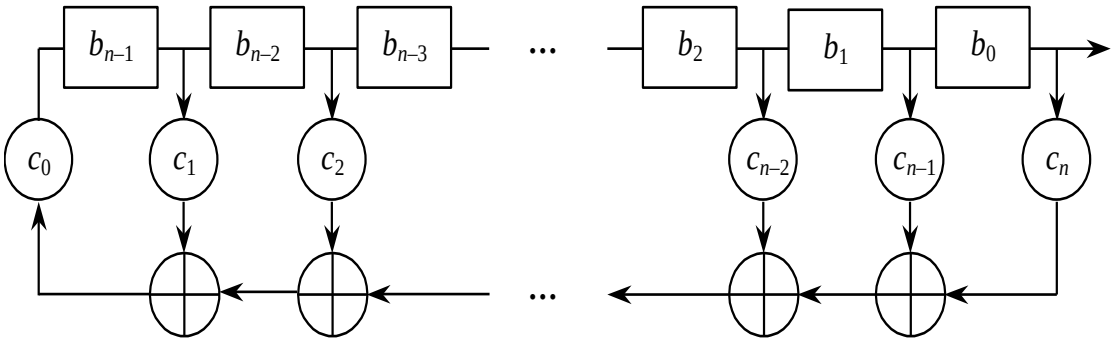


Рис. 2.20. LFSR, схема Фибоначчи

На каждом такте работы схемы Фибоначчи содержимое регистра сдвигается вправо на один бит, так что $b_i = b_{i+1}$ для $i = 0, \ldots, (n-2)$, а содержимое 0-ой ячейки b_0 поступает на выход LFSR. Новое содержимое $(n-1)$-ой ячейки b_{i-1} рассчитывается как сумма по модулю 2 предыдущих состояний определенных ячеек

$$b_{n-1} = \sum_{i=0}^{n-1} c_{n-i} b_i \ .$$

При $c_i = 1$ умножение на c_i равносильно наличию обратной связи, при $c_i = 0$ – отсутствию.

Схема Галуа представлена на рисунке 2.21.

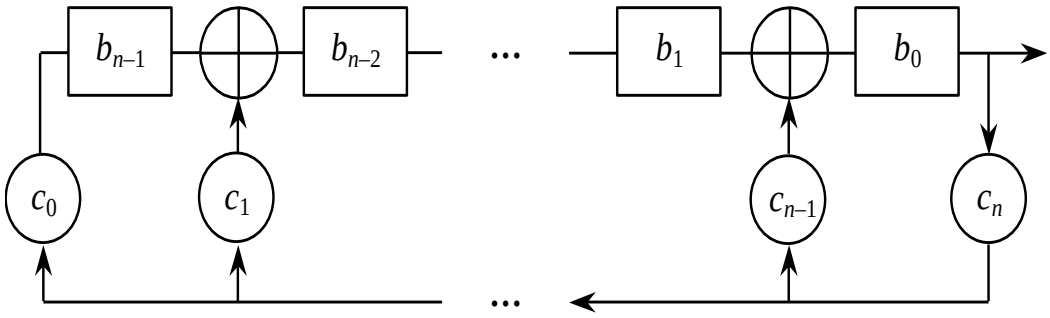


Рис. 2.21. LFSR, схема Галуа

На каждом такте работы схемы Галуа содержимое регистра сдвига сдвигается вправо на 1 бит так, что $b_i = b_{i+1} \oplus b_0 c_{n-i}$, а содержимое 0-ой ячейки b_0 поступает на выход LFSR и в (n – 1) ячейку регистра, т.е. $b_{n-1} = b_0$.

n-битовый LFSR может находиться в одном из (2^n – 1) внутренних состояний. Это значит, что теоретически такой регистр может генерировать псевдослучайную последовательность с периодом (2^n – 1) битов. (Это число равно (2^n – 1), а не 2^n, поскольку заполнение LFSR нулями влечет вывод регистром бесконечной последовательности нулей, что совершенно бесполезно.) Только при определенных последовательностях отводов LFSR циклически пройдет через все 2^n – 1 внутренних состояний. Такие регистры называются регистрами LFSR с максимальным периодом. Получившийся выход называют m-последовательностью.

Для обеспечения максимального периода конкретного LFSR, соответствующий многочлен, образованный из последовательности отводов регистра, должен быть примитивным по модулю 2. Степень многочлена является длиной регистра сдвига [4, 6].

Как бы ни был хорошо подобран полином обратной связи, LFSR остается линейным устройством. А такие устройства обычно легко поддаются криптоанализу независимо от того, насколько много параметров сохраняется в тайне. В современной криптографической литературе LFSR сами по себе не рекомендуются в качестве генераторов псевдослучайных шифрующих последовательностей [6].

Регистры LFSR сами по себе являются хорошими генераторами псевдослучайных последовательностей, но они обладают некоторыми нежелательными неслучайными свойствами. Последовательные биты линейны, что делает их бесполезными для шифрования. Внутреннее состояние LFSR длины n определяет следующие n выходных битов генератора. Даже если схема обратной связи хранится в секрете, она может быть определена по $2n$ выходным битам генератора с помощью высокоэффективного алгоритма Берлекампа-Мэсси [4]. В то же время подавляющее большинство реальных конструкций для поточного шифрования строится на основе LFSR.

Генераторы на основе LFSR

Поточные шифры на основе LFSR подразделяют на:

- системы с генератором с равномерным движением регистров;
- системы с генератором с неравномерным движением регистров.

В генераторе с равномерным движением регистров каждый раз для получения нового бита следует однократно сдвинуть LFSR. Выходной бит представляет собой функцию некоторых битов LFSR. Генераторы с равномерным движением в свою очередь делятся на [4, 5]:

- комбинирующий генератор;
- фильтрующий генератор.

Комбинирующий генератор (рисунок 6) состоит из нескольких параллельно работающих LFSR, выходы которых поступают на вход некоторой булевой функции *f*, комбинирующей эти выходы для генерации ключевого потока.

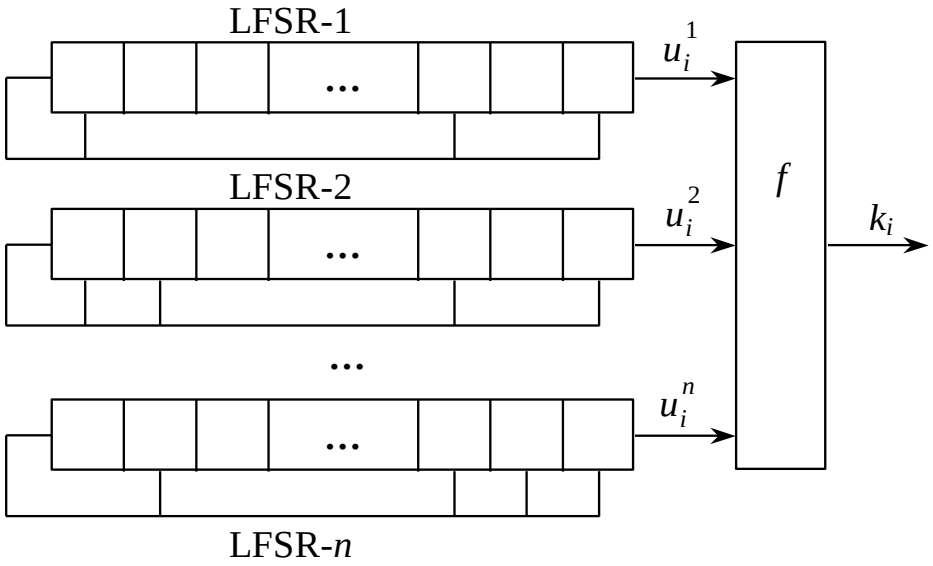


Рис. 2.22. Комбинирующий генератор

Ключом соответствующего поточного шифра является начальное заполнение каждого регистра, реже – начальное заполнение и функции обратных связей.

Результат работы комбинирующего генератора можно представить в виде

$$k_i = f\left(u_i^1, u_i^2, \ldots, u_i^n\right),$$

где k_i – *i*-ый бит ключевого потока, производимого генератором;

n – количество LFSR;

u_i^j – *i*-ый бит, генерируемый *j*-ым LFSR.

Фильтрующий генератор (рисунок 2.23) состоит из одного LFSR. Для генерации ключевого потока используется нелинейная функция *f*, на вход которой подаются значения некоторых ячеек LFSR. Функция *f* в этом случае называется фильтрующей функцией.

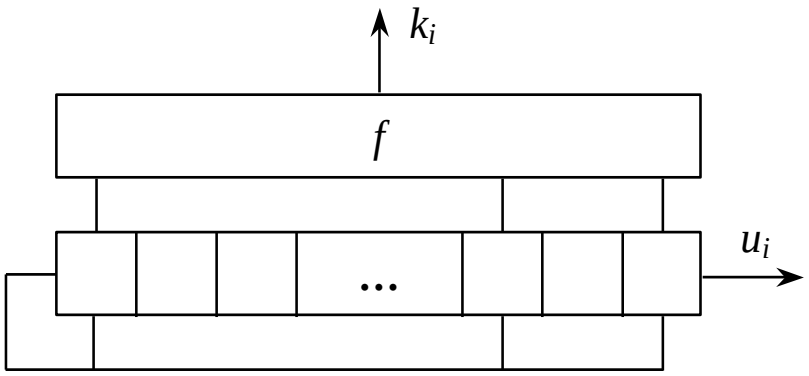


Рис. 2.23. Фильтрующий генератор

Ключом соответствующего поточного шифра является начальное заполнение регистра, реже – начальное заполнение и функция обратных связей.

Результат работы фильтрующего генератора можно представить в виде

$$k_i = f\left(u_i^1, u_i^2, \ldots, u_i^n\right),$$

где k_i – i-ый бит ключевого потока, производимого генератором;

n – длина LFSR;

u_i^j – состояние j-ой ячейки LFSR.

Для того чтобы получить ключевую последовательность, обладающую хорошими статистическими свойствами, необходимо чтобы обратные связи в LFSR были сформированы в соответствии с примитивными многочленами, а комбинирующая и фильтрующая функции формировали равномерно распределенные последовательности.

Параметры LFSR и комбинирующей и фильтрующей функций обычно общеизвестны, секретными являются начальный состояния LFSR, зависящие от ключа. Поэтому целью большинства атак на комбинирующие и фильтрующие генераторы является восстановление начальных состояний всех LFSR.

Фильтрующий генератор можно рассматривать как частный случай комбинирующего генератора, у которого все LFSR одинаковы, а начальное состояние LFSR-i совпадает с состоянием LFSR-1 на i-ом такте работы. Но, поскольку криптоанализ этих схем несколько различается, принято рассматривать эти схемы как два различных типа генераторов ключевого потока.

Примерами комбинирующего генератора являются: генератор Геффе и генератор Дженнингса. Примером фильтрующего генератора является алгоритм Nanoteq.

Регистры сдвига с нелинейной обратной связью

Как уже говорилось выше, регистр сдвига с обратной связью состоит из двух частей: регистра сдвига и функции обратной связи. В качестве функции обратной связи выступает любая булева функция f от n переменных. В случае, когда функция обратной связи f линейна,

регистр сдвига называется регистром сдвига с линейной обратной связью (LFSR), в противном случае – регистром сдвига с нелинейной обратной связью (non-linear feedback shift register, NLFSR).

Регистры сдвига с обратной связью по переносу

Регистр сдвига с обратной связью по переносу (feedback with carry shift register, FCSR) напоминает LFSR. В обоих используется регистр сдвига и функция обратной связи, но в FCSR дополнительно предусмотрен еще и регистр переноса.

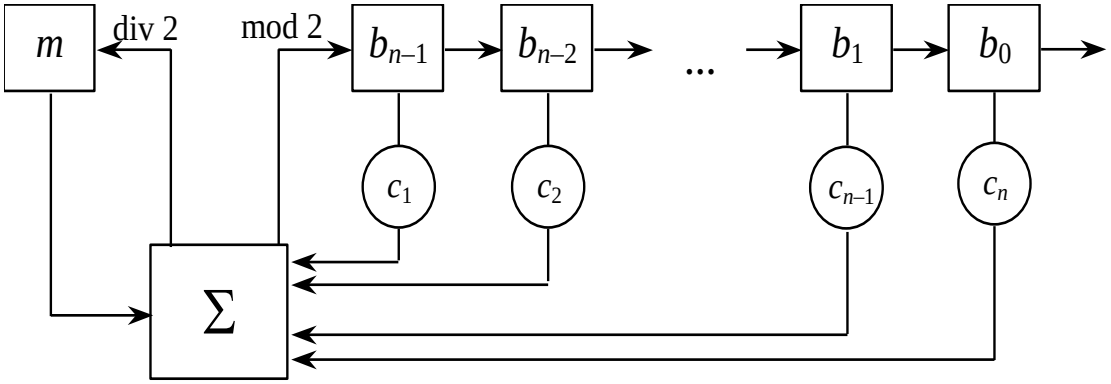


Рисунок 2.24. Регистр сдвига FCSR

На рисунке 2.24 знак Σ означает целочисленное сложение. Содержимое регистра сдвига состоит из n бит, обозначенных $b_{n-1}, b_{n-2}, \ldots, b_1, b_0$.

Работа FCSR описывается следующим образом [6]:

1. Вычисляется сумма $\sigma_n = \sum_{k=1}^{n} c_k b_{n-k} + m$.

2. Содержимое регистра сдвигается на одну позицию вправо. Содержимое крайней правой ячейки FCSR b_0 поступает на выход.

3. Рассчитывается новое содержимое крайней левой ячейки $b_{n-1} = \sigma_n \pmod 2$.

4. В регистр переноса записывается новое значение $m = \frac{\sigma_n - a_n}{2} = \left\lfloor \frac{\sigma_n}{2} \right\rfloor$.

Регистром переноса служит число, а не бит. Размер регистра переноса должен быть не менее $\log_2 t$, где t – количество отводов. Например, если отвода два, то регистр переноса однобитовый, а если отводов четыре, то регистр переноса должен состоять из 2 бит.

Максимальный период последовательности, генерируемой FCSR, равен $(c - 1)$, где c – число обратной связи (connection integer) FCSR. Число с определяется отводами обратной связи:

$c = c_n 2^n + c_{n-1} 2^{n-1} + \ldots + c_1 2^1 - 1$.

Последовательность максимального периода, генерируемая FCSR, называется *l*-

последовательностью. *l*-последовательность генерируется FCSR с числом обратной связи *c*, для которого 2 является корнем примитивным.

Поточные шифры

Обозначения

Ниже используются следующие обозначения:

- +: $(x + y)$ означает $x + y \pmod{2^{32}}$, где $0 \le x < 2^{32}$ и $0 \le y < 2^{32}$.
- ⊕: поразрядное исключающее ИЛИ (XOR).
- ⊗: поразрядное И (AND).
- &: логическое И.
- ||: конкатенация.
- >>: оператор сдвига вправо. x >> n означает, что x сдвигается вправо на n бит.
- <<: оператор сдвига влево. x << n означает, что x сдвигается влево на n бит.
- >>>: оператор циклического сдвига вправо. x >>> n означает $((x >> n) \oplus (x << (32 - n))$, где $0 \le n < 32$, $0 \le x < 2^{32}$.
- <<<: оператор циклического сдвига влево. x <<< n означает $((x << n) \oplus (x >> (32 - n))$, где $0 \le n < 32$, $0 \le x < 2^{32}$.
- ⊟: $(x \boxminus y)$ означает $x - y \pmod{512}$.
- $A^{[g..h]}$ обозначает биты с g по h переменной A.

Поточный шифр HC-128

Поточный шифр HC-128 [6] – упрощенная версия поточного шифра HC-256 для 128-битового уровня безопасности. HC-128 – простой и свободно доступный шифр, ориентированный на программную реализацию. Поточный шифр HC-128 использует 128 битовые ключ и вектор инициализации IV.

HC-128 состоит из двух секретных таблиц, каждая из которых содержит 512 32-разрядных элемента. На каждом шаге один элемент таблицы обновляется с помощью нелинейной функции обратной связи. Все элементы этих двух таблиц обновляются каждые 1024 шага. На каждом шаге, нелинейной функцией фильтрации выхода генерируется один 32-разрядный выходной блок.

Инициализация

Процесс инициализации HC-128 состоит в расширении ключа и вектора инициализации в таблицы P и Q (подобно SHA-256) и выполнении цикла шифрования (1024 раза), без генерации выходной последовательности (выходные последовательности используются для обновления P и Q).

В процессе инициализации выполняются следующие шаги:

1. Пусть $K = K_0||K_1||K_2||K_3$ и $IV = IV_0||IV_1||IV_2||IV_3$, где K_i and IV_i обозначают 32-битовые числа. Ключ и IV расширяются в массив W_i ($0 \le i \le 1279$):

$$W_i = \begin{cases} K_i & 0 \le i \le 7 \\ IV_{i-8} & 8 \le i \le 15 \\ f_2(W_{i-2}) + W_{i-7} + f_1(W_{i-15}) + W_{i-16} + i & 16 \le i \le 1279 \end{cases}$$

где функции $f_1(x)$ и $f_2(x)$ при $x = x_3||x_2||x_1||x_0$, (x – 32-битное слово, x_0, x_1, x_2, и x_3 – четыре байта. x_3 и x_0 обозначают соответственно самый старший байт и самый младший байт величины x) определяются как

$f_1(x) = (x >>> 7) \oplus (x >>> 18) \oplus (x >> 3)$,

$f_2(x) = (x >>> 17) \oplus (x >>> 19) \oplus (x >> 10)$.

2. Обновить таблицы P и Q массивом W.

$P[i] = W_{i+256}$, для $0 \le i \le 511$,

$Q[i] = W_{i+768}$, для $0 \le i \le 511$.

3. Выполнить цикл шифрования 1024 раза и использовать выходные последовательности для замены элементов таблицы следующим образом:

for i = 0 to 511 do
  $P[i] = (P[i] + g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511])) \oplus h_1(P[i \boxminus 12])$;

for i = 0 to 511 do
  $Q[i] = (Q[i] + g_2(Q[i \boxminus 3], Q[i \boxminus 10], Q[i \boxminus 511])) \oplus h_2(Q[i \boxminus 12])$;

где функции $g_1(x)$, $g_2(x)$, $h_1(x)$ и $h_2(x)$ при $x = x_3||x_2||x_1||x_0$ определяются как

$g_1(x, y, z) = ((x >>> 10) \oplus (z >>> 23)) + (y >>> 8)$,

$g_2(x, y, z) = ((x <<< 10) \oplus (z <<< 23)) + (y <<< 8)$,

$h_1(x) = Q[x_0] + Q[256 + x_2]$,

$h_2(x) = P[x_0] + P[256 + x_2]$.

Для функции $h_1(x)$ таблица Q используется как S-блок. Для функции $h_2(x)$, таблица P используется как S-блок.

Процесс инициализации завершен, и шифр готов к генерации ключевой последовательности.

Генерация ключевого потока

На каждом шаге, обновляется один элемент таблицы, и генерируется один 32-битовый блок выходной последовательности. *S*-блок используется для генерации только 512 выходных последовательностей, затем он обновляется за следующие 512 шага. Процесс генерации ключевой последовательности HC-128 описывается следующим псевдокодом (рисунок 1).

```
i = 0;
repeat {
   j = i mod 512;
   if (i mod 1024) < 512{
      P[j] = P[j] + g1(P[j − 3], P[j − 10], P[j − 511]);
      si = h1(P[j − 12]) ⊕ P[j];
    }
   else {
      Q[j] = Q[j] + g2(Q[j − 3], Q[j − 10], Q[j − 511]);
      si = h2(Q[j − 12]) ⊕ Q[j];
    }
   i = i + 1;
 }
```

Рисунок 1. Псевдокод генерации ключевой последовательности

s_i – 32-битовый выходной блок на i-м шаге процесса генерации ключевой последовательности.

Поточный шифр Rabbit

Алгоритм Rabbit в качестве входных данных использует 128-битовый секретный ключ и, если необходимо, 64-битовый вектор *IV* [6]. За одну итерацию генерирует блок 128 псевдослучайных бит. Шифрование/расшифрование производится путем сложения по модулю 2 (XOR) сгенерированной псевдослучайной последовательности с открытым/зашифрованным текстом. Размер внутреннего состояния – 513 бит, поделенных между восьмью 32-разрядными переменными состояния, восьмью 32-разрядными счетчиками и одним битом переноса. Эти восемь переменных состояния обновляются посредством восьми парных нелинейных функций. Счетчики гарантируют нижнюю границу на длине периода для переменных состояния.

Внутреннее состояние поточного шифра состоит из 513 битов. 512 битов разделены между восьмью 32-разрядными переменными состояния $x_{j,i}$ и восьмью 32-разрядными переменными счетчика $c_{j,i}$, где $x_{j,i}$ – переменная состояния j-ой подсистемы в i-ой итерации, а $c_{j,i}$ обозначает соответствующую переменную счетчика. Также есть один бит счетчика по переносу $\phi_{7,i}$, который должен сохраняться/накапливаться между итерациями. Этот бит счетчика по переносу инициализируется путем обнуления. Восемь переменных состояния и восемь счетчиков формируются из ключа при инициализации.

Инициализация

Алгоритм инициализируется разворачиванием 128-битового ключа в восемь переменных состояния и восемь счетчиков так, что есть взаимно однозначное соответствие между ключом и переменными начального состояния $x_{j,i}$ и начальными значениями счетчиков $c_{j,i}$.

Ключ $K^{[127..0]}$ разделен на восемь 16-битовых подключей:

$k_0 = K^{[15..0]}$, $k_1 = K^{[31..16]}$, …, $k_7 = K^{[127..112]}$.

Переменные состояния и счетчика инициализируются из подключей следующим образом:

$$x_{j,0} = \begin{cases} k_{(j+1 \bmod 8)} \parallel k_j, & \text{для четных } j \\ k_{(j+5 \bmod 8)} \parallel k_{(j+4 \bmod 8)}, & \text{для нечетных } j \end{cases}$$

и

$$x_{j,0} = \begin{cases} k_{(j+4 \bmod 8)} \parallel k_{(j+5 \bmod 8)}, & \text{для четных } j \\ k_j \parallel k_{(j+1 \bmod 8)}, & \text{для нечетных } j \end{cases}.$$

Чтобы уменьшить корреляцию между битами ключа и битами переменных внутреннего состояния система повторяется четыре раза, в соответствии с функцией следующего состояния, определенной ранее. Наконец, переменные счетчика повторно инициализируются согласно выражению:

$c_{j,4} = c_{j,4} \oplus x_{(j+4 \bmod 8),4}$

для всех j, чтобы предотвратить восстановление ключа обратным преобразованием системы счетчика.

Обозначим внутреннее состояние после применения схемы установки ключа как основное состояние. Пусть копия этого основного состояния будет изменена в соответствии со схемой установки *IV*. Схема установки *IV* изменяет состояние счетчика как функция *IV*. Это реализуется путем применения операции XOR к 64-битовым *IV* и всеми 256 битами состояния счетчика. 64 бита *IV* обозначаются $IV^{[63..0]}$. Счетчики изменяются следующим образом:

$c_{0,4} = c_{0,4} \oplus IV^{[31..0]}$,

$c_{1,4} = c_{1,4} \oplus (IV^{[63..48]} \parallel IV^{[31..16]})$,

$c_{2,4} = c_{2,4} \oplus IV^{[63..32]}$,

$c_{3,4} = c_{3,4} \oplus (IV^{[47..32]} \parallel IV^{[15..0]})$,

$c_{4,4} = c_{4,4} \oplus IV^{[31..0]}$,

$c_{5,4} = c_{5,4} \oplus (IV^{[63..48]} \parallel IV^{[31..16]})$,

$c_{6,4} = c_{6,4} \oplus IV^{[63..32]}$,

$c_{7,4} = c_{7,4} \oplus (IV^{[47..32]} \,||\, IV^{[15..0]})$.

Чтобы сделать все биты состояния нелинейно зависящими от всех битов *IV* система повторяется четыре раза. Модификация счетчика с помощью *IV* гарантирует, что все 2^{64} различные вектора *IV* приведут к уникальным ключевым последовательностям.

Генерация ключевого потока

Ядром алгоритма Rabbit является повторение функции выработки следующего состояния, определенной уравнениями:

$x_{0,i+1} = g_{0,i} + (g_{7,i} <<< 16) + (g_{6,i} <<< 16)$,

$x_{1,i+1} = g_{1,i} + (g_{0,i} <<< 8) + g_{7,i}$,

$x_{2,i+1} = g_{2,i} + (g_{1,i} <<< 16) + (g_{0,i} <<< 16)$,

$x_{3,i+1} = g_{3,i} + (g_{2,i} <<< 8) + g_{1,i}$,

$x_{4,i+1} = g_{4,i} + (g_{3,i} <<< 16) + (g_{2,i} <<< 16)$,

$x_{5,i+1} = g_{5,i} + (g_{4,i} <<< 8) + g_{3,i}$,

$x_{6,i+1} = g_{6,i} + (g_{5,i} <<< 16) + (g_{4,i} <<< 16)$,

$x_{7,i+1} = g_{7,i} + (g_{6,i} <<< 8) + g_{5,i}$,

$g_{j,i} = ((x_{j,i} + c_{j,i+1})^2 \oplus ((x_{j,i} + c_{j,i+1})^2 >> 32)) \bmod 2^{32}$,

где все операции сложения приводятся по модулю 2^{32}. Эта двойная система приведена на рисунке 2.25. Перед каждой итерацией счетчики увеличиваются в соответствии с описанным ниже правилом.

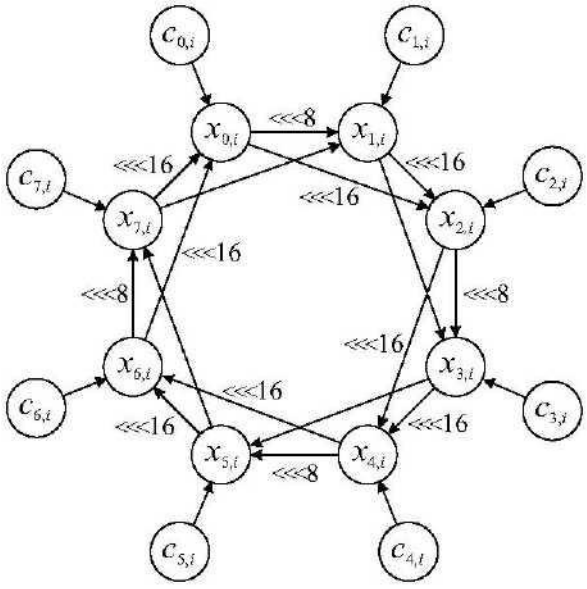


Рис. 2.25. Графическое представление системы

Работа счетчиков определяется следующим образом:

$c_{0,i+1} = c_{0,i} + a_0 + \phi_{7,i} \bmod 2^{32}$,

$c_{1,i+1} = c_{1,i} + a_0 + \phi_{0,i+1} \bmod 2^{32}$,

$c_{2,i+1} = c_{2,i} + a_0 + \phi_{1,i+1} \bmod 2^{32}$,

$c_{3,i+1} = c_{3,i} + a_0 + \phi_{2,i+1} \bmod 2^{32}$,

$c_{4,i+1} = c_{4,i} + a_0 + \phi_{3,i+1} \bmod 2^{32}$,

$c_{5,i+1} = c_{5,i} + a_0 + \phi_{4,i+1} \bmod 2^{32}$,

$c_{6,i+1} = c_{6,i} + a_0 + \phi_{5,i+1} \bmod 2^{32}$,

$c_{7,i+1} = c_{7,i} + a_0 + \phi_{6,i+1} \bmod 2^{32}$,

где бит счетчика по переносу $\phi_{j,i+1}$ задан выражением

$$\phi_{j,i+1} = \begin{cases} 1 & \text{åñëè } c_{0,i} + a_0 + \phi_{7,i} \geq 2^{32} \wedge j = 0 \\ 1 & \text{åñëè } c_{j,i} + a_j + \phi_{j-1,i+1} \geq 2^{32} \wedge j > 0 \\ 0 & \text{âî âñåõ îñòàëüíûõ ñëó÷àÿõ} \end{cases}$$

Кроме того, константы a_j определены как:

a_0 = 0x4D34D34D, a_1 = 0xD34D34D3,

a_2 = 0x34D34D34, a_3 = 0x4D34D34D,

a_4 = 0xD34D34D3, a_5 = 0x34D34D34,

a_6 = 0x4D34D34D, a_7 = 0xD34D34D3.

После каждой итерации результат извлекается следующим образом:

$s_i^{[15..0]} = x_{0,i}^{[15..0]} \oplus x_{5,i}^{[31..16]}$, $s_i^{[31..16]} = x_{0,i}^{[31..16]} \oplus x_{3,i}^{[15..0]}$,

$s_i^{[47..32]} = x_{2,i}^{[15..0]} \oplus x_{7,i}^{[31..16]}$, $s_i^{[63..48]} = x_{2,i}^{[31..16]} \oplus x_{5,i}^{[15..0]}$,

$s_i^{[79..64]} = x_{4,i}^{[15..0]} \oplus x_{1,i}^{[31..16]}$, $s_i^{[95..80]} = x_{4,i}^{[31..16]} \oplus x_{7,i}^{[15..0]}$,

$s_i^{[111..96]} = x_{6,i}^{[15..0]} \oplus x_{3,i}^{[31..16]}$, $s_i^{[127..112]} = x_{6,i}^{[31..16]} \oplus x_{1,i}^{[15..0]}$.

Поточный шифр Salsa20

Ядром шифра Salsa20 является хеш-функция с 64-байтовым входом и 64-байтовым выходом [6]. Хеш-функция в режиме счетчика используется как поточный шифр: Salsa20 шифрует 64-байтовый блок открытого текста хешированием ключа, в данном случае, и номера блока, складывая результат по модулю 2 (XOR) с открытым текстом.

Хеш-функция Salsa20

Хеш-функция Salsa20(x) определяется следующим выражением:

Salsa20(x) = x + doubleround^{10}(x),

где каждая 4-байтовая последовательность x рассматривается как слово в форме littleendian.

Если $b = (b_0, b_1, b_2, b_3)$ – 32-битовое слово, где b_3 и b_0 обозначают соответственно самый старший байт и самый младший байт величины b, тогда

littleendian(b) = $2^{24}b_3 + 2^{16}b_2 + 2^8b_1 + b_0$.

Функция doubleround(x) вычисляется путем последовательного применения к последовательности x из 16 слов (слово – 32-битовый элемент) функций columnround(x) и rowround(x):

doubleround(x) = rowround(columnround(x)).

Функций columnround(x) и rowround(x) в свою очередь строятся на основе функции quarterround(y). Функция quarterround(y) оперирует последовательностями из 4 слов.

Если $y = (y_0, y_1, y_2, y_3)$, тогда quarterround(y) = (z_0, z_1, z_2, z_3), где

$z_1 = y_1 \oplus ((y_0 + y_3) <<< 7)$,

$z_2 = y_2 \oplus ((z_1 + y_0) <<< 9)$,

$z_3 = y_3 \oplus ((z_2 + z_1) <<< 13)$,

$z_0 = y_0 \oplus ((z_3 + z_2) <<< 18)$.

Функцию quarterround можно представить как изменение y следующим образом: сначала y_1 изменяется на z_1, затем y_2 изменяется на z_2, затем y_3 изменяется на z_3, затем y_0 изменяется на z_0. Каждое изменение является обратимым, таким образом, вся функция является обратимой.

Функция rowround(y) оперирует последовательностями из 16 слов.

Если $y = (y_0, y_1, y_2, y_3, \ldots, y_{15})$, тогда rowround($y$) = $(z_0, z_1, z_2, z_3, \ldots, z_{15})$, где

(z_0, z_1, z_2, z_3) = quarterround (y_0, y_1, y_2, y_3),

(z_5, z_6, z_7, z_4) = quarterround (y_5, y_6, y_7, y_4),

$(z_{10}, z_{11}, z_8, z_9)$ = quarterround $(y_{10}, y_{11}, y_8, y_9)$,

$(z_{15}, z_{12}, z_{13}, z_{14})$ = quarterround $(y_{15}, y_{12}, y_{13}, y_{14})$.

Можно представить вход $(y_0, y_1, \ldots, y_{15})$ в виде квадратной матрицы:

$$\begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix}$$

Функция rowround изменяет строки матрицы параллельно, пропуская перестановку каждой строки через функцию quarterround. В первой строке функция rowround изменяет y_1, затем y_2, затем y_3, затем y_0; во второй строке функция rowround изменяет y_6, затем y_7, затем y_4,

затем y_5; в третьей строке функция rowround изменяет y_{11}, затем y_8, затем y_9, затем y_{10}; в четвертой строке функция rowround изменяет y_{12}, затем y_{13}, затем y_{14}, затем y_{15}.

Функция columnround(x) также как и функция rowround(y) оперирует последовательностями из 16 слов.

Если $x = (x_0, x_1, x_2, x_3, \ldots, x_{15})$ тогда columnround(x) = $(y_0, y_1, y_2, y_3, \ldots, y_{15})$, где

(y_0, y_4, y_8, y_{12}) = quarterround (x_0, x_4, x_8, x_{12}),

(y_5, y_9, y_{13}, y_1) = quarterround (x_5, x_9, x_{13}, x_1),

$(y_{10}, y_{14}, y_2, y_6)$ = quarterround $(x_{10}, x_{14}, x_2, x_{16})$,

$(y_{15}, y_3, y_7, y_{11})$ = quarterround $(x_{15}, x_3, x_7, x_{11})$.

Эквивалентная формула: $(y_0, y_4, y_8, y_{12}, y_1, y_5, y_9, y_{13}, y_2, y_6, y_{10}, y_{14}, y_3, y_7, y_{11}, y_{15})$ = rowround$(x_0, x_4, x_8, x_{12}, x_1, x_5, x_9, x_{13}, x_2, x_6, x_{10}, x_{14}, x_3, x_7, x_{11}, x_{15})$.

Можно представить вход $(x_0, x_1, \ldots, x_{15})$ в виде квадратной матрицы:

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}$$

Функция columnround с этого ракурса представляется просто заменой (транспонированием) функции rowround. Функция columnround, изменяет столбцы матрицы параллельно, пропуская перестановку каждого столбца через функцию quarterround. В первом столбце, функция columnround изменяет y_4, затем y_8, затем y_{12}, затем y_0; во втором столбце, функция columnround изменяет y_9, затем y_{13}, затем y_1, затем y_5; в третьем столбце, функция columnround изменяет y_{14}, затем y_2, затем y_6, затем y_{10}; в четвертом столбце, функция columnround изменяет y_3, затем y_7, затем y_{11}, затем y_{15}.

Инициализация

Если ключ k – 32-байтовая или 16-байтовая последовательность, а iv – 16-байтовая последовательность, тогда Salsa20$_k$(iv) является 64-байтовой последовательностью.

Определим σ_0 = (101, 120, 112, 97), σ_1 = (110, 100, 32, 51), σ_2 = (50, 45, 98, 121), и σ_3 = (116, 101, 32, 107). Если k_0, k_1, iv являются 16-байтовыми последовательностями, тогда

Salsa20$_{k_0,k_1}$(iv) = Salsa20(σ_0, k_0, σ_1, iv, σ_2, k_1, σ_3).

Определим τ_0 = (101, 120, 112, 97), τ_1 = (110, 100, 32, 49), τ_2 = (54, 45, 98, 121), и τ_3 = (116, 101, 32, 107). Если k, iv являются 16-байтовыми последовательностями, тогда

Salsa20$_k$(iv) = Salsa20(τ_0, k, τ_1, iv, τ_2, k, τ_3).

Константы $\sigma_0\sigma_1\sigma_2\sigma_3$ и $\tau_0\tau_1\tau_2\tau_3$, равны “expand 32-byte k” и “expand 16-byte k” (в ASCII) соответственно.

Функция шифрования Salsa20

Пусть k – это 32- или 16-байтовый секретный ключ, iv – 8-байтовый вектор инициализации, m – открытый текст. Тогда шифрование последовательности m шифром Salsa20 с помощью вектор инициализации iv на ключе k, обозначается $\text{Salsa20}_k(iv) \oplus m$ – шифрованный текст. Либо m может быть шифрованным текстом, в случае, когда $\text{Salsa20}_k(iv) \oplus m$ является исходным открытым текстом.

Поточный шифр SOSEMANUK

Sosemanuk – синхронный поточный шифр [6]. Шифр Sosemanuk использует принципы базовой конструкции поточного шифра SNOW 2.0 преобразования блочного шифра SERPENT. Поэтому разработчики решили, что название этого шифра должно относиться и к SERPENT и к SNOW. Однако, известно, что снежных змей не существуют, т.к. змеи либо впадают в спячку, либо перемещаются в более теплые страны на зимы. С другой стороны Sosemanuk – популярный спорт, в который играют восточные канадские племена. Он состоит в броске деревянной палки по сугробу как можно дальше. Его название на языке Cree означает snowsnake, так как палка на снегу похожа на змею. Kwakweco-cime win – вариант той же самой игры, но для названия шифра не походит.

Sosemanuk – синхронный поточный шифр, ориентированный на программную реализацию. Размер ключа варьируется между 128 и 256 битами. Утверждается, что при любой длине ключа достигается 128-битная безопасность. Sosemanuk стремится улучшить SNOW 2.0 в двух отношениях. Во-первых, в Sosemanuk избегаются некоторые свойства структуры, которые могут проявиться как потенциальные слабости, даже если шифр SNOW 2.0 с 128-битным ключом сопротивляется всем известным нападениям. Во-вторых, эффективность улучшена относительно нескольких архитектур, уменьшением размера внутреннего состояния, таким образом допускается более прямое отображение данных относительно регистров процессора. Sosemanuk также требует меньшего количества статических данных; соответственно более низкого использования кэша данных, что приводит к лучшей работе на некоторых архитектурах. Другое достоинство Sosemanuk – то, что его процедура установки ключа основана на сокращенной версии блочного шифра SERPENT, улучшая классические процедуры инициализации и с точки зрения эффективности и с точки зрения безопасности.

SERPENT и его производные

SERPENT – блочный шифр, предложенный в качестве AES кандидата. SERPENT работает с 128-битовыми блоками, которые разбиваются на четыре 32-разрядных слова, а

затем объединяются в так называемом “секционном” (“bitslice”) режиме. Таким образом SERPENT может быть определен как шифр, работающий с четверкой 32-битовых слов. Мы нумеруем входные и выходные четверки SERPENT’а от 0 до 3, и пишем им в порядке: (Y_3, Y_2, Y_1, Y_0). Y_0 – самое младшее слово, содержащее младшие разряды 32-ух 4-битовых входных данных для S-блоков SERPENT’а. После того как выходной поток SERPENT’а записывается в 16 байтов, в значения Y_i записываются в соответствии со следующим условным обозначением littleendian (вначале самый младший байт), и сначала выводится Y_0, затем Y_1, и так далее.

Из SERPENT разработчики определили два примитива по имени Serpent1 и Serpent24.

Циклы SERPENT’а состоят из:

- добавления подключа поразрядным исключающим или (XOR);
- применения *s*-блока, которое заключается в ряде поразрядных комбинаций между четырьмя обрабатываемыми 32-битовыми словами, в секционном режиме (bitslice mode);
- линейного биективного преобразования, которое равнозначно нескольким XOR’ам, сдвигам и циклическим сдвигам в секционном режиме (bitslice mode).

Serpent1 – один раунд шифра SERPENT, без добавления ключа и линейного преобразования. SERPENT использует восемь различных *S*-блоков, пронумерованных от S_0 до S_7, рассчитанных на 4-битовые слова. Мы определяем Serpent1 как применение S_2, в секционном режиме (bitslice mode). Это третий S-блоковый уровень шифра SERPENT. Serpent1 использует в качестве входных данных четыре 32-битовых слова, и вырабатывает четыре 32-битовых слова в качестве выхода.

Serpent24 – это SERPENT, сокращенный до 24 раундов, вместо полной 32-х раундовой версии SERPENT. Serpent24 соответствует первым 24 раундам шифра SERPENT, причем последний (24-й) раунд полный – с линейным преобразованием и XOR’ом с 25-ым подключом. Другими словами, 24-ый раунд Serpent24 эквивалентен тридцать второму раунду шифра SERPENT, за исключением того, что содержит линейное преобразование и использует 24-ый и 25-ый подключи (32-ый и 33-ий подключи в SERPENT). Соответствующее уравнение последнего раунда приведено в [32_Serpent]

$$R_{23}(X) = L\left(\hat{S}_{23}\left(X \oplus \hat{K}_{23}\right)\right) \oplus \hat{K}_{24}$$

Serpent24 использует только 25 128-битовых подключей, которые являются первыми 25 подключами, производящимися в соответствии со схемой разворачивания ключа шифра

SERPENT. В Sosemanuk Serpent24 используется на этапе инициализации, только в режиме шифрования. При расшифровании не используется.

Инициализация

Процесс инициализации Sosemanuk разбит на два шага:

- схема разворачивания ключа, которая подвергает обработке секретный ключ, но не зависит от вектора инициализации *IV*;
- добавление вектора инициализации *IV*, которое использует выход схемы разворачивания ключа и вектор *IV*.

Таким образом, инициализируется внутреннее состояние поточного шифра.

Установка ключа соответствует схеме разворачивания ключа в Serpent24, которая производит 25 128-битовых подключей, как 100 32-битовых слова. Эти 25 128-битовых подключей идентичны первым 25 128-битовым подключам, производящимся в соответствии с простой схемой разворачивания ключа в SERPENT.

SERPENT допускает любую длину ключа длину от 1 до 256 битов; следовательно, Sosemanuk может работать с точно такими же ключами. Однако, так как Sosemanuk стремится к 128-битовой безопасности, длина его ключа должна быть по крайней мере 128 битов. Поэтому, 128 битов – стандартная длина ключа. Поддерживается любая длина ключа от 128 битов до 256 битов. Но, уровень безопасности все же соответствует 128-битовой безопасности. Другими словами, использование более длинного секретного ключа не гарантирует обеспечение уровня безопасности, обычно ожидаемого от такого ключа.

Вектор *IV* является 128-битовым значением. Используется в качестве входа для блочного шифра Serpent24, инициализированный в соответствии со схемой разворачивания ключа. Serpent24 состоит из 24 раундов, используются выходы 12-го, 18-го и 24-го раундов. Мы обозначим эти выоды следующим образом:

- $(Y_3^{12}, Y_2^{12}, Y_1^{12}, Y_0^{12})$ – выход 12-ого раунда;
- $(Y_3^{18}, Y_2^{18}, Y_1^{18}, Y_0^{18})$ – выход 18-ого раунда;
- $(Y_3^{24}, Y_2^{24}, Y_1^{24}, Y_0^{24})$ – выход 24-ого раунда.

Выход каждого раунда состоит из четырех 32-битовых слов, берущихся только после линейного преобразования, за исключением 24-ого раунда, для которого выход берется только после добавления 25-ого подключа.

Эти значения используются для инициализации внутреннего состояния Sosemanuk’a со следующими значениями:

$(s_7, s_8, s_9, s_{10}) = (Y_3^{12}, Y_2^{12}, Y_1^{12}, Y_0^{12})$

$(s_5, s_6) = (Y_1^{18}, Y_3^{18})$

$(s_1, s_2, s_3, s_4) = (Y_3^{24}, Y_2^{24}, Y_1^{24}, Y_0^{24})$

$R1_0 = Y_0^{18}$

$R2_0 = Y_2^{18}$

Генерация ключевого потока

Для формирования выходных значений z_t используется конечный автомат (FSM, Finite State Machine) и функция Serpent1.

Конечный автомат (FSM) – компонент с 64 битами памяти, аналогичный двум 32-битовым регистрам $R1$ и $R2$. В каждом шаге FSM берет в качестве входных данных некоторые слова из состояния LFSR; обновляет биты памяти и производит 32-битовый выход. FSM оперирует состоянием LFSR в моменты времени $t \geq 1$ следующим образом:

FSM_t: $(R1_{t-1}, R2_{t-1}, s_{t+1}, s_{t+8}, s_{t+9}) \mapsto (R1_t, R2_t, f_t)$

где

$R1_t = (R2_{t-1} + \text{mux}(\text{lsb}(R1_{t-1}), s_{t+1}, s_{t+1} \oplus s_{t+8})) \bmod 2^{32}$

$R2_t = Trans(R1_{t-1})$

$f_t = (s_{t+9} + R1_t \bmod 2^{32}) \oplus R2_t$

где $\text{lsb}(x)$ – младший бит x,

$\text{mux}(c, x, y)$ равен x если $c = 0$, или y если $c = 1$.

Внутренняя переходная функция *Trans* над полем $F_2{}^{32}$ определяется как

$Trans(z) = (M \times z \bmod 2^{32})_{<<<7}$

где M – постоянное значение, равное 0x54655307 (шестнадцатеричное выражение первых десяти десятичных чисел π).

Выходы конечного автомата FSM группируются по четыре, и Serpent1 применяется к каждой группе; затем результат объединяется XOR’ом с соответствующими отбрасываемыми значениями LFSR, для производства выходных значений z_t:

$(z_{t+3}, z_{t+2}, z_{t+1}, z_t) = \text{Serpent1}(f_{t+3}, f_{t+2}, f_{t+1}, f_t) \oplus (s_{t+3}, s_{t+2}, s_{t+1}, s_t)$

Четыре последовательных раунда Sosemanuk изображены на рисунке 2.26

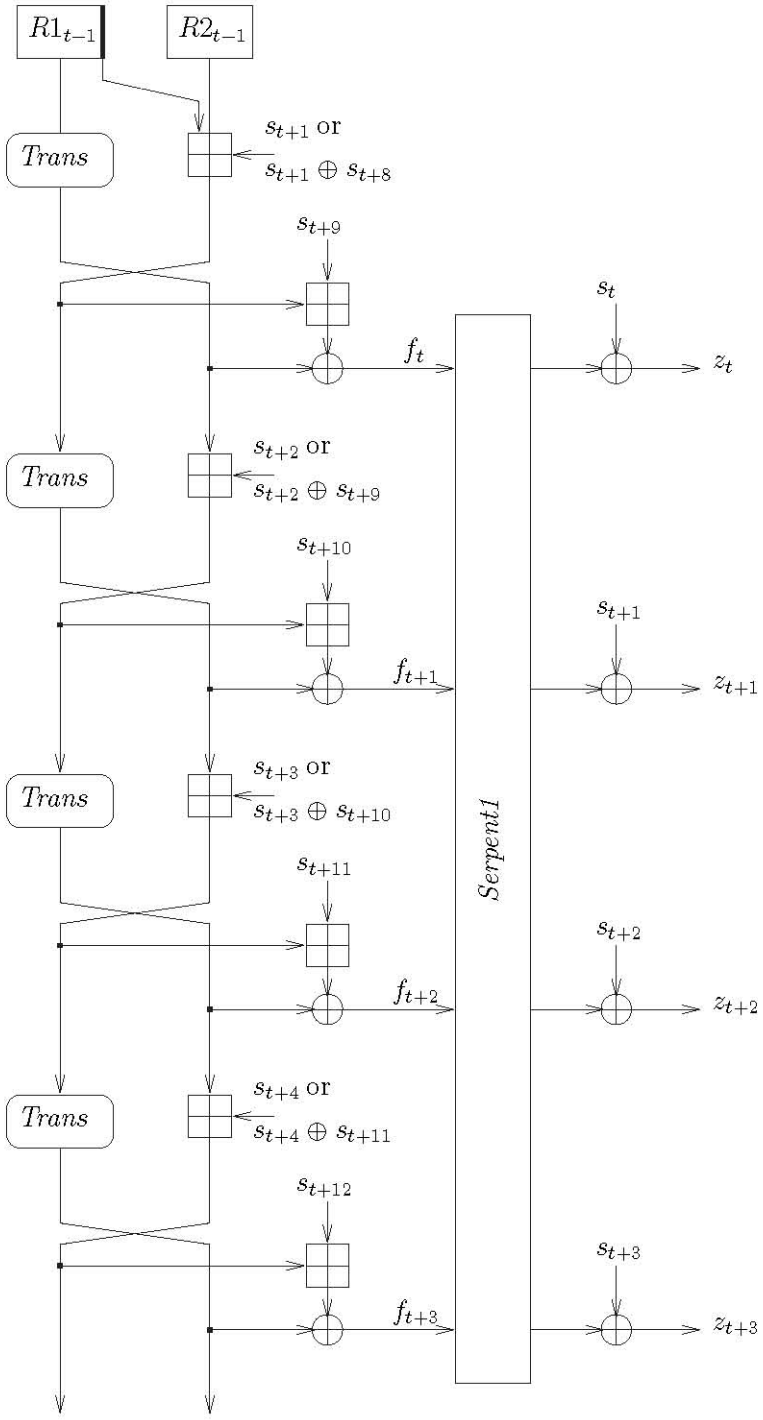


Рис. 2.26. Выходное преобразование четырех последовательных раундов Sosemanuk

Шифр Sosemanuk объединяет FSM и LFSR для производства выходных значений z_t. Время $t = 0$ определяет внутреннее состояние после инициализации; первое выходное значение – z_1. На рисунке 3 представлено описание шифра Sosemanuk.

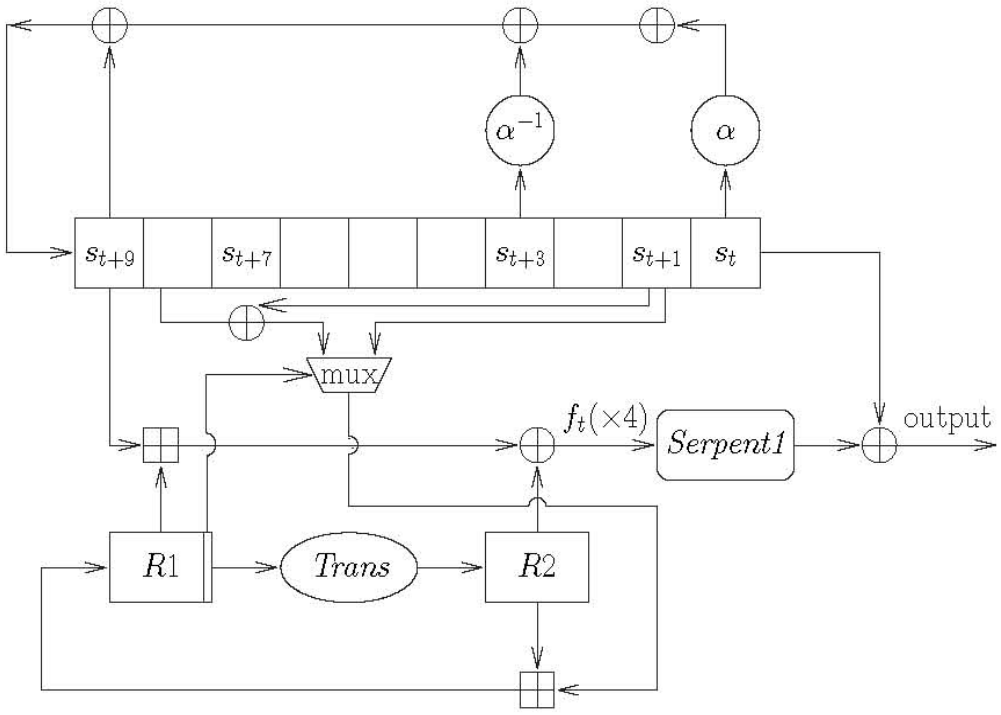


Рис. 2.27. Описание шифра Sosemanuk

В моменты времени $t \geq 1$ выполняются следующие операции:

- обновляется FSM: из $R1_{t-1}$, $R2_{t-1}$, s_{t+1}, s_{t+8} и s_{t+9} вычисляются $R1_t$, $R2_t$ и промежуточные значения f_t.
- обновляется LFSR: из s_t, s_{t+3} и s_{t+9} вычисляется s_{t+10}. Значение s_t передается внутреннему буферу, LFSR сдвигается.

Каждые четыре шага, из накопленных значений f_t, f_{t+1}, f_{t+2}, f_{t+3} и s_t, s_{t+1}, s_{t+2}, s_{t+3} производятся четыре выходных значения z_t, z_{t+1}, z_{t+2}, и z_{t+3}. Таким образом, Sosemanuk производит 32-битовые значения.

Соответственно первые четыре итерации Sosemanuk'а следующие:

- Начальное состояние LFSR содержит значения s_1, …, s_{10}; значение s_0 не определено. Начальное состояние FSM содержит $R1_0$ и $R2_0$.
- В течение первого шага из $R1_0$, $R2_0$, s_2, s_9 и s_{10} вычисляются $R1_1$, $R2_1$ и f_1.
- Первый шаг производит промежуточные значения s_1 и f_1, сохраняемые в буфере.
- В течение первого шага, вычисляется слово обратной связи s_{11} из s_{10}, s_4 и s_1, обновляется внутреннее состояние LFSR, что приводит к новому состоянию, формируемое из s_2, …, s_{11}.

- Первые четыре выходные значения – z_1, z_2, z_3 и z_4, вычисляются, используя однократное применение Serpent1'а к (f_4, f_3, f_2, f_1), выход которого объединяется с (s_4, s_3, s_2, s_1) при помощи XOR.

Поточный шифр F-FCSR-H

F-FCSR-H – аддитивный поточный шифр [5].

Для генерации ключевого потока из ключа длиной 80 бит и вектора инициализации длиной от 32 до 80 бит используется фильтрующий автомат FCSR.

Генерация ключевого потока

В основе шифра F-FCSR-H лежит регистр сдвига с обратной связью по переносу (FCSR) – автомат, который вычисляет двоичное разложение 2-адического числа p/q, где p и q – некоторые целые числа, с q нечетное. Допустим, что $q < 0 < p < |q|$. Размер FCSR n такой, что $n + 1$ является длиной q в битах.

В данном шифре p зависит от секретного ключа (и IV), а q – открытый параметр. Выбор q порождает много свойств ключевого потока. Самое важное – то, что это полностью определяет длину периода ключевого потока. Условия для оптимального выбора:

- q – (отрицательное) простое число размером ($n + 1$) бит.
- ($|q| - 1$) – порядок 2 по модулю q.
- $T = (q - 1)/2$ также является простым числом.
- $d = (1 + |q|)/2$. $W(d)$ – вес Хемминга двоичного разложения, $W(d) > n/2$.

Автомат FCSR содержит два регистра (наборы ячеек): основной регистр M и регистр переноса C. Основной регистр M содержит n ячеек. Обозначим m_i ($0 \le i \le n - 1$) двоичные знаки ($n-1$), содержащиеся в этих ячейках, и назовем числами $m = \sum_{i=0}^{n-1} m_i 2^i$ содержимое (или состояние) регистра M.

Пусть d – положительное целое число $d = (1 - q)/2$, а $d = \sum_{i=0}^{n-1} d_i 2^i$ его двоичное разложение. Регистр переносов содержит l ячеек, где ($l + 1$) – количество чисел d_i отличных от нуля. Более строго, регистр переносов содержит одну ячейку для каждого ненулевого d_i, для $0 \le i \le (n - 2)$. Обозначим c_i – двоичный знак, содержащийся в этой ячейке. Мы также устанавливаем $c_i = 0$ когда $d_i = 0$ или когда $i = (n - 1)$. Назовем числом $c = \sum_{i=0}^{n-2} c_i 2^i$ содержимое (или состояние) регистра C. Вес Хемминга двоичного разложения c не больше, чем l. Функция перехода может быть описана выражениями

$m(t+1) = (m(t) >> 1) \oplus c(t) \oplus m_0(t)d$,

$c(t+1) = (m(t) >> 1) \otimes c(t) \oplus c(t) \otimes m_0(t)d \oplus m_0(t)d \otimes (m(t) >> 1)$.

Отметим, что $m_0(t)$ – самый младший бит в $m(t)$. Числа $m(t)$, $c(t)$ и d – целые числа размером в n бит (или меньше).

Для извлечения псевдослучайных бит ключевого потока из основного регистра автомата FCSR используется фильтр. Этот фильтр описывает, какие ячейки выбраны для производства псевдослучайных битов. Чтобы получить мультиразрядный выход, используются восемь или шестнадцать одноразрядных фильтров, чтобы извлечь 8- или 16-битовые выходные слова после каждого перехода автомата.

Фильтр F – это битовая строка (f_0, …, f_{n-1}) длиной n (что эквивалентно числу $\sum_{i=0}^{n-1} f_i 2^i$). Выходной бит z получается вычислением веса *parity* поразрядного И состояния M основного регистра и фильтра F:

$$z = \bigoplus_{i=0}^{n-1} f_i m_i .$$

Это эквивалентно следующему:

$S = M \otimes F$,

$z = parity(S)$.

Аналогичным способом, предлагается метод извлечения s-битового слова из состояния FCSR. Значение s будет равно 8 для F-FCSR-H, и 16 для F-FCSR-16.

Фильтр F также является битовой строкой (f_0, …, f_{n-1}) длиной n (которая является кратной числу s). Он разбивает на s подфильтров F_0, …, F_{s-1} каждый определяется как

$$F_j = \sum_{i=0}^{n/s-1} f_{si+j} 2^i .$$

Каждый подфильтр F_j выбирает несколько ячеек m_i в основном регистре среди тех, что удовлетворяют выражению $i \equiv j \bmod s$. *Parity* полученного двоичного слова дает j-й псевдослучайный бит:

$$z_j = \bigoplus_{i=0}^{n/s-1} f_{si+j} m_{si+j} .$$

Так как есть s подфильтров, то мы получаем s битов при каждом переходе автомата.

Эта процедура может быть описана эквивалентно следующим образом. Фильтр F и состояние M комбинируются функцией И. Результат разбивается на n/s слова. Псевдослучайное слово получается операцией XOR этих n/s слов:

$S = M \otimes F$

Определим S_i с помощью выражения $S = \sum_{i=0}^{n/s-1} S_i \cdot 2^{si}$, для $0 \le S_i \le 2^8 - 1$.

Выходное слово z будет равно:

$$z = \bigoplus_{i=0}^{n/s-1} S_i .$$

Отметим, что целое слово извлекается быстрее, чем отдельный бит.

Шифр F-FCSR-H использует ключи длиной 80 бит и *IV* размером от 32 до 80 бит. Если *IV* не используется, то по умолчанию можно использовать значение 0.

Длина FCSR (размер основного регистра) – $n = 160$. Регистр переносов содержит $l = 82$ ячейки. Обратное простое число

$q = -$ 1993524591318275015328041611344215036460140087963

таким образом сложение полей и ячеек переносов присутствуют в позициях, соответствующих тем (кроме ведущего) в следующей строке 160 битов (которая имеет вес Хемминга 83),

$d = (1 + |q|)/2 = (\text{AE985DFF 26619FC5 8623DC8A AF46D590 3DD4254E})_{16}$.

Фильтрация

Чтобы извлечь один псевдослучайный байт, мы используем статический фильтр

$F = d = (\text{AE985DFF 26619FC5 8623DC8A AF46D590 3DD4254E})_{16}$

Фильтр F разбит на 8 подфильтров (подфильтр j получен выбором бита j в каждом байте F),

$F_0 = (0011\ 0111\ 0100\ 1010\ 1010)_2$,

$F_1 = (1001\ 1010\ 1101\ 1100\ 0001)_2$,

$F_2 = (1011\ 1011\ 1010\ 1110\ 1111)_2$,

$F_3 = (1111\ 0010\ 0011\ 1000\ 1001)_2$,

$F_4 = (0111\ 0010\ 0010\ 0011\ 1100)_2$,

$F_5 = (1001\ 1100\ 0100\ 1000\ 1010)_2$,

$F_6 = (0011\ 0101\ 0010\ 0110\ 0101)_2$,

$F_7 = (1101\ 0011\ 1011\ 1011\ 0100)_2$.

Повторный вызов бита b_i ($0 \le i \le 7$) каждого извлеченного байта выражается

$$b_i = \bigoplus_{j=0}^{19} f_i^{(j)} m_{8j+i}, \text{ где } F_i = \sum_{j=0}^{19} f_i^{(j)} 2^j$$

где m_k – биты, содержащиеся в основном регистре.

Инициализация

Инициализация шифра F-FCSR-H производится в следующем порядке:

$v \leq 80$)

1. Основной регистр M инициализируется ключом и IV:

$M := K + 2^{80} \cdot IV = (0^{80-v}||IV||K)$

2. Регистр переносов инициализируется в 0:

$C := 0 = (0^{82})$

3. FCSR тактируется 160 раз. (На этом шаге выход отбрасывается),

После фазы установки, псевдослучайный поток производится тактированием FCSR и извлечением по одному псевдослучайных байт, используя фильтр F, как описано выше.

Поточный шифр Grain-128

Grain – двоичный аддитивный поточный шифр, ориентированный на аппаратную реализацию. Версия, которая обозначается Grain v.1 [6], предназначена для приложений, которые имеют очень ограниченные аппаратные ресурсы. Grain v.1 поддерживает 80-битовый размер ключа.

Версия нифра Grain-128 поддерживает размер ключа – 128 бит и размер вектора IV – 96 битов. Шифр Grain-128 также является очень компактным и легко осуществимым в аппаратных средствах. Кроме того, возможно достаточно просто увеличивать скорость за счет большего количества аппаратных средств. Это является отличительной особенность семейства поточных шифров Grain, и во многих других шифрах явно не обосновано. Grain-128 использует линейный регистр сдвига с обратной связью, чтобы гарантировать хорошие статистические свойства и гарантировать нижнюю границу периода ключевой последовательности. Чтобы ввести нелинейность вместе с нелинейным фильтром используется нелинейный регистр сдвига с обратной связью (NFSR). Нелинейный фильтр берет в качестве входных данных значения от обоих сдвиговых регистров.

Генерация ключевого потока

Краткий обзор различных блоков, используемых в шифре, приведен на рисунке 2.28.

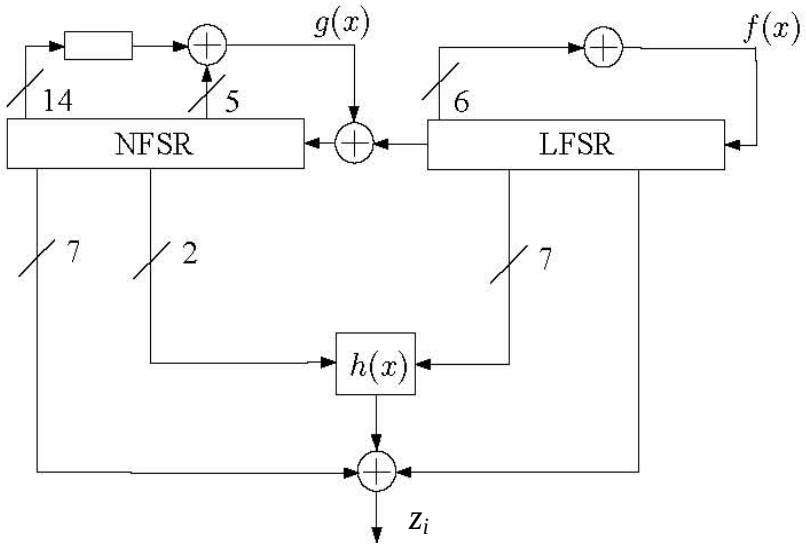


Рис. 2.28. Краткий обзор шифра.

Шифр состоит из трех основных строительных блоков, а именно: LFSR, NFSR и выходной функции. Содержимое LFSR обозначается s_i, s_{i+1}, …, s_{i+127}. Аналогичным образом содержание NFSR обозначается b_i, b_{i+1}, …, b_{i+127}. Полином обратной связи LFSR, обозначенный $f(x)$, является примитивным полиномом степени 128. Он определен как

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}.$$

Чтобы избежать любой возможной неопределенности, разработчики в [6] привели соответствующую функцию обновления LFSR:

$$s_{i+128} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96}.$$

Нелинейный полином обратной связи NFSR, $g(x)$, является суммой одной линейной и одной нелинейной функций. Он определен как

$$g(x) = 1 + x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} +$$
$$+ x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117}.$$

Аналогично, чтобы избежать любой возможной неопределенности, разработчики также привели в [6] соответствующую функцию обновления NFSR. В функции обновления, приведенной ниже, бит s_i, опущенный в полиноме обратной связи, включен и замаскирован со входом NFSR.

$$b_{i+128} = s_i + b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} + b_{i+3}b_{i+67} + b_{i+11}b_{i+13} +$$
$$+ b_{i+17}b_{i+18} + b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} + b_{i+68}b_{i+84}.$$

256 элементов памяти в этих двух сдвиговых регистрах представляют состояние шифра. Из этого состояния берется 9 переменных в качестве входа булевой функции $h(x)$. Два входа $h(x)$ берутся из NFSR, а семь – из LFSR. Эта функция имеет степень 3 и является очень простой. Она определяется как

$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$,

где переменные x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7 и x_8 соответствуют позициям сигнала b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79} и s_{i+95} соответственно. Выходная функция определяется как

$$z_i = \sum_{j \in A} b_{i+j} + h(x) + s_{i+93},$$

где $A = \{2,15,36,45,64,73,89\}$.

Инициализация

Перед генерацией ключевой последовательности шифр должен быть инициализирован с помощью ключа и вектора *IV*. Пусть биты ключа *k* будут обозначаться k_i, $0 \le i \le 127$, а биты вектора *IV* будут обозначаться IV_i, $0 \le i \le 95$. Тогда инициализация ключа и вектора *IV* выполняется следующим образом. 128 элементов NFSR заполняются битами ключа, $b_i = k_i$, $0 \le i \le 127$, затем первые 96 элементов LFSR заполняются битами *IV* битами, $s_i = IV_i$, $0 \le i \le 95$. Последние 32 бита LFSR заполняются единицами, $s_i = 1$, $96 \le i \le 127$. После загрузки битов ключа и *IV*, шифр тактируется 256 раз, без производства ключевой последовательности. Вместо выходной функции производится подача назад и сложение по модулю 2 (XOR'тся) со входом и LFSR и NFSR, см. рисунок 2.29

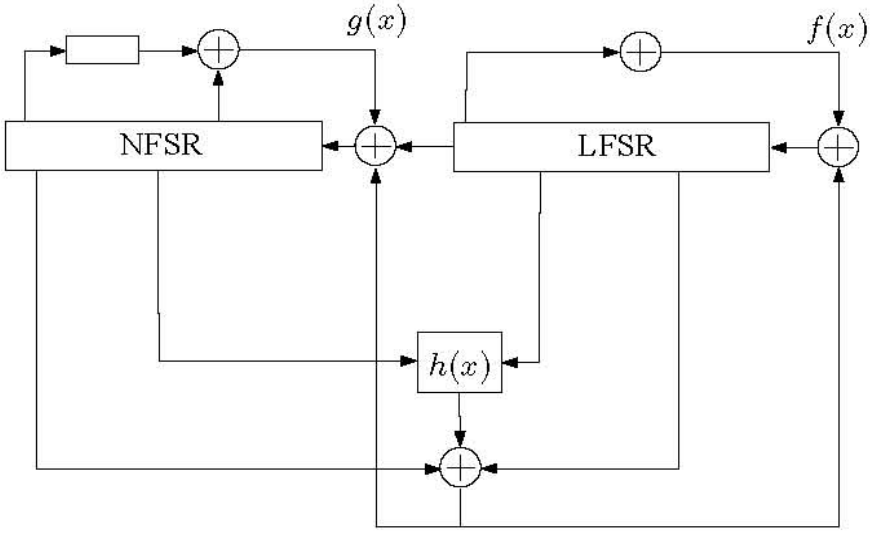


Рис. 2.29. Инициализация ключа

Поточный шифр MICKEY-128

MICKEY-128 – поточный шифр, ориентированный на аппаратную реализацию. для генерации ключевого потока MICKEY-128 использует 128 битовый ключ [6]. Этот шифр использует нерегулярно тактируемые регистры сдвига с новыми методиками, предназначенными обеспечить баланс между необходимостью в гарантированных

периоде и псевдослучайностью и необходимостью избежать определенных криптоаналитических атак.

Инициализация

MICKEY128 2.0 использует два входных параметра:

- 128-битовый секретный ключ K, биты которого обозначаются $k_0 \dots k_{127}$;
- вектор инициализации IV (initialisation variable), длиной от 0 до 128 битов, биты которой обозначаются $iv_0 \dots iv_{\text{IVLENGHT}-1}$, где IVLENGHT – размер вектора инициализации в битах.

Генератор строится из двух регистров R и S. Каждый регистр длиной 160 разрядов. Биты в регистрах обозначаются $r_0 \dots r_{159}$ и $s_0 \dots s_{159}$ соответственно.

Разработчики заявляют, что R – это "линейный регистр", а S – это "нелинейный регистр".

Инициализация генератора начинается с обнуления всех разрядов регистров R и S. Затем производится загрузка вектора инициализации IV путем тактирования всего генератора IVLENGHT раз с помощью операции clock_KG(R, S, Mixing = True, Input_bit = iv_i), описанной в п.1.3.8.2. Затем производится загрузка ключа: весь генератор тактируется 128 раз, так же с помощью операции clock_KG(R, S, Mixing = True, Input_bit = k_i). Заканчивается процесс инициализации 160 кратным применением операции clock_KG(R, S, Mixing = True, Input_bit = 0) тактирования всего генератора.

Генерация ключевого потока

После выполнения операции инициализации можно приступить к генерации битов ключевой последовательности $z_0 \dots z_{L-1}$:

For $0 \le i \le L-1$:

- $z_i = r_0 \oplus s_0$,
- clock_KG(R, S, Mixing = False, Input_bit = 0),

где clock_KG(R, S, Mixing, Input_bit)и – операция тактирования всего генератора, которая определяется в соответствии со следующим псевдокодом:

- Control_bit_R = $s_{54} \oplus r_{106}$
- Control_bit_S = $s_{106} \oplus r_{53}$
- If Mixing = True,
  - clock_R(R, Input_bit_R = Input_bit $\oplus$ s_{80}, Control_bit_R = Control_bit)
  - clock_S(S, Input_bit_S = Input_bit, Control_bit_S = Control_bit)
- If instead Mixing = False,
  - clock_R(R, Input_bit_R = Input_bit, Control_bit_R = Control_bit)
  - clock_S(S, Input_bit_S = Input_bit, Control_bit_S = Control_bit)

clock_R и clock_S – это операции тактирования регистров *R* и *S* соответственно.

Операция тактирования регистра *R* clock_R(*R*, Input_bit_R, Control_bit_R) определяется следующим псевдокодом:

- Пусть $r_0 \ldots r_{159}$ будет состоянием регистра *R* до тактирования, а $r'_0 \ldots r'_{159}$ будет состоянием регистра R после тактирования.
- Feedback_bit = $r_{159} \oplus$ Input_bit_R
- For $1 \le i \le 159$ $\{r'_i = r_{i-1}; r'_0 = 0;\}$
- For $0 \le i \le 159$ {if $i \in Rtaps$, $r'_i = r'_i \oplus$ Feedback_bit}
- If Control_bit_R = 1
  - For $0 \le i \le 159$ $\{r'_i = r'_i \oplus r_i\}$

Rtaps – это набор позиций отводов обратной связи для *R*:

Rtaps = {0, 4, 5, 8, 10, 11, 14, 16, 20, 25, 30, 32, 35, 36, 38, 42, 43, 46, 50, 51, 53, 54, 55, 56, 57, 60, 61, 62, 63, 65, 66, 69, 73, 74, 76, 79, 80, 81, 82, 85, 86, 90, 91, 92, 95, 97, 100, 101, 105, 106, 107, 108, 109, 111, 112, 113, 115, 116, 117, 127, 128, 129, 130, 131, 133, 135, 136, 137, 140, 142, 145, 148, 150, 152, 153, 154, 156, 157}.

Операция тактирования регистра *S* clock_S(*S*, Input_bit_S, Control_bit_S) определяется следующим псевдокодом:

- Пусть $s_0 \ldots s_{159}$ будет состоянием регистра *S* до тактирования, а $s'_0 \ldots s'_{159}$ будет состоянием регистра после тактирования. Также, для упрощения описания шифра, мы будем использовать $\hat{s}_0 \ldots \hat{s}_{159}$ как промежуточные переменные.
- Feedback_bit = $s_{159} \oplus$ Input_bit_R
- For $1 \le i \le 158$ $\{\hat{s}_i = s_{i-1} \oplus ((s_i \oplus comp0_i) \cdot (s_{i+1} \oplus comp1_i))$; $\hat{s}_i = 0$; $\hat{s}_{159} = s_{158};\}$
- If Control_bit_S = 0:
  - For $0 \le i \le 159$ $\{s'_i = \hat{s}_i \oplus (fb0_i \cdot$ Feedback_bit$)\}$
- If instead Control_bit_S = 1
  - For $0 \le i \le 159$ $\{s'_i = \hat{s}_i \oplus (fb1_i \cdot$ Feedback_bit$)\}$

Значения *comp*0, *comp*1, *fb*0 и *fb*1, определенные разработчиками, приведены в [6].

Поточный шифр Trivium

Trivium – синхронный поточный шифр, использующий 80-битовый ключ и 80-битовый вектор инициализации *IV* .

Инициализация

Алгоритм инициализируется загрузкой 80-битового ключа и 80-битового *IV* в 288-битовое начальное состояние и устанавливает все оставшиеся биты в 0, за исключением s_{286}, s_{287}, и s_{288}. Затем состояние изменяется за 4 полных цикла (без генерации бит ключевого потока). В итоге это можно представить в виде псевдокода, приведенного ниже:

```
(s1, s2, …, s93) ← (K1, …, K80, 0, …, 0)
(s94, s95, …, s177) ← (IV1, …, IV80, 0, …, 0)
(s178, s279, …, s288) ← (0, …, 0, 1, 1, 1)
for i = 0 to 4 · 288 do
 {
  t1 ← s66 ⊕ s91 ⊗ s92 ⊕ s93 ⊕ s171
  t2 ← s162 ⊕ s175 ⊗ s176 ⊕ s177 ⊕ s264
  t3 ← s243 ⊕ s286 ⊗ s287 ⊕ s288 ⊕ s69
  (s1, s2, …, s93) ← (t3, s1, …, s92)
  (s94, s95, …, s177) ← (t1, s94, …, s176)
  (s178, s279, …, s288) ← (t2, s178, …, s287)
 }
```

Генерация ключевого потока

Предложенная конструкция содержит 288-битовое внутреннее состояние, обозначенное (s_1, …, s_{288}). Генерация ключевого потока состоит из итерационного процесса, который извлекает значения 15 определенных битов состояния и использует их для обновления 3 битов состояния и вычисления 1 бита ключевого потока z_i. Затем биты состояния циклически сдвигаются и процесс повторяет, пока не будут сгенерированы требуемые *N* бит ключевого потока. Полное описание дается следующим простым псевдокодом:

```
for i = 1 to N do
 {
  t1 ← s66 ⊕ s93
  t2 ← s162 ⊕ s177
  t3 ← s243 ⊕ s288
  zi ← t1 ⊕ t2 ⊕ t3
  t1 ← t1 ⊕ s91 ⊗ s92 ⊕ s171
  t2 ← t2 ⊕ s175 ⊗ s176 ⊕ s264
  t3 ← t3 ⊕ s286 ⊗ s287 ⊕ s69
  (s1, s2, …, s93) ← (t3, s1, …, s92)
  (s94, s95, …, s177) ← (t1, s94, …, s176)
  (s178, s279, …, s288) ← (t2, s178, …, s287)
 }
```

Графическое представление процесса генерации ключевого потока приведено на рис. 2.30.

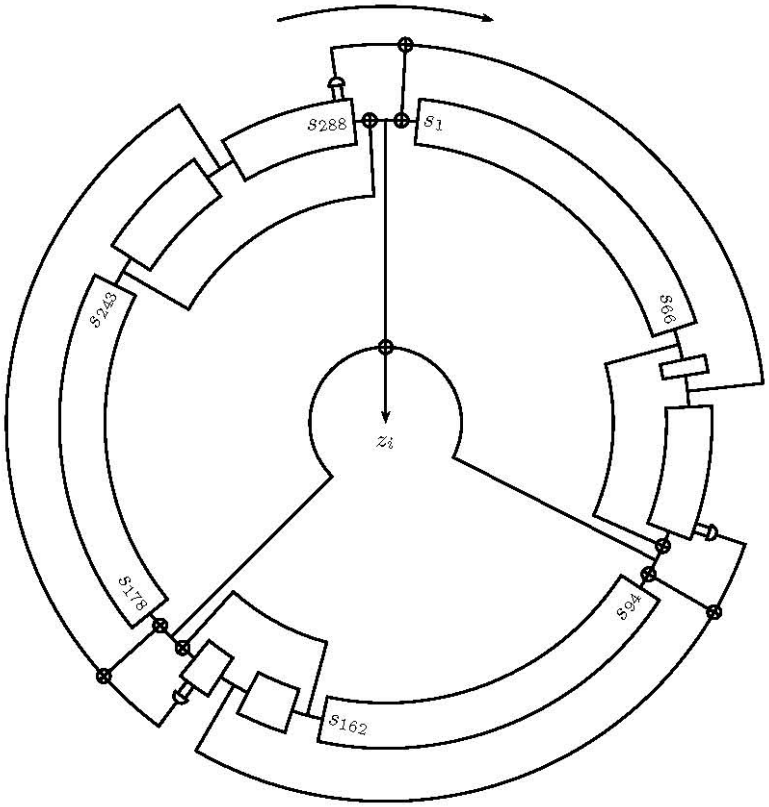


Рис. 2.30. Поточный шифр Trivium

Блочные шифры в поточных режимах

Российский блочный шифр ГОСТ 28147-89 в поточном режиме

В России в качестве стандарта шифрования принят алгоритм криптографического преобразования ГОСТ 28147-89. Этот алгоритм предназначен для аппаратной и программной реализации, удовлетворяет необходимым криптографическим требованиям и не накладывает ограничений на степень секретности защищаемой информации.

Для шифрования используются 256 битовый ключ, который разбивается на 8 32-битовых подключа: K_0, K_1, …, K_7, и таблица блока подстановки H. Заполнение таблиц блока подстановки H является долговременным ключевым элементом.

Алгоритм криптографического преобразования ГОСТ 28147-89 опирается на цикл шифрования (рисунок 2.31). Цикл шифрования построен по принципу сети Фейстеля.

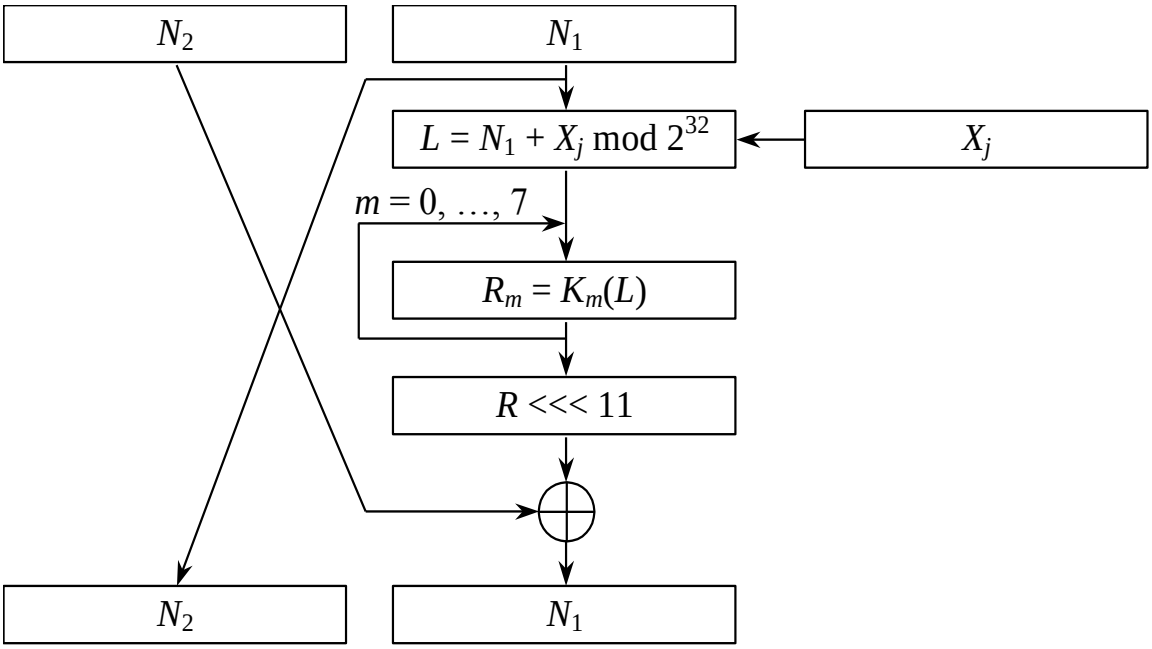


Рис. 2.31. Цикл шифрования

Один цикл шифрования преобразует 64-битовый блок данных, используя один 32-битовый элемент ключа K_j и блок подстановки H.

Преобразуемый 64-битовый блок данных (N) разбивается на две части: старшую (N_2) и младшую (N_1). Значение N_1 суммируется по модулю 2^{32} с одним из 32-битовых элементов ключа K_j, $j = 0, \ldots, 7$. Результат суммирования преобразуется в блоке подстановки H. Блок подстановки H состоит из 8 узлов замены $H_1, \ldots, H_8$ размером 64 бита каждый. Поступающий на блок подстановки 32-разрядный блок разбивается на восемь последовательно идущих 4-разрядных блоков, каждый из которых преобразуется в 4-разрядный блок соответствующим узлом замены, представляющим собой таблицу из шестнадцати строк, содержащих по четыре бита заполнения в строке. Входной блок определяет адрес строки в таблице, заполнение данной строки является выходным блоком. Затем 4-разрядные выходные блоки последовательно объединяются в 32-разрядный блок. Полученное значение R циклически сдвигается на 11 шагов в сторону старших разрядов. Результат сдвига суммируется поразрядно по модулю 2 со значением N_2. Полученный результат N_2 записывается в N_1, при этом старое значение N_1 переписывается в N_2.

Многократное повторение цикла шифрования, при использовании различных элементов ключа, позволяет построить циклы зашифрования (32-З) и расшифрования (32-Р). В цикле зашифрования 32-З отдельный цикл шифрования повторяется 32 раза,

используя 32-битовые элементы ключа в следующем порядке: K_0, …, K_7, K_0, …, K_7, K_0, …, K_7, K_7, …, K_0. Цикл расшифрования 32-Р отличается от цикла порядком использования 32-битовых элементов ключа. В цикле 32-Р этот порядок следующий: K_0, …, K_7, K_7, …, K_0, K_7, …, K_0, K_7, …, K_0.

Ниже приведено описание двух поточных режимов работы криптографического алгоритма ГОСТ 28147-89, а именно: гаммирование и гаммирование с обратной связью.

Гаммирование

Криптосхема, реализующая алгоритм зашифрования в режиме гаммирования, имеет вид, указанный на рисунке 7.32а.

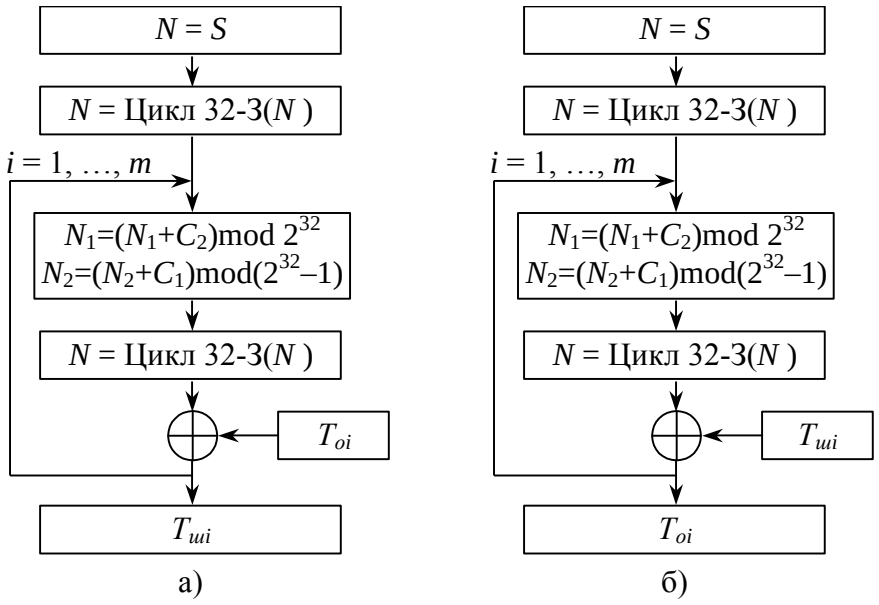


Рис. 2.32. Схема алгоритма зашифрования (а) и расшифрования (б) в режиме гаммирования

Открытые данные, разбитые на 64-разрядные блоки T_{oi}, зашифровываются в режиме гаммирования путем поразрядного суммирования по модулю 2 с гаммой шифра $\Gamma_ш$, которая вырабатывается блоками по 64 бита $\Gamma_{шi}$, i = 1, …, m. m определяется объемом шифруемых данных. Число двоичных разрядов в блоке $T_{от}$ может быть меньше 64, при этом неиспользованная для зашифрования часть гаммы шифра из блока $\Gamma_{шт}$ отбрасывается.

Для инициализации процесса генерации гаммы используется 64-разрядная двоичная последовательность (синхропосылка) S. К синхропосылке применяется цикл зашифрования 32-З. Результат шифрования N разбивается на две части: старшую (N_2) и младшую (N_1). Значение N_2 суммируется по модулю (2^{32} – 1) с 32-разрядной константой $C_1 = 01010104_{16}$. Значение N_1 суммируется по модулю 2^{32} с 32-разрядной константой $C_2 = 01010101_{16}$. К полученному значению N применяется цикл зашифрования 32-З. Полученное в результате зашифрования значение N образует первый 64-разрядный блок гаммы шифра $\Gamma_{ш1}$, который

суммируется поразрядно по модулю 2 с первым 64-разрядным блоком открытых данных T_{o1}. В результате суммирования получается 64-разрядный блок зашифрованных данных $T_{ш1}$. Для получения следующего 64-разрядного блока гаммы шифра $Г_{ш2}$ старшая часть N_2 значения N суммируется по модулю $(2^{32} - 1)$ с константой C_1, а младшая часть N_1 суммируется по модулю 2^{32} с константой C_2. К полученному значению применяется цикл зашифрования 32-З. Полученное в результате зашифрования значение N образует второй 64-разрядный блок гаммы шифра $Г_{ш2}$, который суммируется поразрядно по модулю 2 со вторым блоком открытых данных T_{o2}. Аналогично вырабатываются блоки гаммы шифра $Г_{ш3}$, $Г_{ш4}$, …, $Г_{шm}$ и зашифровываются блоки открытых данных T_{o3}, T_{o4}, …, T_{om}. Если длина последнего m-го блока открытых данных T_{om} меньше 64 бит, то из последнего m-го блока гаммы шифра $Г_{шm}$ для зашифрования используется только соответствующее число разрядов гаммы шифра, остальные разряды отбрасываются.

Аналогичным образом производится расшифрование в режиме гаммирования (рисунок 2.32б).

Гаммирование с обратной связью

Криптосхема, реализующая алгоритм зашифрования в режиме гаммирования с обратной связью, имеет вид, указанный на рисунке 3а.

Открытые данные, разбитые на 64-разрядные блоки T_{oi}, зашифровываются в режиме гаммирования с обратной связью путем поразрядного суммирования по модулю 2 с гаммой шифра $Г_{ш}$, которая вырабатывается блоками по 64 бита $Г_{шi}$, $i = 1, …, m$. m определяется объемом шифруемых данных. Число двоичных разрядов в блоке T_{om} может быть меньше 64.

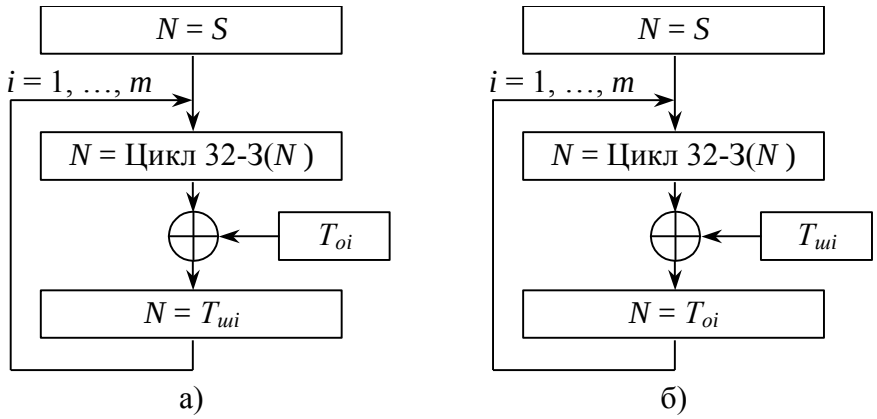


Рис. 2.33. Схема алгоритма зашифрования (а) и расшифрования (б) в режиме гаммирования с обратной связью

Исходное значение – синхропосылка S зашифровывается с помощью цикла зашифрования 32-З. Полученное в результате зашифрования значение N образует первый 64-разрядный блок гаммы шифра $Г_{ш1}$, который суммируется поразрядно по модулю 2 с первым

64-разрядным блоком открытых данных T_{o1}. В результате получается 64-разрядный блок зашифрованных данных $T_{ш1}$.

Блок зашифрованных данных $T_{ш1}$ одновременно является также исходным состоянием N для выработки второго блока гаммы шифра $Г_{ш2}$ и по обратной связи передается на вход. Новое значение N зашифровывается с помощью цикла зашифрования 32-З. Полученное в результате зашифрования значение N образует второй 64-разрядный блок гаммы шифра $Г_{ш2}$, который суммируется поразрядно по модулю 2 со вторым блоком открытых данных T_{o2}.

Выработка последующих блоков данных шифра $Г_{шi}$ и зашифрование соответствующих блоков открытых данных T_{oi} (i = 3, …, m) производится аналогично. Если длина последнего m-го блока открытых данных T_{om} меньше 64 бит, то из последнего m-го блока гаммы шифра $Г_{шm}$ используется только соответствующее число разрядов гаммы шифра, остальные разряды отбрасываются.

Аналогичным образом производится расшифрование в режиме гаммирования с обратной связью (рисунок 2.33б).

Блочный шифр AES в поточном режиме

В основе стандарта шифрования AES лежит алгоритм Rijndael. Rijndael – это итерационный блочный шифр, имеющий архитектуру “Квадрат”. Шифр имеет переменную длину блоков и различные длины ключей. Длина ключа и длина блока могут быть равны независимо друг от друга 128, 192 или 256 битам. В стандарте *AES*-128 определена длина блока данных, равная 128 битам.

Операции алгоритма AES выполнены на основе двумерного массива байтов, называемого состоянием (*state*). Состояние состоит из четырех строк, каждая из которых содержит N_b байт, где N_b – длина блока в битах, деленная на 32. В массиве состояния, обозначенном символом s, каждый байт имеет два индекса: номер строки r, $0 \le r < 4$, и номер столбца c, $0 \le c < N_b$. Это позволяет обращаться к отдельному байту состояния как s_{rc}. Для AES-128 $N_b = 4$, т.е. $0 \le c < 4$.

Перед применением операции зашифрования или расшифрования, входной блок данных, представленный в виде массива байтов in_0, …, in_{15}, копируется в массив состояния (рисунок 2.34):

$s_{rc} = in_{r+4c}$, $0 \le r < 4$, $0 \le c < N_b$,

а в конце операций зашифрования или расшифрования, данные из состояния копируются в выходной массив (рисунок 2.34):

$out_{r+4c} = s_{rc}$, $0 \le r < 4$, $0 \le c < N_b$.

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

Входной блок

s_{00}	s_{01}	s_{02}	s_{03}
s_{10}	s_{11}	s_{12}	s_{13}
s_{20}	s_{21}	s_{22}	s_{23}
s_{30}	s_{31}	s_{32}	s_{33}

Промежуточное состояние

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

Выходной блок

Рис. 2.34. Входной и выходной блоки данных, промежуточное состояние

Ключ шифрования также представляется в виде прямоугольного массива с четырьмя строками (рисунок 2). Число столбцов N_k этого массива равно длине ключа в битах, деленной на 32. В стандарте определены ключи всех трех размеров – 128 бит, 192 бита и 256 бит, то есть соответственно 4, 6 и 8 32-разрядных слова (или столбца – в табличной форме представления). В некоторых случаях ключ шифрования рассматривается как линейный массив 4-байтовых слов. Слова состоят из 4 байтов, которые находятся в одном столбце (при представлении в виде прямоугольного массива).

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

Рис. 2.35. Формат представления ключа шифрования

Для алгоритма AES длина входного и выходного блоков, а также блока состояния – 128 битов, т.е. $N_b = 4$. Длина ключа шифрования равна 128, 192 или 256 битов, т.е. $N_k = 4$, 6 или 8. Количество раундов, которые будут выполнены в течение выполнения алгоритма, зависит от размера ключа. Число раундов обозначается N_r, где $N_r = 10$ при $N_k = 4$, $N_r = 12$ при $N_k = 6$ и $N_r = 14$ при $N_k = 8$.

И для шифрования, и для расшифрования, алгоритм AES использует раундовую функцию, которая составлена из четырех различных преобразований над байтами: 1) замена байта, используя таблицу замены (S-блок), 2) сдвиг строк массива состояния на различные смещениями, 3) смешивание данных в пределах каждого столбца массива состояния, и 4) добавление раундового ключа к состоянию. Эти преобразования (и их инверсии) описаны ниже.

Функция зашифрования

Перед зашифрованием входные данные копируются в массив состояния. После начального добавления раундового ключа, массив состояния преобразуется с помощью раундовой функции 10, 12 или 14 раз (в зависимости от длины ключа). Заключительный раунд немного отличается от первых $N_r - 1$ раундов. Затем полученное состояние копируется в выходной массив.

Раундовая функция состоит из 4 различных преобразований: *SubBytes*, *ShiftRows*, *MixColumns* и *AddRoundKey*. Все N_r раундов идентичны за исключением последнего, который не включает преобразование *MixColumns* (рисунок 2.36).

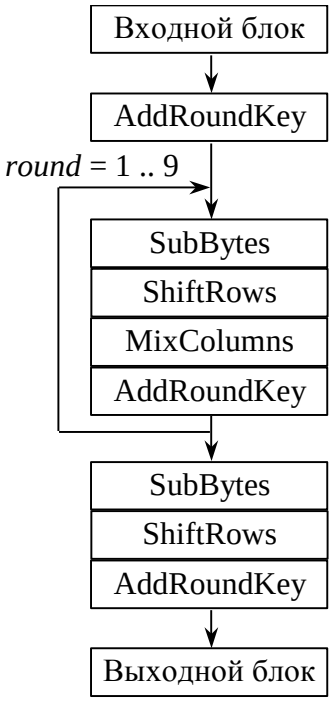


Рис. 2.36. Схема функции зашифрования ($N_b = 4$, $N_k = 4$)

Преобразование *SubBytes* – нелинейная замена байта, которая работает независимо на каждом байте состояния, используя таблицу замены (S-блок). Эта таблица замен создается с использованием двух преобразований:

1. получение обратного элемента относительно умножения в поле GF(2^8), нулевой элемент переходит сам в себя;
2. применение преобразования над $GF(2)$, определенного следующим образом:

$b'_i = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i$,

где $c_0 = c_1 = c_5 = c_6 = 1$, $c_2 = c_3 = c_4 = c_7 = 0$, b_i и b'_i – соответственно исходное и преобразованное значение i-го бита, $i = 0, \ldots, 7$.

В преобразовании *ShiftRows* байты в последних трех строках состояния циклически сдвигаются влево на различное число байт. Первая строка ($r = 0$) не сдвигается. Вторая

строка ($r = 1$) сдвигается на 1 байт, третья строка ($r = 2$) – на 2 байта, четвертая строка ($r = 3$) – на 3 байта.

Преобразование *MixColumns* работает с состоянием столбец за столбцом, обрабатывая каждый столбец как 4-элементный полином. Столбцы рассматриваются как полиномы над $GF(2^8)$ и умножаются по модулю $x^4 + 1$ на фиксированный полином $a(x)$, приведенный ниже

$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

Это может быть представлено в матричном виде:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}, \quad 0 \le c \le 3.$$

В результате этого умножения байты столбца заменяются следующими:

$s'_{0c} = (\{02\} \bullet s_{0c}) \oplus (\{03\} \bullet s_{1c}) \oplus s_{2c} \oplus s_{3c}$,

$s'_{1c} = s_{0c} \oplus (\{02\} \bullet s_{1c}) \oplus (\{03\} \bullet s_{2c}) \oplus s_{3c}$,

$s'_{2c} = s_{0c} \oplus s_{1c} \oplus (\{02\} \bullet s_{2c}) \oplus (\{03\} \bullet s_{3c})$,

$s'_{3c} = (\{03\} \bullet s_{0c}) \oplus s_{1c} \oplus s_{2c} \oplus (\{02\} \bullet s_{3c})$.

В преобразовании *AddRoundKey* раундовый ключ добавляется к состоянию простой поразрядной операцией XOR. Каждый раундовый ключ состоит из N_b 4-байтовых слов развернутого ключа. Сложение раундового ключа и состояния производится следующим образом:

$[s'_{0c}, s'_{1c}, s'_{2c}, s'_{3c}] = [s_{0c}, s_{1c}, s_{2c}, s_{3c}] \oplus [w_{round \cdot Nb+c}]$, $0 \le c < N_b$,

где $[w_i]$ – слова развернутого ключа, описанного ниже, *round* – номер раунда, $0 \le round \le N_r$. При зашифровании, перед первым применением раундовой функции, происходит начальное добавление раундового ключа (*round* = 0). Применение преобразования *AddRoundKey* в N_r раундах зашифрования происходит при $1 \le round \le N_r$.

Расширение ключа

В алгоритме AES для получения раундовых ключей используется алгоритм расширения ключа. Расширенный ключ представляет собой линейный массив $w[i]$ из $N_b(N_r + 1)$ 4-байтовых слов, $i = 0, \ldots, N_b(N_r + 1)$. Алгоритм выработки ключей зависит от величины N_k. Первые N_k слов расширенного ключа заполняются ключом шифрования. Каждое последующее слово $w[i]$ получается посредством XOR предыдущего слова w[i–1] и слова на N_k позиций ранее $w[i - N_k]$

$w[i] = w[i - 1] \oplus w[i - N_k]$.

Для слов, позиция которых кратна N_k, перед применением операции XOR слово $w[i–l]$ подвергается воздействию двух дополнительных функций: *RotWord*, осуществляющей побайтовый сдвиг 32-разрядного слова по формуле $\{a_0\ a_1\ a_2\ a_3\} \rightarrow \{a_1\ a_2\ a_3\ a_0\}$, и *SubWord*, осуществляющей побайтовую замену с использованием S-блока функции *SubBytes*. Затем к полученному значению прибавляется раундовая константа $Rcon[j] = 2^{j-1}$. В итоге значение $w[i]$ равно

$w[i] = SubWord(RotWord(w[i-1])) \oplus Rcon[i/N_k] \oplus w[i-N_k]$.

i-ый раундовый ключ получается из слов массива раундового ключа от $w[N_b i]$ и до $w[N_b(i+1)]$.

Функция расшифрования

Если преобразования, используемые в функции зашифрования, инвертировать и затем применить в обратном порядке, то можно произвести обратное расшифрование. При расшифровании используются следующие преобразования: *InvShiftRows*, *InvSubBytes*, *InvMixColumns* и *AddRoundKey* (рисунок 4а).

InvShiftRows – инверсия преобразования *ShiftRows*. Байты в последних трех строках состояния циклически сдвигаются вправо на различное число байт. Первая строка ($r = 0$) не сдвигается. Вторая строка ($r = 1$) сдвигается на 1 байт, третья строка ($r = 2$) – на 2 байта, четвертая строка ($r = 3$) – на 3 байта.

InvSubBytes – инверсия преобразования замены байта, в котором к каждому байту состояния применяется обратный S-блок.

InvMixColumns – инверсия преобразования *MixColumns*. *InvMixColumns* оперирует состоянием столбец за столбцом, обрабатывая каждый столбец как 4-элементный полином, как описано в ранее. Столбцы рассматривают как полиномы над $GF(2^8)$ и умножаются по модулю $x^4 + 1$ на фиксированный полином $a^{-1}(x)$

$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

Это может быть представлено в матричном виде:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}, \quad 0 \le c \le 3.$$

В результате на выходе получаются следующие байты:

$s'_{0c} = (\{0e\} \bullet s_{0c}) \oplus (\{0b\} \bullet s_{1c}) \oplus (\{0d\} \bullet s_{2c}) \oplus (\{09\} \bullet s_{3c})$,

$s'_{1c} = (\{09\} \bullet s_{0c}) \oplus (\{0e\} \bullet s_{1c}) \oplus (\{0b\} \bullet s_{2c}) \oplus (\{0d\} \bullet s_{3c})$,

$s'_{2c} = (\{0d\} \bullet s_{0c}) \oplus (\{09\} \bullet s_{1c}) \oplus (\{0e\} \bullet s_{2c}) \oplus (\{0b\} \bullet s_{3c})$,

$s'_{3c} = (\{0b\} \bullet s_{0c}) \oplus (\{0d\} \bullet s_{1c}) \oplus (\{09\} \bullet s_{2c}) \oplus (\{0e\} \bullet s_{3c})$.

Преобразование AddRoundKey является собственной инверсией, т.к. использует только операцию XOR.

В функции обратного расшифрования (рисунок 4а) последовательность преобразований отличается от последовательности преобразований при зашифровании, в то время как форма расширенных ключей для зашифрования и расшифрования остается той же самом. Однако, несколько свойств алгоритма AES позволяют функции расшифрования иметь ту же самую последовательность преобразований как и при зашифровании (с преобразованиями, замененными их инверсиями) (рисунок 2.37б). Это достигается при внесении изменений в расширенный ключ.

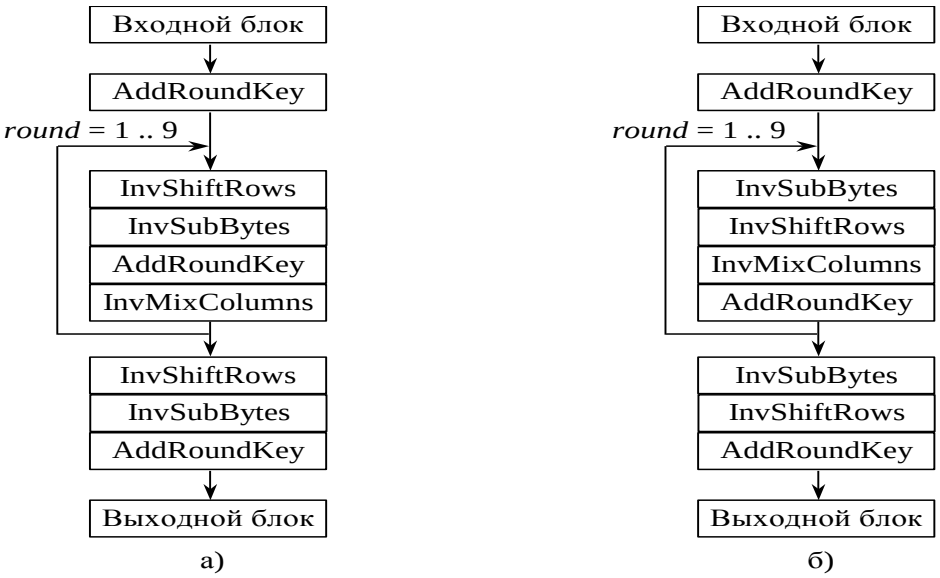


Рис. 2.37. Схемы функций обратного (а) и прямого (б) расшифрования

Существует два свойства, которые делают возможным использование функции прямого расшифрования:

1. Преобразования *SubBytes* и *ShiftRows* можно переставлять местами, т.е. можно сначала применить преобразование *SubBytes*, а затем *ShiftRows*, и наоборот – сначала *ShiftRows*, а затем *SubBytes*. То же самое верно и для их инверсий *InvSubBytes* и *InvShiftRows*.

2. Операции смешивания столбцов *MixColumns* и *InvMixColumns* являются линейными относительно входных данных, т.е.

InvMixColumns(*State* XOR *Round Key*) =

= *InvMixColumns*(*State*) XOR *InvMixColumns*(*Round Key*).

Эти свойства позволяют изменить порядок преобразований *InvSubBytes* и *InvShiftRows*. Порядок преобразований *AddRoundKey* и *InvMixColumns* также может быть полностью

изменен, при условии, что предварительно, используя преобразование *InvMixColumns*, будут изменены столбцы (слова) развернутого ключа расшифрования.

Режим обратной связи по шифртексту (CFB)

В режиме обратной связи по шифртексту (CFB) выходные блоки зашифрованного текста за счет обратной связи становятся входными блоками для последующего шифрования. Чтобы получить зашифрованный текст, открытый текст складывается, с помощью операции XOR, со сгенерированными выходными блоками. В качестве начального входного блока в режиме CFB используется значение вектора инициализации IV. Значение IV может быть открытым, но при этом должно быть непредсказуемым.

В режиме CFB используется целочисленный параметр s, $1 \leq s \leq b$, где b – размер блока (в байтах) открытого и зашифрованного текстов. Значение s иногда включается в название режима, например, 1-битовый режим CFB, 8-битовый режим CFB, 64-битовый режим CFB, или 128-битовый режим CFB.

При зашифровании в режиме CFB первым входным блоком является значение IV (рисунок 2.38). Чтобы произвести первый входной блок к значению IV применяется операция шифрования. Первая доля зашифрованного текста производится операцией XOR между первой долей открытого текста и s старшими битами первого выходного блока. Оставшиеся (b–s) бит первого выходного блока отбрасываются. Затем (b–s) младших бита значения IV соединяются операцией конкатенации с s битами первой доли зашифрованного текста, для формирования второго входного блока. Другими словами для формирования второго входного блока биты первого входного блока циклически сдвигаются на s позиции влево, и затем доля зашифрованного текста заменяет s младших бит.

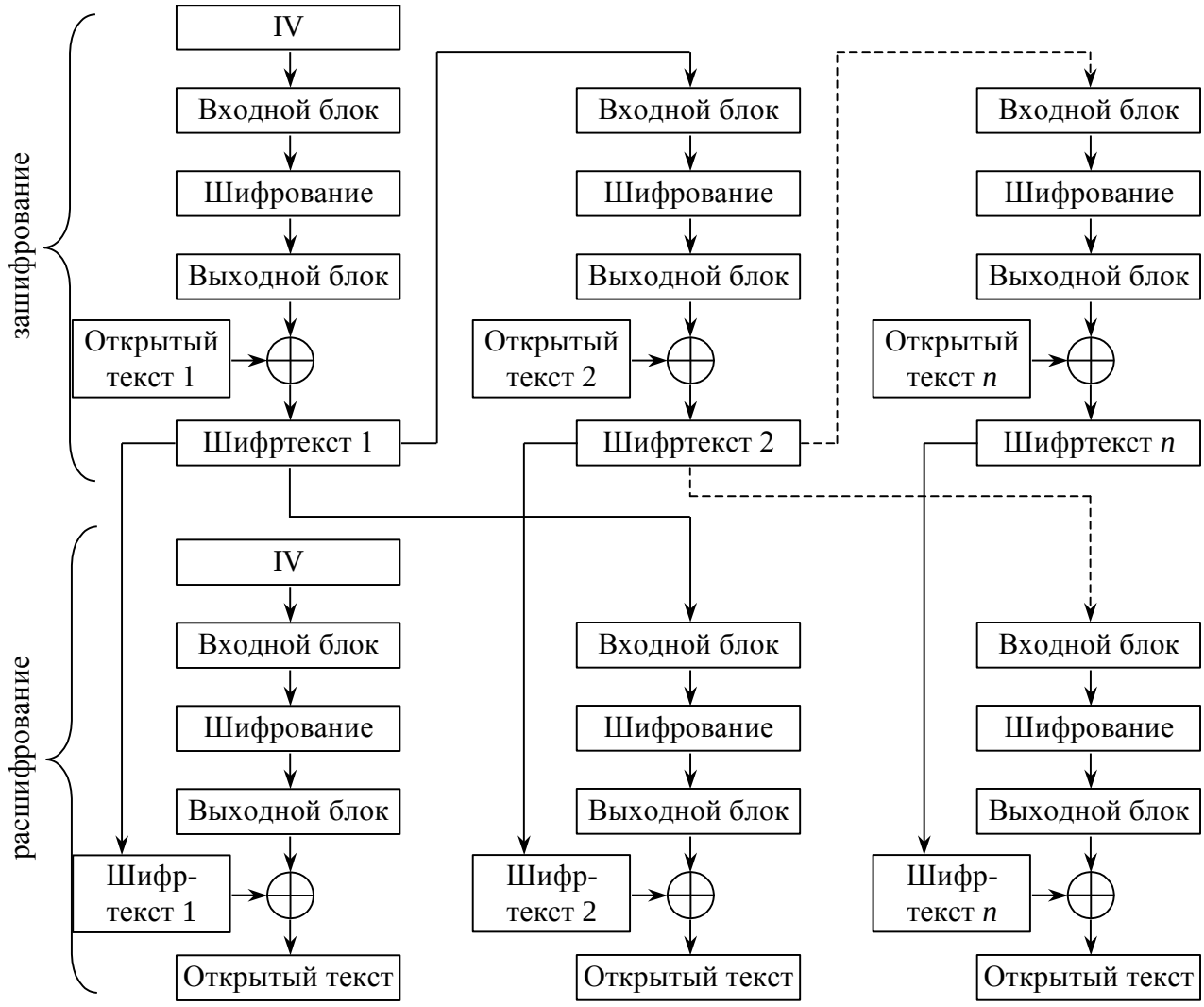


Рис.2.38. Режим CFB

Процесс повторяется до тех пор, пока от каждой доли открытого текста не будет получена доля зашифрованного текста. Чтобы сформировать долю зашифрованного текста s старших бит каждого выходного блока складываются, с помощью операции XOR, с соответствующей долей открытого текста. Каждая доля зашифрованного текста (кроме последней) подается назад, как описано выше, чтобы сформировать новый входной блок. Обратная связь может быть описана в терминах отдельных бит следующим образом: если $i_1 i_2 \ldots i_b$ – j-й входной блок, а $c_1 c_2 \ldots c_s$ – j-ая доля зашифрованного текста, то (j+1)-й входной блок: $i_1 i_2 \ldots i_b c_1 c_2 \ldots c_s$.

При расшифровании в режиме CFB значение IV является первым входным блоком. Каждый последующий входной блок формируется как при CFB зашифровании, соединяя операцией конкатенации (b–s) младших бит предыдущего входного блока с s старшими битами предыдущего зашифрованного текста. Чтобы произвести выходные блоки к каждому входному блоку применяется функция шифрования. Чтобы получить доли открытого текста

s старших бит выходного блока складывается, с помощью операции XOR, с соответствующими долями зашифрованного текста.

При зашифровании в режиме CFB входной блок для каждой функции шифрования (кроме первого) зависит от результата предыдущей функции шифрования, поэтому многократные операции шифрования не могут быть выполнены параллельно. При расшифровании в режиме CFB, необходимые операции шифрования могут быть выполнены параллельно, если из значения IV и зашифрованного текста сначала создать (последовательно) входные блоки.

Режим обратной связи по выходу (OFB)

В режиме обратной связи по выходу (OFB) последовательность выходных блоков генерируется путем последовательного применения операции шифрования к значению IV. Чтобы получить зашифрованный текст открытый текст складывается с помощью операции XOR со сгенерированными выходными блоками и наоборот. Для данного режима шифрования значения IV должны быть случайными, т.е. IV должны быть уникальными для каждого применения режима OFB на данном ключе.

Для зашифрования в режиме OFB, чтобы произвести первый выходной блок, значение IV преобразовывается функцией шифрования. Чтобы произвести первый блок зашифрованного текста первый выходной блок складывается, с помощью операции XOR, с первым блоком открытого текста. Затем, чтобы произвести второй выходной блок, функция шифрования применяется к первому выходному блоку. Второй выходной блок складывается с помощью операции XOR со вторым блоком открытого текста, чтобы произвести второй блок зашифрованного текста. Затем ко второму выходному блоку применяется функция шифрования, чтобы произвести третий выходной блок. Таким образом, последовательные выходные блоки производятся путем применения функции шифрования к предыдущим выходным блокам, а затем выходные блоки складывается с помощью операции XOR с соответствующими блоками открытого текста, чтобы произвести блоки зашифрованного текста. В последнем блоке (который может быть неполным блоком из *u* бит) для операции XOR используются старшие *u* бит последнего выходного блока. Оставшиеся (*b*–*u*) бит последнего выходного блока отбрасываются.

При расшифровании в режиме OFB чтобы произвести первый выходной блок значение IV преобразовывается функцией шифрования (рисунок 2.39). Чтобы получить первый блок открытого текста первый выходной блок складывается, с помощью операции XOR, с первым блоком зашифрованного текста. Затем первый выходной блок преобразовывается функцией шифрования, чтобы произвести второй выходной блок. Второй выходной блок складывается с помощью операции XOR со вторым блоком зашифрованного текста, чтобы произвести

второй блок открытого текста. Второй выходной блок также преобразовывается функцией шифрования, чтобы произвести третий выходной блок. Таким образом, последовательные выходные блоки производятся путем применения функции шифрования к предыдущим выходным блокам, затем выходные блоки складываются, с помощью операции XOR, с соответствующими блоками зашифрованного текста, чтобы получить блоки открытого текста. В последнем блоке (который может быть неполным блоком из *u* бит) для операции XOR используются старшие *u* бит последнего выходного блока. Оставшиеся (*b*–*u*) бит последнего выходного блока отбрасываются.

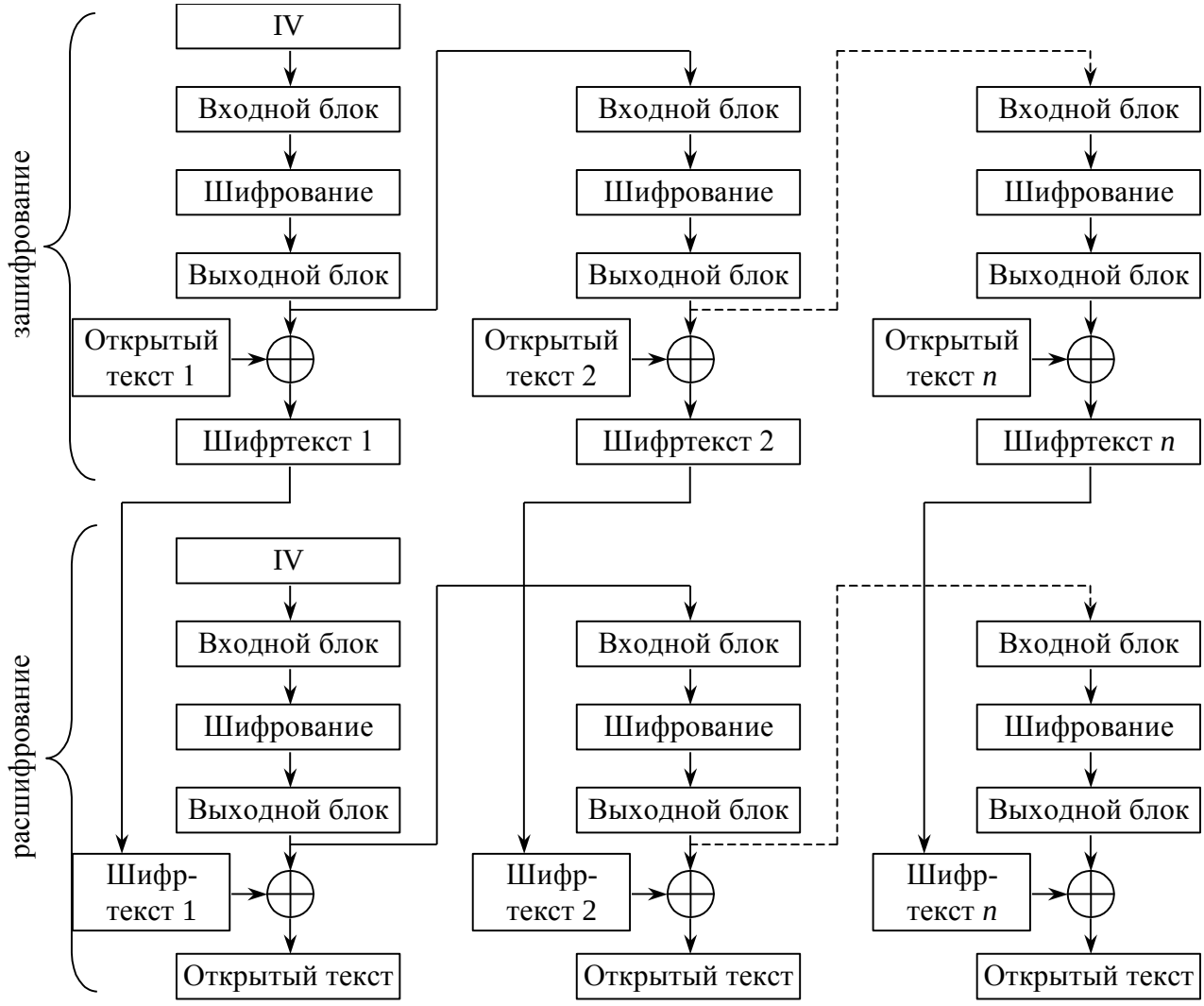


Рис. 2.39. Режим OFB

В режиме OFB и при зашифровании и при расшифровании каждая функция шифрования (кроме первой) зависит от результатов предыдущей функции шифрования, поэтому многократные функции шифрования не могут быть выполнены параллельно. Однако, если значение IV известно, выходные блоки могут быть сгенерированы прежде, чем станет доступным зашифрованный или открытый текст.

Для режима OFB требует уникальное значение IV для каждого сообщения, которое когда-либо будет зашифровано на данном ключе. Если, вопреки этому требованию, используется то же самое значение IV для зашифровывания более, чем одного сообщения, то конфиденциальность этих сообщений может быть поставлена под угрозу. В частности если известен блок открытого текста любого из этих сообщений, скажем, j-й блок открытого текста, то выход j-й функции шифрования может быть определен легко из j-го блока зашифрованного текста сообщения. Эта информация позволяет легко восстановить j-й блок открытого текста любого другого сообщения, зашифрованного с использованием того же самого значения IV, из соответствующего j-го блока зашифрованного текста этого сообщения.

Конфиденциальность может быть аналогичным образом поставлена под угрозу, если любой из входных блоков функции шифрования определяется как IV для зашифрования другого сообщения на данном ключе.

Режим счетчика (Counter mode)

В режиме счетчика (CTR) для производства последовательности выходных блоков операция шифрования применяется к набору выходных блоков, называемых счетчиками. Чтобы получить зашифрованный текст открытый текст складывается с помощью операции XOR со сгенерированными выходными блоками и наоборот. Все блоки в последовательности счетчика должны отличаться друг от друга. Это условие не ограничивается одним сообщением, т.е. для всех сообщений, которые зашифровываются на данном ключе, все счетчики должны быть различными.

При зашифровании в режиме CRT функция шифрования применяется к каждому блоку счетчика (рисунок 2.40). Полученные выходные блоки складываются с помощью операции XOR с соответствующими блоками открытого текста, для того чтобы произвести блоки зашифрованного текста. В последнем блоке (который может быть неполным блоком из u битов) для операции XOR используются старшие u бит последнего выходного блока. Оставшиеся (b–u) бит последнего выходного блока отбрасываются.

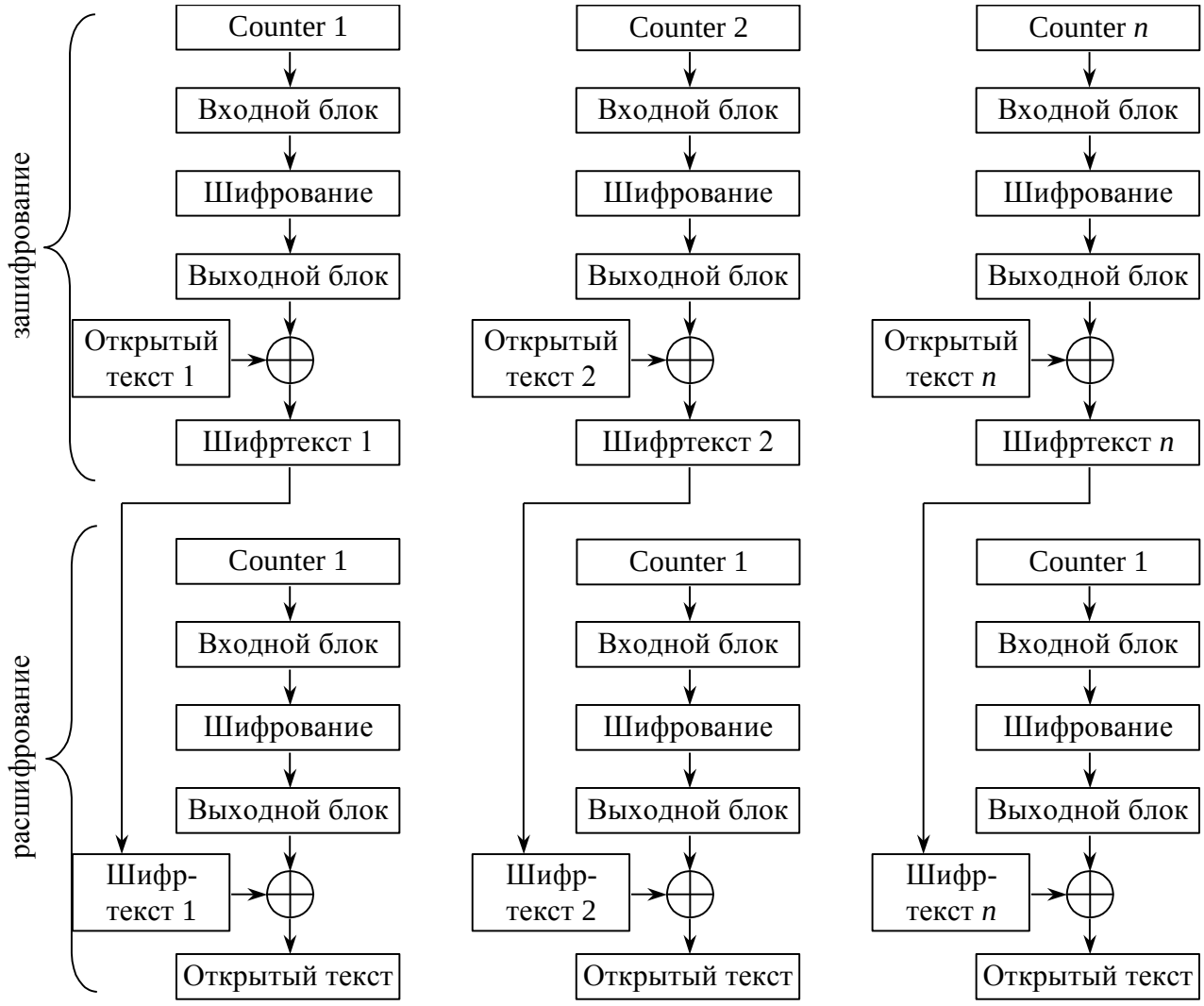


Рис. 2.40. Режим CRT

При расшифровании в режиме CTR выполняются те же самые операции, что и при зашифровании.

В режиме CTR и при зашифровании и при расшифровании функции шифрования могут быть выполнены параллельно. Блок открытого текста, который соответствует какому-то отдельному блоку зашифрованного текста, может быть получен независимо от других блоков открытого текста, если может быть определен соответствующий блок счетчика. Кроме того, функции шифрования могут быть применены к счетчикам прежде, чем станет доступным зашифрованный или открытый текст.

Компьютерный практикум по изучения шифрования с секретным ключом

Изучение стандарта криптографической защиты ГОСТ 28147-89

Цель работы Изучить криптографический стандарт шифрования ГОСТ 28147-89 и его особенности, познакомиться с различными режимами блочного шифрования.

ГОСТ 28147-89 — советский и российский стандарт симметричного шифрования, введённый в 1990 году, также является стандартом СНГ. Полное название — «ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования». С момента опубликования ГОСТа на нём стоял ограничительный гриф «Для служебного пользования», и формально шифр был объявлен «полностью открытым» только в мае 1994 года. К сожалению, история создания шифра и критерии разработчиков до сих пор не опубликованы.

Алгоритм криптографического преобразования предназначен для аппаратной или программной реализации, удовлетворяет криптографическим требованиям и по своим возможностям не накладывает ограничений на степень секретности защищаемой информации.

Стандарт обязателен для организаций, предприятий и учреждений, применяющих криптографическую защиту данных, хранимых и передаваемых в сетях, в отдельных вычислительных комплексах или на персональных компьютерах.

То, что в его названии вместо термина «шифрование» фигурирует более общее понятие «криптографическое преобразование», вовсе не случайно. Помимо нескольких тесно связанных между собой процедур шифрования, в документе описан один построенный на общих принципах с ними алгоритм выработки имитовставки. Последняя является не чем иным, как криптографической контрольной комбинацией, то есть кодом, вырабатываемым из исходных данных с использованием секретного ключа с целью имитозащиты, или защиты данных от внесения в них несанкционированных изменений.

Математические операции

Сложение по модулю 2

Операция поразрядного XOR (обозначается как ⊕) — булева функция и логическая операция. Результат выполнения операции является истинным только при условии, если является истинным в точности один из аргументов. Пример выполнения операции сложения:

$(x^6 + x^4 + x^2 + x + 1) \oplus (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$ (в виде многочленов) (10)

{01010111} ⊕ {10000011} = {11010100} (двоичное представление) (11)

Правила суммирования по модулю 2^{32} и по модулю $(2^{32}-1)$

1. Два целых числа a, b, где $0 \le a$, $b \le <, 2^{32}-1$, представленные в двоичном виде

$$a = (a_{32}, a_{31}, \ldots, a_2, a_1), \; b = (b_{32}, и_{31}, \ldots, b_2, b_1), \quad (12)$$

т.е. $$a = a_{32} \cdot 2^{31} + a_{31} \cdot 2^{30} + \ldots + a_2 \cdot 2 + a_1, \quad (13)$$

$$b = b_{32} \cdot 2^{31} + b_{31} \cdot 2^{30} + \ldots + b_2 \cdot 2 + b_1, \quad (14)$$

суммируются по модулю 2^{32} (операция $\boxplus$) по следующему правилу:

$$a \boxplus b = a + b, \text{ если } a + b < 2^{32}, \quad (15)$$

$$a \boxplus b = a + b - 2^{32}, \text{ если } a + b \ge 2^{32}. \quad (16)$$

где операция +(-) есть арифметическая сумма (разность) двух целых чисел.

2. Два целых числа a, b, где $0 \le a$, $b \le 2^{32}-1$, представленные в двоичном виде

$$a = (a_{32}, a_{31}, \ldots, a_2, a_1), \quad (17)$$

$$b = (b_{32}, b_{31}, \ldots, b_2, b_1), \quad (18)$$

суммируются по модулю $(2^{32}-1)$ (операция $\boxplus'$) по следующему правилу:

$$a \boxplus' b = a + b, \text{ если } a + b < 2^{32}, \quad (19)$$

$$a \boxplus' b = a + b - 2^{32} + 1, \text{ если } a + b \ge 2^{32}. \quad (20)$$

Структура алгоритма

Симметричное шифрование — способ шифрования, в котором для зашифровки и расшифровывания применяется один и тот же криптографический ключ. До изобретения схемы асимметричного шифрования единственным существовавшим способом являлось симметричное шифрование. Ключ алгоритма должен сохраняться в секрете обеими сторонами. Алгоритм шифрования выбирается сторонами до начала обмена сообщениями. К симметричному шифрованию предъявляются следующие требования:

- Отсутствие линейности (то есть условия f(a) xor f(b) == f(a xor b)), в противном случае облегчается применение дифференциального криптоанализа к шифру. (Функция xor – «сложение по модулю 2», «исключающее или» – результат выполнения операции является истинным только при условии, если является истинным в точности один из аргументов).
- Полная утрата всех статистических закономерностей исходного сообщения.

Алгоритм ГОСТ 28147-89 является *блочным шифром* – разновидность симметричного шифра. Особенностью блочного шифра является обработка блока нескольких байт за одну

итерацию (в нашем случае 8). Как и большинство современных блочных шифров, ГОСТ основан на *сети Фейстеля* (рисунок 7.1). Сеть представляет собой определённую 7многократно повторяющуюся (итерированную) структуру, называющуюся ячейкой Фейстеля. При переходе от одной ячейки к другой меняется ключ, причём выбор ключа зависит от конкретного алгоритма. Операции шифрования и расшифрования на каждом этапе очень просты, и при определённой доработке совпадают, требуя только обратного порядка используемых ключей. Шифрование при помощи данной конструкции легко реализуется как на программном уровне, так и на аппаратном, что обеспечивает широкие возможности применения.

Если внимательно изучить оригинал ГОСТ 28147–89, можно заметить, что в нем содержится описание алгоритмов нескольких уровней. На самом верхнем находятся практические алгоритмы, предназначенные для шифрования массивов данных и выработки для них имитовставки. Все они опираются на три алгоритма низшего уровня, называемые в тексте ГОСТа циклами. Эти фундаментальные алгоритмы будут называться «базовые циклы», чтобы отличать их от всех прочих циклов. Они имеют следующие названия и обозначения: (последние приведены в скобках)

- цикл зашифрования (32-З);
- цикл расшифрования (32-Р);
- цикл выработки имитовставки (16-З).

В свою очередь, каждый из базовых циклов представляет собой многократное повторение одной единственной процедуры, называемой «основным шагом криптопреобразования».

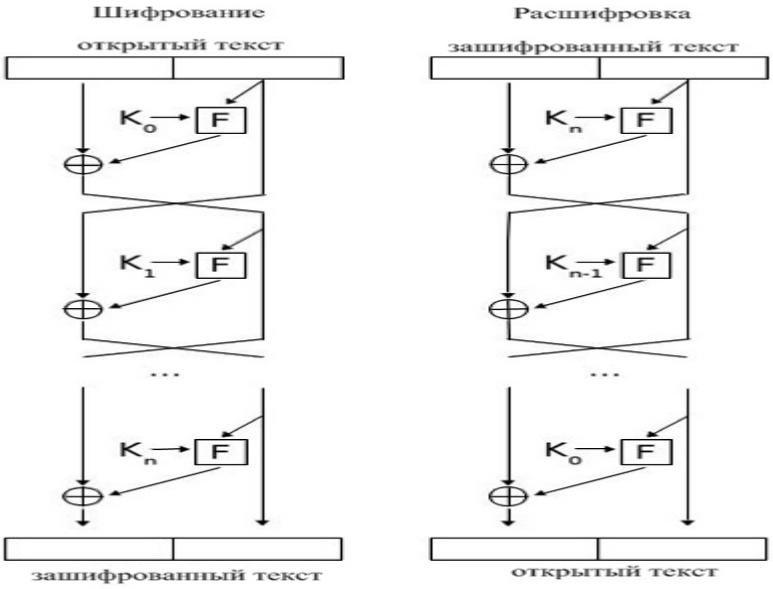


Рис. 2.41. Сеть Фейстеля

- *Ключ* является массивом из восьми 32-битовых элементов кода В ГОСТе элементы ключа используются как 32-разрядные целые числа без знака. Таким образом, размер ключа составляет 256 бит (32 байта). Ключ должен являться массивом статистически независимых битов, принимающих с равной вероятностью значения 0 и 1. При этом некоторые конкретные значения ключа могут оказаться «слабыми», то есть шифр может не обеспечивать заданный уровень стойкости в случае их использования. Однако, предположительно, доля таких значений в общей массе всех возможных ключей ничтожно мала. Поэтому ключи, выработанные с помощью некоторого датчика истинно случайных чисел, будут качественными с вероятностью, отличающейся от единицы на ничтожно малую величину.
- *Таблица замен* является вектором, содержащим восемь *узлов замены*. Каждый узел замены, в свою очередь, является вектором, содержащим шестнадцать 4-битовых элементов замены, которые можно представить в виде целых чисел от 0 до 15, все элементы одного узла замены обязаны быть различными. Таким образом, таблица замен может быть представлена в виде матрицы размера 8x16 или 16x8, содержащей 4-битовые заменяющ Таким образом, общий объем таблицы замен равен 512 бит (64 байта).

Общее количество узлов замены (S-блоков) ГОСТа — восемь. Каждый S-блок представляет собой перестановку чисел от 0 до 15. Первая 4-битная подпоследовательность попадает на вход первого S-блока, вторая — на вход второго и т. д.

Если S-блок выглядит так: 1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12

и на входе S-блока 0, то на выходе будет 1, если 4, то на выходе будет 5, если на входе 12, то на выходе 6 и т. д. (для таблицы 2.24)

Таблица 2.24. S-блоки, приведенные в ГОСТ Р 34.11-94 для целей тестирования

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

Основной шаг криптопреобразования

Основной шаг криптопреобразования по своей сути является оператором, определяющим преобразование 64-битового блока данных. Дополнительным параметром этого оператора является 32-битовый блок, в качестве которого используется какой-либо элемент ключа. Схема алгоритма основного шага приведена на рисунке 2.2. Рассмотрим подробнее этапы основного шага криптопреобразования:

Шаг 0 Определяет исходные данные для основного шага криптопреобразования.

N – преобразуемый 64-битовый блок данных, в ходе выполнения шага его младшая (N_1) и старшая (N_2) части обрабатываются как отдельные 32-битовые целые числа без знака. Таким образом, можно записать $N=(N_1,N_2)$.

X – 32-битовый элемент ключа;

Шаг 1 Сложение с ключом.

Младшая половина преобразуемого блока складывается по модулю 2^{32} с используемым на шаге элементом ключа, результат передается на следующий шаг;

Шаг 2 Поблочная замена.

32-битовое значение, полученное на предыдущем шаге, интерпретируется как массив из восьми 4-битовых блоков кода:

$$S=(S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7) \quad (21)$$

причем S_0 содержит 4 самых младших, а S_7 – 4 самых старших бита S. Далее значение каждого из восьми блоков заменяется новым, которое выбирается по таблице замен следующим образом: значение блока S_i меняется на S_i-тый по порядку элемент (нумерация с нуля) i-того узла замены (т.е. i-той строки таблицы замен, нумерация также с нуля). Другими словами, в качестве замены для значения блока выбирается элемент из таблицы замен с номером строки, равным номеру заменяемого блока, и номером столбца, равным значению

заменяемого блока как 4-битового целого неотрицательного числа. Отсюда становится понятным размер таблицы замен: число строк в ней равно числу 4-битовых элементов в 32-битовом блоке данных, то есть восьми, а число столбцов равно числу различных значений 4-битового блока данных, равному, как известно 2^4 (т. е. шестнадцати).

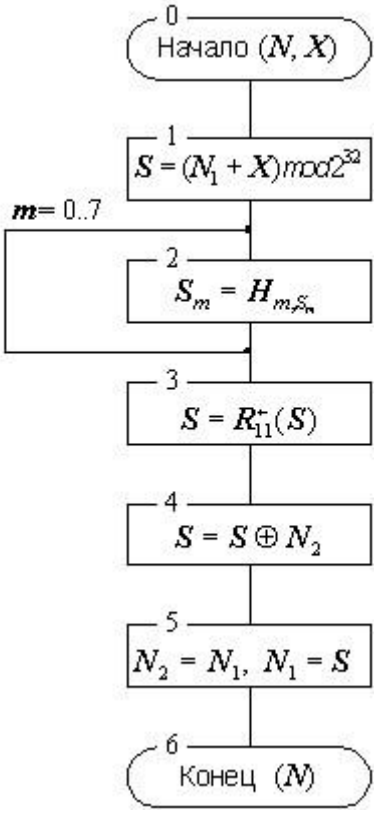


Рис. 2.42. Схема основного шага криптопреобразования алгоритма ГОСТ 28147-89

Шаг 3 Циклический сдвиг на 11 бит влево.

Результат предыдущего шага сдвигается циклически на 11 бит в сторону старших разрядов и передается на следующий шаг. На схеме алгоритма символом «$R_{11}^{\leftarrow}$» обозначена функция циклического сдвига своего аргумента на 11 бит влево, т.е. в сторону старших разрядов.

Шаг 4 Побитовое сложение

Значение, полученное на шаге 3, побитно складывается по модулю 2 со старшей половиной преобразуемого блока.

Шаг 5 Сдвиг по цепочке

Младшая часть преобразуемого блока сдвигается на место старшей, а на ее место помещается результат выполнения предыдущего шага.

Шаг 6 Завершение криптопреобразования.

Полученное значение преобразуемого блока возвращается как результат выполнения алгоритма основного шага криптопреобразования.

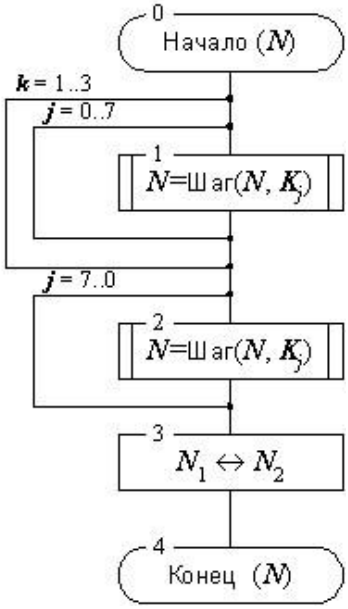


Рис. 2.43. Схема цикла зашифрования 32-З

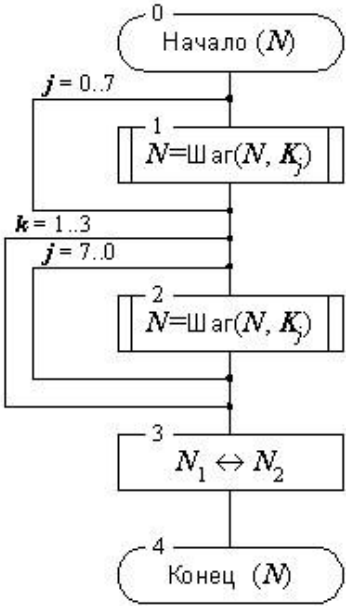


Рис. 2.44. Схема цикла расшифрования 32-Р

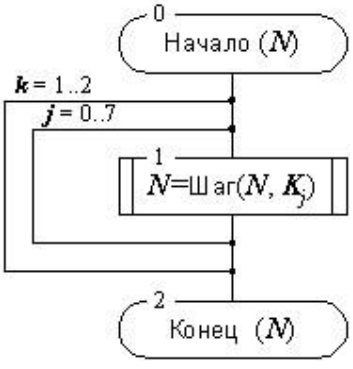


Рис. 2.45. Схема цикла выработки имитовставки 16-З.

Порядок использования ключевых элементов:

- Цикл зашифрования 32-З:

$K_0,K_1,K_2,K_3,K_4,K_5,K_6,K_7,K_0,K_1,K_2,K_3,K_4,K_5,K_6,K_7,K_0,K_1,K_2,K_3,K_4,K_5,K_6,K_7,K_7,K_6,$
K_5,K_4,K_3,K_2,K_1,K_0

- Цикл расшифрования 32-Р:

$K_0,K_1,K_2,K_3,K_4,K_5,K_6,K_7,K_7,K_6,K_5,K_4,K_3,K_2,K_1,K_0,K_7,K_6,K_5,K_4,K_3,K_2,K_1,K_0,K_7,K_6,K_5,$
K_4,K_3,K_2,K_1,K_0

- Цикл выработки имитовставки 16-З:

$K_0,K_1,K_2,K_3,K_4,K_5,K_6,K_7,K_0,K_1,K_2,K_3,K_4,K_5,K_6,K_7$

Режимы шифрования

ГОСТ 28147-89 предусматривает три следующих режима шифрования данных:

- простая замена,
- гаммирование,
- гаммирование с обратной связью,
- дополнительный режим выработки имитовставки

Простая замена

Зашифрование в данном режиме заключается в применении цикла 32-З к блокам открытых данных, расшифрование – цикла 32-Р к блокам зашифрованных данных. Это наиболее простой из режимов, 64-битовые блоки данных обрабатываются в нем независимо друг от друга. Схемы алгоритмов зашифрования и расшифрования в режиме простой замены приведены на рисунках 2.43 и 2.44 соответственно. Размер массива открытых или зашифрованных данных, подвергающийся соответственно зашифрованию или расшифрованию, должен быть кратен 64 битам:

$$|T_o|=|T_ш|=64\cdot n \qquad (21)$$

где n – целое положительное число

после выполнения операции размер полученного массива данных не изменяется.

Режим шифрования простой заменой имеет следующие особенности:

- Так как блоки данных шифруются независимо друг от друга и от их позиции в массиве данных, при зашифровании двух одинаковых блоков открытого текста получаются одинаковые блоки шифртекста и наоборот. Отмеченное свойство позволит криптоаналитику сделать заключение о тождественности блоков исходных данных, если в массиве зашифрованных данных ему встретились идентичные блоки, что является недопустимым для серьезного шифра.

- Если длина шифруемого массива данных не кратна 8 байтам (64 битам), возникает проблема, чем и как дополнять последний неполный блок данных массива до полных 64 бит. Эта задача не так проста, как кажется на первый взгляд. Очевидные решения вроде «дополнить неполный блок нулевыми битами» или, более общее, «дополнить неполный блок фиксированной комбинацией нулевых и единичных битов» могут при определенных условиях дать в руки криптоаналитика возможность методами перебора определить содержимое этого самого неполного блока, и этот факт означает снижение стойкости шифра. Кроме того, длина шифртекста при этом изменится, увеличившись до ближайшего целого, кратного 64 битам, что часто бывает нежелательным.

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

Гаммирование

Как же можно избавиться от недостатков режима простой замены? Для этого необходимо сделать возможным шифрование блоков с размером менее 64 бит и обеспечить зависимость блока шифртекста от его номера, иными словами, рандомизировать процесс шифрования. В ГОСТе это достигается двумя различными способами в двух режимах шифрования, предусматривающих гаммирование.

Гаммирование – это наложение (снятие) на открытые (зашифрованные) данные криптографической гаммы, то есть последовательности элементов данных, вырабатываемых с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных.

Для наложения гаммы при зашифровании и ее снятия при расшифровании должны использоваться взаимно обратные бинарные операции. В ГОСТе для этой цели используется операция побитового сложения по модулю 2, поскольку она является обратной самой себе и, к тому же, наиболее просто реализуется аппаратно. Гаммирование решает обе упомянутые проблемы: во-первых, все элементы гаммы различны для реальных шифруемых массивов и, следовательно, результат зашифрования даже двух одинаковых блоков в одном массиве данных будет различным. Во-вторых, хотя элементы гаммы и вырабатываются одинаковыми порциями в 64 бита, использоваться может и часть такого блока с размером, равным размеру шифруемого блока.

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

РГПЧ, используемый для выработки гаммы, является рекуррентной функцией:

$$\Omega_{i+1}=f(\Omega_i), \tag{22}$$

где Ω_i - элементы рекуррентной последовательности,

f – функция преобразования

Элемент данных «Ω_0» является параметром алгоритма для режимов гаммирования, на схемах он обозначен как S, и называется в криптографии *синхропосылкой,* а в ГОСТе – *начальным заполнением* одного из регистров шифрователя.

Синхропосылка (вектор инициализации) – случайное число, которое регулярно обновляется, передается по каналу управления и используется для инициализации алгоритма шифрования.

Из соображений увеличения стойкости алгоритма, разработчики ГОСТа решили использовать для инициализации РГПЧ не непосредственно синхропосылку, а результат ее преобразования по циклу 32-З.

Последовательность элементов, вырабатываемых РГПЧ, целиком зависит от его начального заполнения, то есть элементы этой последовательности являются функцией своего номера и начального заполнения РГПЧ:

$$\Omega_i=f_i(\Omega_0), \tag{23}$$

где $f_i(X)=f(f_{i-1}(X))$,

$f_0(X)=X$

С учетом преобразования по алгоритму простой замены добавляется еще и зависимость от ключа:

$$\Gamma_i=Ц_{32\text{-}З}(\Omega_i)=Ц_{32\text{-}З}(f_i(\Omega_0))=Ц_{32\text{-}З}(f_i(Ц_{32\text{-}З}(S)))=\varphi_i(S,K),$$

где Γ_i – i-тый элемент гаммы,

K – ключ

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

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

- период повторения последовательности чисел, вырабатываемой РГПЧ, не должен сильно (в процентном отношении) отличаться от максимально возможного при заданном размере блока значения 2^{64};
- соседние значения, вырабатываемые РГПЧ, должны отличаться друг от друга в каждом байте, иначе задача криптоаналитика будет упрощена;
- РГПЧ должен быть достаточно просто реализуем как аппаратно, так и программно на наиболее распространенных типах процессоров, большинство из которых, как известно, имеют разрядность 32 бита

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

- в 64-битовом блоке старшая и младшая части обрабатываются независимо друг от друга, фактически, существуют два независимых РГПЧ для старшей и младшей частей блока:

$$\Omega_i = (\Omega_i^0, \Omega_i^1), \quad (23)$$

$$|\Omega_i^0| = |\Omega_i^1| = 32, \quad (24)$$

$$\Omega_{i+1}^0 = \hat{f}(\Omega_i^0), \quad (25)$$

$$\Omega_{i+1}^1 = \tilde{f}(\Omega_i^1) \quad (26)$$

- рекуррентные соотношения для старшей и младшей частей следующие (нижний индекс в записи числа означает его систему счисления, таким образом, константы, используемые на данном шаге, записаны в 16-ричной системе счисления):

$$\Omega_{i+1}^0 = (\Omega_i^0 + C_0) mod 2^{32} \quad (27)$$

где $C_0=1010101_{16}$

$$\Omega_{i+1}^1 = (\Omega_i^1 + C_1 - 1) mod (2^{32} - 1) + 1 \quad (28)$$

где $C_1=1010104_{16}$

Второе выражение нуждается в комментариях, так как в тексте ГОСТа приведено нечто другое:

$$\Omega_{i+1}^1 = (\Omega_i^1 + C_1) mod (2^{32} - 1) \quad (29)$$

тем же значением константы C_1. Но далее в тексте стандарта дается комментарий, что, оказывается, под операцией взятия остатка по модулю (2^{32}–1) там понимается не то же самое, что и в математике. Отличие заключается в том, что согласно ГОСТу:

$$(2^{32}–1) \text{mod} (2^{32}–1) = (2^{32}–1), \text{а не } 0 \quad (30)$$

На самом деле, это упрощает реализацию формулы, а математически корректное выражение для нее приведено в формуле (30).

Период повторения последовательности для младшей части составляет 2^{32}, для старшей части (2^{32}–1), для всей последовательности период составляет $2^{32}*(2^{32}–1)$. Первая формула из двух реализуется за одну команду, вторая , несмотря на ее кажущуюся громоздкость, за две команды на всех современных 32-разрядных процессорах – первой командой идет обычное сложение по модулю 2^{32} с запоминанием бита переноса, а вторая команда прибавляет бит переноса к полученному значению.

Рассмотрим шаги алгоритма гаммирования, представленного на рисунке 2.46:

Шаг 0 Определяет исходные данные для основного шага криптопреобразования.

$T_{о(ш)}$ – массив открытых (зашифрованных) данных произвольного размера, подвергаемый процедуре зашифрования (расшифрования), по ходу процедуры массив подвергается преобразованию порциями по 64 бита;

S – синхропосылка , 64-битовый элемент данных, необходимый для инициализации генератора гаммы;

Шаг 1 Начальное преобразование синхропосылки.

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

Шаг 2 Один шаг работы РГПЧ, реализующий его рекуррентный алгоритм.

В ходе данного шага старшая (S_1) и младшая (S_0) части последовательности данных вырабатываются независимо друг от друга;

Шаг 3 Гаммирование.

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

Шаг 4 Завершение гаммирования.

Результат работы алгоритма – зашифрованный (расшифрованный) массив данных.

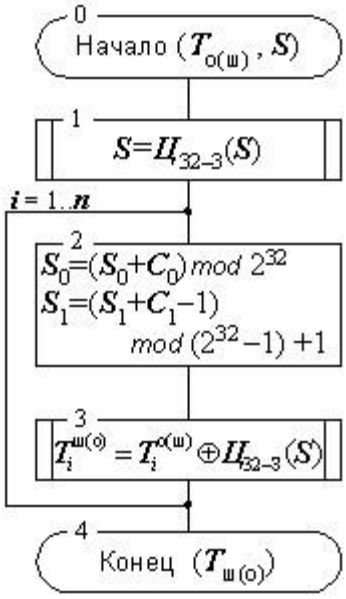


Рис. 2.46. Алгоритм зашифрования (расшифрования) данных в режиме гаммирования

Особенности гаммирования как режима шифрования:

1. Одинаковые блоки в открытом массиве данных дадут при зашифровании различные блоки шифртекста, что позволит скрыть факт их идентичности.

2. Поскольку наложение гаммы выполняется побитно, шифрование неполного блока данных легко выполнимо как шифрование битов этого неполного блока, для чего используется соответствующие биты блока гаммы. Так, для зашифрования неполного блока в 1 бит согласно стандарту следует использовать самый младший бит из блока гаммы.

3. Синхропосылка, использованная при зашифровании, каким-то образом должна быть передана для использования при расшифровании. Это может быть достигнуто следующими путями:

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

4. Биты массива данных шифруются независимо друг от друга. Таким образом, каждый бит шифртекста зависит от соответствующего бита открытого текста и, естественно, порядкового номера бита в массиве. Из этого вытекает, что изменение бита шифртекста на противоположное значение приведет к аналогичному изменению бита открытого текста на противоположный. Данное свойство дает злоумышленнику возможность, воздействуя на биты шифртекста, вносить предсказуемые и даже целенаправленные изменения в соответствующий открытый текст, получаемый после его расшифрования, не обладая при этом секретным ключом. Это иллюстрирует хорошо известный в криптологии факт, что *секретность* и *аутентичность* различные свойства криптографических систем.

Гаммирование с обратной связью

Данный режим очень похож на режим гаммирования и отличается от него только способом выработки элементов гаммы – очередной элемент гаммы вырабатывается как результат преобразования по циклу 32-З предыдущего блока зашифрованных данных, а для зашифрования первого блока массива данных элемент гаммы вырабатывается как результат преобразования по тому же циклу синхропосылки (рисунок 2.7). Этим достигается зацепление блоков – каждый блок шифртекста в этом режиме зависит от соответствующего и всех предыдущих блоков открытого текста, это видно и в уравнении режима шифрования (расшифрования) гаммирования с обратной связью (формула 2.25). Поэтому данный режим

иногда называется *гаммированием с зацеплением блоков*. На стойкость шифра факт зацепления блоков не оказывает никакого влияния.

Шифрование в режиме гаммирования с обратной связью обладает теми же особенностями, что и шифрование в режиме обычного гаммирования, за исключением влияния искажений шифртекста на соответствующий открытый текст (свойство 4).

$$T_i^{o} = T_i^{ш} \oplus Ц_{32-3}(T_{i-1}^{ш}) \qquad (31)$$

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

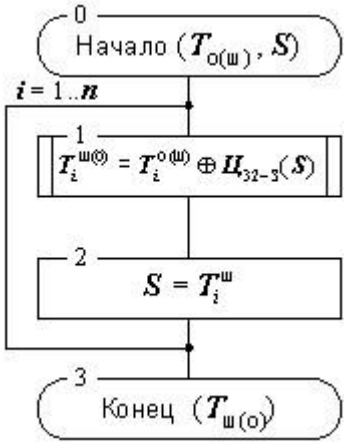


Рис. 2.47. Алгоритм зашифрования (расшифрования) данных в режиме гаммирования с обратной связью.

Выработка имитовставки к массиву данных

Для решения задачи обнаружения искажений в зашифрованном массиве данных с заданной вероятностью в ГОСТе предусмотрен дополнительный режим криптографического преобразования – выработка имитовставки (рисунок 2.48).

Целью использования имитовставки является обнаружение всех случайных или преднамеренных изменений в массиве информации. Для потенциального злоумышленника две следующие задачи практически неразрешимы, если он не владеет ключом:

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

В качестве имитовставки берется часть блока, полученного на выходе, обычно – 32 его младших бита. При выборе размера имитовставки надо принимать во внимание, что вероятность успешного навязывания ложных данных равна величине $2^{-|I|}$ на одну попытку подбора, если в распоряжении злоумышленника нет более эффективного метода подбора, чем простое угадывание. При использовании имитовставки размером 32 бита эта вероятность равна $2^{-32} \approx 0.23 \cdot 10^{-9}$.

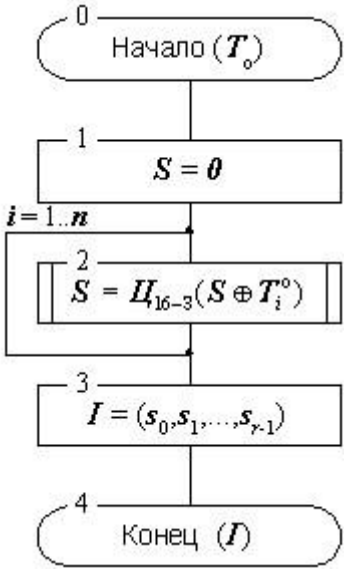


Рис. 2.48. Алгоритм выработки имитовставки для массива данных

Криптоанализ алгоритма

В 1994 г. описание алгоритма ГОСТ 28147-89 было переведено на английский язык и опубликовано в статье Волонгонгского университета (Австралия) «Советский алгоритм шифрования». Именно после этого стали появляться результаты его анализа, выполненного зарубежными специалистами; однако, в течение значительного времени не было найдено каких-либо атак, приближающихся к практически осуществимым.

Анализ таблиц замен

Поскольку таблицы замен в стандарте не приведены, в ряде работ высказывается предположение, что «компетентная организация» может выдать как «хорошие», так и

«плохие» таблицы замен. Ясно, что криптостойкость алгоритма во многом зависит от свойств используемых таблиц замен, соответственно, существуют слабые таблицы замен, применение которых может упростить вскрытие алгоритма. Тем не менее, возможность использования различных таблиц замен кажется весьма достойной идеей, в пользу которой можно привести два следующих факта из истории стандарта шифрования США DES:

- Атаки с помощью как линейного, так и дифференциального криптоанализа алгоритма DES используют конкретные особенности таблиц замен. При использовании других таблиц криптоанализ придется начинать сначала
- Были попытки усилить DES против линейного и дифференциального криптоанализа путем использования более стойких таблиц замен. Такие таблицы, действительно более стойкие, были предложены, например, в алгоритме s^5DES. Но, увы, заменить DES на s^5DES было невозможно, поскольку таблицы замен жестко определены в стандарте, соответственно, реализации алгоритма наверняка не поддерживают возможность смены таблиц на другие

Однако, в работе доказано, что секретные таблицы замен могут быть вычислены с помощью следующей атаки, которая может быть применена практически:

1. Устанавливается нулевой ключ и выполняется поиск «нулевого вектора». Этот этап занимает порядка 2^{32} операций шифрования.
2. С помощью нулевого вектора вычисляются значения таблиц замен, что занимает не более 2^{11} операций.

Модификации алгоритма и их анализ

Алгоритмы:

- GOST-H, в котором, относительно оригинального алгоритма, изменен порядок использования подключей, а именно, в раундах с 25-го по 32-й подключи используются в прямом порядке, т.е. точно так же, как и в предыдущих раундах алгоритма;
- 20-раундовый алгоритм GOSTÅ, в раунде которого для наложения ключа используется операция XOR вместо сложения по модулю 2^{32}.

По результатам анализа сделан вывод о том, что GOST-H и GOSTÅ слабее исходного алгоритма ГОСТ 28147-89, поскольку оба имеют классы слабых ключей.

Анализ полнораундового алгоритма

Существуют атаки и на полнораундовый ГОСТ 28147-89 без каких-либо модификаций. Одна из первых открытых работ, в которых был проведен анализ алгоритма посвящена

атакам, использующим слабости процедур расширения ключа ряда известных алгоритмов шифрования. В частности, полнораундовый алгоритм ГОСТ 28147-89 может быть вскрыт с помощью дифференциального криптоанализа на связанных ключах, но только в случае использования слабых таблиц замен. 24-раундовый вариант алгоритма (в котором отсутствуют первые 8 раундов) вскрывается аналогичным образом при любых таблицах замен, однако, сильные таблицы замен делают такую атаку абсолютно непрактичной.

Отечественные ученые А.Г. Ростовцев и Е.Б. Маховенко в 2001 г. предложили принципиально новый метод криптоанализа (по мнению авторов, существенно более эффективный, чем линейный и дифференциальный криптоанализ) путем формирования целевой функции от известного открытого текста, соответствующего ему шифртекста и искомого значения ключа и нахождения ее экстремума, соответствующего истинному значению ключа. Они же нашли большой класс слабых ключей алгоритма ГОСТ 28147-89, которые позволяют вскрыть алгоритм с помощью всего 4-х выбранных открытых текстов и соответствующих им шифртекстов с достаточно низкой сложностью.

В 2004 г. группа специалистов из Кореи предложила атаку, с помощью которой, используя дифференциальный криптоанализ на связанных ключах, можно получить с вероятностью 91,7% 12 бит секретного ключа. Для атаки требуется 2^{35} выбранных открытых текстов и 2^{36} операций шифрования. Как видно, данная атака, практически, бесполезна для реального вскрытия алгоритма.

Достоинства ГОСТа

- бесперспективность силовой атаки, т.е. полным перебором (XSL-атаки в учёт не берутся, т.к. их эффективность на данный момент полностью не доказана);
- эффективность реализации и соответственно высокое быстродействие на современных компьютерах.
- наличие защиты от навязывания ложных данных (выработка имитовставки) и одинаковый цикл шифрования во всех четырех алгоритмах ГОСТа

Недостатки ГОСТа

Основные проблемы ГОСТа связаны с неполнотой стандарта в части генерации ключей и таблиц замен. Тривиально доказывается, что у ГОСТа существуют «слабые» ключи и таблицы замен, но в стандарте не описываются критерии выбора и отсева «слабых». Также стандарт не специфицирует алгоритм генерации таблицы замен (S-блоков). С одной стороны, это может являться дополнительной секретной информацией (помимо ключа), а с другой, поднимает ряд проблем:

- нельзя определить криптостойкость алгоритма, не зная заранее таблицы замен;
- реализации алгоритма от различных производителей могут использовать разные таблицы замен и могут быть несовместимы между собой;
- возможность преднамеренного предоставления слабых таблиц замен лицензирующими органами;
- потенциальная возможность (отсутствие запрета в стандарте) использования таблиц замены, в которых узлы не являются перестановками, что может привести к чрезвычайному снижению стойкости шифра.

Аппаратные шифраторы серии Криптон

Устройства криптографической защиты данных серии КРИПТОН — это аппаратные шифраторы для IBM PC-совместимых компьютеров. Устройства применяются в составе средств и систем криптографической защиты данных для обеспечения информационной безопасности (в том числе защиты с высоким уровнем секретности) в государственных и коммерческих структурах.

Устройства КРИПТОН гарантируют защиту информации, обрабатываемой на персональном компьютере и/или передаваемой по открытым каналам связи.

Устройства КРИПТОН выполнены в виде плат расширения ISA и PCI персонального компьютера с процессором i386 и выше.

Устройства КРИПТОН разработаны, производятся и реализуются фирмой АНКАД. Они построены на разработанных фирмой АНКАД специализированных 32-разрядных шифрпроцессорах серии БЛЮМИНГ.

За 10 лет работы Фирма АНКАД поставила более 15 тысяч устройств КРИПТОН заказчикам в Центральном Банке, Федеральном агентстве правительственной связи и информации при Президенте РФ, министерствах обороны и внутренних дел, Министерстве по налогам и сборам, Федеральном казначействе, коммерческих банках, финансовых и страховых компаниях, многим корпоративным клиентам.

Сеть кооперационного производства устройств КРИПТОН охватывает наиболее известные предприятия российской электроники (ОАО “Ангстрем” и др.).

Устройства серии КРИПТОН имеют сертификаты соответствия требованиям ФСБ и ФСТЭК (в том числе в составе абонентских пунктов и автоматизированных рабочих мест для защиты информации, содержащей сведения, составляющие государственную тайну).

Таблица 2.24. Основные технические данные и характеристики серии КРИПТОН (данные за 2004-2005г.г.)

Алгоритм шифрования	ГОСТ 28147-89
Размерность секретного ключа шифрования, бит	256 (количество возможных комбинаций ключей — 10^{77})
Размерность открытого ключа, бит	512 или 1024
Количество уровней ключевой системы	3 (главный ключ — пользовательский/сетевой ключ — сеансовый ключ)
Датчик случайных чисел	аппаратный (аттестован экспертной организацией)
Отклонение распределения значения случайных чисел от равновероятного распределения	не более 0,0005
Поддерживаемые операционные системы	MS-DOS, Windows 95(98)/ME/NT 4.0/2000/XP/2003 UNIX (Solaris/Intel) (возможно создание оригинальных программных драйверов для работы устройств)
Скорость передачи данных по сети	10/100 Мбит/с
Скорость шифрования	до 9 Мбайт/с – сетевые шифраторы до 240 Мбит/с – шифраторы жесткого диска
Класс защиты по классификации ФСБ	до КС1
Алгоритм ЭЦП	ГОСТ Р 34.10-94 (с длинной секретного ключа 256 бит)

Описание лабораторной установки и методики измерений

Интерфейс учебно-программного комплекса

Лабораторный комплекс был разработан на языке высокого уровня Visual Basic 6.

Комплекс позволяет работать только с введенным текстом (обработка ведется в двоичном коде, преобразование к которому производится посредством ASCII кодов).

Для удобства работы комплекс оснащен всплывающими подсказками на все входные и выходные параметры, рабочие поля, активные элементы и элементы управления.

Оптимизация кода программы не производилась, поэтому при работе с большими об’емами текста возможны некорректное отображение резальтата или зависание программы.

Основные элементы комплекса:

1. Поле для ввода ключа – максимальная длина ключа 32 символа, при его отсутствии будет использован нулевой вектор
2. Выбор режима шифрования:
    - При выборе простой замены есть возможность проверить кратность текста 8 байтовым блокам;
    - При выборе режимов гаммирование есть возможность ввести вектор инициализации (синхропосылку), при ее отсутствии будет использован нулевой вектор
3. Поле для ввода текста для зашифрования
4. Возможность очистить поле шифрования
5. Возможность узнать об’ем текста в поле шифрования
6. Кнопка для зашифрования текста
7. Время, затраченное на зашифрования – напротив поля текста
8. Поле зашифрованного текста
9. Возможность очистить поле зашифрованного текста
10. Возможность узнать об’ем зашифрованного текста
11. Кнопка для расшифрования текста
12. Время, затраченное на расшифровку – напротив поля шифротекста
13. Системное время вашего компьютера
14. Возможность использовать материалы в электронном виде (рисунок 2.49)

Рис. 2.49. Лабораторный комплекс

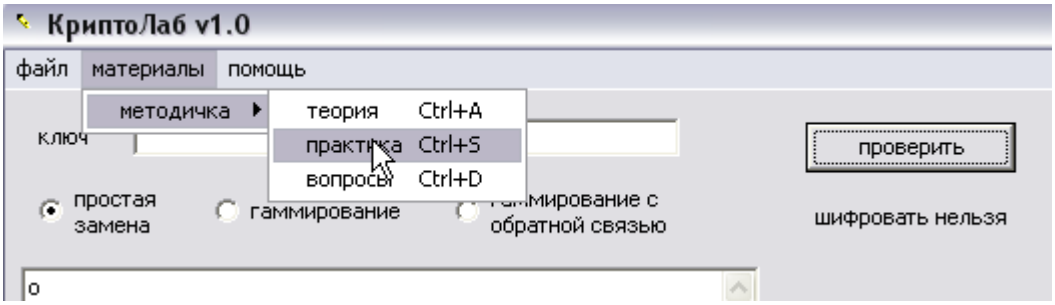


Рис. 2.50. Предоставленные материалы

Порядок выполнения работы

1. Ознакомится с теорией по стандарту ГОСТ 28147-89 (режимы шифрования, строение основного шага криптопреобразования, особенности использования таблицы замен).

2. Взять у преподавателя задание, состоящие из:

уникального ключа для шифрования/дешифрования текстовых сообщений, синхропосылки

3. Исследование стандарта шифрования

3.1 выбрать режим шифрования «простая замена»

3.2 ввести значение ключа

3.3 ввести тестовый текст (взять у преподавателя или пройти **материалы –> методичка –> практика –> тексты –> простая замена**

3.4 проверить кратность 8 байтовых блоков нажатием кнопки «проверить»

3.5 зашифровать текст нажатием соответствующей кнопки

3.6 очистить текстовое поле

3.7 расшифровать текст

3.8 проверить наличие ошибок

3.9 зафиксировать об’емы текстов и время за/расшифрования в таблицу

3.10 Проделать пункты 3.3-3.9 для оставшихся текстов

3.11 выбрать режим шифрования «гаммирование»

3.12 ввести тестовый текст (взять у преподавателя или пройти **материалы –> методичка –> практика –> тексты –> гаммирования**

3.13 ввести значения ключа и синхропосылки

3.14 зашифровать текст нажатием соответствующей кнопки

3.15 очистить текстовое поле

3.16 расшифровать текст

3.17 проверить наличие ошибок

3.18 зафиксировать об’емы текстов и время за/расшифрования в таблицу

3.19 Проделать пункты 3.14-3.18 для оставшихся текстов

3.20 проделать пункты 3.11-3.19 для режима «гаммирование с обратной связью»

3.21 Построить графики зависимости размера файла от времени его шифрования /расшифрования для всех режимов.

3.22 проверить изменение шифротекста при изменении ключа и синхропосылки

3.23 посмотреть как изменится текст после расшифрования если в шифротекст внести изменения (для всех режимов шифрования)

Таблица 2.25. Результаты измерения

Размер файла				
Время зашифрования				
Время расшифрования				

Содержание отчета

1 Цель работы.

2 Описание действий по каждому пункту.

3 Результаты проделанной работы (таблицы, графики, скриншоты).

4 Выводы.

Контрольные вопросы

1. Что такое симметричное шифрование, блочные коды?
2. Что представляет собой стандарт ГОСТ 28147-89 (длина ключа, размер входного блока, ключа, таблица замен)?
3. Что собой представляет архитектура данного стандарта?
4. Как устроен основной шага криптопреобразования?
5. Сколько циклов шифрования предусмотрено стандартом?
6. Что быстрее шифрование или расшифрование? Почему?
7. Какие режимы шифрования применяются в стандарте ГОСТ 28147-89?
8. Какие режимы быстрее при расшифровании? Почему?
9. Какие режимы лучше восстанавливают зашифрованную информацию при ошибке в одном символе? Почему?
10. С какой целью используется синхропосылка или вектор инициализации?

Описание лабораторной установки шифрование AES и методики измерений

Интерфейс учебно-программного комплекса

Учебно-программный комплекс (в дальнейшем просто комплекс) был разработан на языке высокого уровня Visual Basic.

Главное окно

На рисунке 2.51 приведен интерфейс главного окна комплекса.

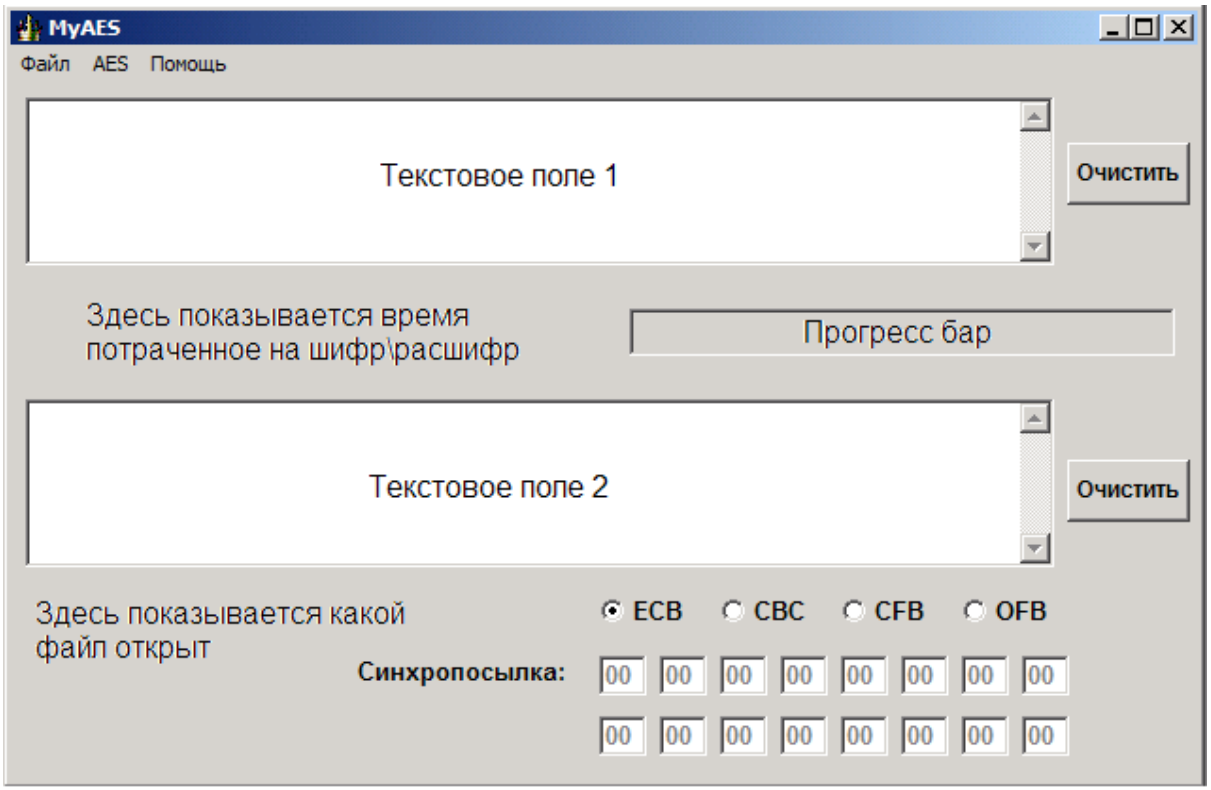


Рис. 2.51. Интерфейс главного окна комплекса

Главное окно состоит из:

1. Текстовое поле 1 – сюда вводится открытый текст, также если открывается файл с расширением txt, то его содержимое помещается сюда же.

2. Текстовое поле 2 – сюда помещается зашифрованный текст, также если открывается файл с расширением aes, то его содержимое помещается сюда же.

3. Прогресс бар – наглядно показывает время требуемое для шифрования \ расшифрования.

4. Кнопки “Очистить” – удаляют весь текст из текстовых полей.

5. Переключатели ECB, CBC, CFB, OFB – показывают, какой режим блочного шифрования используется.

6. Синхропосылка – задает синхропосылку для CBC, CFB и OFB.

В главном окне и происходит шифрование и расшифрование. Также в нем можно выбрать режимы симметричного шифрования (ECB, CBC, CFB, OFB). Про них сказано ниже.

Пункт меню “Файл”

На рисунке 2.52 приведен пункт меню “Файл”:

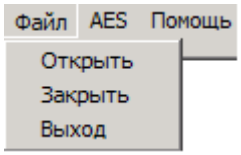


Рис. 2.52. Пункт меню “Файл”

Пункт меню “Файл” содержит следующие опции:

1. Открыть – открывает файл (*.txt, *.aes), но не больше 32kB для текстовых файлов и 64кВ для файлов типа aes.

2. Закрыть – закрывает выбранный файл.

3. Выход – завершает работу комплекса.

Пункт меню “AES”

На рисунке 13 приведен пункт меню “AES”:

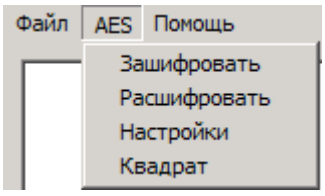


Рис. 2.53. Пункт меню “AES”

Пункт меню “AES” содержит следующие опции:

1. Зашифровать – шифрует текст, расположенный в текстовом поле 1, и помещает его в текстовое поле 2.

2. Расшифровать - расшифровывает текст, расположенный в текстовом поле 2, и помещает его в текстовое поле 1.

3. Настройки – позволяет выбрать свой ключ шифрования, а также включить \ выключить режим логирования. (рисунок 2.54)

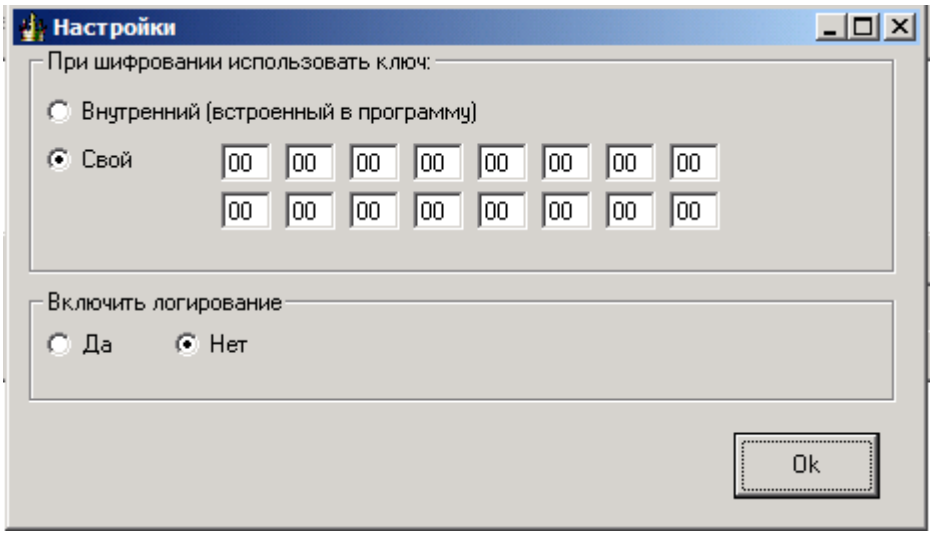


Рис. 2.54. Окно “Настройки”

В этом окне вы можете ввести свой ключ шифрования (например из задания по лабораторной работе) или же использовать внутренний. При использовании внутреннего ключа шифрования вы можете шифровать свои сообщения и отправлять их кому-либо. Причем получателю не обязательно знать ключ, главное чтобы у него была данная программа.

Здесь же вы можете включить логирование. При включенном логировании в папке программы создается Encrypt.txt (Decrypt.txt), в зависимости от того, что вы делаете шифруете или расшифровываете. В этом файле содержатся все преобразования, происходящие с открытым текстом. Пример такого файла приведен на рисунке 2.55:

```
Зашифрование
Входной блок CFF0EEE2E5F0EAE001010101010101
Раундовый ключ: 00000000000000000000000000000000
Состояние после сложения с ключом: CFF0EEE2E5F0EAE001010101010101
Раунд №1
Состояние после SubBytes: 8A8C2898D98C87E17C7C7C7C7C7C7C
Состояние после ShiftRows: 8A8C7C7CD97C7C987C7C28E17C8C877C
Состояние после MixColumns: 80717A8DC93DEE5BB51D68098C916177
Раундовый ключ: 62636363626363636263636362636363
Состояние после сложения с ключом: E21219EEAB5E8D38D77E0B6AEEF20214
Раунд №2
Состояние после SubBytes: 98C9D42862585D070EF32B02288977FA
Состояние после ShiftRows: 98582BFA62F377280E89D40728C95D02
Состояние после MixColumns: 12AF832F952E07724F673D414F445DE8
Раундовый ключ: 9B9898C9F9FBFBAA9B9898C9F9FBFBAA
Состояние после сложения с ключом: 89371BE66CD5FCD8D4FFA588B6BFA642
Раунд №3
Состояние после SubBytes: A79AAF8E5003B061481606C44E08242C
Состояние после ShiftRows: A703062C5016248E4808AF614E9AB0C4
Состояние после MixColumns: 7A87DCAF309E87C546D3A6BD5D6EF86B
Раундовый ключ: 90973450696CCFFAF2F457330B0FAC99
Состояние после сложения с ключом: EA10E8FF59F2483FB427F18E566154F2
```

Рис. 2.55. Зашифрование слова “Проверка”

Логирование работает только в режиме ECB, и автоматически отключается при открытии файла или использовании другого режима.

4. Квадрат – простенькая реализация единственной атаки на AES, под названием квадрат. Реализована для сокращенной версии AES, для 4 раундов. В стандарте же их 10.

Окно Атака «Квадрат»

Смысл атаки “Квадрат” описан выше.

В этом окне (рисунок 16) можно просмотреть ключи для любого раунда. При открытии этого окна в поля “Реальный ключ” загружается ключ, заданный в настройках. При нажатии кнопок “Up” или ”Down” будет меняться номер раунда, а также поля “Реальный ключ”. Они будут равны тому раундовому ключу, который указан в строке “Номер раунда”. Нас будет интересовать ключ 4 раунда поэтому, лучше поставить значение поля “Номер раунда” в 4.

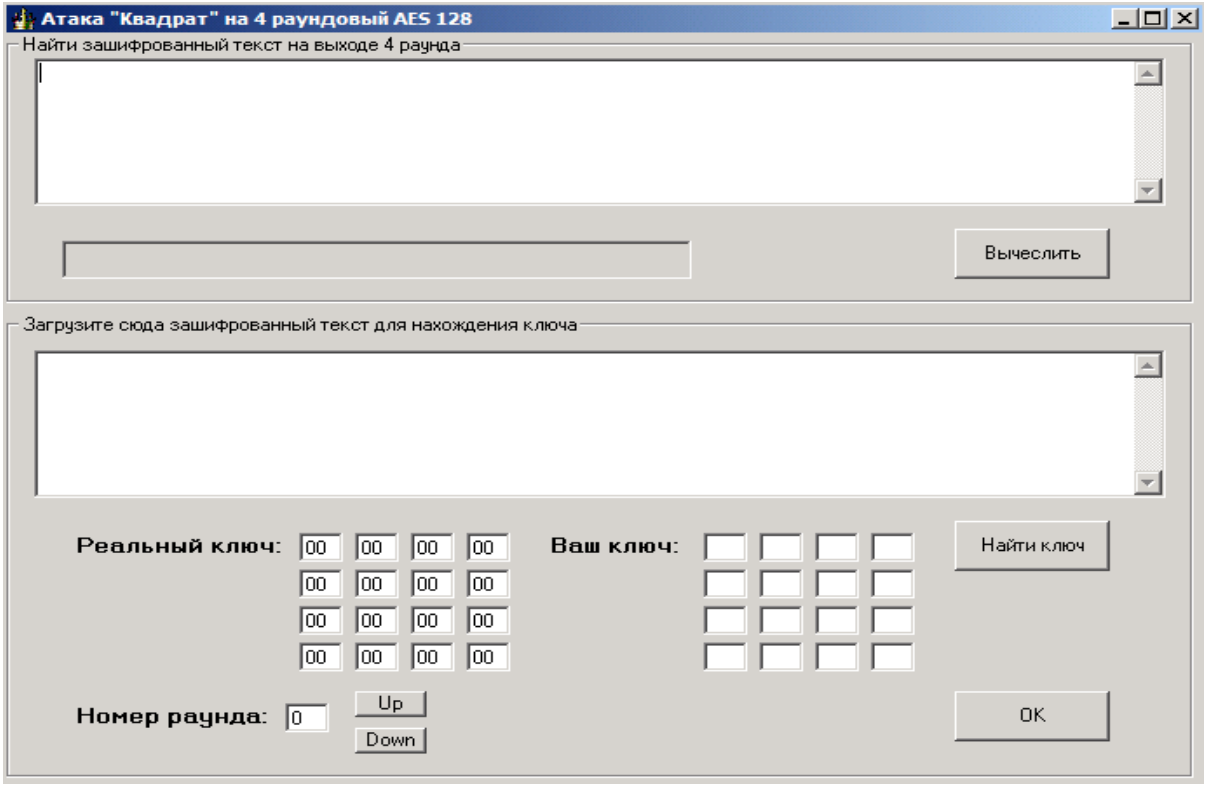


Рис. 2.56. Окно “Квадрат”

При нажатии кнопки “Вычислить” в текстовом поле 1 появляется зашифрованный L набор после 4 раунда. Прогресс бар наглядно показывает время требуемое для его вычисления. Пример L набора, зашифрованного ключом “00”{32}, показан на рисунке 2.57:

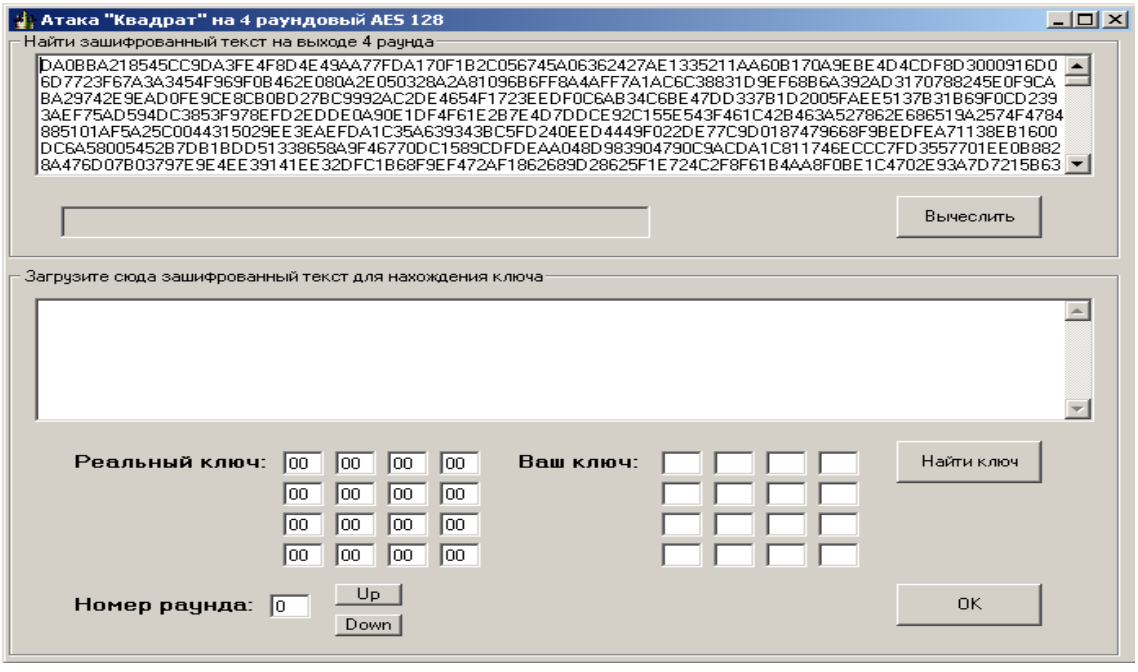


Рис. 2.57. Пример L набора, зашифрованного ключом “00”{32}

Этот текст нужно скопировать и вставить в текстовое поле 2.

При нажатии на кнопку “Найти ключ” начнется вычисление ключа 4 раунда на основе текста в текстовом поле 2. Прогресс бар наглядно показывает время, требуемое для его вычисления.

После вычисления ключа он появится в окошках “Ваш ключ”. Можно сравнить окна “Ваш ключ” и “Реальный ключ” и увидеть, что они практически совпадают (рисунок 2.58).

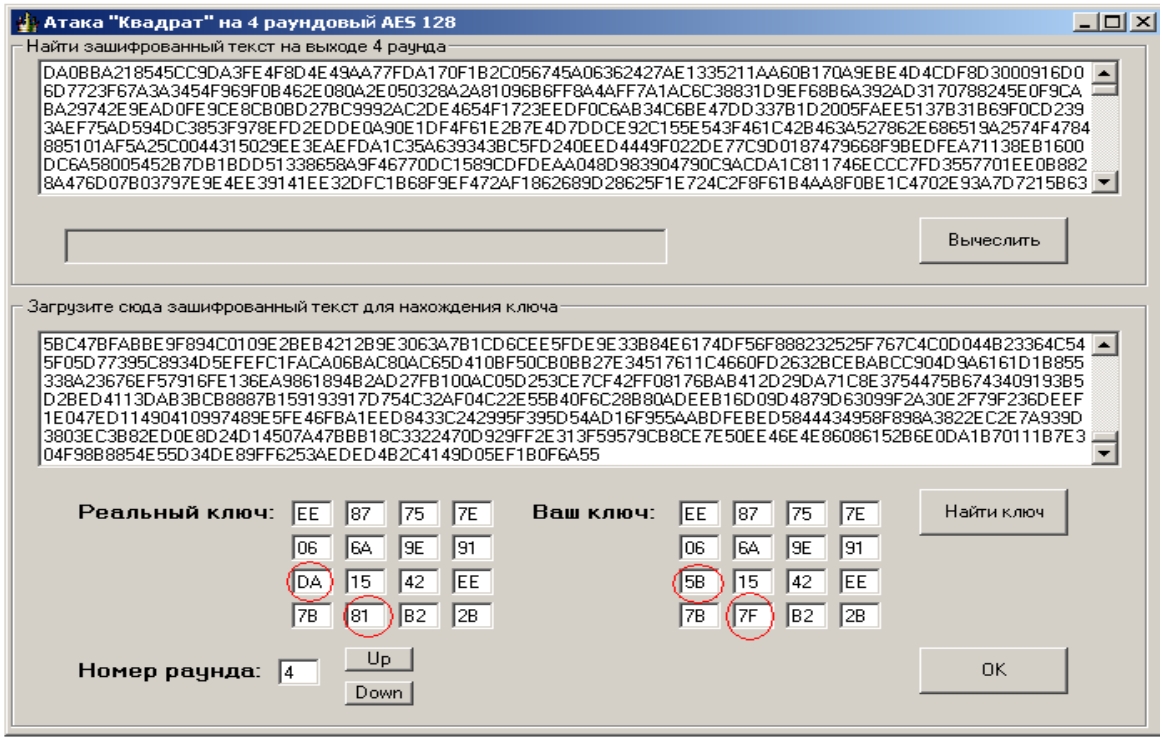


Рис. 2.58. Пример вычисления раундового ключа

Режимы ECB, CBC, CFB, OFB

Для различных ситуаций, встречающихся на практике, разработано значительное количество режимов шифрования.

Режим ECB (Electronic Code Book – режим электронной кодовой книги)

В режиме ECB каждый блок открытого текста заменяется блоком шифротекста. А так как один и тот же блок открытого текста заменяется одним и тем же блоком шифротекста, теоретически возможно создать кодовую книгу блоков открытого текста и соответствующих шифротекстов. Но если размер блока составляет n бит, кодовая книга будет состоять из 2n записей.

Режим ECB - простейший режим шифрования. Все блоки открытого текста шифруются независимо друг от друга. Это важно для шифрованных файлов с произвольным доступом,

например, файлов баз данных. Если база данных зашифрована в режиме ECB, любая запись может быть добавлена, удалена, зашифрована или расшифрована независимо от любой другой записи (при условии, что каждая запись состоит из целого числа блоков шифрования). Кроме того, обработка может быть параллельной: если используются несколько шифровальных процессоров, они могут шифровать или расшифровывать различные блоки независимо друг от друга. Режим ECB показан на рисунке 2.59:

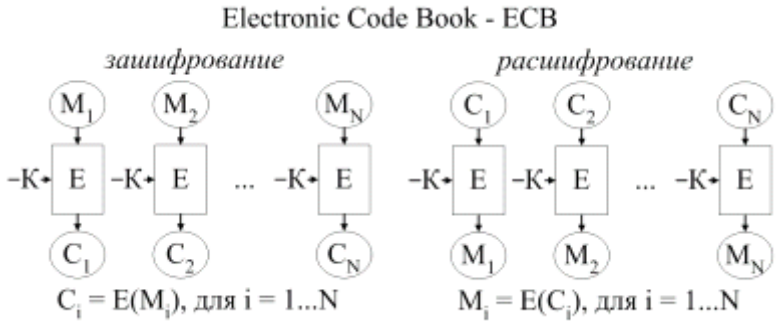


Рис. 2.59. Режим ECB

К недостаткам режима ECB можно отнести то обстоятельство, что если у криптоаналитика есть открытый текст и шифротекст нескольких сообщений, он может, не зная ключа, начать составлять шифровальную книгу. В большинстве реальных ситуаций фрагменты сообщений имеют тенденцию повторяться. В различных сообщениях могут быть одинаковые битовые последовательности, В сообщениях, которые, подобно электронной почте, создаются компьютером, могут быть периодически повторяющиеся структуры. Сообщения могут быть высоко избыточными или содержать длинные строки нулей или пробелов.

К достоинствам режима ECB можно отнести возможность шифрования нескольких сообщений одним ключом без снижения надежности. По существу, каждый, блок можно рассматривать как отдельное сообщение, шифрованное тем же самым ключом. При расшифровании ошибки в символах шифротекста ведут к некорректному расшифрованию соответствующего блока открытого текста, однако не затрагивают остальной открытый текст. Но если бит шифротекста случайно потерян или добавлен, весь последующий шифротекст будет дешифрован некорректно, если только для выравнивания границ блоков не используется какое-нибудь выравнивание по границам блока. Большинство сообщений не делятся точно на n - битовые блоки шифрования – в конце обычно оказывается укороченный блок. Однако режим ECB требует использовать строго n - битовые блоки. Для решения этой проблемы используют дополнение (padding), Чтобы создать полный блок, последний блок дополняют некоторым стандартным шаблоном - нулями, единицами, чередующимися нулями и единицами.

Режим CBC (Ciphertext Block Chaining – режим сцепления блоков шифротекста)

Сцепление добавляет в блочный шифр механизм обратной связи: результаты шифрования предыдущих блоков влияют на шифрование текущего блока, т.е. каждый блок используется для модифицирования шифрования следующего блока. Каждый блок шифротекста зависит не только от шифруемого блока открытого текста, но и от всех предыдущих блоков открытого текста.

В режиме сцепления блоков шифротекста перед шифрованием над открытым текстом и предыдущим блоком шифротекста выполняется операция XOR. Процесс шифрования в режиме CBC. Когда блок открытого текста зашифрован, полученный шифротекст сохраняется в регистре обратной связи. Следующий блок открытого текста перед шифрованием подвергается операции XOR с содержимым регистра обратной связи. Результат операции XOR используется как входные данные для следующего этапа процедуры шифрования. Полученный шифротекст снова сохраняется в регистре обратной связи, чтобы подвергнуться операции XOR вместе со следующим блоком открытого текста, и так до конца сообщения. Шифрование каждого блока зависит от всех предыдущих блоков.

Расшифрование выполняется в обратном порядке. Блок шифротекста расшифровывается обычным путем, но сохраняется в регистре обратной связи. Затем следующий блок расшифровывается и подвергается операции XOR с содержимым регистра обратной связи. Теперь следующий блок шифротекста сохраняется в регистре обратной связи и т.д. до конца сообщения.

На рисунке 2.60 показан режим CBC (IV – синхропосылка):

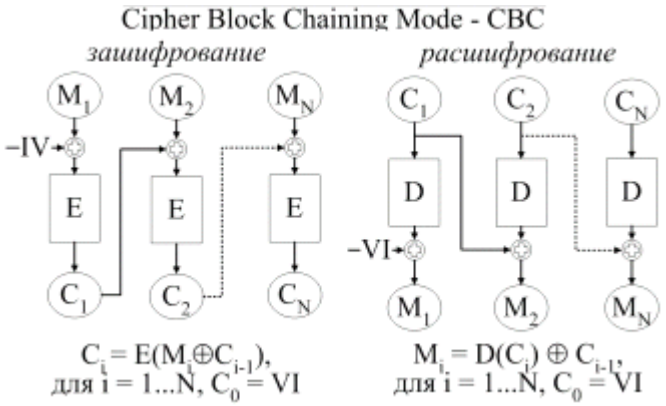


Рис. 2.60. Режим CBC

При шифровании в режиме CBC одинаковые блоки открытого текста превращаются в различающиеся друг от друга блоки шифротекста только в том случае, если различались какие-то предшествующие блоки открытого текста. Однако при шифровании двух 48 идентичных сообщений создается один и тот же шифротекст. Хуже того, два одинаково начинающихся сообщения будут шифроваться одинаково вплоть до первого различия.

Чтобы избежать этого, можно зашифровать в первом блоке какие-то произвольные данные. Этот блок случайных данных называют вектором инициализации (ВИ) (Initialization Vector, IV, русский термин - синхропосылка), инициализирующей переменной или начальным значением сцепления. Вектор ВИ не имеет какого-то смыслового значения, он используется только для того, чтобы сделать каждое сообщение уникальным. Когда получатель расшифровывает этот блок, он использует его только для заполнения регистра обратной связи. В качестве вектора ВИ удобно использовать метку времени, либо какие-то случайные биты.

Если используется вектор инициализации, сообщения с идентичным открытым текстом после шифрования превращаются в сообщения с разными шифротекстами. Следовательно, злоумышленник не может попытаться повторить блок, и создание шифровальной книги затруднится. Хотя для каждого сообщения, шифруемого одним и тем же ключом, рекомендуется выбирать уникальный вектор ВИ, это требование необязательное. Вектор ВИ не обязательно хранить в секрете, его можно передавать открыто - вместе с шифротекстом.

Режим CFB (Ciphertext Feedback – обратная связь по шифротексту)

В режиме CBC начать шифрование до поступления полного блока данных невозможно. Для некоторых сетевых приложений это создает проблемы. Например, в защищенном сетевом окружении терминал должен иметь возможность передавать хосту каждый символ сразу после ввода. Если же данные нужно обрабатывать блоками в несколько байт, режим CBC просто не работает.

На рисунке 2.61 показан режим CFB (IV – синхропосылка):

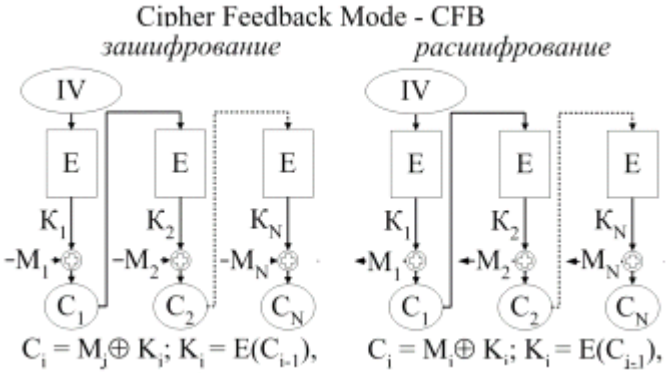


Рис. 2.61. Режим CFB

Как и в режиме CBC, первоначально очередь заполнена вектором инициализации ВИ. Очередь шифруется, затем выполняется операция XOR над *n* старшими (крайними левыми) битами результата и первым *n* -битовым символом открытого текста. В результате появляется первый *n* -битовый символ шифротекста. Теперь этот символ можно передать. Кроме того, полученные *n* битов попадают в очередь на место *n* младших битов, а все

остальные биты сдвигаются на *n* позиций влево. Предыдущие *n* старших битов отбрасываются. Затем точно также шифруется следующие *n* битов открытого текста. Расшифрование выполняется в обратном порядке. Обе стороны - шифрующая и расшифровывающая - использует блочный алгоритм в режиме шифрования. Как и режим CBC, режим CFB сцепляет символы открытого текста с тем, чтобы шифротекст зависел от всего предыдущего открытого текста.

Для инициализации процесса шифрования в режиме CFB в качестве входного блока алгоритма можно использовать вектор инициализации ВИ. Как и в режиме CBC, хранить в тайне вектор ВИ не нужно. Однако вектор ВИ должен быть уникальным. (В отличие от режима CBC, где уникальность вектора ВИ необязательна, хотя и желательна). Если вектор ВИ в режиме CFB не уникален, криптоаналитик может восстановить соответствующий открытый текст. Вектор инициализации должен меняться в каждом сообщении. Например, вектором ВИ может служить порядковый номер, возрастающий в каждом новом сообщении и не повторяющийся все время жизни ключа. Если данные шифруются с целью последующего хранения, вектор ВИ может быть функцией индекса, используемого для поиска данных.

Режим OFB (Output Feedback – режим обратной связи по выходу)

Режим OFB представляет собой метод использования блочного шифра в качестве синхронного потокового шифра. Этот режим подобен режиму CFB.

Блочный алгоритм работает в режиме шифрования как на шифрующей, так и на расшифровывающей сторонах. Такую обратную связь иногда называют внутренней, поскольку механизм обратной связи не зависит ни от потока открытого текста, ни от потока шифротекста.

К достоинствам режима OFB относится то, что большую часть работы можно выполнить оффлайново, даже когда открытого текста сообщения еще вовсе не существует. Когда, наконец, сообщение поступает, для создания шифротекста над сообщением и выходом алгоритма необходимо выполнить операцию XOR.

В сдвиговый регистр OFB сначала надо загрузить вектор ВИ. Вектор должен быть уникальным, но сохранять его в тайне не обязательно.

На рисунке 7.62 показан режим OFB (IV – синхропосылка):

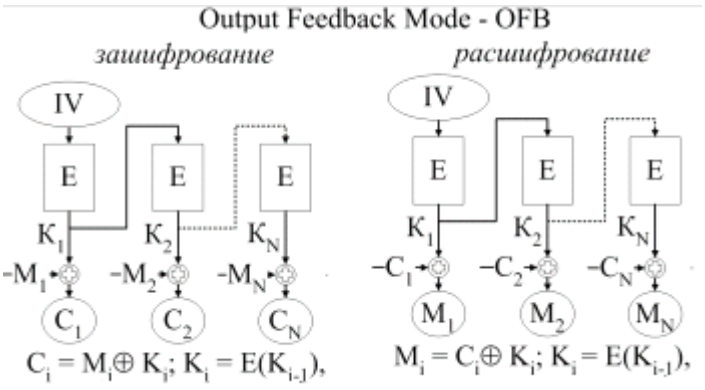


Рис. 2.62. Режим OFB

Анализ режима OFB показывает, что OFB целесообразно использовать только если разрядность обратной связи совпадает с размером блока.

В режиме OFB над гаммой и текстом выполняется операция XOR. Эта гамма, в конце концов, повторяется. Существенно, чтобы она не повторялась для одного и того же ключа, в противном случае секретность не обеспечивается ничем.

Порядок выполнения работы

Ознакомится с теорией по стандарту AES (из чего состоит раунд шифрования, как работают раундовые преобразования, с необходимым математическим аппаратом).

Взять у преподавателя задание, состоящие из:

- уникального ключа для шифрования/дешифрования текстовых сообщений
- набора файлов разного размера для исследования стандарта
- файла для исследования режимов шифрования
- вектора инициализации
- файла для проведения атаки «Квадрат»

Исследование стандарт шифрования

1. Открыть окно «Настройки» (AES>Настройки>), установить переключатель в положение «Свой ключ» и ввести посимвольное значение ключа (32 символа, 16 ячеек). Нажать «ОК».
2. В главном окне поставить переключатель в положение ECB.
3. Открыть файл из папки программы (Файл >Открыть)
4. Зашифровать и расшифровать его. (AES > Зашифровать (Расшифровать)).
5. Проделать пункты 3.2-3.4 для оставшихся трех файлов.
6. По полученным результатам заполнить таблицу

Размер файла				
Время зашифрования				
Время расшифрования				

7. Построить графики зависимости шифрования \ расшифрования от времени.

8. По полученным данным оценить примерное время взлома стандарта AES с помощью всего перебора ключей. Построить график.

\Пример:

На расшифрование 5кВ (5120 байт) текста на Athlon 2600+ тратится 9с. Поэтому один блок (16 байт) программа расшифровывает за 9/(5120/16) = (9*16)/5120 = 28 мс. Таким образом, на перебор 1 ключа, а это 128 бит, потребуется 28 мс.

За 1 секунду программа способна перебрать 4571 бит или примерно 35 вариантов ключей.

За год непрерывной работы программа переберет (35*60*60*24*365) = 1 103 760 000 ключей.

Если поделить 2^{128} на 1 103 760 000 , то получим $3.08*10^{29}$ лет непрерывной работы программы.

Исследование быстродействия различных режимов шифрования

1. Поставить переключатель в режим CBC.
2. Открыть файл из папки программы (Файл >Открыть)
3. Зашифровать и расшифровать его (AES > Зашифровать (Расшифровать))
4. Повторить пункт 4.2-4.3 для оставшихся трех файлов (10, 15, 30 кБ)
5. Поставить переключатель в режим CFB.
6. Повторить пункты 4.2 – 4.3.
7. Поставить переключатель в режим OFB.
8. Повторить пункты 4.2 – 4.3
9. По полученным результатам заполнить таблицу:

Размер файла				
ECB время заш/расш				

CBC время заш/расш				
CFB время заш/расш				
OFB время заш/расш				

10. Построить на одном графике зависимость шифрования \ расшифрования от времени.

11. Для каждого режима загрузить еще раз файл (любой). Зашифровать его. В текстовом поле 2 изменить какой-нибудь символ. Нажать AES > Расшифровать.

Исследование свойств различных режимов шифрования

Загрузить файл «Свойства режимов».

Поочередно зашифровать текстовое сообщение во всех четырех режимах.

Сравнить полученные шифротексты и сделать выводы.

Исследование пораундовой работы алгоритма

1 Открыть любой из предоставленных для работы файлов

2. Включить функцию логирования (AES>Настройки>Включить логирование>Да).

3. В главном окне поставить переключатель в положение ECB.

4. Зашифровать и расшифровать.

5. Открыть с помощью текстового редактора файлы Encrypt.txt и Decrypt.txt из папки программы

6. Проанализировать полученный результат.

Знакомство с атакой «Квадрат»

1. В главном окне в текстовое поле 1 загрузить файл «Атака Квадрат» (Файл>Открыть)

2. Открыть окно Атака «Квадрат» (AES > Квадрат)

3. С помощью переключателя “Номер раунда” установить 4 раундовый ключ.

4. Нажать кнопку “вычислить”.

5. Поместить текст из поля 1 в поле 2.

6. Нажать кнопку найти ключ.

7. Сравнить значения текстовых полей “Ваш ключ” и значения текстовых полей “Реальный ключ”.

8. Сделать выводы.

Содержание отчета

1. Цель работы.
2. Описание действий по каждому пункту.
3. Результаты проделанной работы (таблицы, графики, расчетная часть).
4. Выводы.

Контрольные вопросы

1. Что такое симметричное шифрование?
2. Что представляет собой стандарт AES (длина ключа, размер входного блока)?
3. Какой алгоритм выбран в качестве стандарта AES?
4. Что собой представляет архитиктура данного стандарта?
5. Из чего состоит один раунд?
6. Сколько раундов шифрования предусмотрено стандартом?
7. Что быстрее шифрование или разшифрование? Почему?
8. Какие режимы шифрования применяются в стандарте AES?
9. Какие режимы быстрее при разшифровании? Почему?
10. Какие режимы лучше восстанавливают зашифрованную информацию при ошибке в одном символе? Почему?
11. С какой целью используется синхропосылка или вектор инициализации?
12. Что представляет собой атака Квадрат? Какие ее особенности?

1. Ознакомится с теорией по стандарту AES (из чего состоит раунд шифрования, как работают раундовые преобразования, с необходимым математическим аппаратом).

2. Взять у преподавателя задание, состоящие из:

- уникального ключа для шифрования/дешифрования текстовых сообщений
- набора файлов разного размера для исследования стандарта
- файла для исследования режимов шифрования
- вектора инициализации
- файла для проведения атаки «Квадрат»

3. Исследование стандарт шифрования

3.1. Открыть окно «Настройки» (AES>Настройки>), установить переключатель в положение «Свой ключ» и ввести посимвольное значение ключа (32 символа, 16 ячеек). Нажать «ОК».

3.2. В главном окне поставить переключатель в положение ECB.

3.3. Открыть файл из папки программы (Файл >Открыть)

3.4. Зашифровать и расшифровать его. (AES > Зашифровать (Расшифровать)).

3.5. Проделать пункты 3.2-3.4 для оставшихся трех файлов.

3.5. По полученным результатам заполнить таблицу

Размер файла				
Время зашифрования				
Время расшифрования				

3.6. Построить графики зависимости шифрования \ расшифрования от времени.

3.7. По полученным данным оценить примерное время взлома стандарта AES с помощью всего перебора ключей. Построить график.

Пример:

На расшифрование 5кВ (5120 байт) текста на Athlon 2600+ тратится 9с. Поэтому один блок (16 байт) программа расшифровывает за 9/(5120/16) = (9*16)/5120 = 28 мс. Таким образом, на перебор 1 ключа, а это 128 бит, потребуется 28 мс.

За 1 секунду программа способна перебрать 4571 бит или примерно 35 вариантов ключей.

За год непрерывной работы программа переберет (35*60*60*24*365) = 1 103 760 000 ключей.

Если поделить 2^{128} на 1 103 760 000 , то получим $3.08*10^{29}$ лет непрерывной работы программы.

3.8. Сделать выводы

Исследование быстродействия различных режимов шифрования

1. Поставить переключатель в режим CBC.

2. Открыть файл из папки программы (Файл >Открыть)

3. Зашифровать и расшифровать его (AES > Зашифровать (Расшифровать))

4. Повторить пункт 4.2-4.3 для оставшихся трех файлов (10, 15, 30 кБ)

5. Поставить переключатель в режим CFB.

6. Повторить пункты 4.2 – 4.3.

7. Поставить переключатель в режим OFB.

8. Повторить пункты 4.2 – 4.3

9. По полученным результатам заполнить таблицу:

Размер файла				
ECB время заш/расш				
CBC время заш/расш				
CFB время заш/расш				
OFB время заш/расш				

10. Построить на одном графике зависимость шифрования \ расшифрования от времени.

11. Для каждого режима загрузить еще раз файл (любой). Зашифровать его. В текстовом поле 2 изменить какой-нибудь символ. Нажать AES > Расшифровать.

12. Сделать выводы.

Исследование свойств различных режимов шифрования

Загрузить файл «Свойства режимов».

Поочередно зашифровать текстовое сообщение во всех четырех режимах.

Сравнить полученные шифротексты и сделать выводы.

Исследование пораундовой работы алгоритма

1 Открыть любой из предоставленных для работы файлов

2. Включить функцию логирования (AES>Настройки>Включить логирование>Да).

3. В главном окне поставить переключатель в положение ECB.

4. Зашифровать и расшифровать.

5. Открыть с помощью текстового редактора файлы Encrypt.txt и Decrypt.txt из папки программы

6. Проанализировать полученный результат.

Знакомство с атакой «Квадрат»

1. В главном окне в текстовое поле 1 загрузить файл «Атака Квадрат» (Файл>Открыть)

2. Открыть окно Атака «Квадрат» (AES > Квадрат)

3. С помощью переключателя “Номер раунда” установить 4 раундовый ключ.

4. Нажать кнопку “вычислить”.

5. Поместить текст из поля 1 в поле 2.

6. Нажать кнопку найти ключ.

7. Сравнить значения текстовых полей “Ваш ключ” и значения текстовых полей “Реальный ключ”.

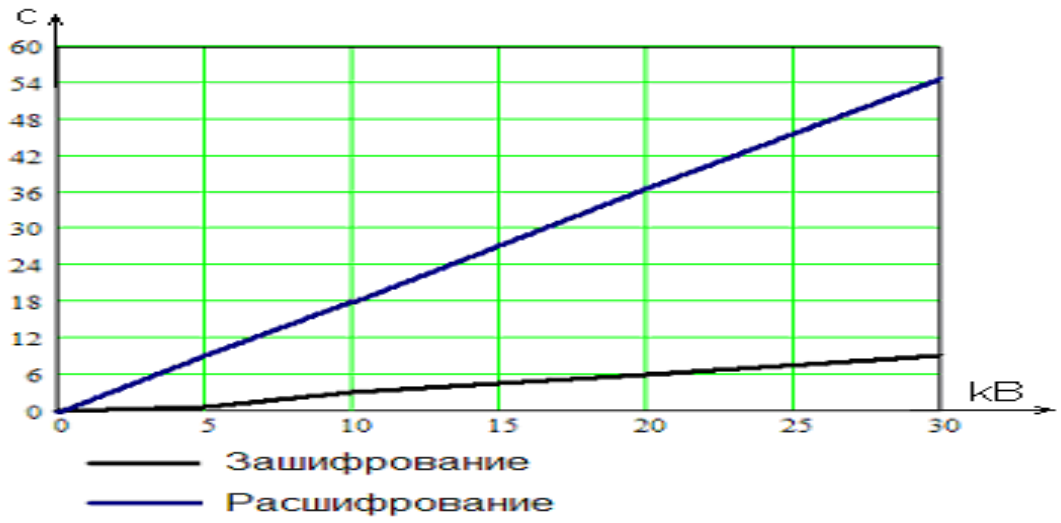
8. Сделать выводы.

Содержание отчета

1. Цель работы.
2. Описание действий по каждому пункту.
3. Результаты проделанной работы (таблицы, графики, расчетная часть).
4. Выводы.

Исследование стандарт шифрования

Размер файла	5kB	10kB	20kB	30kB
Время зашифрования, с	1	3	6	9
Время расшифрования, с	9	18	37	55



2.2.3. Практикум по поточным шифрам

Потоковый шифр RC4

Целью данной лабораторной работы является изучение алгоритма RC4, рассмотрение его слабых и сильных сторон, определение в каких продуктах и каким образом он используется, а так же знакомство с программной реализацией на языке C++, с использованием программного обеспечения Borland C++ Builder 6.

RC4 (англ. Rivest Cipher 4 или англ. Ron’s Code, также известен как ARCFOUR или ARC4 (англ. Alleged RC4)) — это потоковый шифр, широко применяющийся в различных системах защиты информации в компьютерных сетях (например, в протоколах SSL и TLS, алгоритме безопасности беспроводных сетей WEP, для шифрования паролей).

Шифр разработан компанией RSA Security и для его использования требуется лицензия.

Алгоритм RC4 строится, как и любой потоковый шифр на основе параметризованного ключом генератора псевдослучайных битов с равномерным распределением. Длина ключа обычно составляет от 5 до 64 байт. Максимальная длина ключа 256 байт.

Основные преимущества шифра — высокая скорость работы и переменный размер ключа. RC4 довольно уязвим, если используются не случайные или связанные ключи, один ключевой поток используется дважды. Эти факторы, а также способ использования могут сделать криптосистему небезопасной (например, WEP).

Потоковый шифр RC4 был создан Роном Ривестом из RSA Security в 1987 году. Хотя официально сокращение обозначает Rivest Cipher 4, его часто считают сокращением от Ron’s Code.

Шифр являлся коммерческой тайной, но в сентябре 1994 года его описание было анонимно отправлено в рассылку Cypherpunks. Вскоре описание RC4 было опубликовано в ньюс-группе sci.crypt. Именно оттуда исходный код попал на множество сайтов в сети Интернет. Опубликованный шифр давал те же шифротексты на выходе, какие давал подлинный RC4. По-видимому, данный текст был получен в результате анализа исполняемого кода. Опубликованный шифр совместим с имеющимися продуктами, использующими RC4, а некоторые участники телеконференции, имевшие, по их словам, доступ к исходному коду RC4, подтвердили идентичность алгоритмов при различиях в обозначениях и структуре программы.

Поскольку данный алгоритм известен, он более не является коммерческой тайной. Однако, название «RC4» является торговой маркой компании RSA. Поэтому иногда шифр

называют «ARCFOUR» или «ARC4» (имея ввиду Alleged RC4 — предполагаемый RC4, поскольку RSA официально не опубликовала алгоритм), чтобы избежать возможных претензий со стороны владельца торговой марки.

Шифр RC4 применяется в некоторых широко распространённых стандартах и протоколах шифрования таких, как WEP, WPA и TLS.

Главными факторами, способствовавшими широкому применению RC4, были простота его аппаратной и программной реализации, а также высокая скорость работы алгоритма в обоих случаях.

В США длина ключа для использования внутри страны рекомендуется равной 128 битов, но соглашение, заключённое между Software Publishers Association (SPA) и правительством США даёт RC4 специальный статус, который означает, что разрешено экспортировать шифры длиной ключа до 40 бит. 56-битные ключи разрешено использовать заграничным отделениям американских компаний.

Описание алгоритма

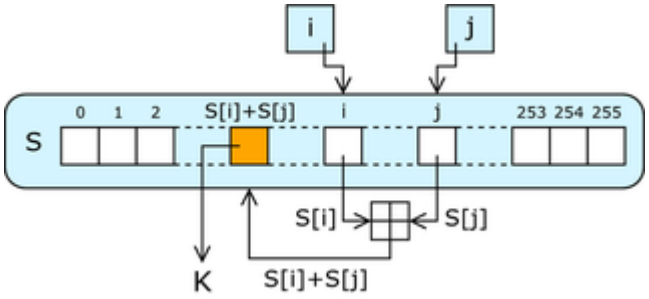


Рис. 2.62. Генератор ключевого потока RC4

Ядро алгоритма состоит из функции генерации ключевого потока. Эта функция генерирует последовательность битов (ki), которая затем объединяется с открытым текстом (mi) посредством суммирования по модулю два. Так получается шифрограмма (ci):

$$c_i = m_i \oplus k_i.$$

Расшифровка заключается в регенерации этого ключевого потока (ki) и сложении его и шифрограммы (ci) по модулю два. В силу свойств суммирования по модулю два на выходе мы получим исходный незашифрованный текст (mi):

$$m_i = c_i \oplus k_i = (m_i \oplus k_i) \oplus k_i.$$

Другая главная часть алгоритма — функция инициализации, которая использует ключ переменной длины для создания начального состояния генератора ключевого потока.

RC4 — фактически класс алгоритмов, определяемых размером его блока. Этот параметр n является размером слова для алгоритма. Обычно, n = 8, но в целях анализа можно уменьшить его. Однако для повышения безопасности необходимо увеличить эту величину.

Внутреннее состояние RC4 представляется в виде массива слов размером 2n и двух счетчиков, каждый размером в одно слово. Массив известен как S-бокс, и далее будет обозначаться как S. Он всегда содержит перестановку 2n возможных значений слова. Два счетчика обозначены через i и j.

Алгоритм инициализации RC4 приведен ниже. Этот алгоритм также называется алгоритмом ключевого расписания (Key-Scheduling Algorithm or KSA). Этот алгоритм использует ключ, сохраненный в Key, и имеющий длину l байт. Инициализация начинается с заполнения массива S, далее этот массив перемешивается путем перестановок определяемых ключом. Так как только одно действие выполняется над S, то должно выполняться утверждение, что S всегда содержит все значения кодового слова.

Начальное заполнение массива:

```
for i = 0 to 2n − 1
  S[i] = i
```

Скремблирование:

```
  j = 0
for i = 0 to 2n − 1
  j = (j + S[i] + Key[i mod l]) mod 2n
      Перестановка (S[i], S[j])
```

Генератор ключевого потока RC4 переставляет значения, хранящиеся в S, и каждый раз выбирает различное значение из S в качестве результата. В одном цикле RC4 определяется одно n-битное слово K из ключевого потока, которое в последующем суммируется с исходным текстом для получения зашифрованного текста. Эта часть алгоритма называется генератором псевдослучайной последовательности (Pseudo-Random Generation Algorithm or PRGA).

Инициализация:

```
i = 0
j = 0
```

Цикл генерации:

```
i = (i + 1) mod 2n
j = (j + S[i]) mod 2n
  Перестановка (S[i], S[j])
  Результат: K = S[(S[i] + S[j]) mod 2n]
```

Безопасность

В отличие от современных шифров (таких, как в eSTREAM), RC4 не использует отдельной оказии (nonce) наряду с ключом. Это значит, что если один ключ должен

использоваться в течение долгого времени для шифрования нескольких потоков, сама криптосистема, использующая RC4, должна комбинировать оказию и долгосрочный ключ для получения потокового ключа для RC4. Один из возможных выходов — генерировать новый ключ для RC4 с помощью хэш-функции от долгосрочного ключа и оказии. Однако, многие приложения, использующие RC4, просто конкатенируют ключ и оказию. Из-за этого и слабого расписания ключей, используемого в RC4, приложение может стать уязвимым.

Здесь будут рассмотрены некоторые атаки на шифр и методы защиты от них.

Манипуляция битами

Шифр RC4 крайне уязвим к манипуляции битами, если он не реализован верным образом. И поэтому он был признан устаревшим многими софтверными компаниями, такими как Microsoft. Например, в .NET Framework от Microsoft отсутствует реализация RC4.

Исследования Руза и восстановление ключа из перестановки

В 1995 году Андрю Руз (Andrew Roos) экспериментально пронаблюдал, что первый байт ключевого потока коррелирован с первыми тремя байтами ключа, а первые несколько байт перестановки после алгоритма расписания ключей (KSA) коррелированны с некоторой линейной комбинацией байт ключа. Эти смещения не были доказаны до 2007 года, когда Пол, Рафи и Мэйтрэ доказали коррелированность ключа и ключевого потока. Также Пол и Мэйтрэ доказали коррелированность перестановки и ключа. Последняя работа также использует коррелированность ключа и перестановки для того, чтобы создать первый алгоритм полного восстановления ключа из последней перестановки после KSA, не делая предположений о ключе и векторе инициализации(V or Initial Vector). Этот алгоритм имеет постоянную вероятность успеха в зависимости от времени, которая соответствует квадратному корню из сложности полного перебора. Позднее было сделано много работ о восстановлении ключа из внутреннего состояния RC4.

Атака Флурера, Мантина и Шамира (ФМШ)

В 2001 году, Флурер, Мантин и Шамир опубликовали работу об уязвимости ключевого расписания RC4. Они показали, что среди всех возможных ключей, первые несколько байт ключевого потока являются совсем неслучайными. Из этих байт можно с высокой вероятностью получить информацию о используемом шифром ключе. И если долговременный ключ и оказия (nonce) просто конкатенируются для создания ключа шифра RC4, то этот долговременный ключ может быть получен с помощью анализа достаточно большого количества сообщений, зашифрованных с использованием данного ключа. Эта

уязвимость и некоторые связанные с ней эффекты были использованы при взломе шифрования WEP в беспроводных сетях стандарта IEEE 802.11. Это показало необходимость скорейшей замены WEP, что повлекло за собой разработку нового стандарта безопасности беспроводных сетей WPA.

Криптосистему можно сделать невосприимчивой к этой атаке, если отбрасывать начало ключевого потока. Таким образом, модифицированный алгоритм называется «RC4-drop[n]», где n — количество байт из начала ключевого потока, которые следует отбросить. Рекомендовано использовать n = 768, консервативная оценка составляет n = 3072.

Атака Кляйна

В 2005 году Андреас Кляйн представил анализ шифра RC4, в котором он указал на сильную коррелированность ключа и ключевого потока RC4. Кляйн проанализировал атаки на первом раунде (подобные атаке ФМШ), на втором раунде и возможные их улучшения. Он также предложил некоторые изменения алгоритма для усиления стойкости шифра. В частности, он утверждает, что если поменять направление цикла на обратное в алгоритме ключевого расписания, то можно сделать шифр более стойким к атакам типа ФМШ.

Комбинаторная проблема

В 2001 году Ади Шамир и Ицхак Мантин первыми поставили комбинаторную проблему, связанную с количеством всевозможных входных и выходных данных шифра RC4. Если из всевозможных 256 элементов внутреннего состояния шифра известно x элементов из состояния ($x \leq 256$), то, если предположить, что остальные элементы нулевые, максимальное количество элементов, которые могут быть получены детерминированным алгоритмом за следующие 256 раундов также равно x. В 2004 году это предположение было доказано Сорадюти Полом (Souradyuti Paul) и Бартом Прэнилом (Bart Preneel).

Программная реализация

Работа многих поточных шифров основана на линейных регистрах сдвига с обратной связью (LFSR). Это позволяет достичь высокой эффективности реализаций шифра в виде ИС. Но затрудняет программную реализацию таких шифров. Поскольку шифр RC4 не использует LFSR и основан на байтовых операциях, его удобно реализовывать программно. Типичная реализация выполняет от 8 до 16 машинных команд на каждый байт текста, поэтому программная реализация шифра должна работать очень быстро.

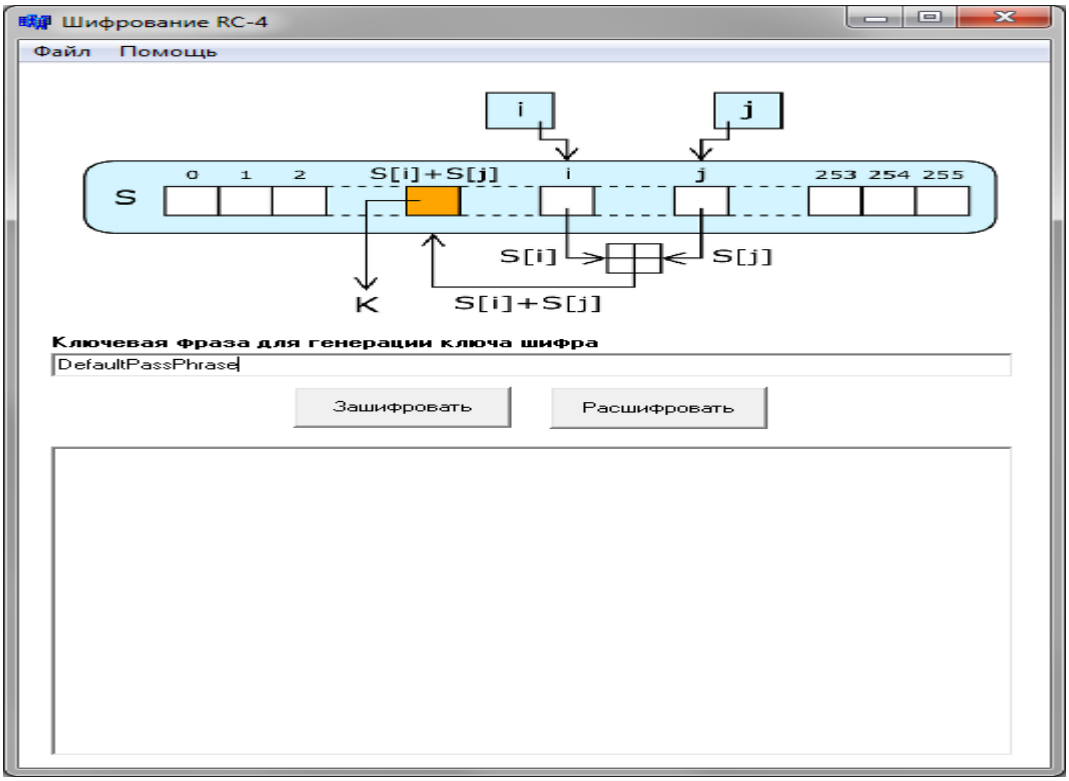


Рис. 2.63. Общий вид программы

Файл

Содержит:

Зашифровать – выбор файла, который необходимо зашифровать. При нажатии открывается интерфейс, позволяющий выбрать файл в проводнике.

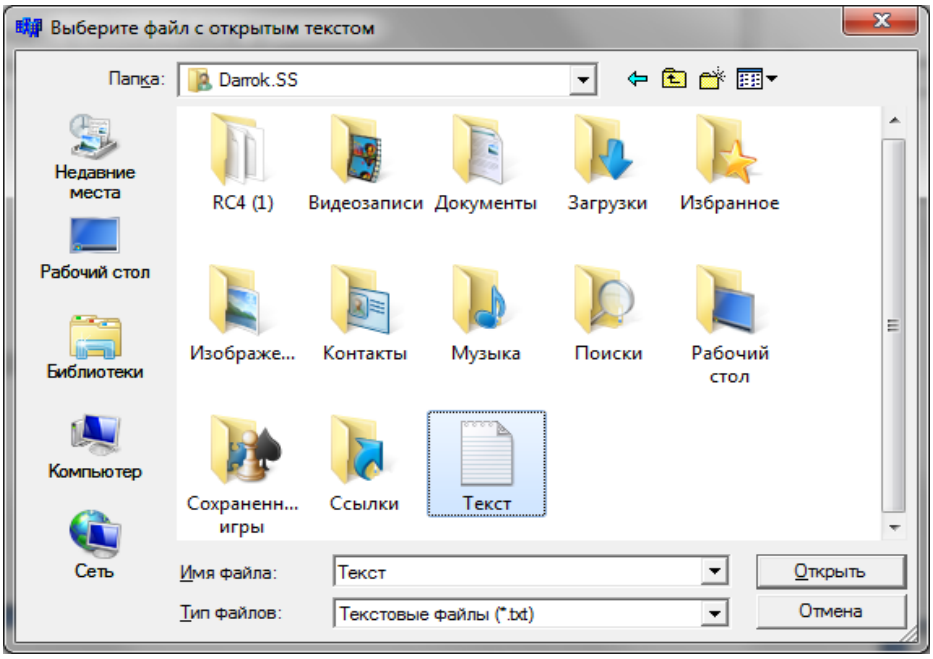


Рис. 2.64. Выбор текстового файла для шифрования

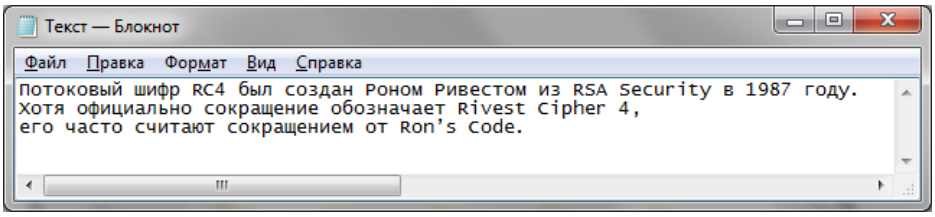


Рис. 2.65. Шифруемый текст

Расшифровать – выбор файла, который необходимо расшифровать. При нажатии открывается интерфейс, позволяющий выбрать файл в проводнике.

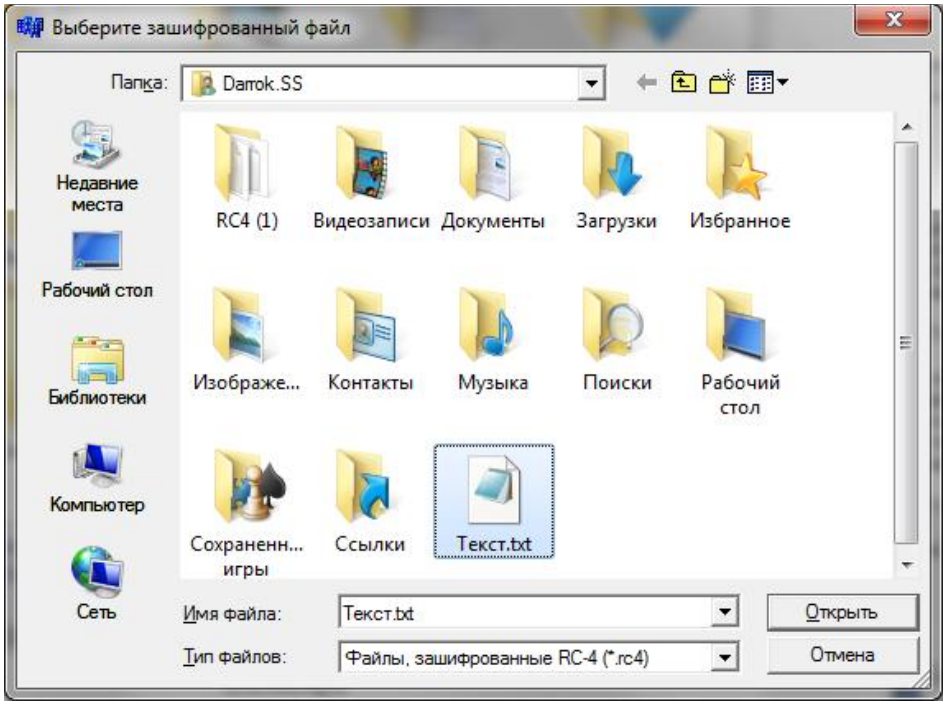


Рис. 2.66. Выбор текстового файла для расшифрования

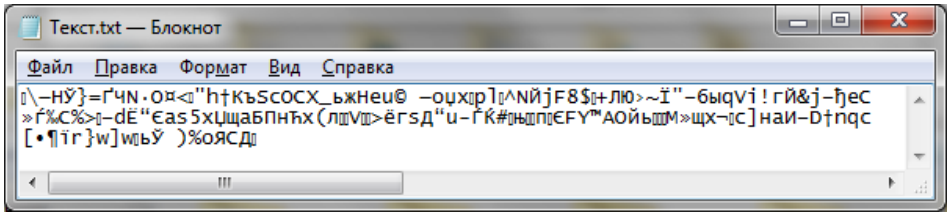


Рис. 2.67. Зашифрованный текст

Выход – выход из программы.

Помощь

Содержит:

Алгоритм – отображает теоретическую информацию об алгоритме, реализуемом в лабораторной работе.

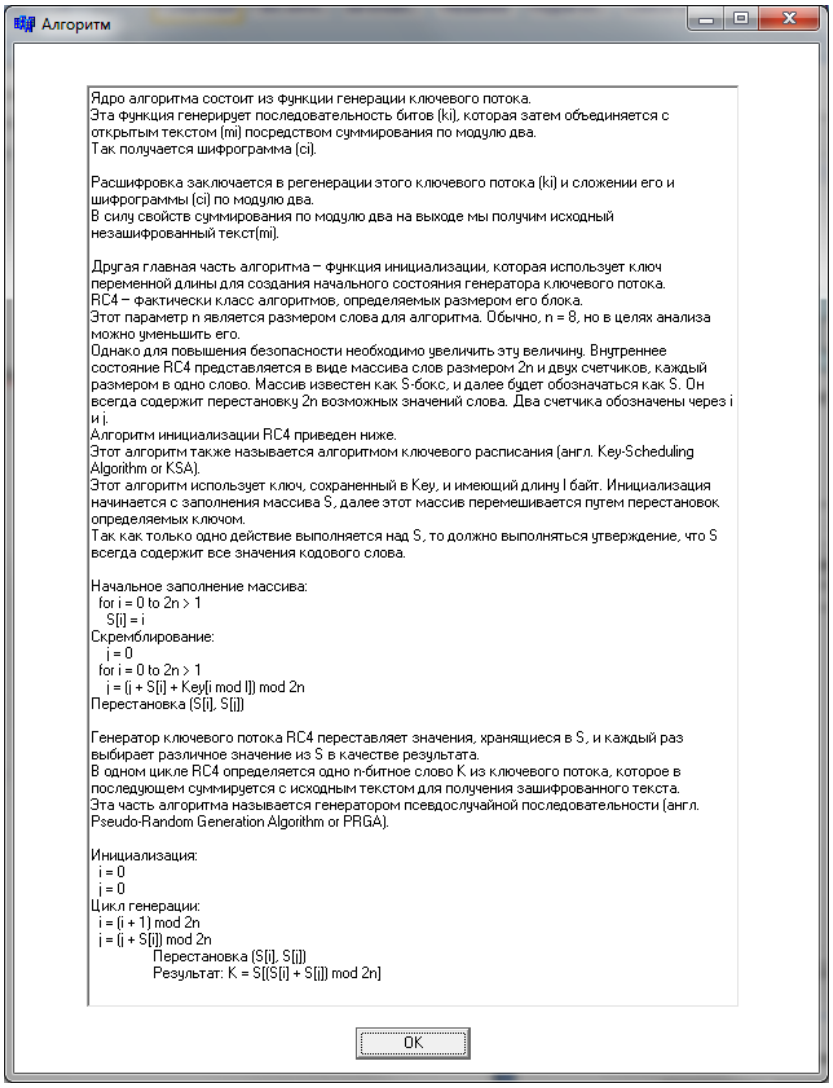


Рис. 2.68. Описание работы алгоритма

Автор – отображает информацию об авторе лабораторной работы и его научного руководителя.

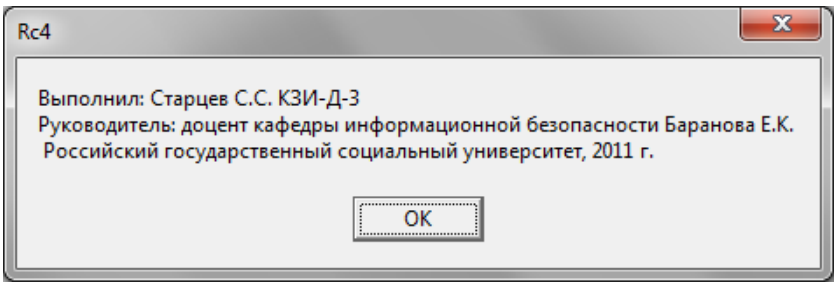


Рис. 2.69. Информация об авторе и руководителе

Поле “Ключ”

В это поле вводятся символы, являющиеся Ключом, который необходим для реализации шифрования методом, используемым в алгоритме.

Окно процесса шифрования

Отображает процесс шифрования и расшифрования, а именно: инициализацию матрицы, выполнение перестановок, считывание блока, количество обработанных байт и запись блока.

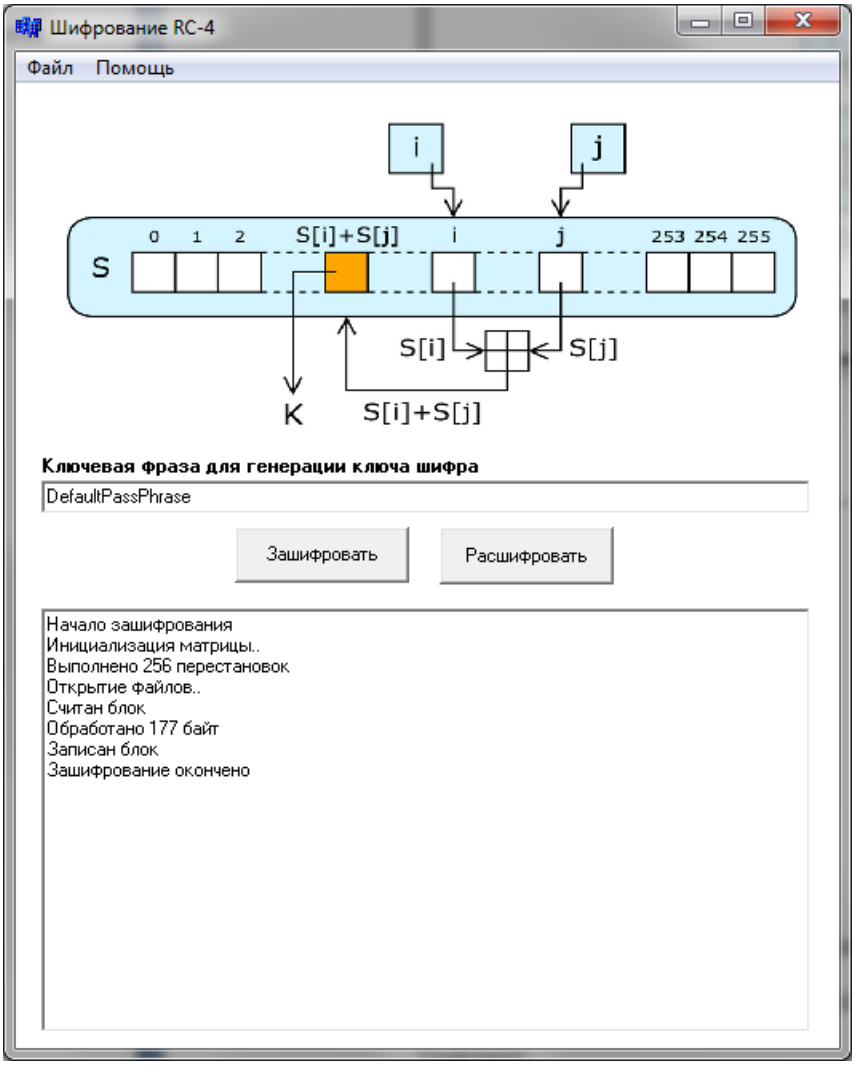


Рис. 2.70. Отображение процесса шифрования

Последовательность действий

Выбрать входной и выходный файлы, ввести ключ, нажать «Зашифровать/Расшифровать». Т.к. алгоритм симметричный, то процессы шифрования и расшифрования аналогичны.

Была поставлена и, в ходе подготовки данной лабораторной работы, достигнута цель: разработка программы, реализующей алгоритм шифрования RC4, а так же рассмотрены и изучены особенности этого алгоритма.

Потоковый шифр RC4 был разработан Роном Ривестом в 1987 году. Этот шифр позволяет использовать ключи размером от 8 до 2048 бит (с шагом 8). В RC4 для

зашифрования и расшифрования применяются одни и те же действия: генерируется гамма, которая накладывается на шифруемое сообщение путем сложения по модулю 2 (операция XOR).

RC4 применяется в таких продуктах, как Microsoft Office, Lotus Notes, Adobe Acrobat и др.

Алгоритм RC4 является собственностью компании RSA Data Security, Inc. Его описание никогда не было опубликовано и предоставлялось партнерам только после подписания соглашения о неразглашении. Однако в сентябре 1994 года в списке рассылки Cipherpunks (Шифропанки) кто-то анонимно опубликовал алгоритм шифрования, который на всех известных тестовых значениях совпадал с RC4. С тех пор сам алгоритм перестал быть секретом, но название RC4 остается торговой маркой. То есть, чтобы получить право заявлять, что в коммерческом программном продукте используется RC4, необходимо приобрести лицензию на этот алгоритм у RSA Data Security. А без лицензии можно утверждать лишь то, что "используется алгоритм, похожий на RC4 и совпадающий с ним на всем известном множестве тестов". Именно поэтому на языке ADA был реализован Alleged (предполагаемый) RC4.

3. ШИФРОВАНИЕ С ОТКРЫТЫМ КЛЮЧОМ

Теория шифров с открытым ключом

Асимметричные криптосистемы

Предпосылки появления асимметричных криптосистем

Появлению нового направления в криптологии - асимметричной криптографии с открытым ключом - способствовали две проблемы, которые не удавалось решить в рамках классической симметричной одноключевой криптографии.

Первая из этих проблем связана *с распространением секретных ключей.* Как передать участникам обмена информацией сменяемые секретные ключи, которые требуются им для осуществления этого обмена? В общем случае для передачи ключа опять же требуется использование какой-то криптосистемы, то есть задача в рамках симметричной криптографии неразрешима.

Вторая из этих проблем связана с распространением электронного документооборота. Возникла проблема обеспечения подлинности и авторства электронных документов. В обычном, бумажном документообороте эта прооблема решается с помощью подписи на бумаге. Подделать подпись человека на бумаге совсем не просто, а скопировать цепочку цифр на ЭВМ - несложная операция. Возникла проблема цифровой подписи, которая бы выполняла все те задачи, которые выполняет подпись, поставленная на документе рукой. Обе эти проблемы были успешно решены с помощью криптографии с открытыми ключами. В опубликованной в 1976 г. статье "Новые направления в криптографии" У.Диффи и М.Хеллман впервые показали, что секретная связь возможна без передачи секретного ключа между отправителем и получателем.

На основе результатов, полученных классической и современной алгеброй, были предложены *системы с откр*ытым *ключом,* называемые также *асимметричными криптосистемами.*

Суть их состоит в том, что каждым адресатом ИС генерируются два ключа, связанные между собой по определенному правилу. Один ключ объявляется *откр*ытым, а другой *зак*р*ытым*. Открытый ключ публикуется и доступен любому, кто желает послать сообщение

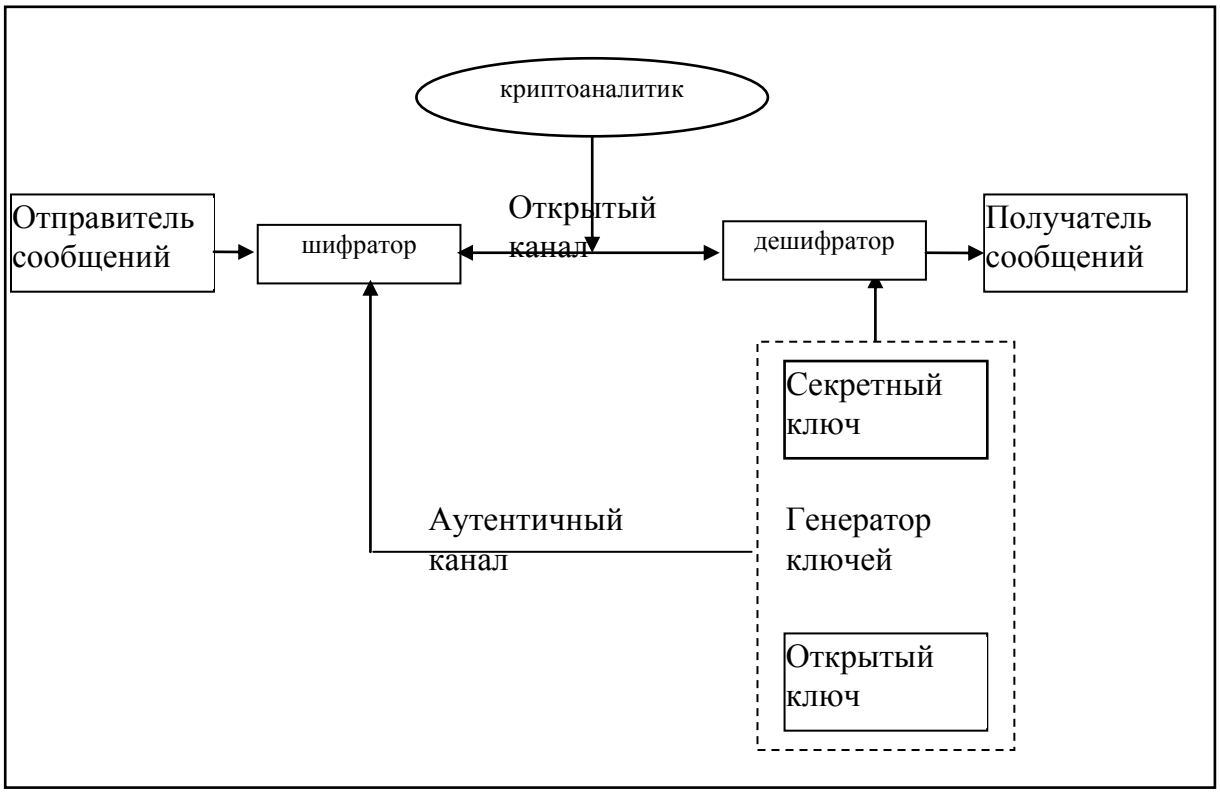


Рис.3.1. Обобщенная схема асимметричной крипосистемы

адресату. Секретный ключ сохраняется в тайне. Если генератор ключей расположить на стороне получателя, то отпадает необходимость пересылки секретного ключа по каналу связи.

Исходный текст шифруется открытым ключом адресата и передается ему. Зашифрованный текст *в принципе не может быть расшифрован* тем же открытым ключом. Дешифрование сообщения возможно только с использованием закрытого ключа, который известен только самому адресату. Таким образом, не требуется секретный канал связи для передачи ключа, но необходимо обеспечить подлинность открытого ключа, так как его искажение или подмена не позволит расшифровать информацию на парном с ним закрытом ключе. Кроме того, замена злоумышленником законного открытого ключа на свой открытый ключ предоставляет ему полный доступ к шифруемой информации.

Обобщенная схема асимметричной крипосистемы

Ниже (рис.3.1) приведена *обобщенная схема асимметричной крипосистемы*.

Здесь для передачи ключа используется открытый канал связи, обеспечивающий аутентичность передаваемой информации.

Чтобы гарантировать надежную защиту информации, к системам с открытым ключом (СОК) предъявляются два важных и очевидных требования:

1. Преобразование исходного текста должно быть необратимым и исключать его восстановление на основе открытого ключа.

2. Определение закрытого ключа на основе открытого также должно быть невозможным на современном технологическом уровне. При этом желательна точная нижняя оценка сложности раскрытия шифра.

Алгоритмы шифрования с открытым ключом получили широкое распространение в современных информационных системах. Их используют в следующих основных напрвлениях:

1. Как самостоятельные средства защиты передаваемых и хранимых данных.

2. Как средства для распределения криптографических ключей. Алгоритмы асимметричных крипосистем более трудоемки, чем традиционные криптосистемы. Поэтому часто на практике имеет смысл с помощью асимметричных крипосистем распределять ключи, объем которых незначителен. А потом с помощью менее трудоемких симметричных алгоритмов осуществлять обмен большими информационными потоками.

3. Как средства аутентификации пользователей информационных систем, в том числе для решения проблемы электронной подписи.

4. Как средства для построения сложных криптографических протоколов для решения различных задач защиты инфомации в современных информационных системах.

Алгебраическая обобщенная модель шифра

Ранее мы рассматривали алгебраическую модель шифра К.Шеннона как трехосновную универсальную алгебру A=(M,K,C,E), где

M - множество открытых текстов,

K - множество ключей,

C - множество криптограмм и

E – инъективная (взаимно однозначная) функция шифрования:

E_k: MxK $\rightarrow$ C $E_k(m)=c$, где $m \in M$, $k \in K$, $c \in C$.

Для того, чтобы эта модель могла быть применима к асимметричным криптографическим системам, необходимо ее расширение. Основная концептуальная идея построения такой модели, естественна и очевидна. Она состоит в отдельном описании моделей двух шифров: шифра шифрования и шифра расшифрования, совокупность которых и составляет обобщенную алгебраическую модель шифра.

Шифром зашифрования (алгеброй зашифрования) назовем алгебру

$A_ш=(M, M_c, К_ш C, C_c, E)$, где

- ✧ множество $M_c \subseteq M$ трактуется как подмножество всех содержательных текстов из множества «открытых текстов» M.
- ✧ функция шифрования E осуществляет отображение $MxК_ш$ на C:

$E: MxК_ш \rightarrow C$, $E_к(m)=c$,

то есть является сюръективной, причем $\forall k \in К_ш$ отображение $E_k(m)$ инъективно (образы двух различных элементов различны), а множество C_c состоит из шифрограмм, которые могут быть получены в результате шифрования содержательных текстов: $C_c = E(M_c x К_ш)$, то есть результатов шифрования тех открытых текстов m, для которых определено значение $E_k(m)$ для всех ключей шифра $k \in К_ш$.

Введение подмножества $M_c \subseteq M$ как множества содержательных текстов позволяет корректно вводить критерии на содержательные тексты.

Таким образом, шифр зашифрования есть некоторое уточнение модели шифра Шеннона $A=(M,К_ш,C,E)$.

Шифром расшифрования (алгеброй расшифрования) для $A_ш$ назовем алгебру

$A_р=(M_р, К_р, C_р, D)$, где

- ➢ $C \subseteq C_р$ и введение множества C_p — шифртекстов «правильных» сообщений (а точнее, $C_p \backslash C$ — множества искаженных шифртекстов) обеспечивает возможность описания реакции приемной стороны на поступление искаженного шифрованного сообщения $c_p \notin C$;
- ➢ $M \subseteq M_р$, и введение множества $M_p \backslash M$ обеспечивает возможность описания результата расшифрования приемной стороной искаженного шифрованного сообщения $c_p \notin C$;
- ➢ функция дешифрования D- сюръективное отображение

$D: C_р x К_р \rightarrow M_р$, $D_k(c)= m$,

для которого выполняются следующие условия:

1) существует биекция $f: К_ш \rightarrow К_р$;

2) для любых $m \in M$, $k \in К_ш$ из условия $E_k(m)=c$ вытекает

$D(c,f(k))=m$.

При отсутствии искажений в канале связи функция расшифрования D полностью определена на всем множестве $CxК_р$.

Отметим, что в определении шифра расшифрования не содержится требований инъективности функции f по переменной $k \in K_p$.

Алгебраической обобщенной моделью шифра назовем тройку

$(A_ш, A_р, f)$.

К положительным свойствам этой модели относится возможность моделирования шифров как с симметричным, так и с асимметричным ключом.

При этом учитываются следующие соображения:

- ключ $k_ш \in K_ш$ несекретен, а ключ $k_р = f(k_ш) \in K_p$ является секретным;
- определение значения k связано с решением сложных проблем;
- синтез пар ключей $(k_ш, k_р)$ проводится достаточно просто.

Заметим, что здесь проявляется возможность классификации шифров по параметру сложности вычисления значения f $(k_ш)$ ключа расшифрования, что определяет основной параметр криптографической стойкости шифров с ассиметричным ключом.

Односторонние функции

Концепция асимметричных криптографических систем с открытым ключом основана на применении однонаправленных или односторонних функций. Последнее название было дано по ассоциации с односторонним движением, когда легко проехать в одну сторону и нельзя в другую. При этом в криптографии, как в жизни, «нельзя» не означает «невозможно ни при каких условиях», но говорит о том, что это сопряжено с серьезными трудностями.

Определение. Функция f: $X \to Y$ называется *односторонней (oneway function),* если существует эффективный алгоритм для вычисления $f(x)$ $\forall x$, но не существует эффективного алгоритма для вычисления хотя бы одного элемента прообраза $f^{-1}(y)$.

Никто не знает, существуют ли вообще односторонние функции. Основным критерием отнесения функции f к классу односторонних или необратимых является отсутствие эффективных *с вычислительной точки зрения* алгоритмов обратного преобразования $Y \to X$. В криптографии под *необратимостью* понимается не теоретическая необратимость функции, а практическая невозможность вычислить обратное значение, используя современные вычислительные средства за заданный интервал времени. Таким образом, проблемы построения односторонних функций связаны с теоретико-вероятностной сложностью алгоритмов и алгоритмическими вопросами теории чисел.

Множество классов необратимых функций и порождает все разнообразие систем с открытым ключом. Большинство предлагаемых сегодня криптосистем с открытым ключом опираются на один из следующих типов необратимых преобразований:

1. Разложение больших целых чисел на простые множители.
2. Вычисление логарифма в конечном поле.
3. Вычисление корней алгебраических уравнений.

Факторизация

В качестве первого примера однонаправленной функции рассмотрим целочисленное умножение. Прямая задача — вычисление произведения двух очень больших целых чисел p и q, т.е. нахождение значения $n=p\cdot q$, является относительно несложной задачей.

Обратная задача, называемая задачей факторизации, - разложение на множители большого целого числа, т.е. нахождение делителей p и q большого целого числа

$n = p\cdot q,$

является практически неразрешимой задачей при достаточно больших значениях n. По современным оценкам теории чисел при целом $n\approx 2^{664}$ и $p\approx q$ для разложения числа n потребуется около 10^{23} операций, т.е. задача практически неразрешима на современных ЭВМ.

Если простые сомножители имеют специальный вид, известны более эффективные алгоритмы факторизации. Речь идет о сомножителях p, таких, у которых величины p-1 или p+1 являются «гладкими», т. е. имеют только малые простые делители.

Однако с появлением алгоритма факторизации с использованием эллиптических кривых класс чисел, допускающих быструю факторизацию, расширился и простые критерии проверки принадлежности данному классу утратили свою значимость. Поэтому, как правило, единственным разумным критерием может служить размер простых множителей, поскольку с увеличением размера уменьшается вероятность выбрать число специального вида.

Дискретный логарифм

Другой пример однонаправленной функции — это модульная экспонента с фиксированными основанием и модулем. Пусть a и n - целые числа, такие, что $1\leq a \leq n$. Тогда модульная экспонента с основанием a по модулю n представляет собой функцию

$y = a^x \bmod n,$

где x – целое число. Естественно записать $x = log_a (y)$.

Задачу обращения этой функции в множестве целых чисел называют задачей нахождения дискретного логарифма.

Определение. Число x называют дискретным логарифмом числа y по основанию a и модулю n, если для всех $a\in Z_n$ найдется такое целое y, что

$y = a^x \bmod n.$

Вычисление дискретных логарифмов (когда заданы a, y и n) примерно такая же труднорешаемая задача, как и разложение на множители.

Определение. Односторонняя функция $f: X \to Y$ называется *односторонней функцией с ловушкой*, если $f^{-1}(y)$ можно вычислить за полиномиальное время, имея некоторую дополнительную информацию, т. е. существует функция $g(y,t)$, вычислимая за полиномиальное время и такая, что $g(y,t) = f^{-1}(y)$ для некоторой ловушки t.

Эффективное вычисление обратной функции возможно, если известен "потайной ход" (секретное число, строка или другая информация, ассоциирующаяся с данной функцией). В качестве примера однонаправленной функции с "потайным ходом" можно привести использование функции Эйлера в криптосистеме RSA.

Криптосистема RSA

Алгоритм RSA стал первым полноценным алгоритмом с открытым ключом, который может работать как в режиме шифрования данных, так и в режиме электронной цифровой подписи. Основанная на этом алгоритме популярная криптосистема RSA разработана в 1977 году и получила название в честь ее создателей: Рональда Ривеста (в настоящее время он возглавляет компанию RSA Data Security), Ади Шамира и Леонарда Эйдельмана.

В настоящее время RSA является наиболее распространенной криптосистемой с открытым ключом — стандартом де-факто для многих криптографических прложений. Статус де-факто послужил причиной включения криптосистемы RSA в принятые ранее криптографические стандарты, например в финансовые стандарты США и Франции, австралийский стандарт управления ключами и многие другие. Криптосистема RSA применяется в различных протоколах Internet. В криптографические стандарты, действующие на территории России, RSA не входит, что осложняет ее применение с точки зрения правовых норм. Тем не менее, выбор этой криптосистемы признается оправданным отечественными авторами [8].

Основные определения и теоремы

Надежность алгоритма основывается на трудновычислимых задачах факторизации (разложения на множители) больших чисел и вычисления дискретных логарифмов. Определения и теоремы из алгебры, использованные при создании данной криптосистемы рассмотрены в приложении. Приведем здесь только основные утверждения.

1. Криптографические системы являются стойкими, если определенные их параметры являются ***простыми числами***. Число a называется простым, если оно не имеет целых делителей, кроме единицы. Числа a и b называются взаимно простыми, если их наибольший общий делитель НОД(a, b)=1.

2. ***Функцией Эйлера*** $\varphi(n)$ называется число положительных целых чисел меньших n и взаимно простых с n.

Вычисление функции Эйлера $\varphi(n)$ для больших n в общем случае представляет собой трудоемкую процедуру перебора всех чисел меньших n и проверки для каждого взаимной простоты с n. Однако, эта функция обладает следующими свойствами:

- $\varphi(p) = p - 1 \ \forall p$ – простого числа.
- $\varphi(a, b) = \varphi(a)\ \varphi(b)$ для любых натуральных взаимно простых a и b,

которые позволяют легко вычислить значение функции Эйлера $\varphi(n)$ с помощью трех арифметических действий, если известно разложение числа n на простые сомножители p и q:

$\varphi(n)=(p-1)(q-1)$.

3. В RSA используется теорема, которая носит название ***китайской теоремы об остатках,*** так как этот результат был известен еще в древнем Китае, где теорема была предложена китайским математиком первого века Сун Це. Она утверждает, что любое неотрицательное целое число, не превосходящее произведения модулей, можно однозначно восстановить, если известны его вычеты по этим модулям, и названа так потому, что результатом приведения числа a по модулю n является остаток от деления a на n.

Фактически в RSA используется следствие из этой теоремы, утверждающее, что если известно разложение числа n на простые множители $n=n_1n_2...n_k$, где все n_i попарно взаимно просты, и результат приведения числа x по модулю $n_i \ \forall i=1,...,k$ одинаков, то результатом приведения числа x по модулю n будет то же число. То есть $\forall x,a$ – целых чисел

$x \equiv a \bmod n \ \Leftrightarrow \ x \equiv a \bmod n_i \ \forall i=1,...,k.$

4. ***Теорема Эйлера.*** Если $n>1$, то $\forall \ x \in Z_n^*$ (x взаимно простого с n), выполняется сравнение

$x^{\varphi(n)} \equiv 1 \bmod n.$

Следствие. $\forall \ x < n$ и взаимно простого с n можно легко вычислить обратный элемент x^{-1} в кольце вычетов Z_n из сравнения

$x^{-1} \equiv x^{\varphi(n)-1} \bmod n.$

Малая теорема Ферма. $\forall \ x \in GF(p), \ x \neq 0,$ выполняется сравнение

$x^{p-1} \equiv 1 \bmod p.$

Малая теорема Ферма является следствием из теоремы Эйлера, хотя исторически она была доказана раньше, затем Эйлер её обобщил.

Следствие 1: Если p — простое число, то $\forall \ x$, взаимно простого с p:

$x^p = x \bmod p.$

Следствие 2: если НОД$(e,\varphi(n))=1$ (e - простое относительно $\varphi(n)$)

то $\exists \ d$-целое, такое, что

$ed = 1 \bmod n.$

На этих математических фактах основан алгоритм RSA.

Алгоритм *RSA*

В криптосистеме RSA сообщение m, криптограмма c, открытый ключ K_o, и секретный ключ K_c, принадлежат множеству целых чисел $Z_n=\{0, 1, 2,..., n-1\}$. Множество Z_n с операциями сложения и умножения по модулю n образует кольцо.

Модуль n определяется как составное число равное произведению $n=p\cdot q$ двух больших простых чисел $p\cdot$и q. Модуль n является открытым параметром алгоритма, а чисела $p\cdot$и q — секретными параметрами. То есть множители p и q хранят в секрете, а их произведение n известно всем, кто пользуется данной криптосистемой. Здесь используется односторонняя функция с ловушкой. Зная секретные параметры алгоритма p и q можно легко вычислить функцию Эйлера $\varphi(n)$ по формуле

$\varphi(n)=(p-1)(q-1)$,

тогда как вычисление $\varphi(n)$ только по большому числу n является трудновычислимой задачей.

Функция Эйлера используется в RSA при вычислении ключей.

Открытый ключ $K_o = e$ выбирают случайным образом из множества

$Z^*_{\varphi(n)}$ — чисел меньших $\varphi(n)$ и взаимно простых с $\varphi(n)$.

Иначе условие $e\in Z^*_{\varphi(n)}$ раносильно выполнению двух условий

$1<K_o\le\varphi(n)$, и НОД(K_o, $\varphi(n)$)=1,

которым должено удовлетворять случайно выбранное число $K_o = e$, чтобы оно могло служить открытым ключом в RSA.

Секретный ключ $K_c = d$ вычисляют так, чтобы выполнялось условие

$$K_c \cdot K_o = e \cdot d \equiv 1 \bmod \varphi(n). \qquad (*)$$

То есть, секретный ключ d является обратным элементом к открытому ключу e в множестве $Z_{\varphi(n)}$: $d = e^{-1} \bmod \varphi(n)$. Решение сравнения (*) можно найти с помощью расширенного алгоритма Евклида.

Заметим, что d и n также взаимно простые числа. Вычисление секретного ключа по открытому является трудновычислимой задачей, если неизвестны секретные параметры алгоритма p и q, так как при этом трудно вычислить значение функции Эйлера, то есть модуля, по которому приводятся результаты операций при вычислении ключей.

При выполнении шифрования и дешифрования вычисления приводятся по модулю n. Открытый ключ K_o и модуль n сообщают всем, с кем предполагают обмениваться

сообщениями и используют для шифрования данных, а секретный ключ K_c хранят в секрете на стороне получателя и используют для дешифрования.

Преобразование шифрования определяет криптограмму c через пару (открытый ключ K_o, сообщение m) в соответствии со следующей формулой:

$c = E_{Ko}(m) = m^{Ko} \bmod n$.

В качестве алгоритма быстрого вычисления значения c используют ряд последовательных возведений в квадрат целого m и умножений на m с приведением по модулю n.

Обращение функции $c = m^{Ko} \bmod n$, т.е. определение значения m по известным значениям c, K_o и n, является задачей дискретного логарифмирования и практически неосуществимо при $n \approx 2^{512}$.

Однако задачу расшифрования криптограммы c, можно легко решить, используя секретный ключ K_c, по следующей формуле:

$m = D_{Kc}(c) = c^{Kc} \bmod n$.

Докажем, что в результате возведения криптограммы c в степень секретного ключа K_c получается исходный текст m. Процесс расшифрования можно записать так:

$D_{Kc}(E_{Ko}(m)) = D_{Kc}(m^e) \bmod n = (m^e)^d \bmod n = m^{ed} \bmod n$.

По условию выбора ключей

$e \cdot d \equiv 1 \bmod \varphi(n)$ (*)

мы можем написать, что $\exists\, k$ такое что, с учетом свойств $\varphi(n)$:

$e \cdot d = k \cdot \varphi(n) + 1 = k\,(p-1)(q-1) + 1$.

Подставим это выражение в показатель степени:

$m^{ed} \equiv m\,(m^{(p-1)})^{k\,(q-1)} = (m^{(q-1)})^{k\,(p-1)}$.

По малой теореме Ферма ($x^{p-1} \equiv 1 \bmod p$, p – простое):

$m^{ed} \bmod p \equiv m\,(1)^{k\,(q-1)} \bmod p = m \bmod p$.

Аналогично, заменив p на q, получим:

$m^{ed} \bmod q \equiv m\,(1)^{k\,(p-1)} \bmod q = m \bmod q$.

Далее, по следствию из китайской теоремы об остатках так как $n = pq$:

$m^{ed} \bmod n = m \bmod n$, ч.т.д.

Таким образом, если криптограмму c возвести в степень K_c:

$c^{Kc} \bmod n = m$,

то в результате восстанавливается исходный открытый текст m.

Именно поэтому для вычисления секретного ключа K_c используют соотношение (*).

Процедуры шифрования и расшифрования в криптосистеме *RSA*

В реальных системах алгоритм RSA реализуется следующим образом. Предположим, что пользователь А хочет передать пользователю В сообщение в зашифрованном виде, используя криптосистему RSA. В таком случае пользователь А выступает в роли отправителя сообщения, а пользователь В - в роли получателя. Криптосистему RSA должен сформировать получатель сообщения, т.е. пользователь В, так как в этом случае не будет необходимости в передаче секретного ключа. Рассмотрим последовательность действий пользователя В и пользователя А.

Формирование критпосистемы (на строне получателя информации) состоит в выборе параметров алгоритма и вычислении пары ключей:

1. Пользователь В выбирает два больших простых числа p и q. Это *секретные параметры* алгоритма, они хранятся в секрете на стороне получатателя.

2. Пользователь В вычисляет значение модуля n криптосистемы как результат умножения первых двух чисел:

$n = p \cdot q$.

Это *общедоступный параметр* криптосистемы**.** Иногда его включают в открытый ключ.

3. Пользователь В, зная секретные параметры алгоритма p и q, вычисляет функцию Эйлера:

$\varphi(n)=(p-1)(q-1)$

и выбирает случайным образом простое число e как значение открытого ключа K_o с учетом выполнения условий:

$K_o<\varphi(n)$ и НОД$(K_o, \varphi(n))=1$.

4. Пользователь В вычисляет значение секретного ключа $K_c = d$ при решении сравнения

$K_c \equiv K_o^{-1} \bmod \varphi(n)$.

Для вычисления обратного элемента в кольце $Z_{\varphi(n)}$ используют частный режим работы расширенного алгоритма Евклида для определения НОД. Это можно осуществить, так как получатель В знает пару простых чисел (p,q) и может легко найти $\varphi(n)$. Заметим, что K_c и n должны быть взаимно простыми.

(В принципе открытый и закрытый ключи можно поменять местами, главное, чтобы выполнялось соотношение $e \cdot d \equiv 1 \bmod \varphi(n)$).

5. Пользователь В пересылает пользователю А пару чисел $\{e,n\}$ по незащищенному каналу.

Шифрование информации (на строне отправителя). Если пользователь А хочет передать пользователю В сообщение m, он выполняет следующие шаги.

6. Пользователь А разбивает исходный открытый текст m на блоки m_i, i=1,…,N, каждый из которых может быть представлен в виде числа меньшего n

$m_i \in \{0, 1, 2, \ldots, n\text{-}1\}$.

7. Пользователь А шифрует текст, представленный в виде последовательности чисел m_i с помощью ключа $K_o = e$ по формуле

$c_i = m_i^e \mod n$

и отправляет криптограмму $c_1, \ldots, c_i$. пользователю В.

Дешифрование информации (на строне получателя):

8. Пользователь В расшифровывает принятую криптограмму $c_1, \ldots, c_i$, используя секретный ключ K_c=d, по формуле

$m_i = c_i^d \mod n$.

В результате будет получена последовательность чисел, которые представляют собой исходное сообщение m.

Таким образом, когда получатель В, который создает криптосистему, он защищает следующие параметры:

1. секретный ключ K_c;
2. пару чисел (p, q);

Открытый ключ K_o и значение модуля n публикуются и доступны каждому, кто желает послать владельцу ключа сообщение, которое зашифровывается указанным алгоритмом. После шифрования сообщение невозможно раскрыть с помощью открытого ключа. Владелец же закрытого ключа без труда может расшифровать принятое сообщение. Противнику же длятого, чтобы определить значение секретного ключа K_c нужно суметь разложить число n на множители p и q, чтобы узнать бы "потайной ход" и вычислить значение функции Эйлера как $\varphi(n) = (p\text{-}1)(q\text{-}1)$.

Пример. Рассмотрим небольшой пример, иллюстрирующий применение алгоритма RSA. Зашифруем сообщение "CAB". Для простоты будем использовать маленькие числа (на практике применяются гораздо большие).

1. Выберем p=3 и q=11.
2. Определим n=3·11=33.
3. Найдем $\varphi(n) = (p\text{-}1)(q\text{-}1)= 2\cdot 10 = 20$.
4. Выберем в качестве секретного ключа d произвольное число взаимно простое с $\varphi(n)$=20, например, d=3.
5. Выберем открытый ключ e = 7. В качестве такого числа может быть взято любое число, для которого удовлетворяется соотношение

$e \cdot d \mod \varphi(n) = 7 \cdot 3 \mod 20 = 1$.

6. Представим шифруемое сообщение как последовательность целых чисел с помощью отображения: A=1, B=2, C=3. Тогда сообщение принимает вид (3,1,2). Зашифруем сообщение с помощью ключа {e=7, n=33}:

ШТ1 = (3^7) mod 33 = 2187 mod 33 = 9,

ШТ2 = (1^7) mod 33 = 1 mod 33 = 1,

ШТ3 = (2^7) mod 33 = 128 mod 33 = 29.

7. Расшифруем полученное зашифрованное сообщение (9, 1, 29) на основе закрытого ключа { d=3, n=33}:

ИТ1 = (9^3) mod 33 = 729 mod 33 = 3,

ИТ2= (1^3) mod 33 = 1 mod 33 = 1,

ИТ3 = (29^3) mod 33 = 24389 mod 33 = 2.

Надежность RSA. RSA многие годы противостоит интенсивному криптоанализу. Доказано, что раскрытие шифра RSA эквивалентно решению задачи факторизации больших (100-200 двоичных разрядов) чисел. Важно, что в этой задаче для любой длины ключа можно дать нижнюю оценку числа операций для раскрытия шифра, а с учетом производительности современных компьютеров оценить и необходимое на это время.

Возможность гарантированно оценить защищенность алгоритма RSA стала одной из причин популярности этой СОК на фоне десятков других схем. Поэтому алгоритм RSA используется в банковских компьютерных сетях, особенно для работы с удаленными клиентами (обслуживание кредитных карточек). В настоящее время алгоритм RSA активно реализуется как в виде самостоятельных криптографических продуктов, так и в качестве встроенных средств в популярных приложениях, а также используется во многих стандартах.

Криптосистема Эль-Гамаля

Данная система является альтернативой RSA и при равном значении ключа обеспечивает ту же криптостойкость. Однако общего мнения по поводу предпочтительности того или иного метода нет.

В отличие от RSA метод Эль-Гамаля основан на проблеме дискретного логарифма. Этим он похож на *алгоритм Диффи-Хелмана*. Если возводить число в степень в конечном поле достаточно легко, то восстановить аргумент по значению (то есть найти логарифм в множестве целых чисел) довольно трудно.

Основу системы составляют параметры p и g - числа, первое из которых p — простое, а второе ($g \in Z_p$) — целое. Данные параметры не являются секретными и могут быть общими

для группы пользователей. В реальных схемах шифрования необходимо использовать в качестве модуля большое целое простое число, имеющее в двоичном представлении длину 512...1024 бит.

Секретный ключ $x \in Z_{p-1}$ генерируется случайным образом, а открытый ключ y вычисляется по формуле

$y = g^x \bmod p$.

Для шифрования сообщения m сначала выбирается случайное число k, взаимно простое с $p-1$, которое называют также сеансовым ключом.

Шифртекстом является пара чисел (a,b), вычисляемая по формулам:

$a = g^k \bmod p$ и

$b = y^k m \bmod p$.

Таким образом, шифртекст в два раза длиннее открытого текста.

Для дешифрования вычисляется

$$m = \frac{b}{a^x} \bmod p.$$

Преобразование обратимо, так как

$a^x = g^{kx} \bmod p$

и $$\frac{b}{a^x} \equiv \frac{y^k m}{a^x} \equiv \frac{g^{xk} m}{a^x} = m \bmod p.$$

Число k называют также рандомизатором. Его использование означает, что здесь реализован мнозначный шифр замены. При этом для зашифрования различных блоков (чисел) открытого текста необходимо использовать различные значения рандомизатора. При использовании одного и того же значения соответсвующие шифртексты (a, b) и (a', b'), полученные для блоков открытых текстов m и m', связаны соотношением

$b(b')^{-1} = m(m')^{-1}$

и текст m' можно вычислить, если известен текст m.

Стойкость криптосистемы Эль Гамаля основана на сложности задачи логарифмирования в мультипликативной группе конечного простого поля. Эта криптосистема может быть обобщена для применения в любой конечной циклической группе. В качестве такой группы, помимо рассмотренной Z^*_p, чаще всего используется мультипликативная группа конечного поля $GF(2^m)$ и группа точек на эллиптической кривой над конечным полем (см. приложение П.8).

Алгоритм не запатентован, но попадает под действие патента на метод экспоненциального ключевого обмена Диффи-Хеллмана, рассмотренный ниже. Преобразование

шифрования/дешифрования Эль Гамаля по сути то же самое, что ключевой обмен по Диффи-Хеллману, за исключением того, что y – это часть ключа, а при шифровании сообщение умножается на y^k. Алгоритм цифровой подписи DSA, разработанный NIST (National Institute of Standard and Technology USA) и являющийся частью стандарта DSS частично опирается на рассмотренный метод.

Комбинированный метод шифрования

Главным достоинством криптосистем с открытым ключом является их потенциально высокая безопасность: нет необходимости ни передавать, ни сообщать кому бы то ни было значения секретных ключей, ни убеждаться в их подлинности. В симметричных криптосистемах существует опасность раскрытия секретного ключа во время передачи.

Однако алгоритмы, лежащие в основе криптосистем с открытым ключом, имеют следующие недостатки:

• генерация секретных и открытых ключей основана на генерации больших простых чисел, а проверка простоты чисел занимает много процессорного времени;

• процедуры шифрования и расшифрования, связанные с возведением в степень многозначного числа, достаточно трудоемки.

Поэтому быстродействие (скорость шифрования и дешифрования) в криптосистемах с открытым ключом обычно в сотни и тысячи раз меньше быстродействия симметричных криптосистем с секретным ключом.

Комбинированный метод шифрования позволяет сочетать преимущества высокой секретности, предоставляемые асимметричными криптоистемами с открытым ключом, с преимуществами высокой скорости работы, присущими симметричным криптосистемам с секретным ключом. При таком подходе криптосистема с открытым ключом применяется для шифрования, передачи и последующего расшифрования только секретного ключа симметричной криптосистемы. А симметричная криптосистема применяется для шифрования и передачи исходного открытого текста. В результате криптосистема с открытым ключом не заменяет симметричную криптосистему с секретным ключом, а лишь дополняет ее, позволяя повыить в целом защищенность передаваемой информации.

Пользователи А и В, использующие комбинированный метод шифрования, имеют каждый по паре асимметричных ключей шифрования

(K_{Ao}, K_{Ac}) и (K_{Bo}, K_{Bc}).

Если пользователь А хочет передать зашифрованное комбинированым методом сообщение m пользователю В, то порядок его действий будет таков.

1. Создать (например, сгенерировать случайным образом) симметричный ключ, называемый в этом методе сеансовым ключом K_s.

2. Зашифровать сообщение m на сеансовом ключе K_s.

3. Зашифровать сеансовый ключ K_s на открытом ключе K_{Bo} пользователя B и своем секретном ключе K_{Ac}.

4. Передать по открытому каналу связи в адрес пользователя B зашифрованное сообщение вместе с зашифрованным сеансовым ключом.

Действия пользователя B при получении зашифрованного сообщения и зашифрованного сеансового ключа должны быть обратными:

5. Расшифровать на своем секретном ключе K_{Bc} и открытом ключе K_{Ao} пользователя A сеансовый ключ K_s.

6. С помощью полученного сеансового ключа K_s расшифровать и прочитать сообщение m.

При использовании комбинированного метода шифрования можно быть уверенным в том, что только пользователь B сможет правильно расшифровать ключ K_s и прочитать сообщение m.

Выбор длины ключей в комбированном методе шифрования. Таким образом, при комбинированном методе шифрования применяются криптографические ключи как симметричных, так и асимметричных криптосистем. Очевидно, выбор длин ключей для каждого типа криптосистемы следует осуществлять таким образом, чтобы злоумышленнику было одинаково трудно атаковать любой механизм защиты комбинированной криптосистемы.

В таблице 3.1. приведены распространенные длины ключей симметричных и асимметричных криптосистем, для которых трудность атаки полного перебора примерно равна трудности факторизации соответствующих модулей асимметричных криптосистем.

Таблица 3.1. Длины ключей для симметричных и асимметричных криптосистем при одинаковой их криптостойкости

Симметричные криптосистемы	56	64	80	112	128
Асимметричные криптосистемы	384	512	786	1792	2304

Метод экспоненциального ключевого обмена Диффи-Хеллмана

Одни из авторов идеи криптосистем с открытым ключом, Диффи и Хеллман предложили новую идею - *открытое распределение ключей.*

Они задались вопросом: можно ли организовать такую процедуру взаимодействия абонентов А и В по открытым каналам связи, чтобы решить следующие задачи:

1) вначале у А и В нет никакой общей секретной информации, но в конце процедуры такая общая секретная информация (общий ключ) у А и В появляется, т. е. вырабатывается;

2) пассивный противник, который перехватывает все передачи информации и знает, что хотят получить А и В, тем не менее не может восстановить выработанный общий ключ А и В.

Метод получил название метода экспоненциального ключевого обмена. Он был первой криптосистемой с открытым ключом, хотя для обмена ключами можно использовать любые криптосистемы с открытым ключом, например, тот же алгоритм RSA.

Криптостойкость данного метода определяется трудоемкостью вычисления дискретного логарифма:

$f(x) = g^x \bmod p,$

где p — большое простое число, x — произвольное натуральное число, g — некоторый *примитивный элемент* поля Галуа $GF(p)$. Общепризнано, что инвертирование функции $g^x \bmod p$, т.е. дискретное логарифмирование, является трудной математической задачей.

Сама процедура или *протокол выработки общего ключа* заключается в следующем. Значения p и g являются общедоступными параметрами протокола. Абоненты А и В независимо друг от друга случайно выбирают по одному большому натуральному числу x_A и x_B. Это их секретные ключи. Далее каждый из них вычисляет открытые ключи:

$y_A = g^{xA} \bmod p$ и $y_B = g^{xB} \bmod p.$

Потом они обмениваются этими элементами по каналу связи. Теперь абонент А, получив y_B и зная свой секретный элемент X_A, вычисляет новый элемент:

$y_B^{xA} \bmod p = (g^{xB})^{xA} \bmod p.$

Аналогично поступает абонент *B:*

$y_A^{xB} \bmod p = (g^{xA})^{xB} \bmod p.$

Тем самым у А и В появился общий элемент поля, равный $g^{xA\ xB}$. Этот элемент и объявляется общим ключом абонентов А и В.

Необратимость преобразования в этом случае обеспечивается тем, что достаточно легко вычислить показательную функцию в конечном поле Галуа, состоящем из p элементов (p — простое число). Обратная задача вычисления x из y будет достаточно сложной. Если p выбрано достаточно правильно, то извлечение логарифма потребует вычислений, пропорциональных $L(p) = exp\{ (ln\ p\ ln\ ln\ p)^{0.5}\}$.

При всей простоте алгоритма Диффи-Хелмана его недостатком по сравнению с системой RSA является отсутствие гарантированной нижней оценки трудоемкости раскрытия ключа.

Алгоритмы практической реализации криптосистем с открытым ключом

Возведение в степень по модулю m

В криптографии используются вычисления по mod *m*, так как алфавит любой криптографической системы представляет собой конечное множество целых чисел. Арифметика вычетов к тому же легче реализуется на компьютерах, поскольку она ограничивает диапазон промежуточных значений и результата. Для *k*-битовых вычетов *m* промежуточные результаты любого сложения, вычитания или умножения будут не длиннее, чем 2*k* бит. Поэтому в арифметике вычетов мы можем выполнить возведение в степень без огромных промежуточных результатов. Вычисление степени некоторого числа по модулю другого числа,

$a^d \bmod m$,

представляет собой просто последовательность умножений и делений, но существуют приемы, ускоряющие это действие. Один из таких приемов стремится минимизировать количество умножений по модулю, другой - оптимизировать отдельные умножения по модулю. Так как операции дистрибутивны, быстрее выполнить возведение в степень как поток последовательных умножений, каждый раз получая вычеты.

Например, для того, чтобы вычислить $a^8 \bmod m$, не выполняйте семь умножений и одно приведение по модулю; вместо этого выполните три меньших умножения и три меньших приведения по модулю:

$a^8 \bmod m = ((a^2 \bmod m)^2 \bmod m)^2 \bmod m$.

Точно также, $a^{16} \bmod m = (((a^2 \bmod m)^2 \bmod m)^2 \bmod m)^2 \bmod m$.

Вычисление $a^d \bmod m$, где d не является степенью 2, не намного труднее. Двоичная запись представляет x в виде суммы степеней 2: число 25 — это бинарное 11001, поэтому 25 = 16 + 8 + 1. Тогда

$a^{25} \bmod m = (a\, a^8\, a^{16}) \bmod m = (a*(((\, a*a^2)^2)^2)^2) \bmod m$.

С продуманным сохранением промежуточных результатов нам понадобится только шесть умножений:

$(((((((a^2 \bmod m)*a)^2 \bmod m)^2 \bmod m)^2 \bmod m)^2 \bmod m)^2 \bmod m\, *a) \bmod m$.

Такой прием называется методом двоичных квадратов и умножений. Он использует простую и очевидную цепочку сложений, в основе которой лежит двоичное представление

числа. Увеличение скорости вычислений при умножении 200-битовых чисел будет очень заметным.

Алгоритм вычисления $a^d \bmod m$.

Пусть натуральные числа a и d не превосходят по величине m.

1. Представим d в двоичной системе счисления:

$d = d_o 2^r + \ldots + d_{r-1} 2 + d_r$,

где r – число двоичных разрядов, d_i - цифры в двоичном представлении, равные 0 или 1, и $d_o = 1$.

2. Положим $a_o = a$ и затем для $i = 1, \ldots r$ вычислим

$a_i \equiv a^2_{i-1}\, a^{d_i} \bmod m$.

Тогда a_r есть искомый вычет $a^d \bmod m$.

Справедливость этого алгоритма вытекает из легко доказываемого индукцией по i сравнения:

$a_i \equiv a^{d_o 2^i + \ldots + d_i} \bmod m$.

Так как каждое вычисление на шаге 2 требует не более трех умножений по модулю m и этот шаг выполняется $r \le \log_2 m$ раз, то сложность алгоритма может быть оценена величиной $0(\ln m)$. Говоря о сложности алгоритмов, мы имеем в виду количество арифметических операций.

Второй алгоритм — это классический алгоритм Евклида вычисления наибольшего общего делителя целых чисел. Мы предполагаем заданными два натуральных числа a и b и вычисляем их наибольший общий делитель НОД(a, b).

Алгоритм Евклида вычисления НОД

Одним из способов вычисления НОД двух чисел является алгоритм Евклида, который описал его с своей книге «Начала» около 300 лет до н.э. Полагают, что он не изобрел его, а сам алгоритм ешё старше лет на 200. Это древнейший нетривиальный алгоритм, который актуален и в наше время.

1. Вычислим r — остаток от деления числа a на b $(a>b)$:

$a = b\, q + r, \qquad 0 \le r < b$.

2. Если $r = 0$, то b есть искомое число.

3. Если $r \neq 0$, то заменим пару чисел (a, b) парой (b,r) и перейдем к шагу 1.

Остановка гарантируется, поскольку остатки от делений образуют строго убывающую последовательность.

Оценку сложности этого алгоритма дает следующая теорема.

Теорема. При вычислении наибольшего общего делителя НОД (*a, b*) с помощью алгоритма Евклида будет выполнено не более 5*p* операций деления с остатком, где *p* есть количество цифр в десятичной записи меньшего из чисел *a* и *b*.

Вычисление обратных величин в кольце целых чисел

При вычислении ключей в асимметричных криптографических системах необходимо находить обратные элементы в кольце целых чисел Z_m. Элемент $x \in Z_m$ является обратным к $a \in Z_m$, ($x=a^{-1}$), если

$x \cdot a \equiv 1 \bmod m$ или $a^{-1} \equiv x \bmod m$.

В общем случае это сравнение может не иметь решений или иметь несколько решений. Оно имеет единственное решение тогда и только тогда, если числа *a* и *m* взаимно простые, то есть НОД (*a, m*) =1.

Рассмотрим основные способы нахождения обратных величин в Z_m.

1. Метод перебора. Проверить поочередно значения $x \in (1, 2, \ldots . m-1)$, пока не выполнится сравнение $x \cdot a \equiv 1 \bmod m$.

2. Если известна функция Эйлера (для RSA) $\varphi(m)$, то можно вычислить $a^{-1} \bmod m \equiv a^{\varphi(m)-1} \bmod m$, используя алгоритм быстрого возведения в степень. (Это следует из утверждения теоремы Эйлера: если *a* и *m* взаимно простые, то $a^{\varphi(m)} \equiv 1 \bmod m$.)

3. Нахождение обратной величины $a^{-1} \bmod m$ с помощью расширенного алгоритма Евклида.

Алгоритм Евклида можно обобщить способом, который имеет большое практическое значение. При этом способе во время вычисления НОД можно попутно вычислить такие целые числа *x* и *y*, что

$a \cdot x + b \cdot y =$ НОД (*a, b*).

Этот вариант называется расширенным алгоритмом Евклида. Для вычисления обратной величины используется частный режим работы алгоритма Евклида, при котором

$b = m$ и НОД (*a, m*)=1.

В самом деле сравнение $x \cdot a \equiv 1 \bmod m$ означает, что ∃ целое *k*, что верно:

$a x + m k = 1 \bmod m$.

Эта задача равносильна поиску целых решений уравнения

$ax + mk = 1$.

Алгоритм поиска целых решений уравнения a x + m k = 1

Немного подправив алгоритм Евклида, можно достаточно быстро решать сравнения

$ax + mk = 1 \bmod m$

при условии, что НОД(*a, m*)=1.

0. Определим матрицу $E = I$ -единичная матрица размером 2x2.

1. Вычислим r — остаток от деления числа a на m:

$a = m\,q + r, \qquad 0 \le r < m$.

2. Если $r = 0$, то второй столбец матрицы E дает вектор $[x,k]^T$ решений уравнения.

3. Если $r \neq 0$, то заменим матрицу E матрицей

$$E = E \bullet \begin{pmatrix} 1 & 0 \\ 0 & -q \end{pmatrix}.$$

4. Заменим пару чисел (a, m) парой (m, r) и перейдем к шагу 1.

Три приведенных выше алгоритма относятся к разряду так называемых полиномиальных алгоритмов. Это название носят алгоритмы, сложность которых оценивается сверху степенным образом в зависимости от длины записи входящих чисел (см. подробности в [2]). Если наибольшее из чисел, подаваемых на вход алгоритма, не превосходит m, то сложность алгоритмов этого типа оценивается величиной $O(ln^c m)$, где c — некоторая абсолютная постоянная. Во всех приведенных выше примерах c=1.

Полиномиальные алгоритмы в теории чисел — большая редкость. Да и оценки сложности алгоритмов чаще всего опираются на какие-либо недоказанные, но правдоподобные гипотезы, обычно относящиеся к аналитической теории чисел.

Генерация простых чисел

Для алгоритмов с открытыми ключами нужны простые числа. Их нужно множество для любой достаточно большой сети, хватит ли их?

Да, существует приблизительно 10^{151} простых чисел длиной до 512 бит включительно. Для чисел, близких m, вероятность того, что случайно выбранное число окажется простым, равна $1/ln\ m$. Поэтому полное число простых чисел, меньших m, равно $m/(ln\ m)$. При выборе из 10^{151} простых чисел вероятность совпадения выбора практически равна нулю.

Но если так трудоемко разложение на множители, как может быть простой генерация простых чисел? Дело в том, что ответить "да" или "нет" на вопрос "является ли число простым?" гораздо проще, чем ответить на более сложный вопрос "каковы множители m?"

Генерация случайных чисел с последующей попыткой разложения их на множители - это неправильный способ поиска простых чисел. Существуют различные вероятностные проверки на простоту чисел, определяющие, является ли число простым, с заданной степенью достоверности. При условии, что эта «степень достоверности» достаточно велика, такие способы проверки достаточно хороши.

Предлагаются [6] следующие тесты проверки чисел на простоту:

Тест Леманна (Lehmann)

Вот последовательность действий при проверке простоты числа *p;*

(1) Выберите случайно число *a,* меньшее *p.*

(2) Вычислите $a^{(p-1)/2} \bmod p$.

(3) Если $a^{(p-1)/2} \neq 1$ или -1 mod *p*, то *p* не является простым.

(4) Если $a^{(p-1)/2} \equiv 1$ или -1 mod *p*, то вероятность того, что число *p* не является простым, не больше 50 процентов.

Число *a*, которое не показывает, что число *p* не является простым числом, называется свидетелем. Если *p*— составное число, вероятность случайного числа *a* быть свидетелем составной природы числа *p* не меньше 50 процентов. Повторите эту проверку *t* раз с *t* различными значениями *a*. Если результат вычислений равен 1 или -1, но не всегда равен 1, то *p* является простым числом с вероятностью ошибки $1/2^t$.

Алгоритм Рабина-Миллера (Rabin-Miller)

Повсеместно используемым является простой алгоритм, разработанный М. Рабином, частично основанным на идеях Миллера.

Выберите для проверки случайное число *p.* Вычислите *b* – число делений *p*-1 на 2 (т.е. 2^b – это наибольшая степень числа 2, на которую делится *p*-1). Затем вычислите *m,* такое, что $p = 1 + 2^b \cdot m$.

1) Выберите случайное число *a,* меньшее *p.*

2) Установите $j = 0$ и $z = a \cdot m \bmod p$.

3) Если $z = 1$ или если $z = p$-1, то *p* проходит проверку и может быть простым числом.

4) Если $j > 0$ и $Z = 1$, то *p* не является простым числом.

5) Установите $j = j + 1$. Если $j < b$ и $z < p - 1$, установите $z = z^2 \bmod p$ и вернитесь на этап (4). Если $z = p - 1$, то *p* проходит проверку и может быть простым числом.

6) Если $j = b$ и $z \neq p - 1$, то *p* не является простым числом.

В этом тесте вероятность прохождения проверки составным числом убывает быстрее, чем в предыдущих. Гарантируется, что три четверти возможных значений *a* окажутся свидетелями. Это означает, что составное число проскользнет через *t* проверок с вероятностью, не большей $1/4^t$, где *t* - число итераций. На самом деле и эти оценки слишком пессимистичны. Для большинства случайных чисел около 99.9 процентов возможных значений являются свидетелями [4].

Существуют более точные оценки [4]. Для *n*-битового кандидата в простые числа (где $n > 100$) вероятность ошибки в одном тесте меньше, чем $4n2^{\sqrt{k/2}}$. И для 256-битового *n* вероятность ошибки в шести тестах меньше, чем $1/2^{51}$.

Практические соображения

В реальных приложениях генерация простых чисел происходит быстро. В [6] описан следующий алгоритм:

1) Сгенерируйте случайное n-битовое число p.

2) Установите старший и младший биты равными 1. (Старший бит гарантирует требуемую длину простого числа, а младший бит обеспечивает его нечетность.)

3) Убедитесь, что p не делится на небольшие простые числа: 3, 5, 7, 11 и т.д. Во многих реализациях проверяется делимость p на все простые числа, меньшие 256. Наиболее эффективной является проверка на делимость для всех простых чисел, меньших 2000.

4) Выполните тест Рабина-Миллера для некоторого случайного a. Если p проходит тест, сгенерируйте другое случайное a и повторите проверку. Выбирайте небольшие значения a для ускорения вычислений. Выполните пять тестов. (Одного может показаться достаточным, но выполните пять.) Если p не проходит одной из проверок, сгенерируйте другое p и попробуйте снова.

Иначе, можно не генерировать p случайным образом каждый раз, но последовательно перебирать числа, начиная со случайно выбранного до тех пор, пока не будет найдено простое число.

Этап 3 не является обязательным, но это хорошая идея. Проверка, что случайное нечетное p не делится на 3, 5 и 7 отсекает 54% нечетных чисел еще до этапа 4. Проверка делимости на все простые числа, меньшие 100, убирает 76% нечетных чисел, проверка делимости на все простые числа, меньшие 256, убирает 80% нечетных чисел. В общем случае, доля нечетных кандидатов, которые не делятся ни на одно простое число, меньшее m, равна $1.12/1n\ m$. Чем больше проверяемое m, тем больше предварительных вычислений нужно выполнить до теста Рабина-Миллера.

Сильные простые числа.

Если m – произведение двух простых чисел p и q, то может понадобиться использовать в качестве и q *сильные простые числа*.

Такие простые числа обладают рядом свойств, которые усложняют разложение произведения m определенными методами разложения на множители. Среди таких свойств были предложены [4]:

1. Наибольший общий делитель p-1 и q-1 должен быть небольшим.
2. И p-1,и q-1 должны иметь среди своих множителей большие простые числа, соответственно p'и q'.
3. И p'-1,и q'-1 должны иметь среди своих множителей большие простые числа.

4. И $p+1$,и $q+1$ должны иметь среди своих множителей большие простые числа.

5. И $(p-1)/2$,и $(q-1)/2$ должны быть простыми (обратите внимание, при выполнении этого условия выполняются и два первых).

Насколько существенно применение именно сильных простых чисел остается предметом продолжающихся споров. Эти свойства были разработаны, чтобы затруднить выполнение ряда старых алгоритмов разложения на множители. Однако самые быстрые алгоритмы одинаково быстры при разложении на множители любых чисел, как удовлетворяющих приведенным условиям, так и нет.

Некоторые авторы [4] выступают против специальной генерации сильных простых чисел, утверждая, что длина простых чисел гораздо важнее их структуры. Более того, сама структура уменьшает случайность числа и может снизить устойчивость системы.

Но все может измениться. Могут быть созданы новые методы разложения на множители, которые лучше работают с числами, обладающими определенными свойствами. В этом случае снова могут потребоваться сильные простые числа.

3.2. Компьютерный практикум для шифров с открытым ключом

Изучение отечественных стандартов функции хэширования
и цифровой подписи.

Цель работы

Целью данной лабораторной работы является изучение процессов вычисления функции хэширования по ГОСТ Р 34.11-94, формирования и проверки ЭЦП на эллиптических кривых по ГОСТ Р 34.10-2001.

Краткие теоретические сведения

Электронная Цифровая Подпись

При обмене электронными документами по сети связи существенно снижаются затраты на обработку и хранение документов, убыстряется их поиск. Но при этом возникает проблема аутентификации автора документа и самого документа, т.е. установления подлинности автора и отсутствия изменений в полученном документе. В обычной (бумажной) информатике эти проблемы решаются за счет того, что информация в документе и рукописная подпись автора жестко связаны с физическим носителем (бумагой). В электронных документах на машинных носителях такой связи нет.

Целью аутентификации электронных документов является их защита от возможных видов злоумышленных действий, к которым относятся:

- *активный перехват* – нарушитель, подключившийся к сети, перехватывает документы (файлы) и изменяет их;

- *маскарад* – абонент *C* посылает документ абоненту *B* от имени абонента *A*;
- *ренегатство* – абонент *A* заявляет, что не посылал сообщения абоненту *B*, хотя на самом деле послал;
- *подмена* – абонент *B* изменяет или формирует новый документ и заявляет, что получил его от абонента *A*;
- *повтор* – абонент *C* повторяет ранее переданный документ, который абонент *A* посылал абоненту *B*.

Эти виды злоумышленных действий могут нанести существенный ущерб банковским и коммерческим структурам, государственным предприятиям и организациям, частным лицам, применяющим в своей деятельности компьютерные информационные технологии.

При обработке документов в электронной форме совершенно непригодны традиционные способы установления подлинности по рукописной подписи и оттиску печати на бумажном документе. Принципиально новым решением является электронная цифровая подпись (ЭЦП).

Электронная цифровая подпись предназначена для аутентификации лица, подписавшего электронное сообщение. Функционально она аналогична обычной рукописной подписи.

Электронная цифровая подпись обладает следующими свойствами:

- осуществляет контроль целостности передаваемого подписанного сообщения;
- предоставляет возможность доказательно подтвердить авторство лица, подписавшего сообщение;
- не дает самому этому лицу возможности отказаться от обязательств, связанных с подписанным текстом;
- обеспечивает защиту от возможностей подделки сообщения.

Цифровая подпись представляет собой относительно небольшое количество дополнительной цифровой информации, передаваемой вместе с подписываемым текстом.

Система ЭЦП включает две процедуры:

- процедуру формирования подписи;
- процедуру проверки подписи.

В процедуре постановки подписи используется секретный ключ отправителя сообщения, в процедуре проверки подписи – открытый ключ отправителя.

При формировании ЭЦП отправитель прежде всего вычисляет хэш-функцию h(*M*) подписываемого текста *M*. Вычисленное значение хэш-функции h(*M*) представляет собой один короткий блок информации *m,* характеризующий весь текст *M* в целом. Затем число *m* шифруется секретным ключом отправителя. Получаемая при этом пара чисел представляет собой ЭЦП для данного текста *M*.

При проверке ЭЦП получатель сообщения снова вычисляет хэш-функцию *m* = h(*M*) принятого по каналу текста *M*, после чего при помощи открытого ключа отправителя проверяет, соответствует ли полученная подпись вычисленному значению *m* хэш-функции.

Принципиальным моментом в системе ЭЦП является невозможность подделки ЭЦП пользователя без знания его секретного ключа подписывания.

В качестве подписываемого документа может быть использован любой файл.

Функция хэширования

Функция хэширования (хэш-функция, или дайджест-функция) представляет собой отображение, на вход которого подается сообщение переменной длины M, а выходом является строка фиксированной длины h (M). Иначе говоря, хэш-функция h (·) принимает в качестве аргумента сообщение (документ) M произвольной длины и возвращает хэш-значение h (M) фиксированной длины.

Хэш-значение h (M) – это сжатое двоичное представление (дайджест) основного сообщения M произвольной длины. Функция хэширования позволяет сжать подписываемый документ M до нескольких десятков или сотен бит (например, 128 или 256 бит), тогда как M может быть размером в мегабайт или более. Следует отметить, что значение хэш-функции h (M) зависит от документа M сложным образом и не позволяет восстановить сам документ M.

Функция хэширования может использоваться для обнаружения модификации сообщения, то есть служить в качестве криптографической контрольной суммы (также называемой кодом обнаружения изменений или кодом аутентификации сообщения). В этом качестве функция хэширования применяется при формировании и проверке электронной цифровой подписи.

Модулярная арифметика

Пусть *a* и *n* – натуральные числа. «Разделить число *a* на число *n* с остатком» – это значит найти целые числа *q* и *r*, удовлетворяющие условию

$$a = q \cdot n + r,\ 0 \le r < n. \qquad (32)$$

При этом число *q* называют неполным частным, а *r* – остатком от деления числа *a* на число *n*.

Если остаток r равен нулю, то говорят, что число n делит число a, или, по-другому, n является делителем числа a.

Целые числа a и b называются *сравнимыми по модулю* n, если их остатки при делении на n совпадают. Обычно для обозначения этого факта используется запись

$$a \equiv b \pmod{n}. \qquad (33)$$

Отсюда, в частности, следует, что число n делит разность чисел a и b. Для обозначения остатка обычно используют бесскобочную запись

$$b = a \bmod n. \qquad (34)$$

Операцию нахождения числа $b = a \bmod n$ называют *приведением числа a по модулю n.*

Конечное множество целых чисел $a_0, \ldots, a_{n-1}$, таких, что для любого целого числа b найдется $k \in \{0, \ldots, n-1\}$ со свойством $a_k \equiv b \pmod{n}$, называется *полной системой вычетов по модулю n.* Обычно используется полная система вычетов $\{0, \ldots, n-1\}$.

Эллиптические кривые

Эллиптические кривые применяются в криптографии с 1985 года. Интерес к ним обусловлен тем, что на их основе обеспечиваются те же криптографические свойства, которыми обладают числовые или полиномиальные криптосистемы, но при существенно меньшем размере ключа.

Пусть задано простое число $p > 3$. Тогда *эллиптической кривой* E, определённой над конечным простым полем F_p, называется кривая, которая задается уравнением вида

$$y^2 \equiv x^3 + ax + b \pmod{p}, \qquad (35)$$

где $a, b \in F_p$, и $4a^3 + 27b^2$ не сравнимо с нулем по модулю p.

Эллиптическая кривая E состоит из всех точек (x, y), $x, y \in F_p$, удовлетворяющих тождеству (1.4), и бесконечно удаленной точки O, которая называется *нулевой точкой*.

Точки эллиптической кривой будем обозначать $Q = (x, y)$ или просто Q. Две точки эллиптической кривой равны, если равны их соответствующие x- и y-координаты.

Инвариантом эллиптической кривой называется величина $J(E)$, удовлетворяющая тождеству

$$J(E) \equiv 1728 \cdot \frac{4a^3}{4a^3 + 27b^2} \pmod{p} \qquad (36)$$

Коэффициенты a, b эллиптической кривой E, по известному инварианту $J(E)$, определяются следующим образом

$$\begin{cases} a \equiv 3k \pmod{p}, \\ b \equiv 2k \pmod{p}, \end{cases} \quad \text{где } k \equiv \frac{J(E)}{1728 - J(E)} \pmod{p},\ J(E) \neq 0 \text{ или } 1728. \qquad (37)$$

Таким образом эллиптическая кривая E может быть однозначно задана либо своим инвариантом $J(E)$, либо коэффициентами $a, b \in F_p$.

Функция хэширования ГОСТ Р 34.11-94

Российский стандарт ГОСТ Р 34.11-94 определяет алгоритм и процедуру вычисления хэш-функции для любых последовательностей двоичных символов, применяемых в криптографических методах обработки и защиты информации. Этот стандарт базируется на блочном алгоритме шифрования ГОСТ 28147-89, хотя в принципе можно было бы использовать и другой блочный алгоритм шифрования с 64-битовым блоком и 256-битовым ключом.

Введем обозначения, которые будут использоваться в данном пункте:

$\{01\}^*$ – множество двоичных строк произвольной конечной длины;

$\{01\}^n$ – множество двоичных строк длиной n бит;

$\{0\}^n$ – двоичная строка из n нулей;

$|A|$ – длина слова A в битах;

$A \oplus B$ – побитное сложение слов A и B по mod 2, или попросту *XOR*;

$A \otimes B$ –сложение слов A и B по mod 2^k, где $k = |A| = |B|$;

$A \boxplus B$ – сложение слов A и B по mod 2^{256};

$\leftarrow$ оператор присвоения;

$||$ – конкатенация.

Сообщения с произвольной длины можно сжать используя хэш-функцию с фиксированным размером входа при помощи двух методов:

- последовательного (итерационного);
- параллельного.

Создатели ГОСТ Р 34.11-94 пошли по первому пути и использовали метод последовательного хэширования использующий хэш-функцию с фиксированным размером входа h: $\{01\}^{2n} \rightarrow \{01\}^n$ (см. рисунок 1.1), то есть функцию сжатия с коэффициентом 2.

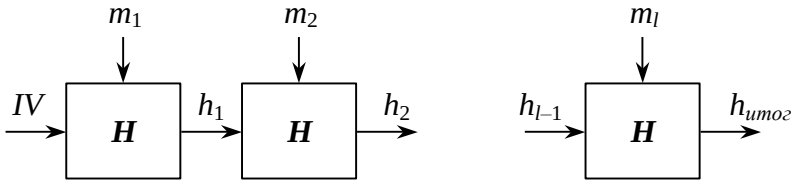


Рис. 3.1. Метод последовательного хэширования

Если необходимо хэшировать сообщение $m = (m_1, m_2, \ldots, m_l)$, то хэширование выполняется следующим образом:

$h_0 \leftarrow IV$,

$h_i \leftarrow H(m_i, h_{i-1})$, для $i = 1, 2, \ldots, l$,

$h_{итог} \leftarrow h_l$.

Здесь H – функция сжатия, а h_i – переменная сцепления, IV – начальный вектор.

Если последний блок меньше чем n бит, то он набивается одним из существующих методов до достижения длины кратной n. В отличие от стандартных предпосылок, что сообщение разбито на блоки и произведена набивка последнего блока (форматирование входа априори), если необходимо, до начала хэширования, то в ГОСТ Р 34.11-94 процедура хэширования ожидает конца сообщения (форматирование входного сообщения постериори). Набивка производится следующим образом: последний блок сдвигается вправо, а затем набивается нулями до достижения длины в 256 бит (рисунок 8.3).

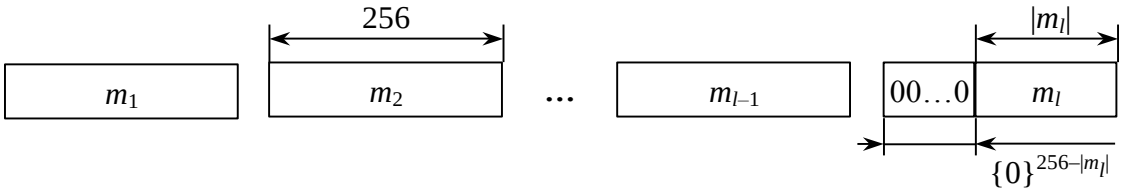
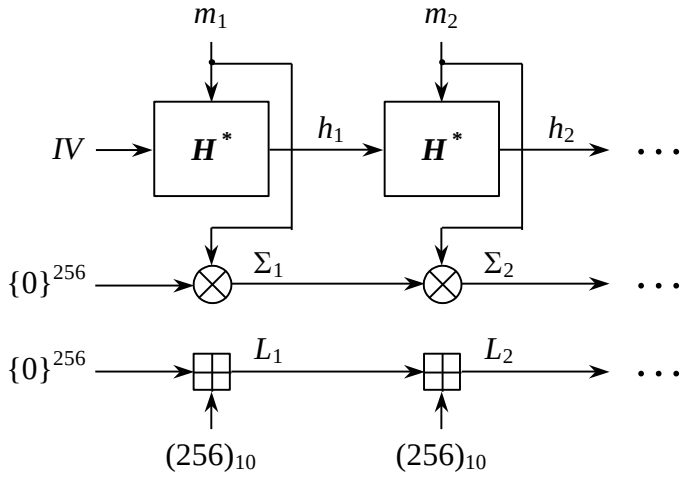


Рис. 3.2. Набивка сообщения

Указывать в передаваемом сообщении сколько было добавлено нулей к последнему блоку не требуется, так как длина сообщения участвует в хэшировании.

Параллельно рассчитываются контрольная сумма, представляющая собой сумму всех блоков сообщения (последний суммируется уже набитым) по правилу $A \otimes B$, и битовая длина хэшируемого сообщения, приводимая по mod 2^{256}, которые в финальной функции сжатия используются для вычисления итогового хэша (рисунок 1.3).



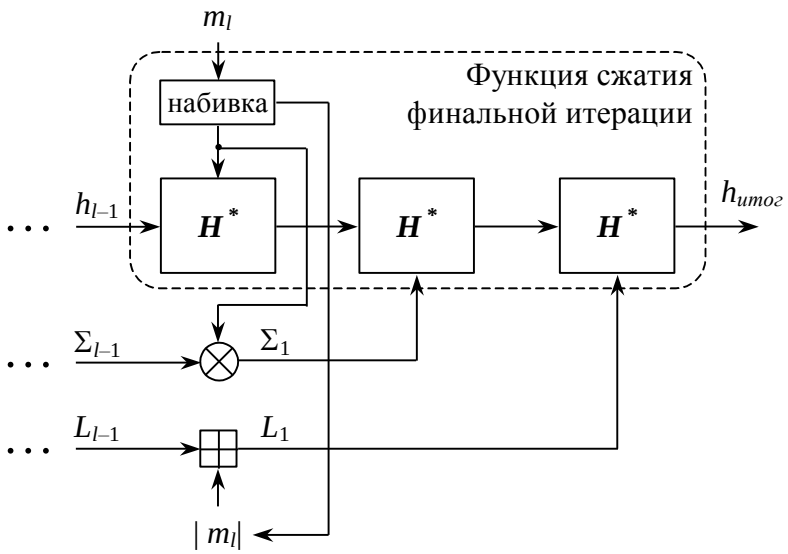


Рис. 3.3. Общая схема функции хэширования по ГОСТ Р 34.11-94

Согласно ГОСТ Р 34.11-94 *IV* – произвольное фиксированное слово длиной 256 бит ($IV \in \{01\}^{256}$). В таком случае, если он априорно не известен верифицирующему целостность сообщения, то он должен передаваться вместе с сообщением с гарантией целостности. При небольших сообщениях можно усложнить задачу противнику, если *IV* выбирается из небольшого множества допустимых величин (но при этом увеличивается вероятность угадывания хэш величины противником). Также он может задаваться в рамках организации, домена как константа (обычно как в тестовом примере).

Функция сжатия внутренних итераций

Функция сжатия внутренних итераций (по ГОСТ “шаговая функция хэширования”) *H** отображает два слова длиной 256 бит в одно слово длиной 256 бит:

$H^*: \{01\}^{256} \times \{01\}^{256} \rightarrow \{01\}^{256}$,

и состоит из трех процедур (рисунок 3.5):

- перемешивающего преобразования;
- шифрующего преобразования;
- генерирования ключей.

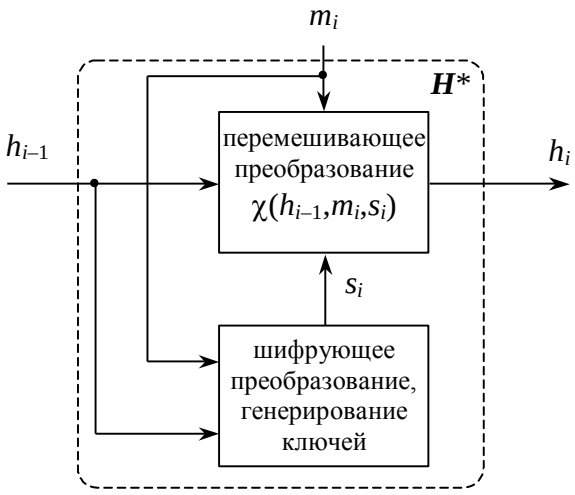


Рис. 3.4. Структура функции сжатия в ГОСТ Р 34.11-94

Генерирования ключей

Данная подпрограмма использует следующие функции и константы:

- Константы

$C_2 = C_4 = \{0\}^{256}$ и $C_3 = 1^8 0^8 1^{16} 0^{24} 1^{16} 0^8 (0^8 1^8)^2 1^8 0^8 (0^8 1^8)^4 (1^8 0^8)^4$,

где степень обозначает количество повторений 0 или 1.

- Функции

  - P: $\{01\}^{256} \to \{01\}^{256}$.

  Пусть $X = \xi_{32} \| \xi_{31} \| \dots \| \xi_1$,

  тогда $P(X) = \xi_{\varphi(32)} \| \xi_{\varphi(31)} \| \dots \| \xi_{\varphi(1)}$, где $\varphi(i+1+4(k-1)) = 8i+k$, $i = 0\dots3$, $k = 1\dots8$.

  - A: $\{01\}^{256} \to \{01\}^{256}$.

  Пусть $X = x_4 \| x_3 \| x_2 \| x_1$,

  тогда $A(X) = (x_1 \oplus x_2) \| x_4 \| x_3 \| x_2$.

При вычислении ключей реализуется следующий алгоритм (рисунок 1.5):

1. $j \leftarrow 1$, $V \leftarrow m_i$, $U \leftarrow h_{i-1}$;
2. $W \leftarrow U \oplus V$, $K_1 \leftarrow P(W)$;
3. $j \leftarrow j + 1$;
4. Проверить условие $j = 5$. При положительном исходе перейти к шагу 7, при отрицательном – к шагу 5;
5. $U \leftarrow A(U) \oplus C_j$, $V \leftarrow A(A(V))$, $W \leftarrow U \oplus V$, $K_j \leftarrow P(W)$;
6. Перейти к шагу 3;
7. Выход (K_1, K_2, K_3, K_4).

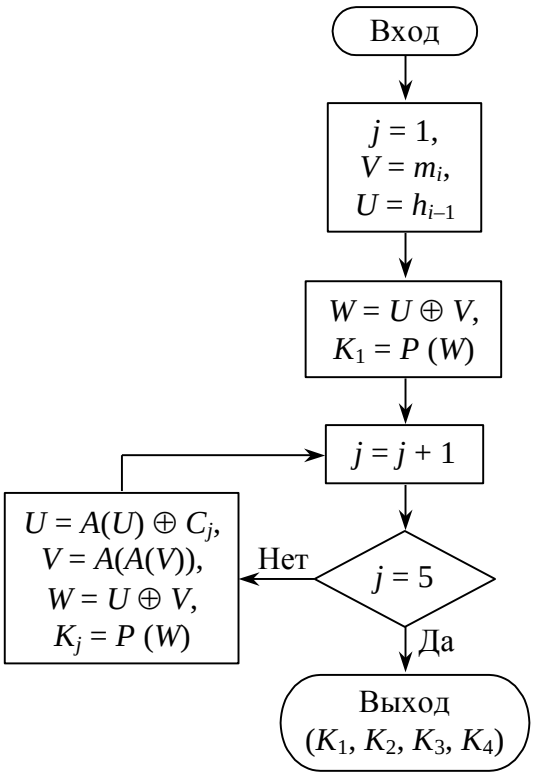


Рис. 3.5. Алгоритм генерирования ключей

Шифрующее преобразование

Основным функциональным предназначением является получение s_i из h_{i-1}.

Пусть $h_{i-1} = h_{i-1}^4 \parallel h_{i-1}^3 \parallel h_{i-1}^2 \parallel h_{i-1}^1$, где $h_{i-1}^j \in \{01\}^{64}, j = 1\ldots4$, а

$s_i = s_i^4 \parallel s_i^3 \parallel s_i^2 \parallel s_i^1$, где $s_i^j \in \{01\}^{64}, j = 1\ldots4$.

Тогда

$s_i^j \leftarrow E_{K_j}(h_{i-1}^j)$,

где $j = 1\ldots4$, E_{K_j} – шифрование по ГОСТ 28147-89 в режиме простой замены.

Схематически это изображено на рисунке 1.6.

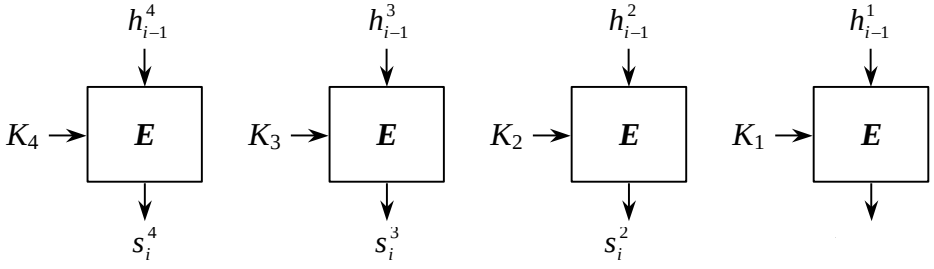


Рис. 3.6. Шифрующее преобразование

ГОСТ 28147-89 предусматривает три следующих режима шифрования данных:

- простая замена,
- гаммирование,

- гаммирование с обратной связью,
- и один дополнительный режим выработки имитовставки.

В любом из этих режимов данные обрабатываются блоками по 64 бита, на которые разбивается массив, подвергаемый криптографическому преобразованию, именно поэтому ГОСТ относится к блочным шифрам.

Если внимательно изучить оригинал ГОСТа 28147-89, можно заметить, что в нем содержится описание алгоритмов нескольких уровней. На самом верхнем находятся практические алгоритмы, предназначенные для шифрования массивов данных и выработки для них имитовставки. Все они опираются на три алгоритма низшего уровня, называемые в тексте ГОСТа *циклами*. Эти фундаментальные алгоритмы упоминаются в данной работе как *базовые циклы*, чтобы отличать их от всех прочих циклов. Они имеют следующие названия и обозначения:

- $Ц_{32\text{-З}}$ – цикл зашифрования (32-З);
- $Ц_{32\text{-Р}}$ – цикл расшифрования (32-Р);

В свою очередь, каждый из базовых циклов представляет собой многократное повторение одной единственной процедуры, называемой для определенности далее в настоящей работе *основным шагом криптопреобразования*.

В соответствии с принципом Кирхгофа, которому удовлетворяют все современные известные широкой общественности шифры, секретность зашифрованного сообщения обеспечивается секретностью ключевой информации. В ГОСТе ключевая информация состоит из двух структур данных. Помимо собственно *ключа*, необходимого для всех шифров, она содержит еще и *таблицу замен*:

1. **Ключ** является массивом из восьми 32-битных элементов кода: $K = \{K_i\}_{0 \le i \le 7}$. В ГОСТе элементы ключа используются как 32-разрядные целые числа без знака: $0 \le K_i < 2^{32}$. Таким образом, размер ключа составляет 32·8=256 бит или 32 байта.

2. **Таблица замен** является матрицей 8×16, содержащей 4-битовые элементы, которые можно представить в виде целых чисел от 0 до 15: $H = \{H_{i,j}\}_{\substack{0 \le i \le 7 \\ 0 \le j \le 15}}$, $0 \le H_{i,j} \le 15$. Строки *таблицы замен* называются *узлами замен*, они должны содержать различные значения, то есть каждый узел замен должен содержать 16 различных чисел от 0 до 15 в произвольном порядке. Таким образом, общий объем таблицы замен равен: 8 узлов × 16 элементов/узел × 4 бита/элемент = 512 бит или 64 байта.

Основной шаг криптопреобразования по своей сути является оператором, определяющим преобразование 64-битового блока данных. Дополнительным параметром этого оператора является 32-битовый блок, в качестве которого используется какой-либо элемент ключа. Схема алгоритма основного шага приведена на рисунке 1.7.

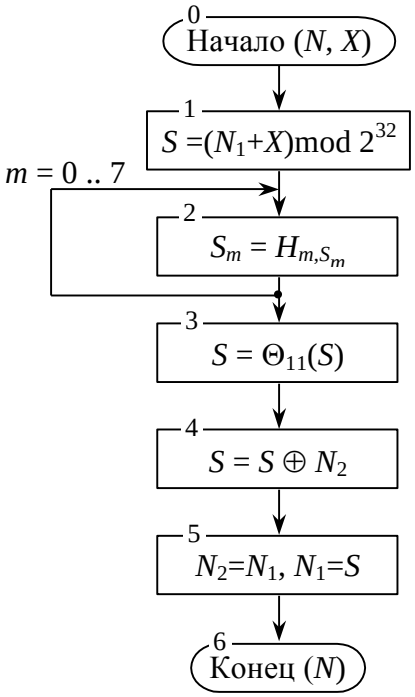


Рис. 3.7. Схема основного шага криптопреобразования
алгоритма ГОСТ 28147-89

Ниже даны пояснения к алгоритму основного шага:

0. Определяет исходные данные для основного шага криптопреобразования:

- N – преобразуемый 64-битовый блок данных, в ходе выполнения шага его младшая (N_1) и старшая (N_2) части обрабатываются как отдельные 32-битовые целые числа без знака. Таким образом, можно записать $N = (N_1, N_2)$.
- X – 32-битовый элемент ключа.

1. Сложение с ключом. Младшая половина преобразуемого блока складывается по модулю 2^{32} с используемым на шаге элементом ключа, результат передается на следующий шаг.

2. Поблочная замена. 32-битовое значение, полученное на предыдущем шаге, интерпретируется как массив из восьми 4-битовых блоков кода:

$S = (S_7, S_6, S_5, S_4, S_3, S_2, S_1, S_0)$.

Далее значение каждого из восьми блоков заменяется на новое, которое выбирается по таблице замен следующим образом: значение блока S_i заменяется на S_i-тый по порядку элемент (нумерация с нуля) i-того узла замен (т.е. i-той строки

таблицы замен, нумерация также с нуля). Другими словами, в качестве замены для значения блока выбирается элемент из таблицы замен с номером строки, равным номеру заменяемого блока, и номером столбца, равным значению заменяемого блока как 4-битового целого неотрицательного числа. Теперь становится понятным размер таблицы замен: число строк в ней равно числу 4-битных элементов в 32-битном блоке данных, то есть восьми, а число столбцов равно числу различных значений 4-битного блока данных, равному как известно 2^4, шестнадцати.

3. Циклический сдвиг на 11 бит влево. Результат предыдущего шага сдвигается циклически на 11 бит в сторону старших разрядов и передается на следующий шаг. На схеме алгоритма символом Θ_{11} обозначена функция циклического сдвига своего аргумента на 11 бит в сторону старших разрядов.

4. Побитовое сложение: значение, полученное на шаге 3, побитно складывается по модулю 2 со старшей половиной преобразуемого блока.

5. Сдвиг по цепочке: младшая часть преобразуемого блока сдвигается на место старшей, а на ее место помещается результат выполнения предыдущего шага.

6. Полученное значение преобразуемого блока возвращается как результат выполнения алгоритма основного шага криптопреобразования.

Базовые циклы заключаются в многократном выполнении *основного шага* с использованием разных элементов ключа и отличаются друг от друга только числом повторения шага и порядком использования ключевых элементов. Ниже приведен этот порядок для различных циклов.

1. Цикл зашифрования 32-З:

$K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0$.

2. Цикл расшифрования 32-Р:

$K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0$.

Каждый из циклов имеет собственное буквенно-цифровое обозначение, где первый элемент обозначения, задает число повторений основного шага в цикле, а второй элемент обозначения, буква, задает порядок зашифрования («З») или расшифрования («Р») в использовании ключевых элементов.

Цикл расшифрования должен быть обратным циклу зашифрования, то есть последовательное применение этих двух циклов к произвольному блоку должно дать в итоге исходный блок, что отражается следующим соотношением:

$Ц_{32\text{-}Р}(Ц_{32\text{-}З}(T))=T$,

где T – произвольный 64-битный блок данных,

$Ц_{X}(T)$ – результат выполнения цикла X над блоком данных T.

Для выполнения этого условия для алгоритмов, подобных ГОСТу, необходимо и достаточно, чтобы порядок использования ключевых элементов соответствующими циклами был взаимно обратным. Из двух взаимно обратных циклов любой может быть использован для зашифрования, тогда второй должен быть использован для расшифрования данных, однако стандарт ГОСТ28147-89 закрепляет роли за циклами и не предоставляет пользователю права выбора в этом вопросе.

Схемы базовых циклов приведены на рисунках 1.8а, 1.8б. Каждый из них принимает в качестве аргумента и возвращает в качестве результата 64-битный блок данных, обозначенный на схемах N.

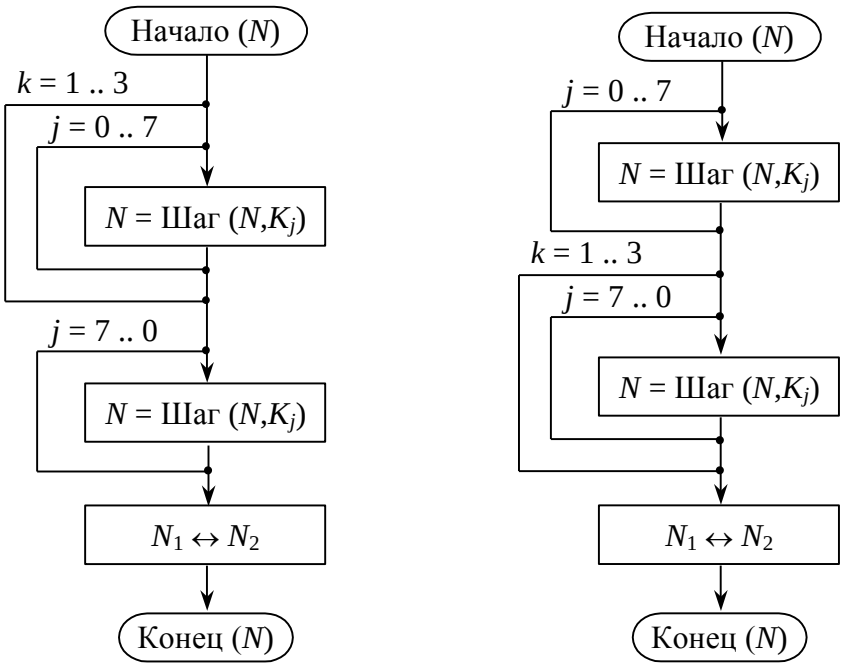


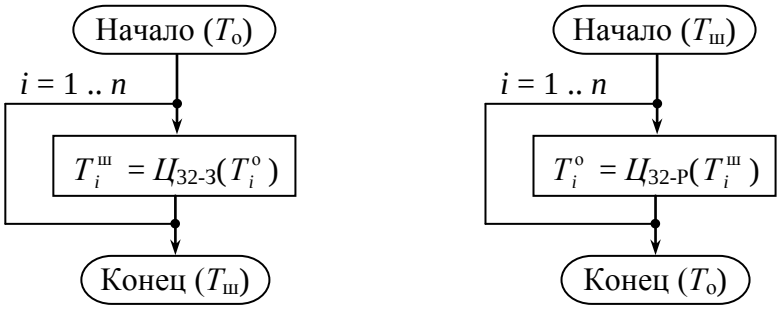
Рис. 3.8. Схемы цикла зашифрования 32-З (а) и расшифрования 32-Р (б)

Символ Шаг(N,X) обозначает выполнение основного шага криптопреобразования для блока N с использованием ключевого элемента X.

В конце базовых циклов шифрования старшая и младшая часть блока результата меняются местами, это необходимо для их взаимной обратимости.

Как уже упоминалось выше, нас интересует шифрование в режиме простой замены. Зашифрование в режиме простой замены заключается в применении цикла 32-З к блокам открытых данных, расшифрование – цикла 32-Р к блокам зашифрованных данных. Это наиболее простой из режимов, 64-битовые блоки данных обрабатываются в нем независимо

друг от друга. Схемы алгоритмов зашифрования и расшифрования в режиме простой замены приведены на рисунках 3.10а и 3.10б соответственно.



а) б)

Рис. 3.9. Алгоритмы зашифрования (а) и расшифрования (б)

данных в режиме простой замены

На рисунке 3.10 используются следующие обозначения:

- $T_о$, $T_ш$ – массивы соответственно открытых и зашифрованных данных;
- $T_i^о$, $T_i^ш$ – i-тые по порядку 64-битные блоки соответственно открытых и зашифрованных данных: $T_о = (T_n^о, \ldots, T_2^о, T_1^о)$, $T_ш = (T_n^ш, \ldots, T_2^ш, T_1^ш)$, $1 \le i \le n$;
- n – число 64-битных блоков в массиве данных.

Размер массива открытых или зашифрованных данных, подвергающийся соответственно зашифрованию или расшифрованию, должен быть кратен 64 битам: $|T_о| = |T_ш| = 64 \cdot n$, после выполнения операции размер полученного массива данных не изменяется.

Перемешивающее преобразование

Перемешивающее преобразование имеет вид

$h_i = \chi(m_i, h_{i-1}, s_i) = \psi^{61}(h_{i-1} \oplus \psi(m_i \oplus \psi^{12}(s_i)))$,

где ψ^j – j-я степень преобразования ψ.

Схематически данное преобразование представлено на рисунке 3.10.

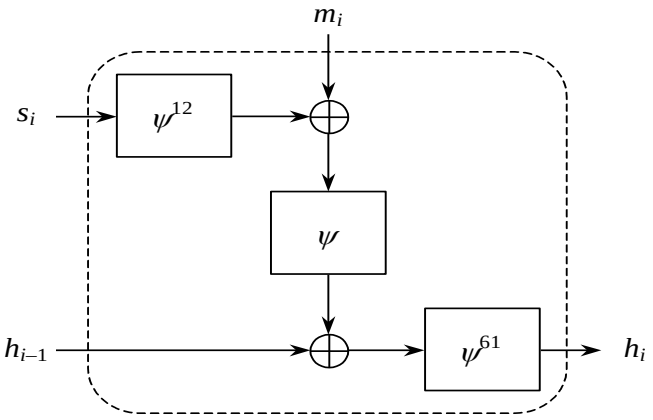


Рис. 3.10. Перемешивающее преобразование ГОСТ Р 34.11-94

Заметим, что s_i выводится из h_{i-1} (см. рисунок 3.4).

Функция ψ: $\{01\}^{256} \rightarrow \{01\}^{256}$ преобразует слово $\eta_{16} \| \eta_{15} \| \ldots \| \eta_1 \in \{01\}^{16}$, i =1…16 в слово $\eta_1 \oplus \eta_2 \oplus \eta_3 \oplus \eta_4 \oplus \eta_{13} \oplus \eta_{16} \| \eta_{16} \| \eta_{15} \| \ldots \| \eta_2$ (рисунок 1.11).

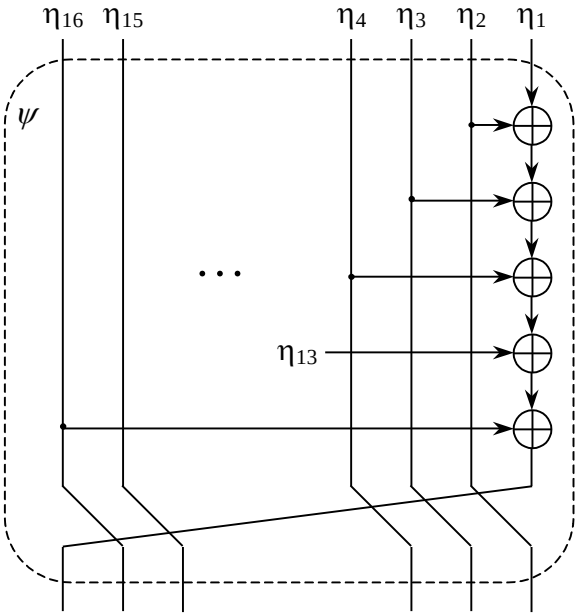


Рис. 3.11. Функция ψ перемешивающего преобразования

Функция сжатия финальной итерации

Функция сжатия финальной итерации производит итоговую хэш величину, зависящую от результата хэширования последовательным методом, контрольной суммы по mod 2 и длины сообщения в битах приведенной по mod 2^{256}:

$m_l^{/} \times h_{l-1} \times \sum_l \times L_l \rightarrow h_{итог}$,

где $m_l^{/}$, h_{l-1}, $\sum_l$, $L_l \in \{01\}^{256}$, $m_l^{/}$ – последний набитый нулями блок (если необходимо, см. рисунок 1.2).

Сначала вычисляется битовая длина последнего (ненабитого) блока сообщения $|m_l| \leq 256$ бит. Если $|m_l| < 256$, то производится его набивка справа нулями до достижения длины 256 бит и получается новый блок $m_l^{/} \leftarrow \{0\}^{256-|m_l|} \| m_l$. Вычисляются итоговая контрольная сумма $\sum_l \leftarrow \sum_{l-1} \otimes m_l^{/}$ и длина всего сообщения $L_l \leftarrow L_{l-1} + |m_l| \pmod{2^{256}}$. Параллельно выполняется последняя итерация $h_l = H^*(m^{/}, h_{l-1})$. Затем вычисляются значения $h_{l+1} \leftarrow H^*(\sum_l, h_l)$ и $h_{итог} \leftarrow H^*(L_l, h_{l+1})$, давая в результате итоговую хэш величину $h_{итог}$.

Теперь дадим формальное описание алгоритма:

Вход: двоичное сообщение M, блоки замен для шифрования в режиме простой замены по ГОСТ 28147-89, начальный вектор $IV \in \{01\}^{256}$.

Выход: хэш величина $h_{итог}$ для сообщения M.

1. *Инициализация алгоритма*: $h \leftarrow IV$, $\sum \leftarrow \{0\}^{256}$, $L \leftarrow \{0\}^{256}$.

2. *Функция сжатия внутренних итераций*: Пока |M| > 256 выполняем следующее:

2.1. $h \leftarrow H^*(m_s, h)$ (итерация метода последовательного хэширования);

2.2. $L \leftarrow L + 256 \pmod{2^{256}}$ (итерация вычисления длины сообщения);

2.3. $\sum \leftarrow \sum \otimes m_s$ (итерация вычисления контрольной суммы).

3. Иначе (*функция сжатия финальной итерации*)

3.1. $L \leftarrow L + |m| \pmod{2^{256}}$ (вычисление полной длины сообщения);

3.2. $m' \leftarrow \{0\}^{256-|m|} \| m$ (набивка последнего блока);

3.3. $\sum \leftarrow \sum \otimes m'$ (вычисление контрольной суммы сообщения);

3.4. $h \leftarrow H^*(m', h)$;

3.5. $h \leftarrow H^*(\sum, h)$;

3.6. $h_{итог} \leftarrow H^*(L, h)$.

4. *Выход* $(h_{итог})$.

Российский стандарт ЭЦП ГОСТ Р 34.10-2001

Новый отечественный стандарт цифровой подписи обозначается как ГОСТ Р 34.10-2001. В нем используются следующие параметры:

- простое число p – модуль эллиптической кривой, удовлетворяющее неравенству $p>2^{255}$. Верхняя граница данного числа должна определяться при конкретной реализации схемы цифровой подписи;
- эллиптическая кривая E, задаваемая своим инвариантом $J(E)$ или коэффициентами $a, b \in F_p$;
- целое число m – порядок группы точек эллиптической кривой E;
- простое число q – порядок циклической подгруппы группы точек эллиптической кривой E, для которого выполнены следующие условия:

$$\begin{cases} m = nq, \quad n \in Z, \quad n \geq 1, \\ 2^{254} < q < 2^{256}; \end{cases} \tag{38}$$

- точка $P \neq O$ эллиптической кривой E, с координатами (x_p, y_p), удовлетворяющая равенству $qP = O$.
- хэш-функция, отображающая сообщения, представленные в виде двоичных векторов произвольной конечной длины, в двоичные вектора длины 256 бит. Хэш-функция определена в ГОСТ Р 34.11, ее алгоритм будет рассмотрен ниже.

Каждый пользователь схемы цифровой подписи должен обладать личными ключами:

- ключом подписи – целым числом d, удовлетворяющим неравенству $0 < d < q$;
- ключом проверки – точкой эллиптической кривой Q с координатами (x_q, y_q), удовлетворяющей равенству $dP = Q$.

На приведенные выше параметры схемы цифровой подписи накладываются следующие требования:

- должно быть выполнено условие $p^t \neq 1 \pmod q$, для всех целых $t = 1, 2, \ldots B$, где B удовлетворяет неравенству $B \geq 31$;
- должно быть выполнено неравенство $m \neq p$;
- инвариант кривой должен удовлетворять условию $J(E) \neq 0$ или 1728.

Для получения цифровой подписи под сообщением M необходимо выполнить следующие действия (рис. 1.12а):

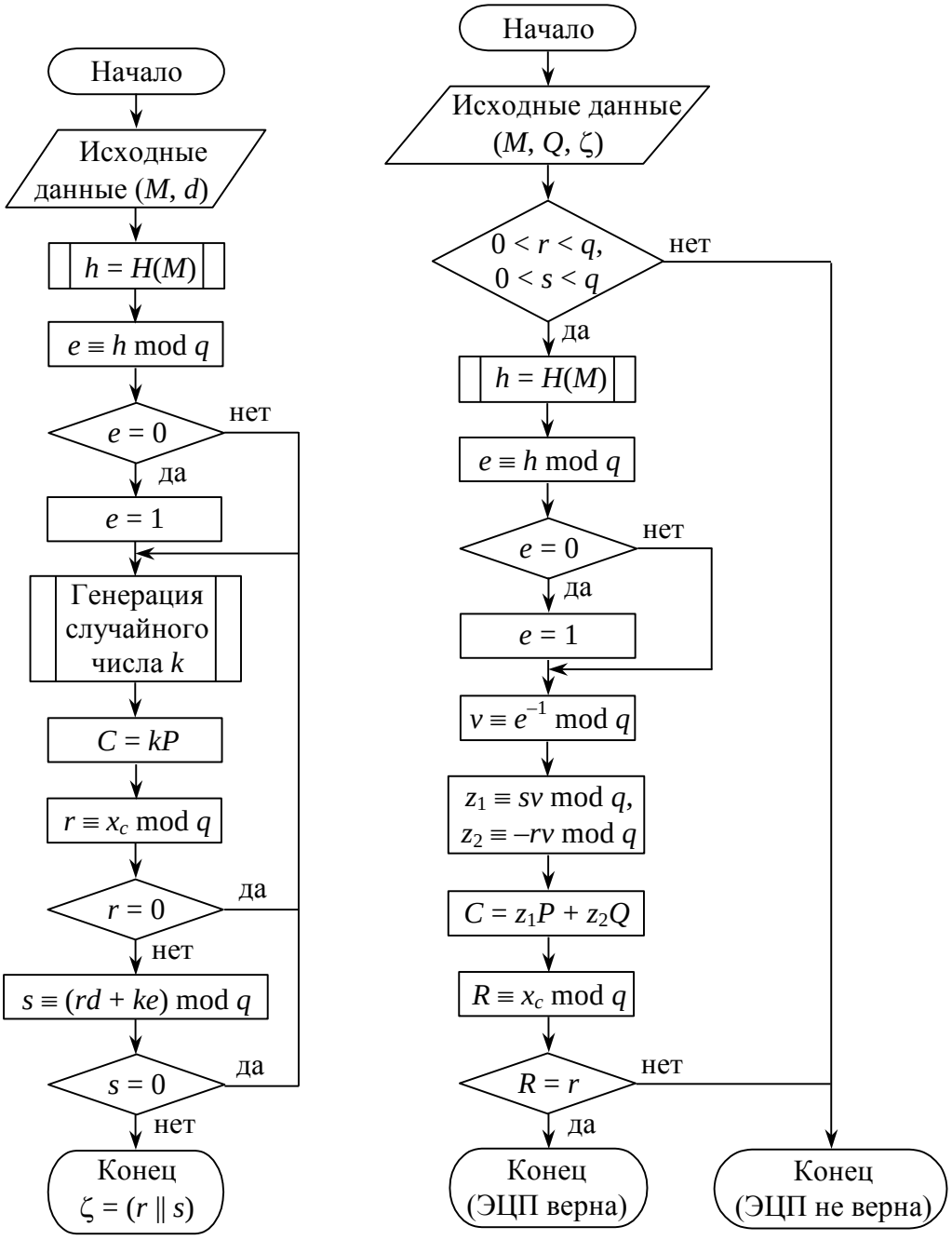
1. вычислить хэш-значение сообщения M: $\bar{h} = h(M)$;
2. вычислить целое число α, двоичным представлением которого является вектор $\bar{h}$, и определить $e \equiv \alpha \pmod q$. Если $e = 0$, то определить $e = 1$;
3. сгенерировать случайное (псевдослучайное) целое число k, удовлетворяющее неравенству $0 < k < q$;
4. вычислить точку эллиптической кривой $C = kP$ и определить $r \equiv x_c \pmod q$, где x_c - x-координата точки C. Если $r = 0$, то вернуться к шагу 3;
5. вычислить значение $s \equiv (rd + ke) \pmod q$. Если $s = 0$, то вернуться к шагу 3;

Подпись для сообщения M составит пара чисел (r,s).

Для проверки цифровой подписи ζ под полученным сообщением M необходимо выполнить следующие действия (рис. 1.12б):

1. Если выполнены неравенства $0 < r < q$, $0 < s < q$, то перейти к следующему шагу. В противном случае подпись неверна;
2. вычислить хэш-значение полученного сообщения M: $\bar{h} = h(M)$;
3. вычислить целое число α, двоичным представлением которого является вектор $\bar{h}$ и определить $e \equiv \alpha \pmod q$. Если $e = 0$, то определить $e = 1$;
4. вычислить значение $v = e^{-1} \pmod q$;
5. вычислить значения $z_1 \equiv sv \pmod q$, $z_2 \equiv -rv \pmod q$;
6. вычислить точку эллиптической кривой $C = z_1P + z_2Q$ и определить $R \equiv x_c \pmod q$, где x_c – x-координата точки C;

7. если выполнено равенство $R = r$, то подпись принимается, в противном случае, подпись неверна.



а) б)

Рис. 3.12. Процесс формирования цифровой подписи (а), процесс проверки цифровой подписи (б).

Интерфейс учебного программного комплекса DigitSign

На рисунке 8.13 приведен интерфейс главного окна учебного программного комплекса.

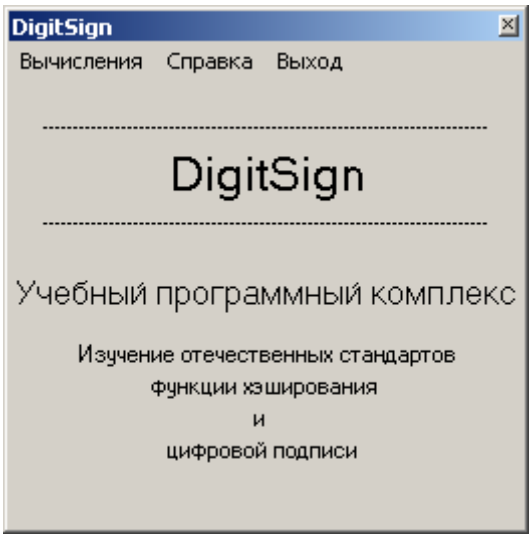


Рис. 3.13. Интерфейс главного окна учебного программного комплекса

Пункт меню “Вычисления” содержит следующие опции:

- Вычисление хэш-функции.

При выборе данной опции открывается окно “Вычисление хэш-функции” (рисунок 8.15), которое позволяет определить хэш-значение произвольного файла при заданных пользователем начальном хэш-векторе и таблице замен. Полученный результат может быть сохранен в файл. Также пользователь получает информацию о времени хэширования сообщения.

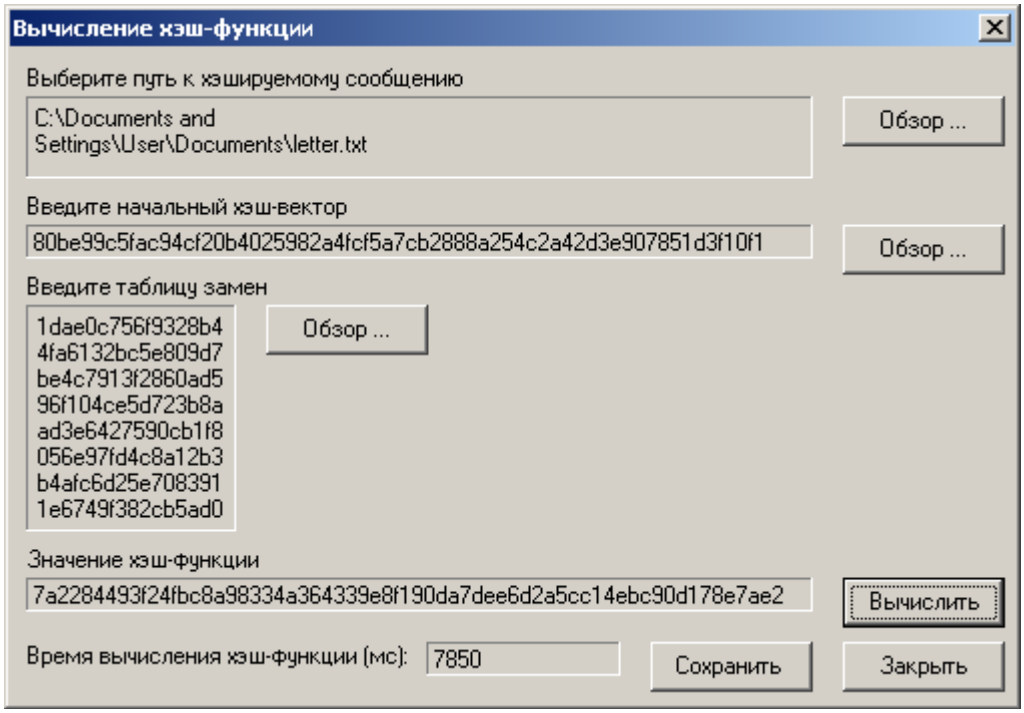


Рис. 3.14. Окно “Вычисление хэш-функции”

- Генератор псевдослучайных чисел.

При выборе данной опции открывается окно “Генератор псевдослучайных чисел” (рисунок 3.16).

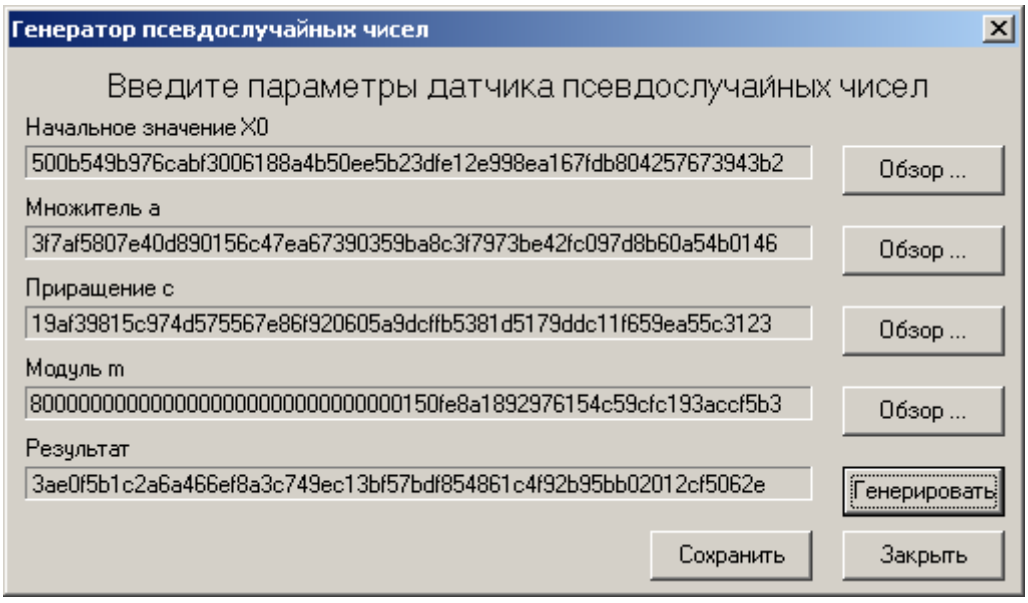


Рис. 3.15. Окно “Генератор псевдослучайных чисел”

Пользователь имеет возможность ввести параметры своего генератора, произвести вычисления и сохранить полученный результат в файл.

- Вычисление открытого ключа.

При выборе данной опции открывается окно “Вычисление открытого ключа” (рисунок 8.16).

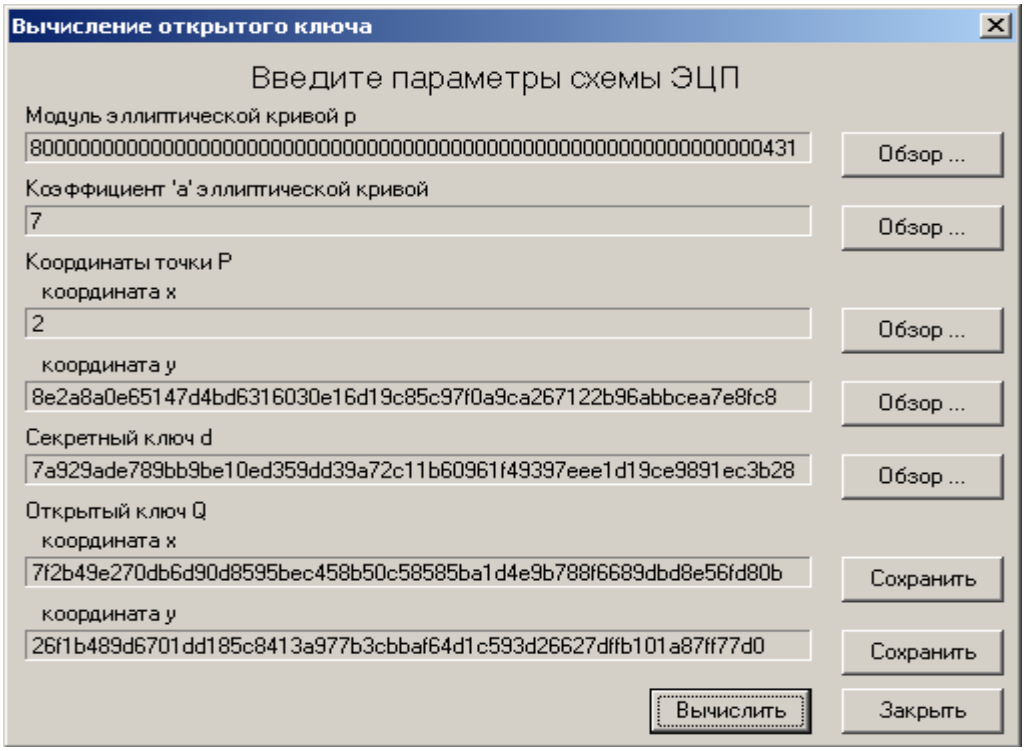


Рис. 3.16. Окно “Вычисление открытого ключа”

Пользователь имеет возможность вычислить открытый ключ для имеющегося у него секретного ключа и параметров схемы ЭЦП.

- Формирование ЭЦП.

При выборе данной опции открывается окно “Формирование ЭЦП” (рисунок 3.17).

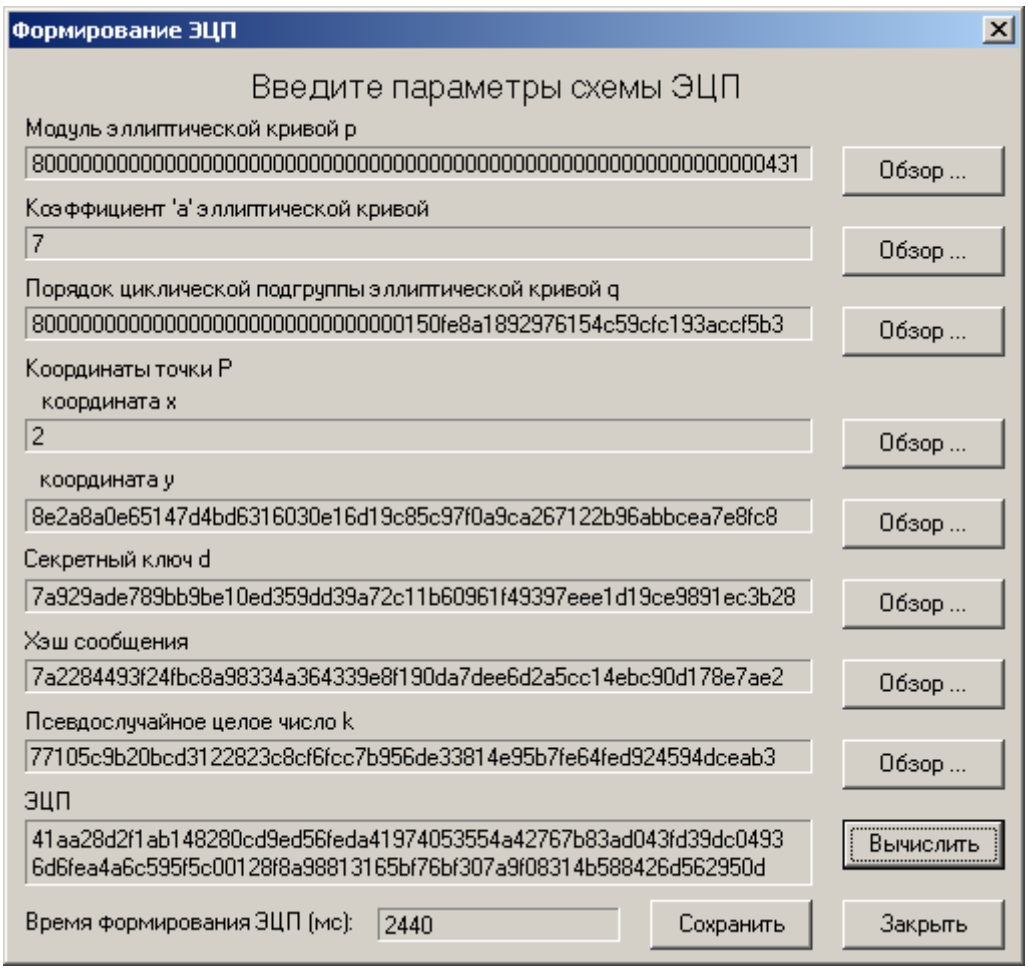


Рис. 3.17. Окно “Формирование ЭЦП”

Пользователь имеет возможность, после ввода всех необходимых для расчетов параметров, вычислить цифровую подпись и сохранить полученный результат в файл.

- Проверка ЭЦП.

При выборе данной опции открывается окно “Проверка ЭЦП” (рисунок 8.18).

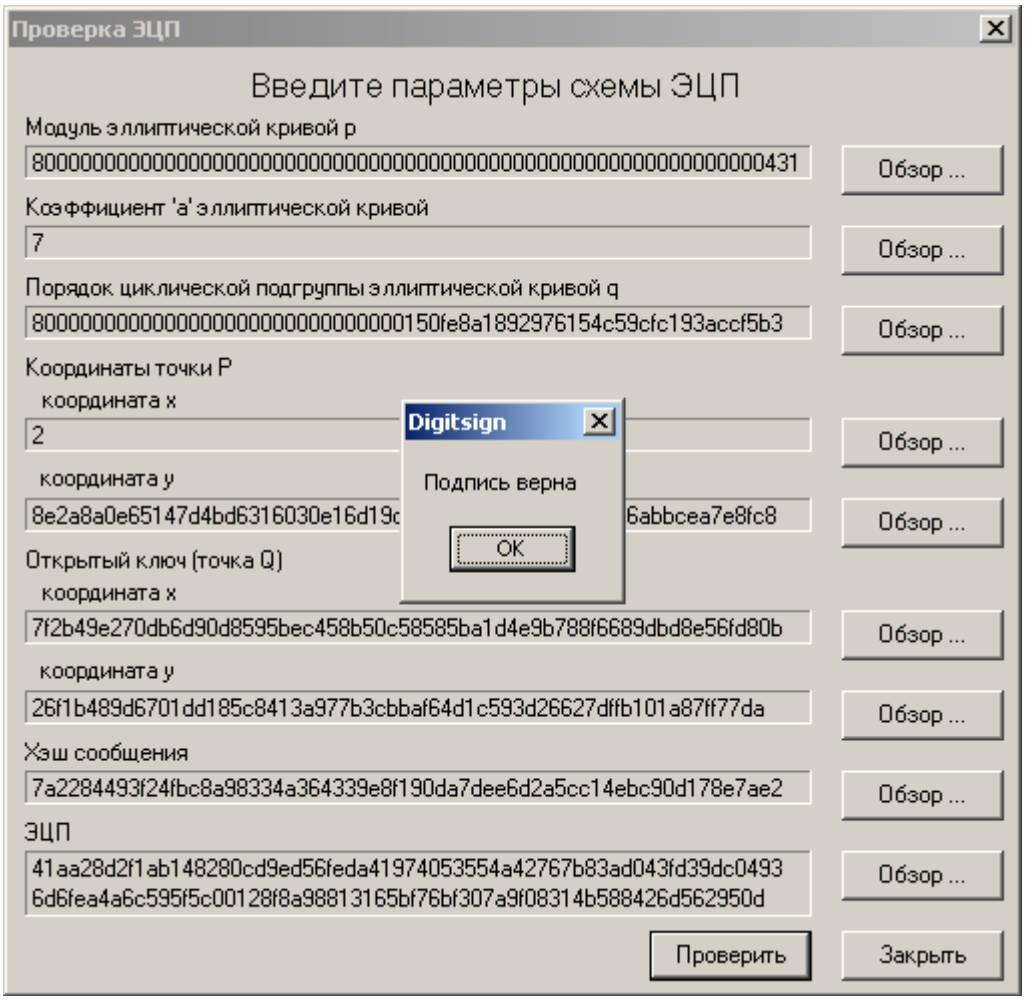


Рис. 3.18. Окно “Проверка ЭЦП”

Пользователь имеет возможность, после ввода всех необходимых для расчетов параметров, проверить подлинность цифровой подписи.

Пункт меню “Справка” содержит следующие опции:

- Задание на лабораторную работу.

Данная опция позволяет открыть файл справки, содержащий задание на лабораторную работу.

- Справка.

Данная опция позволяет открыть файл справки, содержащий информацию о хэш-функции и цифровых подписях.

- О программе.

Пункт меню “Выход” позволяет выйти из программы.

Порядок выполнения работы

Во время выполнения лабораторной работы делайте скриншоты. После вставить в отчет.

1. Ознакомьтесь с теорией по функциям хэширования, эллиптическим кривым и цифровым подписям.
2. Ознакомьтесь с ГОСТ Р 34.11-94 и ГОСТ Р 34.10-2001.
3. Откройте папку DigitSign и запустите программу DigitSign.exe.
4. Изучение функции хэширования по ГОСТ Р 34.11-94.

4.1 Открываем пункт меню «Вычисления» и выбираем «Вычисление хэш-функции».

4.2 Задать начальные данные для вычисления хэш-функции (файлы находятся в папке «Хэширование»):

- путь к файлу, который подвергается сжатию (название text0.txt);
- начальный хэш-вектор (HeshVHex.txt);
- таблицу замен (TabZamen.txt).

4.3 Вычислить хэш-функцию.

4.4 Повторить пункты 4.2 – 4.3 для файлов с названием text1, text2, text3, text4, text5. Сохранить полученные значения хэш-функции в папку «Временное», назвать hash1.txt, hash2.txt, hash3.txt, hash4.txt, hash5.txt соответственно. Определить время хэширования. Полученные результаты занести в первые две строки таблицы (l – размер текстового файла, t_h – время хэширования, для text0 делать данный пункт не обязательно). ВНИМАНИЕ, хэширование может производиться до 3х минут, лучше в этот момент не чего не делать на компьютере, иначе возможно получение недостоверных значений.

l, Мбайт					
t_h, мс					
$t_{ЭЦП}$, мс					
t_{Σ}, мс					

5. Изучение цифровой подписи ГОСТ Р 34.10-2001.

5.1 Вычислить секретный ключ.

5.1.1 Открываем пункт меню «Вычисления» и выбираем «Генератор псевдослучайных чисел».

5.1.2 Ввести параметры датчика (файлы находятся в папке «ГПСЧ»):

- начальное значение X_0 (X0.txt);

- множитель *a* (a.txt);
- приращение *c* (c.txt);
- модуль *m* (m.txt).

5.1.3 Вычислить псевдослучайное число, оно же является секретным ключом.

5.1.4 Сохранить полученное значение в папку «Временное», назвав «KeySekr.txt».

5.2 Вычислить открытый ключ.

5.2.1. Открываем пункт меню «Вычисления» и выбираем “Вычисление открытого ключа”.

5.2.2. Задать параметры, необходимые для вычисления (файлы находятся в папке «Открытый ключ»).

- Модуль эллиптической кривой p (modP.txt);
- Коэффициент эллиптической кривой a (kofA.txt);
- Координаты точки P (координата x – Px.txt, координата y – Py.txt);
- Секретный ключ d (ранее полученный в пункте 5.1.4).

5.2.3. Вычислить открытый ключ.

5.2.4. Сохранить полученное значение в папку «Временное», назвав «Qx.txt» для координаты X и «Qy.txt» для координаты Y.

5.3 Вычислить цифровую подпись.

5.3.1. Выполнить предварительное вычисление случайного числа *k*, необходимого для формирования цифровой подписи. Для этого следует выполнить п. 5.1. Сохранить результат в папку «Временное», назвав «psK.txt».

5.3.2. Открываем пункт меню «Вычислить» и выбираем «Формирование цифровой подписи».

5.3.3. Задать параметры, необходимые для вычисления (файлы находятся в папке «Открытый ключ»):

- Модуль эллиптической кривой p (modP.txt);
- Коэффициент эллиптической кривой a (kofA.txt);
- Порядок циклической подгруппы эллиптической кривой q (q.txt);

- Координаты точки P (координата X – Px.txt, координата Y – Py.txt);
- Секретный ключ d (ранее полученный в пункте 5.1.4);
- Хэш сообщения (ранее полученное в пункте 4.4, с названием hash1.txt);
- Псевдо случайное число k (ранее полученное в пункте 5.4.1).

5.3.4. Сформировать цифровую подпись и занести в таблицу время создания ЭЦП (строка $t_{ЭЦП}$).

5.3.5. Сохранить значение ЭЦП в папку «Временное», назвав «podpis.txt»

5.3.6. Повторить пункты 5.4.3-5.4.4 для хэш-значений hash2.txt; hash3.txt; hash4.txt; hash5.txt. Обратите внимание на то, что пункт 5.4.5 выполнять повторно не нужно.

5.3.7. Рассчитать суммарное время вычисления ЭЦП и занести результаты расчетов в таблицу ($t_{\Sigma}= t_h+ t_{ЭЦП}$).

5.4 Передайте другому студенту, который сидит за другим компьютером, открытый ключ (Qx.txt; Qy.txt), файл (text1.txt) и соответствующую ему цифровую подпись (podpis.txt) при помощи дискеты или сети и получите от него такой же набор. Полученные файлы переименовать:

- «Qx.txt» в «QxPol.txt»;
- «Qy.txt» в «QyPol.txt»;
- «text1.txt» в «text1Pol.txt»;
- «podpis.txt» в «podpisPol.txt»;
- И поместить в папку «Временное».

5.5 Проверить цифровую подпись.

5.5.1 Сделать пункт 4, для файла text1Pol.txt, результат сохранить в папку «Временное», назвав «hashPol.txt»;

5.5.2 Открываем пункт меню «Вычислить», и выбираем «Проверка цифровой подписи».

5.5.3 Задать параметры, необходимые для вычисления (файлы находятся в папке «Открытый ключ»):

- Модуль эллиптической кривой p (modP.txt);
- Коэффициент эллиптической кривой a (kofA.txt);

- Порядок циклической подгруппы эллиптической кривой q (q.txt);
- Координаты точки P (координата X – «Px.txt», Y – «Py.txt»);
- Открытый ключ Q (полученный в пункте 5.4, координата X – «QxPol.txt»; координата Y – «QyPol.txt»);
- Хэш сообщения (поученный в пункте 5.4.1, «hashPol.txt»);
- ЭЦП (полученное в пункте 5.4, «podpisPol.txt»).

5.5.4 Осуществить проверку.

ВНИМАНИЕ! Данная программа неверно вычисляет открытый ключ Q, координату y. Если «Подпись не верна»: открываете файл QyPol.txt через блокнот и добавляете к **последней** цифре единицу (программа все время выдает последнюю цифру равную 0) и снова проверяете, если опять неверна, то еще добавляете единицу до тех пор пока подпись не будет верна или пока не достигнете f и подпись по прежнему не верна, тогда подпись действительно не верна (программа использует 16-ти ричную систему счисления, {0; 1; 2; 3; 4; 5; 6; 7; 8; 9; a; b; c; d; e; f}).

5.6 Подделка сообщения.

5.6.1. Откройте файл «text1Pol.txt» (полученный в пункте 5.4) с помощью блокнота и добавьте, удалите или измените любой символ в тексте, после сохраните изменение в этот же файл.

5.6.2. Повторите пункт 5.5.

6. По полученным результатам построить два графика: на первом графике зависимость времени вычисления хэш-значения от размера файла $t_h = f(l)$, на втором графике зависимость $t_{ЭЦП} = f(l)$.

7. По проделанной работе сделать выводы.

8. Оформить отчет по проделанной работе.

9. Очистить папку «Временное».

Контрольные вопросы

1. Что такое асимметричная криптографическая система? В чем ее преимущества перед симметричной системой? В чем недостатки?

2. Что такое хэш-функция? Какими свойствами она должна обладать и в чем они заключаются?

3. Опишите процесс формирования хэш-функции ГОСТ Р 34.11-94.

4. Что такое “функция сжатия внутренних итераций”? В чем она заключается?

5. Что такое цифровая подпись? В чем ее отличие от рукописной подписи?

6. Опишите процесс формирования цифровой подписи ГОСТ Р 34.10-2001.

7. Опишите процесс проверки цифровой подписи ГОСТ Р 34.10-2001.

8. От каких злоумышленных действий позволяет защититься ЭЦП?

Выполнение работы

10. Изучение функции хэширования по ГОСТ Р 34.11-94.

1. Открываем пункт меню «Вычисления» и выбираем «Вычисление хэш-функции».

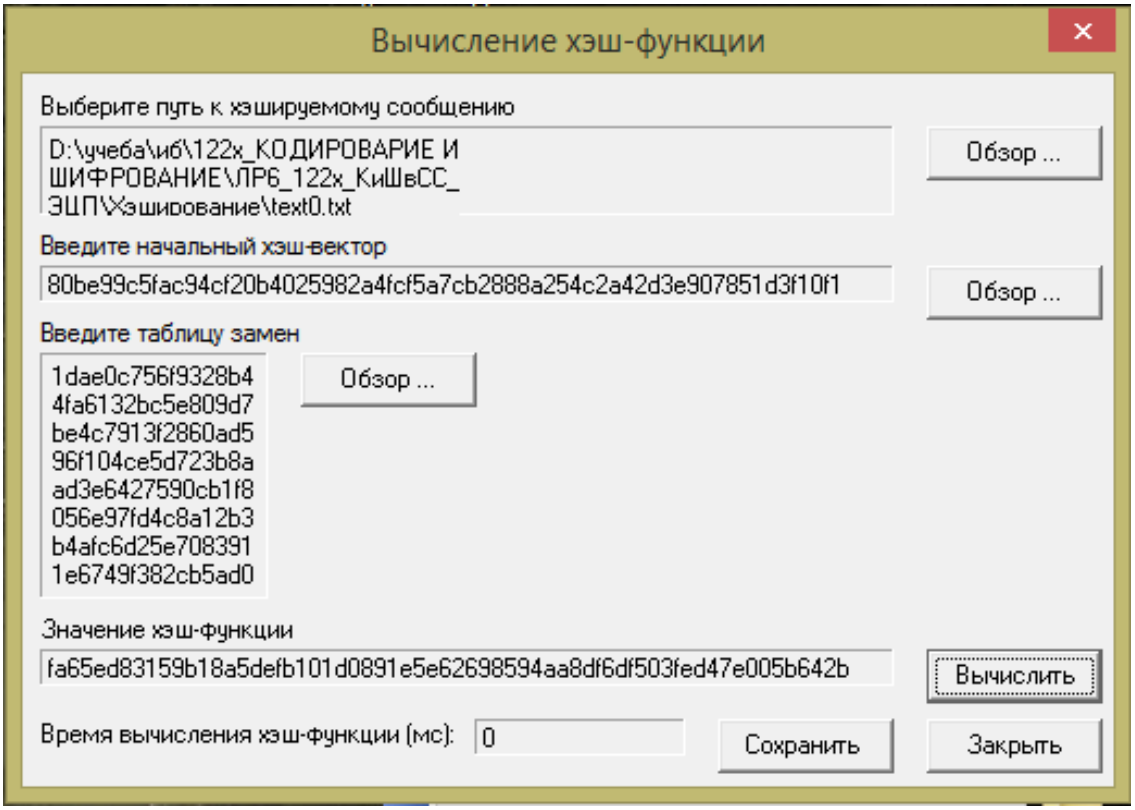
2. Задать начальные данные для вычисления хэш-функции (файлы находятся в папке «Хэширование»):

3. путь к файлу, который подвергается сжатию (название text0.txt);

4. начальный хэш-вектор (HeshVHex.txt);

5. таблицу замен (TabZamen.txt).

6. Вычислить хэш-функцию.



7. Повторить пункты 4.2 – 4.3 для файлов с названием text1, text2, text3, text4, text5. Сохранить полученные значения хэш-функции в папку «Временное», назвать hash1.txt, hash2.txt, hash3.txt, hash4.txt, hash5.txt соответственно.

Определить время хэширования. Полученные результаты занести в первые две строки таблицы (l – размер текстового файла, t_h – время хэширования, для text0 делать данный пункт не обязательно). ВНИМАНИЕ, хэширование может производиться до 3х минут, лучше в этот момент не чего не делать на компьютере, иначе возможно получение недостоверных значений.

l, байт	64	64	64	64	64
t_h, мс	15550	31390	46490	63940	78150
$t_{ЭЦП}$, мс	540	530	550	540	550
t_{Σ}, мс	16090	31920	47040	64480	78700

11. Изучение цифровой подписи ГОСТ Р 34.10-2001.

5.2 Вычислить секретный ключ.

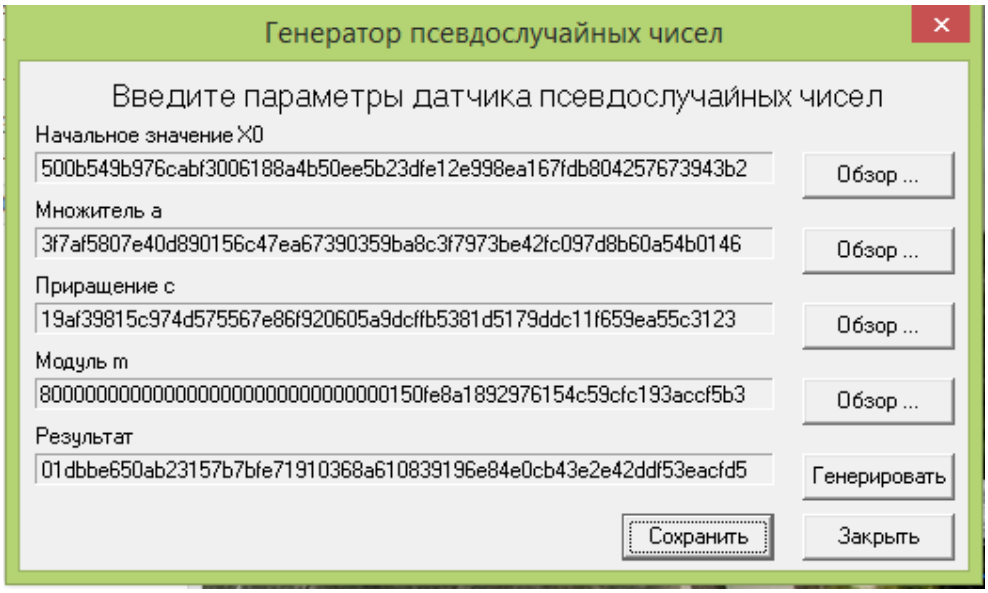
5.1.5 Открываем пункт меню «Вычисления» и выбираем «Генератор псевдослучайных чисел».

5.1.6 Ввести параметры датчика (файлы находятся в папке «ГПСЧ»):

- начальное значение X_0 (X0.txt);
- множитель a (a.txt);
- приращение c (c.txt);
- модуль m (m.txt).

5.1.7 Вычислить псевдослучайное число, оно же является секретным ключом.

5.1.8 Сохранить полученное значение в папку «Временное», назвав «KeySekr.txt».



5.3 Вычислить открытый ключ.

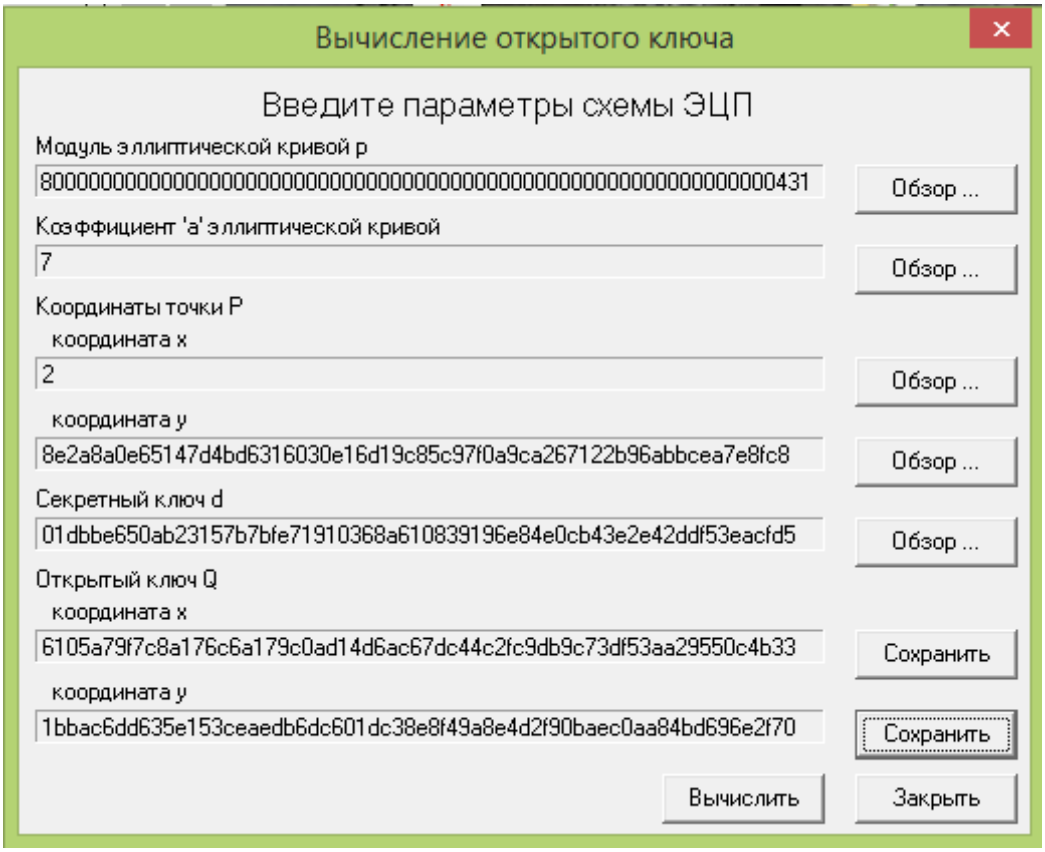
5.2.5. Открываем пункт меню «Вычисления» и выбираем “Вычисление открытого ключа”.

5.2.6. Задать параметры, необходимые для вычисления (файлы находятся в папке «Открытый ключ»).

- Модуль эллиптической кривой p (modP.txt);
- Коэффициент эллиптической кривой a (kofA.txt);
- Координаты точки P (координата x – Px.txt, координата y – Py.txt);
- Секретный ключ d (ранее полученный в пункте 5.1.4).

5.2.7. Вычислить открытый ключ.

5.2.8. Сохранить полученное значение в папку «Временное», назвав «Qx.txt» для координаты X и «Qy.txt» для координаты Y.



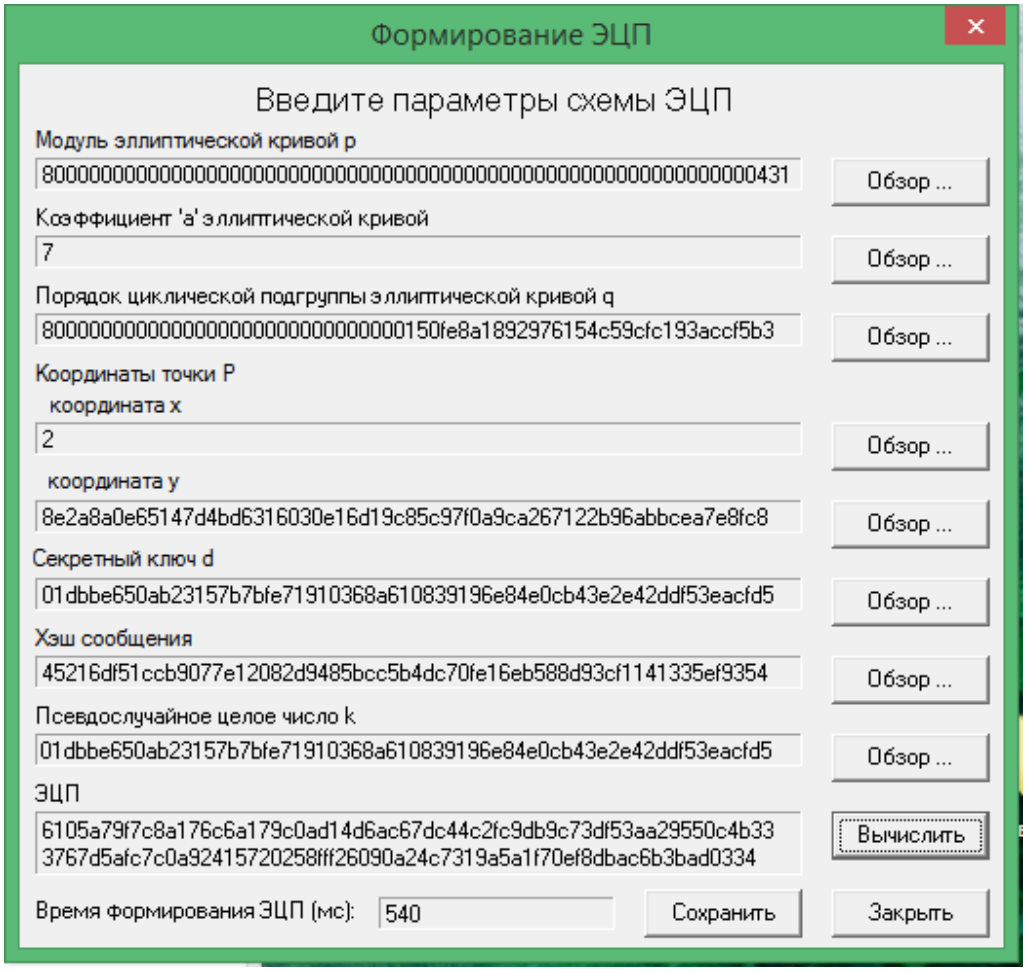
3.19. Вычислить цифровую подпись.

5.3.1. Выполнить предварительное вычисление случайного числа *k*, необходимого для формирования цифровой подписи. Для этого следует выполнить п. 5.1. Сохранить результат в папку «Временное», назвав «psK.txt».

5.3.2. Открываем пункт меню «Вычислить» и выбираем «Формирование цифровой подписи».

5.3.3. Задать параметры, необходимые для вычисления (файлы находятся в папке «Открытый ключ»):

- Модуль эллиптической кривой p (modP.txt);
- Коэффициент эллиптической кривой a (kofA.txt);
- Порядок циклической подгруппы эллиптической кривой q (q.txt);
- Координаты точки P (координата X – Px.txt, координата Y – Py.txt);
- Секретный ключ d (ранее полученный в пункте 5.1.4);
- Хэш сообщения (ранее полученное в пункте 4.4, с названием hash1.txt);
- Псевдо случайное число k (ранее полученное в пункте 5.4.1).



5.3.4. Сформировать цифровую подпись и занести в таблицу время создания ЭЦП (строка $t_{ЭЦП}$).

5.3.5. Сохранить значение ЭЦП в папку «Временное», назвав «podpis.txt»

5.3.6. Повторить пункты 5.4.3-5.4.4 для хэш-значений hash2.txt; hash3.txt; hash4.txt; hash5.txt. Обратите внимание на то, что пункт 5.4.5 выполнять повторно не нужно.

5.3.7. Рассчитать суммарное время вычисления ЭЦП и занести результаты расчетов в таблицу ($t_{\Sigma}= t_h+ t_{ЭЦП}$).

5.6 Передайте другому студенту, который сидит за другим компьютером, открытый ключ (Qx.txt; Qy.txt), файл (text1.txt) и соответствующую ему цифровую подпись (podpis.txt) при помощи дискеты или сети и получите от него такой же набор. Полученные файлы переименовать:

- «Qx.txt» в «QxPol.txt»;
- «Qy.txt» в «QyPol.txt»;
- «text1.txt» в «text1Pol.txt»;
- «podpis.txt» в «podpisPol.txt»;
- И поместить в папку «Временное».

5.7 Проверить цифровую подпись.

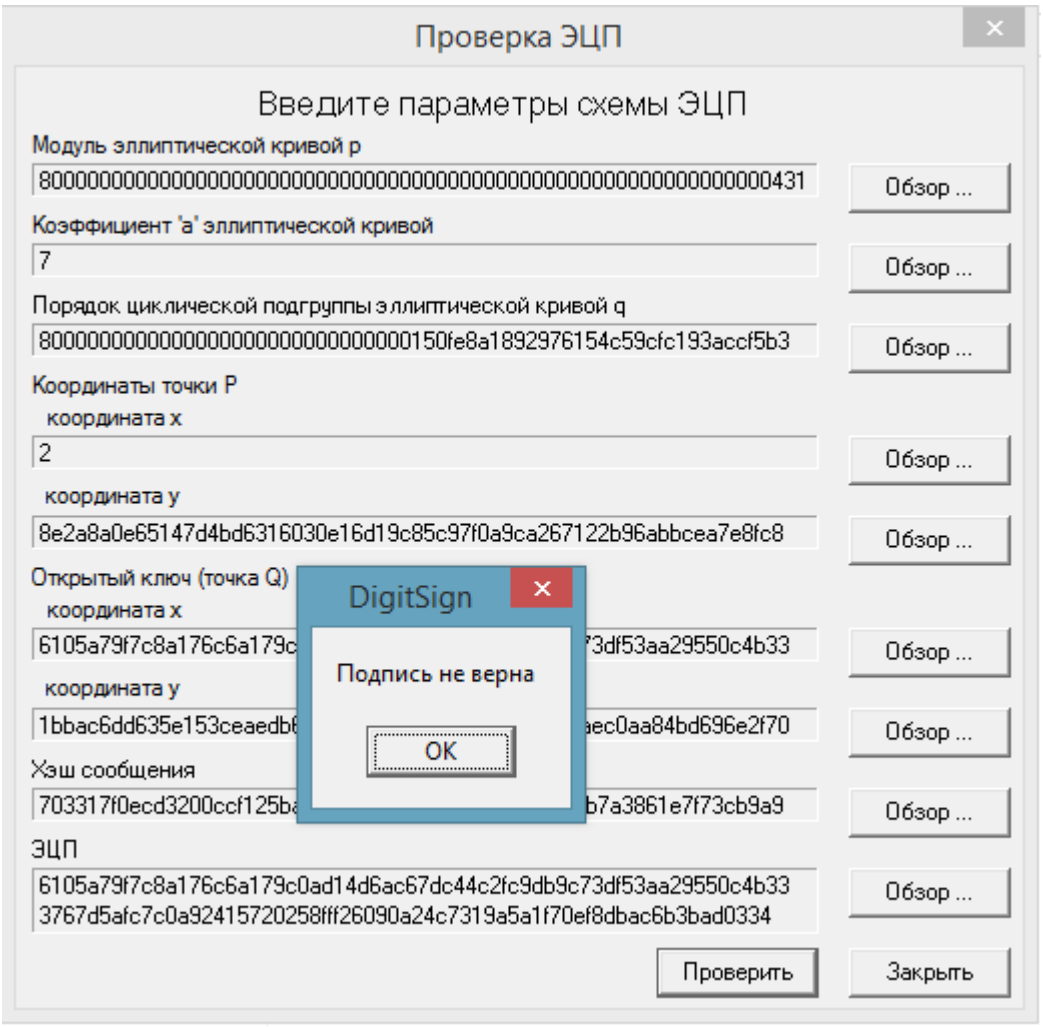
5.5.5 Сделать пункт 4, для файла text1Pol.txt, результат сохранить в папку «Временное», назвав «hashPol.txt»;

5.5.6 Открываем пункт меню «Вычислить», и выбираем «Проверка цифровой подписи».

5.5.7 Задать параметры, необходимые для вычисления (файлы находятся в папке «Открытый ключ»):

- Модуль эллиптической кривой p (modP.txt);
- Коэффициент эллиптической кривой a (kofA.txt);
- Порядок циклической подгруппы эллиптической кривой q (q.txt);
- Координаты точки P (координата X – «Px.txt», Y – «Py.txt»);
- Открытый ключ Q (полученный в пункте 5.4, координата X – «QxPol.txt»; координата Y – «QyPol.txt»);
- Хэш сообщения (поученный в пункте 5.4.1, «hashPol.txt»);
- ЭЦП (полученное в пункте 5.4, «podpisPol.txt»).

5.5.8 Осуществить проверку.



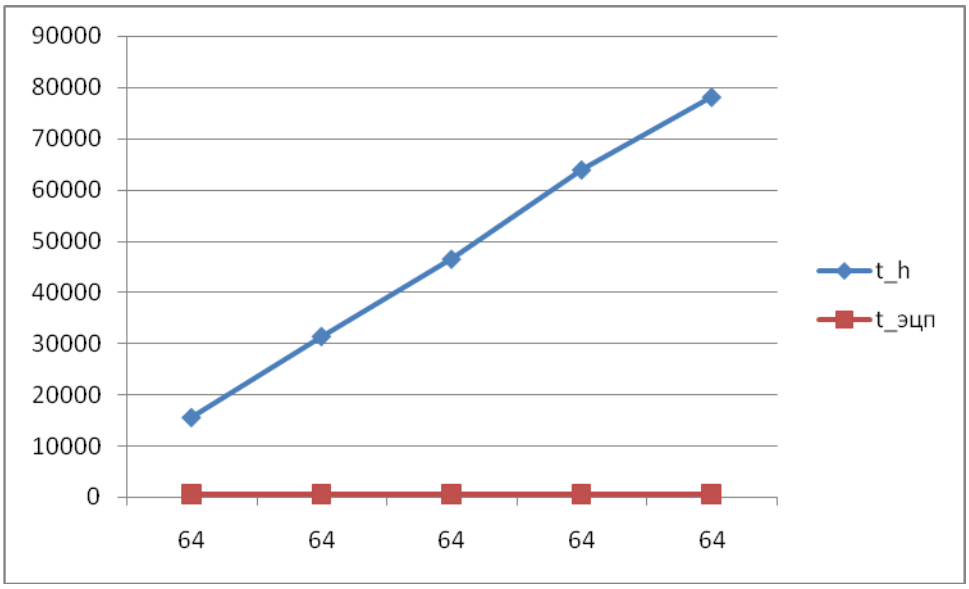
ВНИМАНИЕ! Данная программа неверно вычисляет открытый ключ Q, координату y. Если «Подпись не верна»: открываете файл QyPol.txt через блокнот и добавляете к **последней** цифре единицу (программа все время выдает последнюю цифру равную 0) и снова проверяете, если опять неверна, то еще добавляете единицу до тех пор пока подпись не будет верна или пока не достигнете f и подпись по прежнему не верна, тогда подпись действительно не верна (программа использует 16-ти ричную систему счисления, {0; 1; 2; 3; 4; 5; 6; 7; 8; 9; a; b; c; d; e; f}).

5.7 Подделка сообщения.

5.6.3. Откройте файл «text1Pol.txt» (полученный в пункте 5.4) с помощью блокнота и добавьте, удалите или измените любой символ в тексте, после сохраните изменение в этот же файл.

5.6.4. Повторите пункт 5.5.

12. По полученным результатам построить два графика: на первом графике зависимость времени вычисления хэш-значения от размера файла $t_h = f(l)$, на втором графике зависимость $t_{ЭЦП} = f(l)$.



Вывод: В результате выполнения данной лабораторной работы мы изучили процессы вычисления функции хэширования по ГОСТ Р 34.11-94, формирования и проверки ЭЦП на эллиптических кривых по ГОСТ Р 34.10-2001.

3.3 PGP кодирование и шифрование с открытым ключом

PGP (англ. Pretty Good Privacy) — компьютерная программа, также библиотека функций, позволяющая выполнять операции шифрования и цифровой подписи сообщений, файлов и другой информации, представленной в электронном виде, в том числе прозрачное шифрование данных на запоминающих устройствах, например, на жёстком диске.

Первоначально разработана Филиппом Циммерманном в 1991 году.

Общие сведения

PGP имеет множество реализаций, совместимых между собой и рядом других программ (GnuPG, FileCrypt и др.) благодаря стандарту OpenPGP (RFC 4880), но имеющих разный набор функциональных возможностей. Существуют реализации PGP для всех наиболее распространённых операционных систем. Кроме свободно распространяемых реализаций есть еще и коммерческие.

Совместимость

Так как PGP развивается, некоторые системы позволяют создавать зашифрованные сообщения с использованием новых возможностей, которые отсутствуют в старых системах. Отправитель и получатель должны знать возможности друг друга или, по крайней мере, согласовать настройки PGP.

Защищённость

В 1996 году криптограф Брюс Шнайер охарактеризовал раннюю версию PGP как «ближайшую к криптосистемам военного уровня». На данный момент не известно ни одного способа взлома данных, зашифрованных PGP, при помощи полного перебора или уязвимости криптоалгоритма. Ранние версии PGP обладали теоретическими уязвимостями, поэтому рекомендуется пользоваться современными версиями.

Криптографическая стойкость PGP основана на предположении, что используемые алгоритмы устойчивы к криптоанализу на современном оборудовании. Например, в PGP первых версий для шифрования ключей сессии использовался алгоритм RSA, основанный на односторонней функции (факторизация). В PGP версии 2 дополнительно можно использовать алгоритм IDEA. В последующем были добавлены дополнительные алгоритмы шифрования. Ни у одного используемого алгоритма нет известных уязвимостей.

В 2010 году группе учёных из Швейцарии, Японии, Франции, Нидерландов, Германии и США удалось декодировать данные, зашифрованные по алгоритму RSA при помощи ключа длиной 768 бит. Нахождение простых сомножителей осуществлялось общим методом решета числового поля. На первый шаг (выбор пары полиномов степени 6 и 1) было потрачено около полугода вычислений на 80 процессорах, что составило около 3 % времени, потраченного на главный этап алгоритма (просеивание), который выполнялся на сотнях компьютеров в течение почти двух лет. Если интерполировать это время на работу одного процессора AMD Opteron 2.2ГГц с 2Гб памяти, то получилось бы порядка 1500 лет. Обработка данных после просеивания для следующего ресурсоёмкого шага (линейной алгебры) потребовала несколько недель на малом количестве процессоров. Заключительный шаг после нахождения нетривиальных решений ОСЛУ занял не более 12 часов.

Решение ОСЛУ проводилось с помощью метода Видемана на нескольких раздельных кластерах и длилось чуть менее 4 месяцев. При этом размер разреженной матрицы составил 192 796 550x192 795 550 при наличии 27 795 115 920 ненулевых элементов. Для хранения матрицы на жёстком диске понадобилось около 105 гигабайт. В то же время понадобилось около 5 терабайт сжатых данных для построения данной матрицы.

В итоге группе удалось вычислить 232-цифровой ключ, открывающий доступ к зашифрованным данным.

Исследователи уверены, что с использованием их метода факторизации взломать 1024-битный RSA-ключ будет возможно в течение следующего десятилетия.

По словам исследователей, после их работы в качестве надежной системы шифрования можно рассматривать только RSA-ключи длиной 1024 бита и более. Причём от шифрования ключом длиной в 1024 бит стоит отказаться в ближайшие три-четыре года.

Зная разложение модуля на произведение двух простых чисел, противник может легко найти секретную экспоненту и тем самым взломать RSA. Однако на сегодняшний день самый быстрый алгоритм факторизации — решето обобщённого числового поля (General Number Field Sieve), скорость которого для k-битного целого числа составляет $\exp((c+o(1))k^{\frac{1}{3}}\log^{\frac{2}{3}}k)$ для некоторого $c<2$, не позволяет разложить большое целое за приемлемое время.

Механизм работы PGP

Шифрование PGP осуществляется последовательно хешированием, сжатием данных, шифрованием с симметричным ключом, и, наконец, шифрованием с открытым ключом, причём каждый этап может осуществляться одним из нескольких поддерживаемых алгоритмов. Симметричное шифрование производится с использованием одного из семи симметричных алгоритмов (AES, CAST5, 3DES, IDEA, Twofish, Blowfish, Camellia) на сеансовом ключе. Сеансовый ключ генерируется с использованием криптографически стойкого генератора псевдослучайных чисел. Сеансовый ключ зашифровывается открытым ключом получателя с использованием алгоритмов RSAили Elgamal (в зависимости от типа ключа получателя). Каждый открытый ключ соответствует имени пользователя или адресу электронной почты. Первая версия системы называлась Сеть Доверия и противопоставлялась системе X.509, использовавшей иерархический подход, основанной на удостоверяющих центрах, добавленный в PGP позже. Современные версии PGP включают оба способа.

Ключи

Пользователь PGP создаёт ключевую пару: открытый и закрытый ключ. При генерации ключей задаются их владелец (имя и адрес электронной почты), тип ключа, длина ключа и срок его действия. Открытый ключ используется для шифрования и проверки цифровой подписи. Закрытый ключ - для декодирования и создания цифровой подписи.

PGP поддерживает три типа ключей RSA v4, RSA legacy (v3) и Diffie-Hellman/DSS (Elgamal в терминологии GnuPG).

Для ключей RSA legacy длина ключа может составлять от 1024 до 2048 бит, а для Diffie-Hellman/DSS и RSA — от 1024 до 4096. Ключи RSA legacy содержат одну ключевую пару, а ключи Diffie-Hellman/DSS и RSA могут содержать один главный ключ и дополнительные ключи для шифрования. При этом ключ электронной подписи в ключах Diffie-Hellman/DSS всегда имеет размер 1024. Срок действия для каждого из типов ключей может быть определён как неограниченный или до конкретной даты. Для защиты ключевого контейнера используется секретная фраза.

Цифровая подпись

PGP поддерживает аутентификацию и проверку целостности посредством цифровой подписи. По умолчанию она используется совместно с шифрованием, но также может быть применена и к открытому тексту. Отправитель использует PGP для создания подписи алгоритмом RSA или DSA. При этом сначала создаётся хеш открытого текста (также известный как дайджест), затем — цифровая подпись хеша при помощи закрытого ключа отправителя. Для формирования хеша могут использоваться алгоритмы MD5, SHA-1, RIPEMD-160, SHA-256, SHA-384, SHA-512. В новых версиях PGP поддержка MD5 осуществляется для сохранения совместимости с ранними версиями. Для подписи используются алгоритмы RSA или DSA (в зависимости от типа ключа).

Сжатие данных

В целях уменьшения объёма сообщений и файлов и, возможно, для затруднения криптоанализа PGP производит сжатие данных перед шифрованием. Сжатие производится по одному из алгоритмов ZIP, ZLIB, BZIP2. Для сжатых, коротких и слабосжимаемых файлов сжатие не выполняется.

Сеть доверия

Как при шифровании сообщений, так и при проверке цифровой подписи, необходимо, чтобы принятый получателем открытый ключ действительно принадлежал отправителю. При простом скачивании открытого ключа он может быть подменён. С первых версий PGP поддерживает сертификаты открытых ключей, с помощью которых подмены (или случайные ошибки передачи) легко распознаются. Однако недостаточно просто создать сертификат, защищённый от модификации, так как при этом гарантируется лишь целостность сертификата после его создания. Пользователи также должны каким-нибудь способом проверить, что открытый ключ в сертификате действительно принадлежит отправителю. С первых версий продукты PGP включают в себя внутреннюю схему проверки сертификатов, названную сеть доверия. Заданная пара «имя пользователя — открытый ключ» может быть подписана третьим лицом, удостоверяющим соответствие ключа и владельца. В таких подписях может быть несколько вложенных уровней доверия. Хотя многие программы читают и пишут эту информацию, очень немногие учитывают этот уровень сертификата, принимая решение о принятии или отклонении сертификата.

Протокол сети доверия был впервые описан Циммерманном в 1992 году в руководстве PGP версии 2.0: «С течением времени вы будете накапливать ключи других людей, которых вы можете назвать доверенными рекомендателями. Кто-нибудь ещё может выбрать своих доверительных рекомендателей. И все будут постепенно накапливать и распространять со своими ключами набор заверенных подписей других людей, ожидая, что любой получатель

доверяет по крайней мере одной или двум подписям. Это позволяет создать децентрализованную устойчивую к сбоям сеть всех открытых ключей.»

Механизм сети доверия обладает преимуществами над централизованной инфраструктурой управления открытыми ключами, например, используемой в S/MIME, но не получил повсеместного применения. Пользователи хотели проверять корректность сертификатов вручную или не проверять вовсе.

Сертификаты

В последних спецификациях OpenPGP доверенные подписи могут использоваться для поддержки создания центров сертификации. Доверенность сертификата означает, что ключ действительно принадлежит указанному владельцу и может использоваться для подписи сертификатов одним уровнем ниже. Сертификат уровня 0 означает обычную подпись. Уровень 1 означает, что при помощи подписанного ключа можно создавать сертификаты уровня 0. При помощи сертификата уровня 2 можно создавать сертификаты уровня 1. Уровень 2 практически идентичен степени доверия, с которой полагаются пользователи на списки доверенных сертификатов, встроенные в браузеры.

Все версии PGP включают в себя способ отмены сертификата. Это необходимо, если требуется сохранять безопасность связи при потере или компрометации закрытого ключа. Отмена сертификата похожа на списки отзыва сертификатов в централизованной инфраструктуре открытых ключей. Современные версии PGP также поддерживают сроки истечения сертификатов.

Проблема корректного определения принадлежности открытого ключа владельцу характерна для всех криптографических систем с асимметричным шифрованием. У неё не существует достаточно хороших решений. Оригинальная схема PGP позволяет решить пользователю, использовать ли схему проверки сертификатов, в то время как большинство других инфраструктур открытых ключей требуют проверки каждого сертификата.

История

В 1991 году Филипп Циммерман создал первую версию PGP. Первая версия включала в себя симметричный алгоритм шифрования BassOmatic, созданный самим Циммерманом. Циммерман участвовал в движении против ядерной энергии и создал PGP для защищённого использования BBS и хранения файлов и сообщений. Для некоммерческого использования не требовалось лицензии, со всеми копиями распространялся весь исходный код. PGP распространилась в Usenet, а затем и в Интернете.

Уголовное расследование

Вскоре после выпуска PGP стала использоваться за пределами США, и в 1993 году правительство США начало расследование против Циммермана по подозрению в

нарушении экспортного законодательства, которое регулирует распространение криптографических систем с длиной ключа более 40 бит. В PGP использовались ключи длиной 128 бит и более.

Циммерман остроумно обошёл ограничения законодательства США. Он опубликовал исходный код в книге, изданной MIT Press. Код можно было сосканировать, распознать и скомпилировать. Экспорт книг не может быть запрещён, так как защищён первой поправкой к Конституции США.

OpenPGP

PGP Inc. была обеспокоена по поводу патентов. В компании был создан внутренний стандарт Unencumbered PGP («необременённый PGP»), не использующий алгоритмы, имеющие проблемы с лицензиями. Так как PGP широко использовалась во всём мире, многие хотели создавать собственное ПО, совместимое с PGP 5. В 1997 году PGP Inc. предложила IETF стандарт, названный OpenPGP. В IETF был созданы стандарты RFC 2440 (1998 год) и RFC 4880 (2007 год).

В 1999 году силами Фонда свободного программного обеспечения была создана свободная реализация OpenPGP под названием GNU Privacy Guard (GnuPG).

Поглощение Network Associates

В декабре 1997 года PGP Inc. была поглощена Network Associates Inc (ныне McAfee). NAI продолжила экспорт посредством печати исходных текстов. В составе NAI команда PGP разработала средства для шифрования дисков, брандмауэр, средства для обнаружения вторжений и IPsec VPN. После легализации экспорта криптографического ПО в 2000 году NAI прекратила публикацию исходных текстов, несмотря на возражения команды PGP.

В 2001 году Циммерман покинул NAI, NAI объявила о продаже PGP и остановке разработки PGP. В 2002 году NAI прекратила поддержку всех продуктов PGP PGP E-Business Server (исходной консольной версии PGP).

Современное состояние

В 2002 году несколько бывших разработчиков PGP основали PGP Corporation и выкупили PGP (кроме консольной версии). В 2003 году PGP Corporation разработала новый серверный продукт, PGP Universal.

В 2010-м году Symantec Corp. выкупил PGP за 300 млн долларов.

Криптографические приложения PGP Corporation

PGP изначально разрабатывалась для шифрования электронной почты на стороне клиента, но с 2002 года включает также шифрование жёстких дисков переносных компьютеров, файлов и директорий, сессий программ мгновенного обмена сообщениям, пакетной передачи файлов, защиту файлов и директорий в сетевых хранилищах, а в

современных версиях — ещё и шифрование HTTP-запросов и ответов на стороне сервера (mod openpgp) и клиента (Enigform).

Клиентские программы объединены в семейство PGP Desktop (включает в себя PGP Desktop EMail, PGP Whole Disk Encryption и PGP NetShare).

PGP Universal Server позволяет из командной строки централизованно администрировать клиенты на основе PGP Desktop.

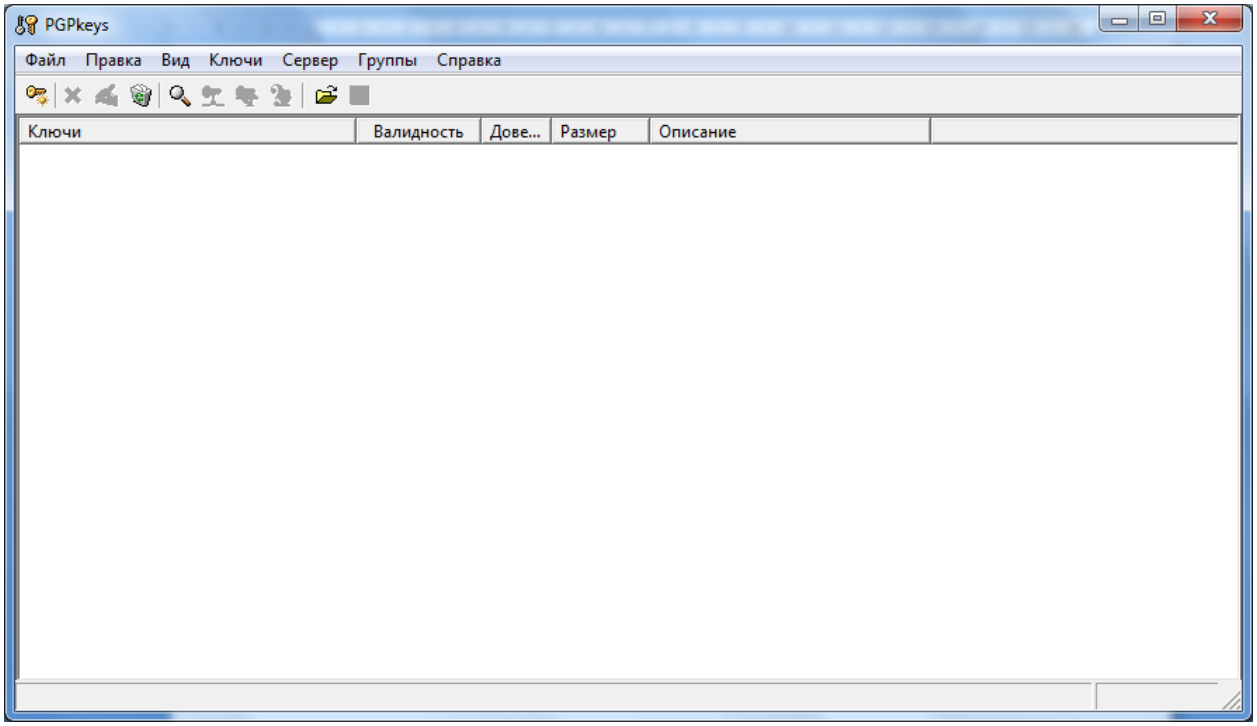
В 2010 году права на приложение были приобретены компаний Symantec за 300 млн. долларов.

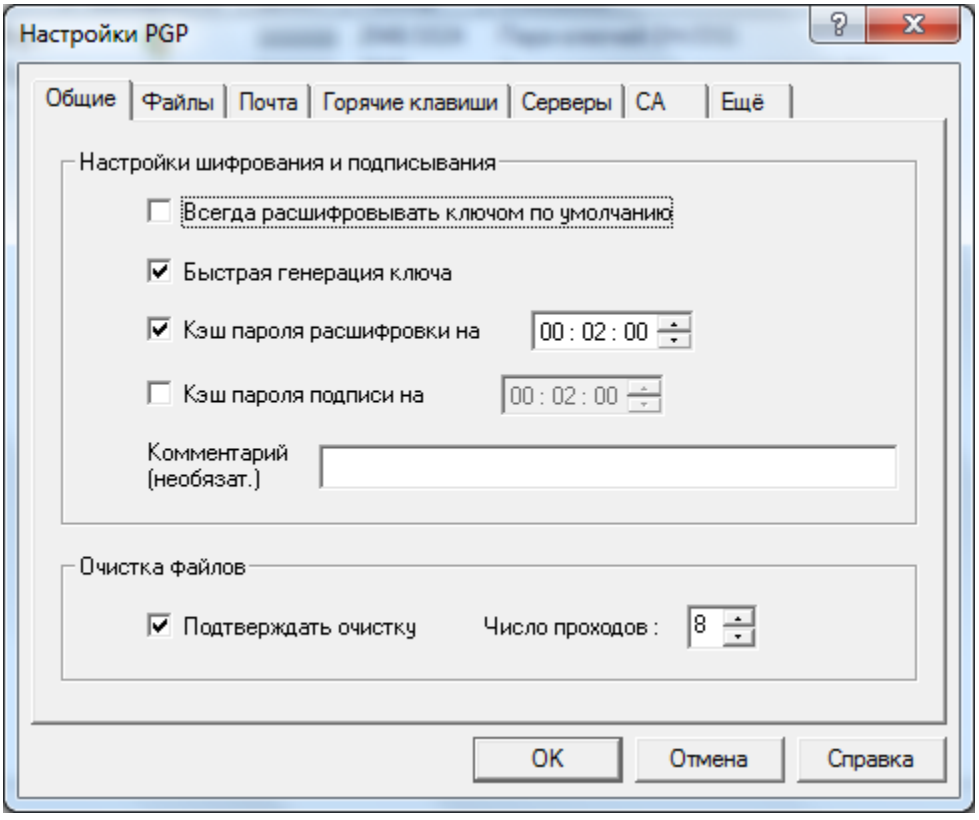
Правовые аспекты использования в России

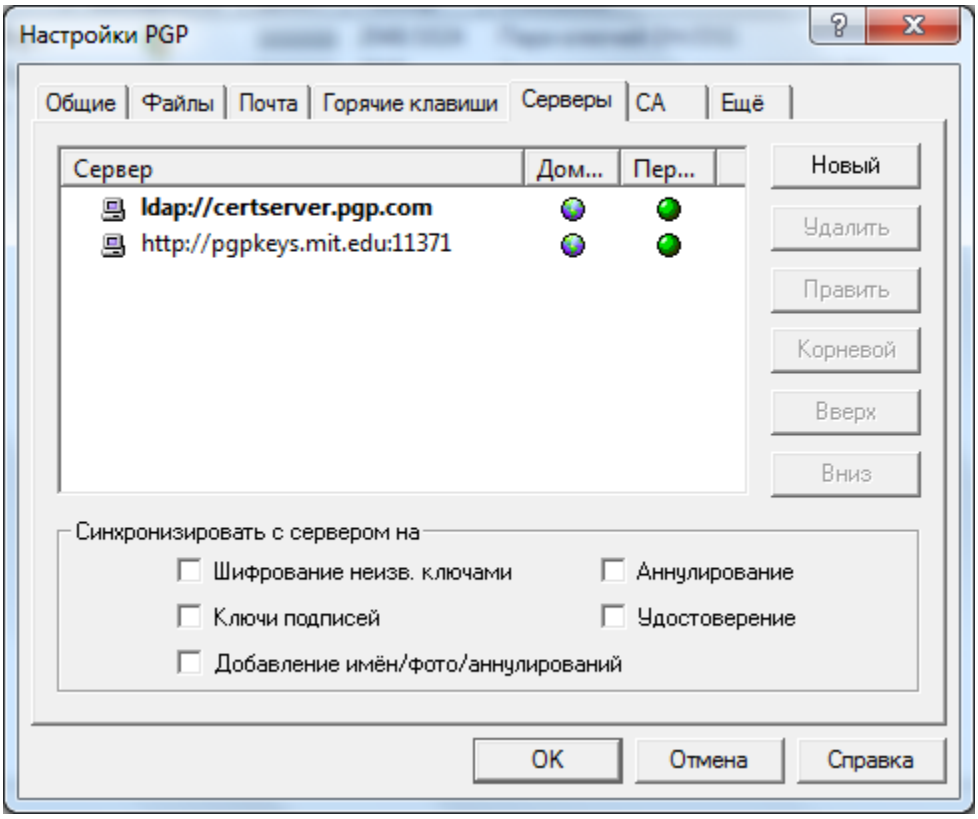
На сегодняшний день прямых законодательных запретов на использование PGP в России нет. Законодательно ограничивается использование криптографии только в государственных и муниципальных учреждениях. ФСБ предписывает всем государственным структурам использовать только сертифицированные средства криптографии. Физические лица и компании сами устанавливают какая информация является для них коммерческой тайной, методы хранения и передачи такой информации Закон «Об информации, информационных технологиях и защите информации» также указывает, что способ защиты информации, представляющей тайну, для негосударственных структур определяется оператором. Информационный ресурс Helpdesk24 в статье «Правомерность использования криптографических средств защиты информации» приводит выдержки из федеральных законов, поясняющие данный вопрос. Также авторы проекта «openPGP в России» утверждают, что не существует законов, запрещающих использование PGP . Указ от 3 апреля 1995 г. N 334 «О мерах по соблюдению законности в области разработки, производства, реализации и эксплуатации шифровальных средств» отменен. Электронная подпись, генерируемая с помощью PGP и её несертифицированных аналогов, имеет юридическую силу в Российской Федерации, т.к. согласно пункту 3 статьи 5 63-ФЗ "Об электронной подписи" попадает под определение усиленной неквалифицированной электронной подписи. Согласно пункту 2 статьи 6 этого ФЗ для признания такой ЭП необходимо соглашение между участниками электронного взаимодействия.

ХОД РАБОТЫ

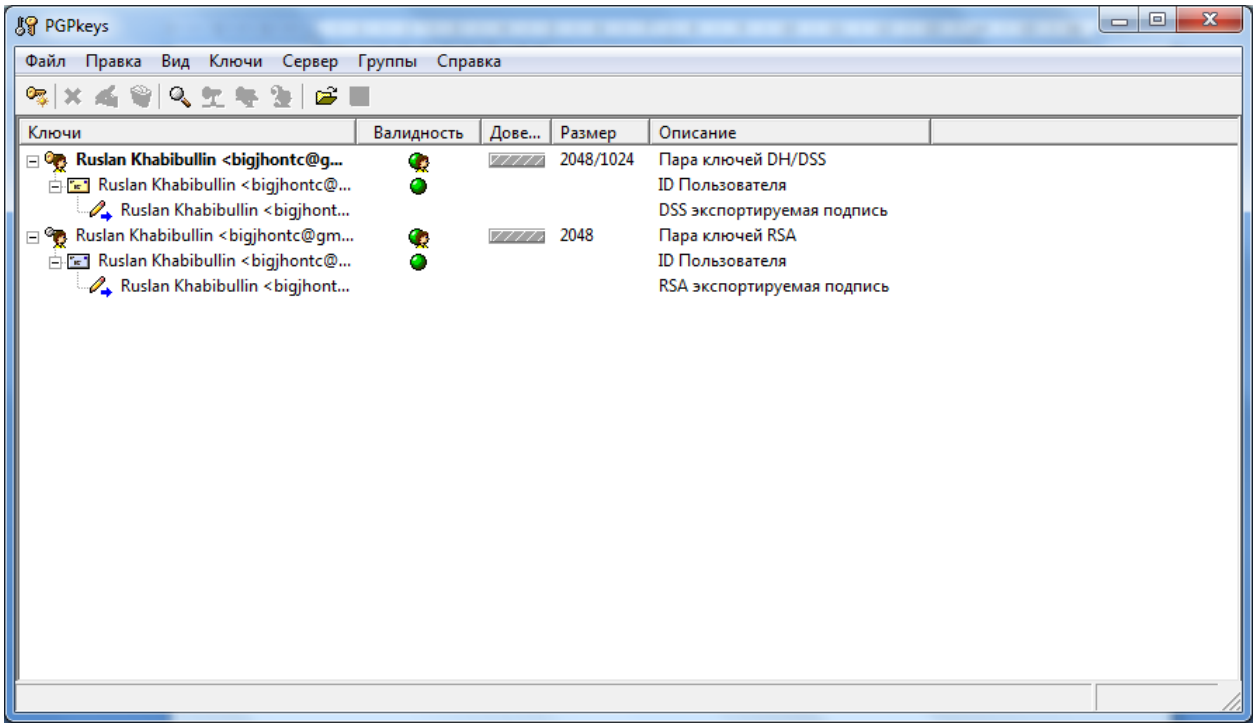
1. Изучите вкладки окна “Настройки PGP”.



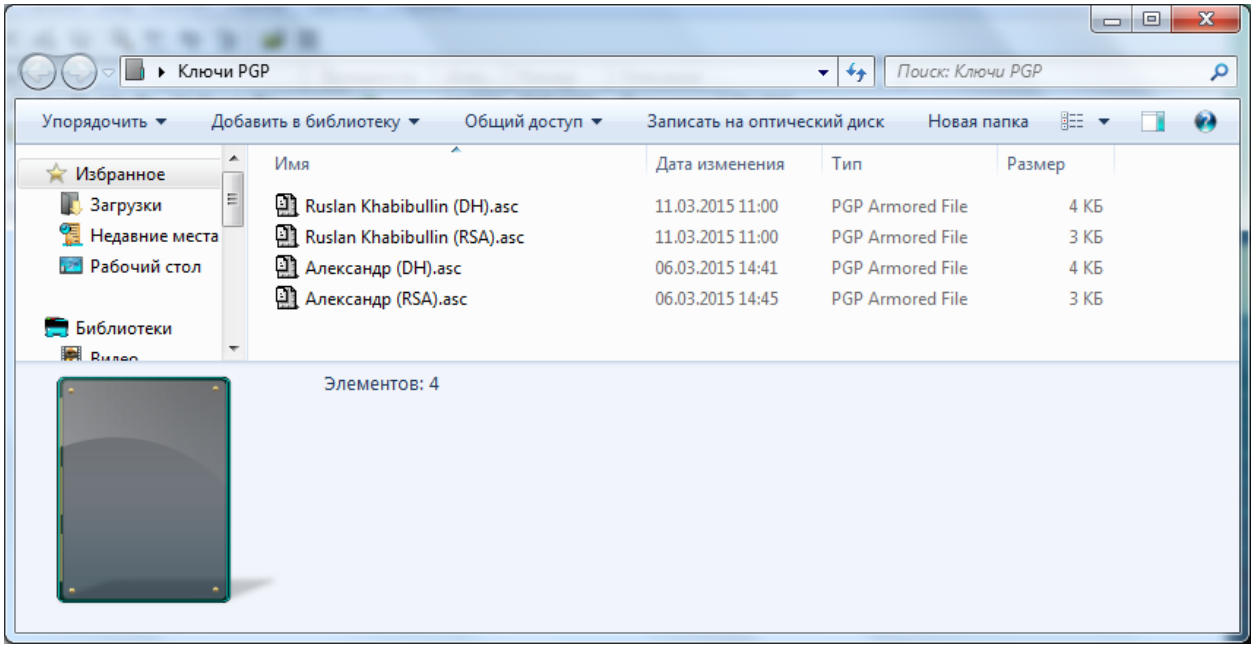




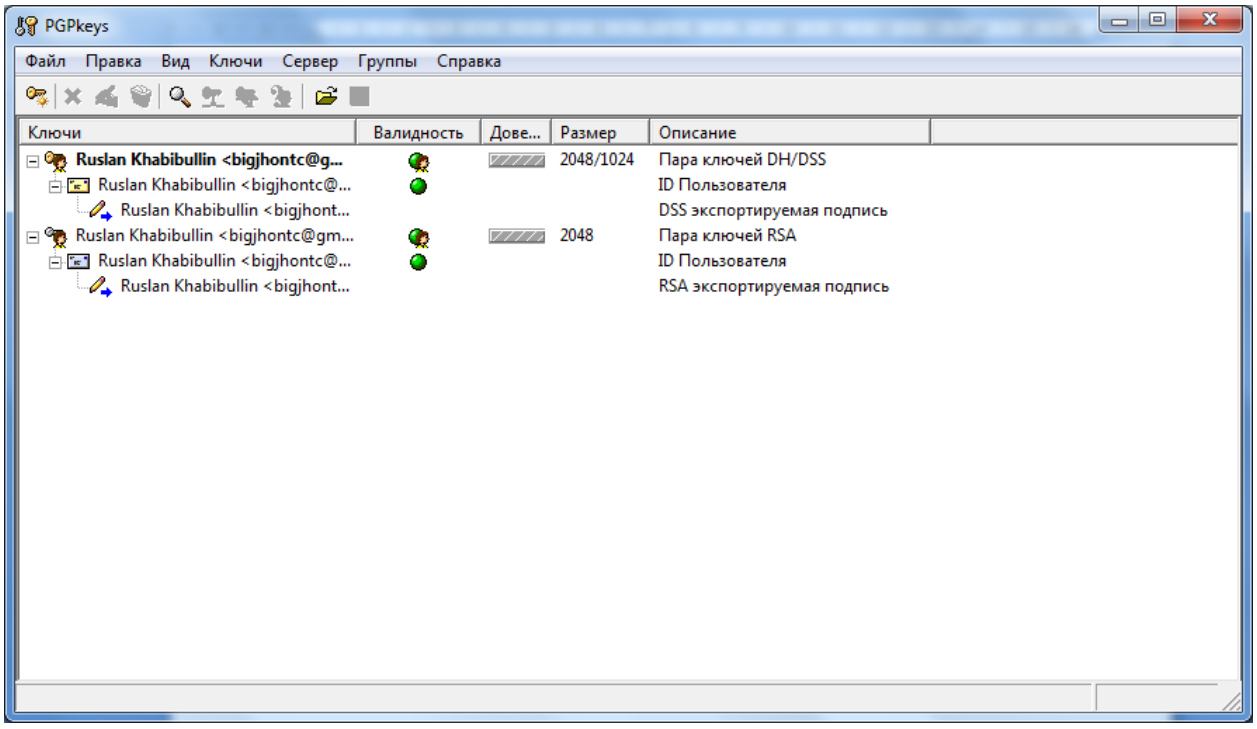
Создайте с помощью менеджера PGPkeys ключи шифрования двух типов: RSA и DH/DSS.



Сохраните полученные ключи (открытые и секретные) в отдельный файл.

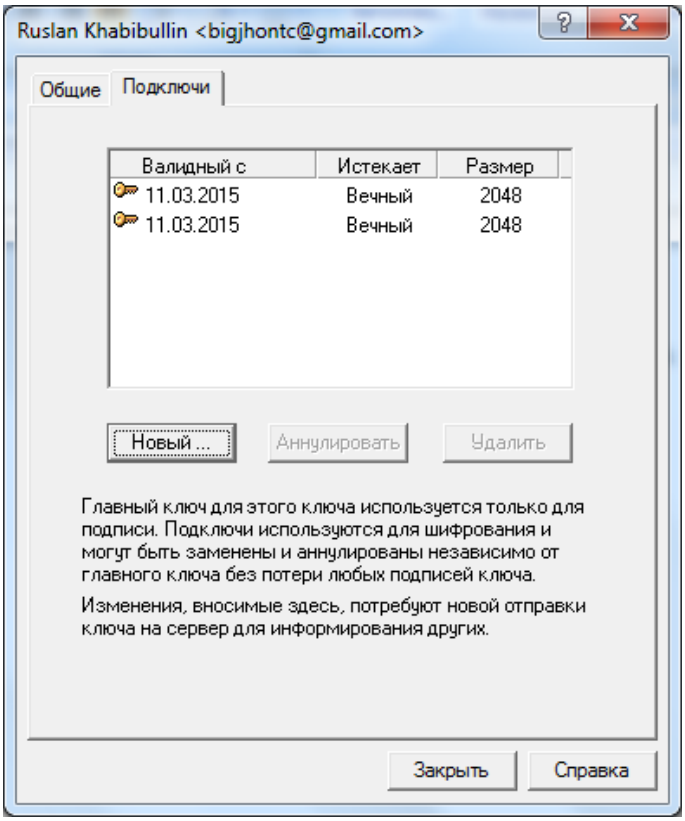


Назначьте ключ DH/DSS используемым по умолчанию.

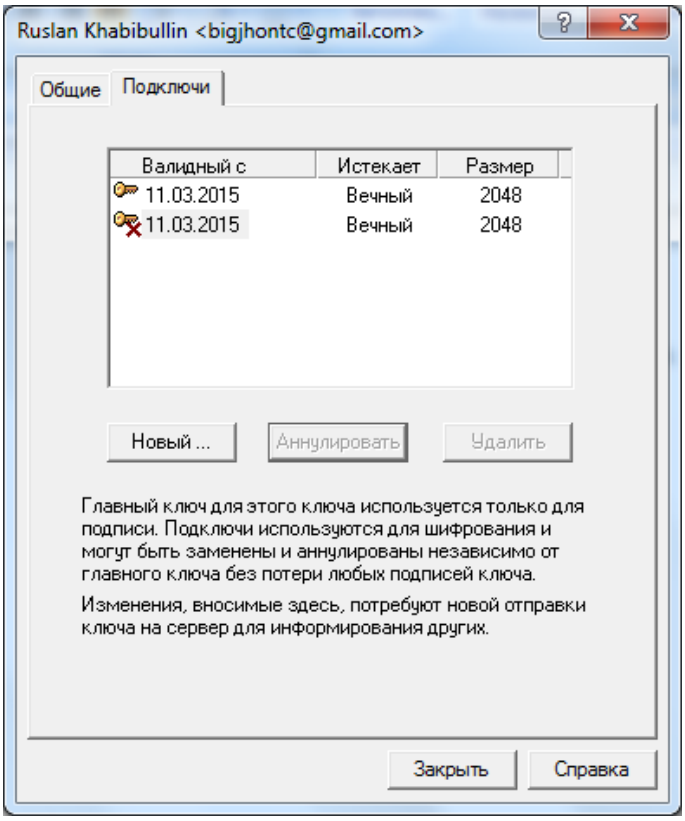


Изучите свойства ключевой пары DH/DSS.

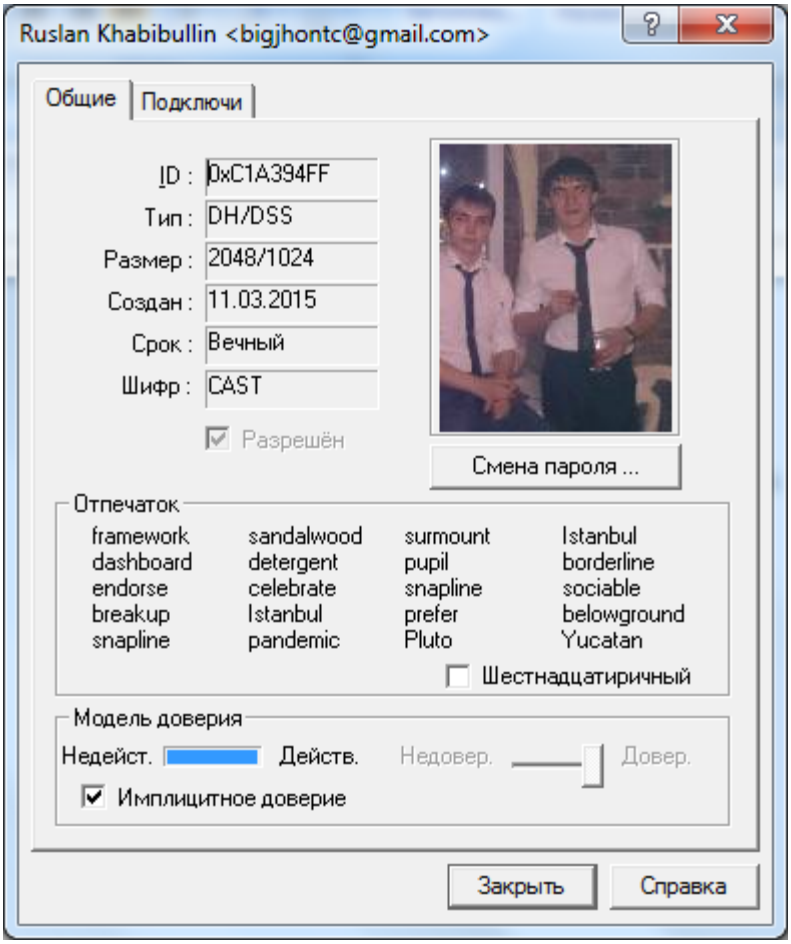
Создание подключа



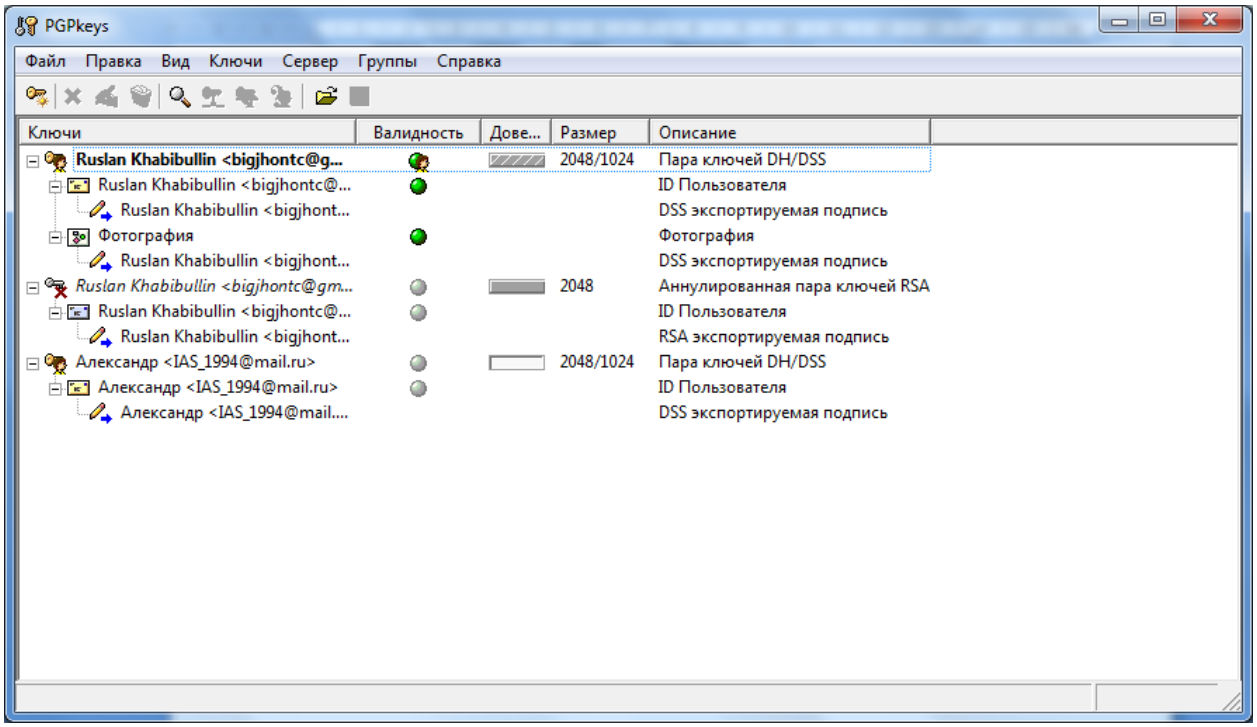
Аннулирование подключа



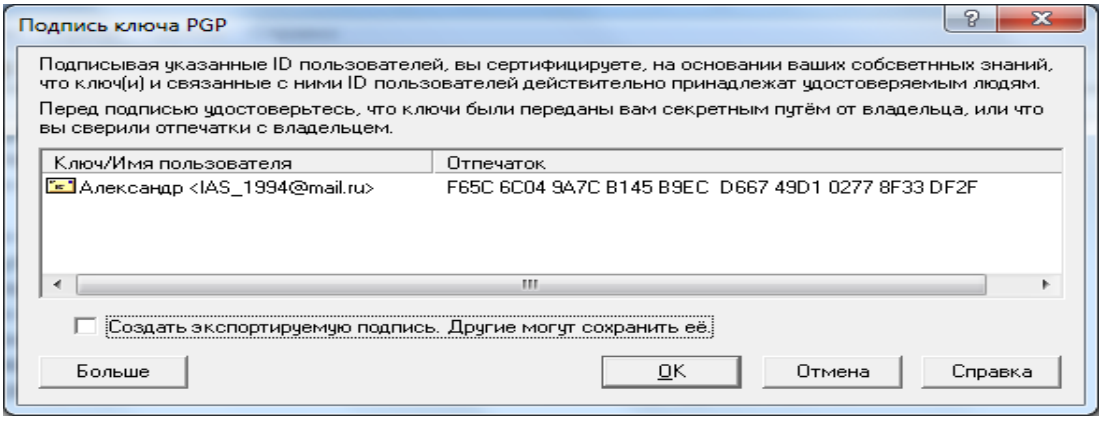
Добавление фотографии



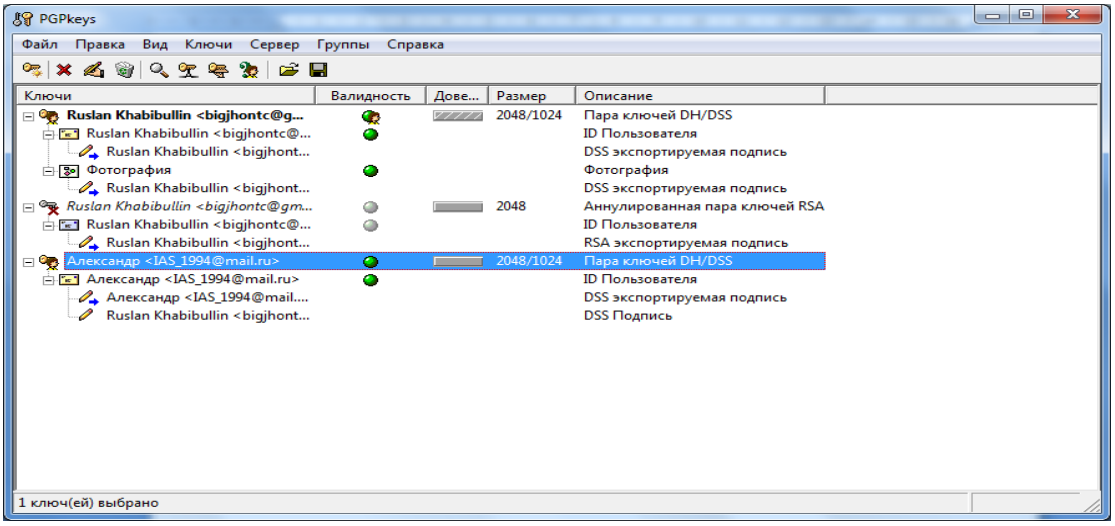
Обменяйтесь с другим студентом открытыми ключами DH/DSS.



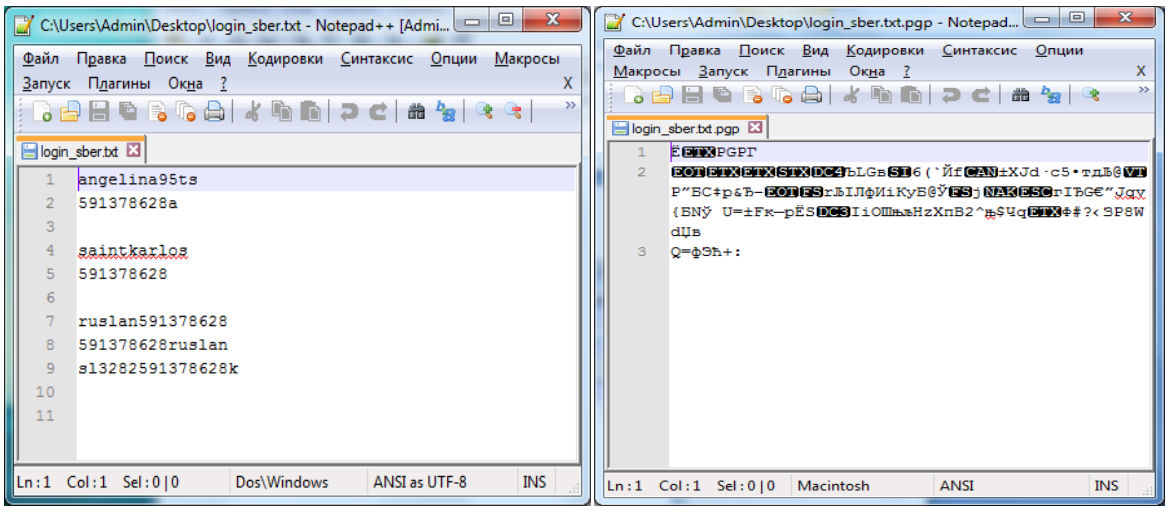
Установите подлинность полученного ключа с помощью его отпечатка.

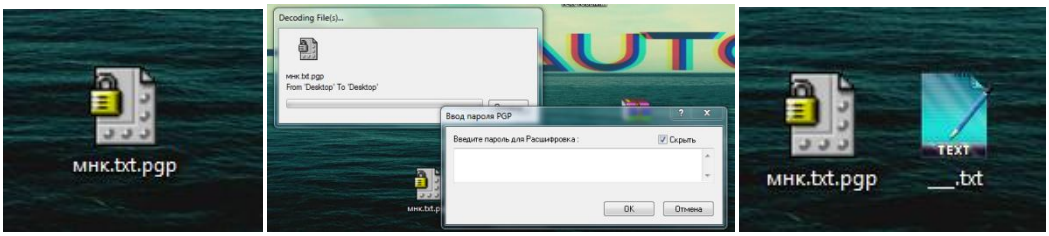


Установите степень доверия к владельцу полученного ключа на максимальный уровень.

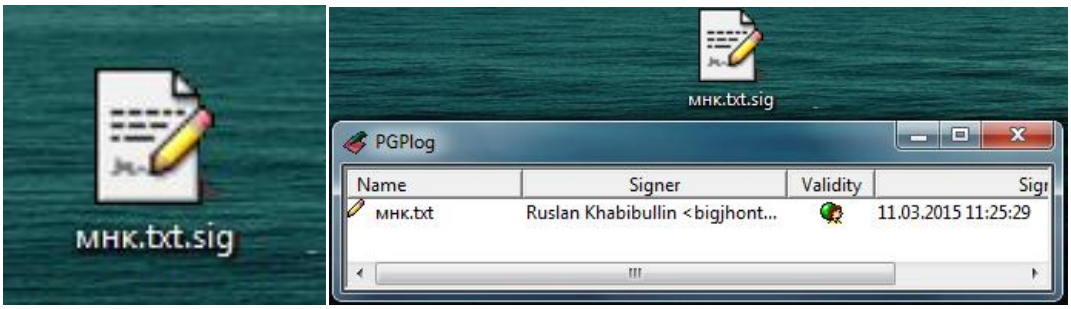


Зашифруйте произвольное сообщение/файл и обменяйтесь полученным результатом с другим студентом. Расшифруйте полученное сообщение.

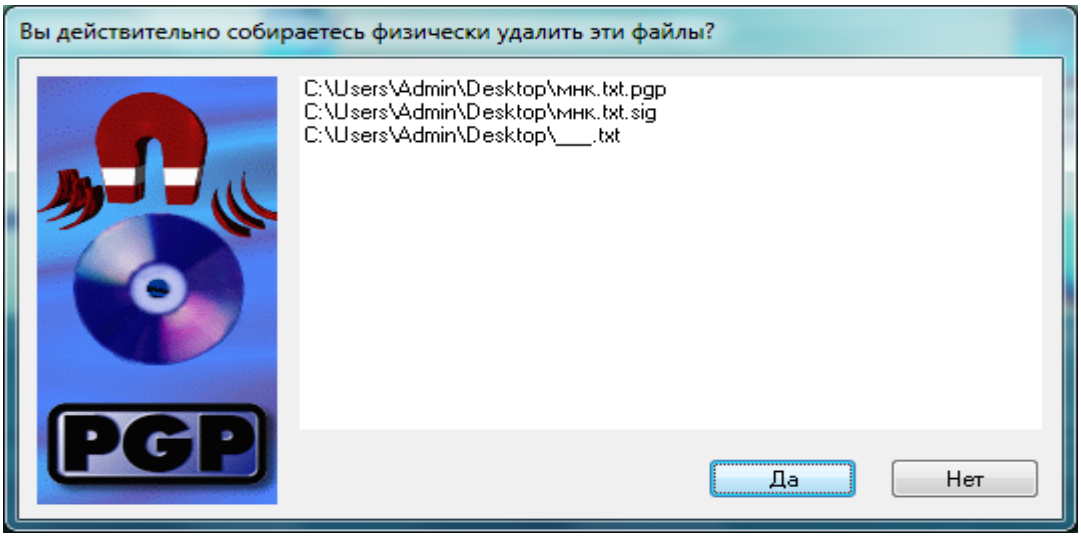




Подпишите произвольное сообщение/файл и обменяйтесь полученным результатом с другим студентом. Проверьте достоверность источников полученного сообщения.



Уничтожьте все ненужные файлы, используя утилиту PGP Wipe.



Контрольные вопросы.

1. Криптографическая система с открытым ключом (или асимметричное шифрование, асимметричный шифр) — система шифрования и/или электронной подписи (ЭП), при которой открытый ключ передаётся по открытому (то есть незащищённому, доступному для наблюдения) каналу и используется для проверки ЭП и для шифрования сообщения. Для генерации ЭП и для расшифровки сообщения используется закрытый ключ. Криптографические системы с открытым ключом в настоящее время широко применяются в различных сетевых протоколах, в частности, в

протоколах TLS и его предшественнике SSL (лежащих в основе HTTPS), в SSH. Также используется в PGP, S/MIME.

2. Электронная подпись (ЭП), Электронная цифровая подпись (ЭЦП) — реквизит электронного документа, полученный в результате криптографического преобразования информации с использованием закрытого ключа подписи и позволяющий установить отсутствие искажения информации в электронном документе с момента формирования подписи и проверить принадлежность подписи владельцу сертификата ключа подписи.

3. Аннулирование – после аннулирования открытого ключа PGP синхронизирует его с сервером, дабы в дальнейшем ваши корреспонденты не могли его применять. **Дезактивация –** временное отключение неиспользуемого ключа или ключевой пары.

4. Отпечаток ключа — серия знаков, которой сопровождается каждый ключ. Сам по себе он не секретный, но уникальный. Если отпечаток того ключа, который вы получили (например) по электронной почте, и отпечаток, которым поделился ваш друг в Skype, совпадут, значит, у вас в руках правильный, настоящий ключ.

5. Имплицитное доверие – полное доверие зарезервировано для ключевых пар, расположенных на локальном ключе. Если одна часть ключевой пары находится на вашей связке, PGP предполагает, что вы владелец пары ключей и что Вы без проблем сможете доверять себе.

4. КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ В СЕТЯХ ПЕРЕДАЧИ ДАННЫХ

Теория шифров с открытым ключом криптографические протоколы в сетях передачи данных

Безопасность сети передачи данных на транспортном уровне SSL и TLS

Безопасность транспортного уровня обеспечивает услуги безопасности "из конца в конец" для приложений, которые используют протоколы транспортного уровня, такие как TCP. Основные идеи предназначены для того, чтобы обеспечить услуги безопасности на сети Интернет. Например, когда в сети имеются интерактивно работающие онлайн(online)-магазины, то желательны следующие услуги безопасности:

1. Клиент должен убедиться, что сервер принадлежит фактическому продавцу, а не самозванцу. Клиент не хочет сообщать самозванцу номер кредитной карточки (установление подлинности объекта).

2. Клиент и продавец должны быть убеждены, что содержание сообщения не изменено в течение передачи (целостность сообщения).

3. Клиент и продавец должны быть убеждены, что самозванец не перехватит чувствительную информацию, такую как номер кредитной карточки (конфиденциальность).

Сегодня применяются в основном два протокола обеспечения безопасности на транспортном уровне: *Протокол "Уровень безопасных розеток" (SSL - Secure Socket Layer)* и *Протокол Безопасности Транспортного уровня (TLS - Transport Layer Security).* Мы сначала обсудим SSL, затем TLS, а потом их сравним и покажем их отличия друг от друга.

Одна из целей этих протоколов состоит в том, чтобы обеспечить сервер и клиента услугами установления подлинности, конфиденциальности и целостности данных. Прикладной уровень программ клиент-сервер (client-server), таких как Язык передачи гипертекста (HTTP), который использует услуги TCP, может инкапсулировать свои данные в пакеты SSL. Если сервер и клиент согласованы с функционирующими программами SSL (или TLS), то клиент может использовать URL https: // ... вместо http:// ... , для того чтобы разрешить сообщениям HTTP инкапсулироваться в пакеты SSL (или TLS). Например, номера кредитной карточки могут быть безопасно переданы через Интернет для онлайн-покупателей.

Протокол SSL

SSL (англ. secure sockets layer — уровень защищённых сокетов) — криптографический протокол, который подразумевает более безопасную связь. Он использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для

сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений. Протокол широко использовался для обмена мгновенными сообщениями и передачи голоса через IP (англ. Voice over IP — VoIP), в таких приложениях, как электронная почта, Интернет-факс и др. В настоящее время известно, что протокол не является безопасным. SSL должен быть исключен из работы в пользу TLS (см. CVE-2014-3566).

SSL изначально разработан компанией Netscape Communications для добавления протокола HTTPS в свой веб-браузер Netscape Navigator. Впоследствии, на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший имя TLS.

Протокол SSL обеспечивает защищенный обмен данных за счет двух следующих элементов:

- Аутентификация
- Шифрование

SSL использует асимметричную криптографию для аутентификации ключей обмена, симметричный шифр для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений.

Протокол SSL предоставляет "безопасный канал", который имеет три основных свойства:

1. Канал является частным. Шифрование используется для всех сообщений после простого диалога, который служит для определения секретного ключа.

2. Канал аутентифицирован. Серверная сторона диалога всегда аутентифицируется, а клиентская делает это опционно.

3. Канал надежен. Транспортировка сообщений включает в себя проверку целостности.

Преимуществом SSL является то, что он независим от прикладного протокола. Протоколы приложений (HTTP, FTP, TELNET и т.д.) могут работать поверх протокола SSL совершенно прозрачно, т е SSL может согласовывать алгоритм шифрования и ключ сессии, а также аутентифицировать сервер до того, как приложение примет или передаст первый байт сообщения.

Принцип работы

SSL использует среду с несколькими слоями, что обеспечивает безопасность обмена информацией. Конфиденциальность общения присутствует за счет того, что безопасное соединение открыто только целевым пользователям.

Многослойная среда

Протокол SSL размещается между двумя протоколами: протоколом, который использует программа-клиент (HTTP, FTP, LDAP, TELNET etc) и транспортным протоколом TCP/IP.

SSL защищает данные выступая в роли фильтра для обеих сторон и передает их далее на транспортный уровень. Работу протокола можно разделить на два уровня:

1. Слой протокола подтверждения подключения (Handshake Protocol Layer)
2. Слой протокола записи

Первый слой, в свою очередь, состоит из трех подпротоколов:

1. Протокол подтверждения подключения (Handshake Protocol)
2. Протокол изменения параметров шифра (Cipher Spec Protocol)
3. Предупредительный протокол (Alert Protocol)

Протокол подтверждения подключения используется для согласования данных сессии между клиентом и сервером. К данным сессии относятся:

- Идентификационный номер сессии
- Сертификаты обеих сторон
- Параметры алгоритма шифрования
- Алгоритм сжатия информации
- "Общий секрет" применен для создания ключей; открытый ключ

Протокол подтверждения подключения производит цепочку обмена данными, что в свою очередь начинает аутентификацию сторон и согласовывает шифрование, хэширование и сжатие. Следующий этап - аутентификация участников, которая осуществляется также протоколом подтверждения подключения.

Протокол изменения параметров шифра используется для изменения данных ключа (keyingmaterial) - информации, которая используется для создания ключей шифрования. Протокол состоит всего из одного сообщения, в котором сервер говорит, что отправитель хочет изменить набор ключей.

Предупредительный протокол содержит сообщение, которое показывает сторонам изменение статуса или сообщает о возможной ошибке. Обычно предупреждение отсылается тогда, когда подключение закрыто и получено неправильное сообщение, сообщение невозможно расшифровать или пользователь отменяет операцию.

Цифровые сертификаты

Протокол SSL использует сертификаты для проверки соединения. Сертификаты расположены на безопасном сервере и используются для шифрования данных и идентификации Web-сайта.

Способы получения SSL-сертификата:

1. Использовать сертификат, выданный CA
2. Использовать самоподписанный сертификат
3. Использовать "пустой" сертификат

Самоподписанный сертификат - сертификат, созданный самим пользователем - в этом случае издатель сертификата совпадает с владельцем сертификата. “Пустой” сертификат - сертификат, содержащий фиктивную информацию, используемую в качестве временной для настройки SSL и проверки его функциональности в данной среде.

Механизмы образования ключа для текущей сессии в SSL/TLS

Для обмена подлинными и конфиденциальными сообщениями клиенту и серверу нужны шесть криптографических объектов секретности (четыре ключа и два вектора инициализации). Однако чтобы создать их, между этими двумя сторонами должен быть установлен один предварительный главный секретный код (pre-master secret). SSL определяет шесть методов обмена ключами, чтобы установить этот предварительный объект секретности: NULL, RSA, анонимный Диффи-Хеллман (Diffie-Hellman), кратковременный Диффи-Хеллман, фиксированный Диффи-Хеллман и Fortezza

RSA

В этом методе предварительный главный секретный код - 48-байтовое случайное число, созданное клиентом, зашифрованное открытым ключом RSA-сервера и передаваемое серверу. Сервер должен передать свой сертификат шифрования/дешифрования RSA.

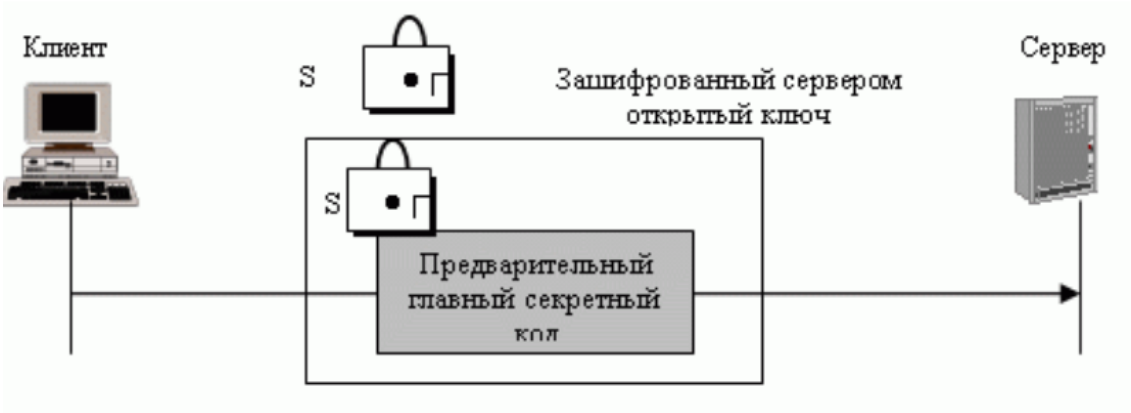


Рис. 4.1. RSA – смена ключа; открытый ключ сервера

Анонимный протокол Диффи-Хеллмана

Это самый простой и наи**б**олее ненадежный метод. Предварительный главный секретный код устанавливают между клиентом и сервером, используя протокол Диффи-Хеллмана. При этом передают половину ключа в исходном тексте - это называется анонимным протоколом Диффи-Хеллмана, потому что ни одна сторона не известна другой. Самый серьезный недостаток этого метода - возможность атаки "посредника".

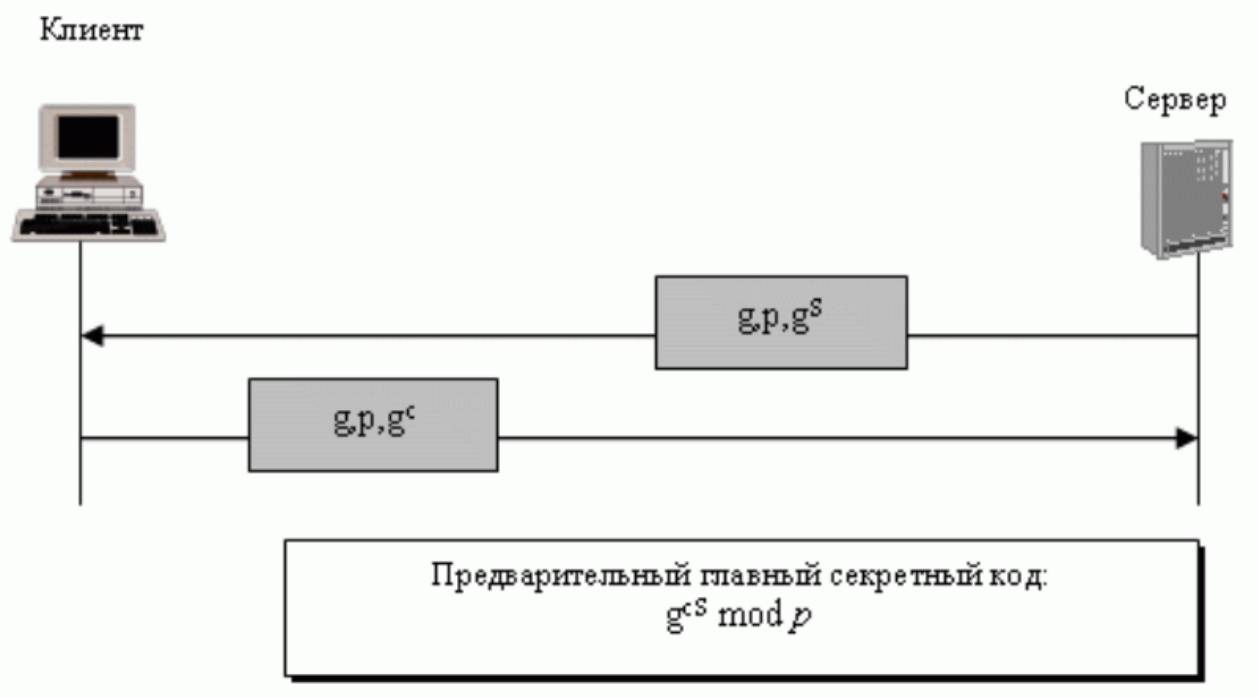


Рис. 4.2. Анонимный протокол Диффи-Хеллмана смены ключей

Кратковременный метод Диффи-Хеллмана

Чтобы сорвать атаку "посредника", может быть использована кратковременная смена ключей методом Диффи-Хеллмана. Каждая сторона передает ключ Диффи-Хеллмана, подписанный своим секретным ключом. На приемной стороне должны проверить подпись, используя открытый ключ передатчика. Обмен открытыми ключами для проверки использует либо RSA-, либо DSS-сертификат цифровой подписи.

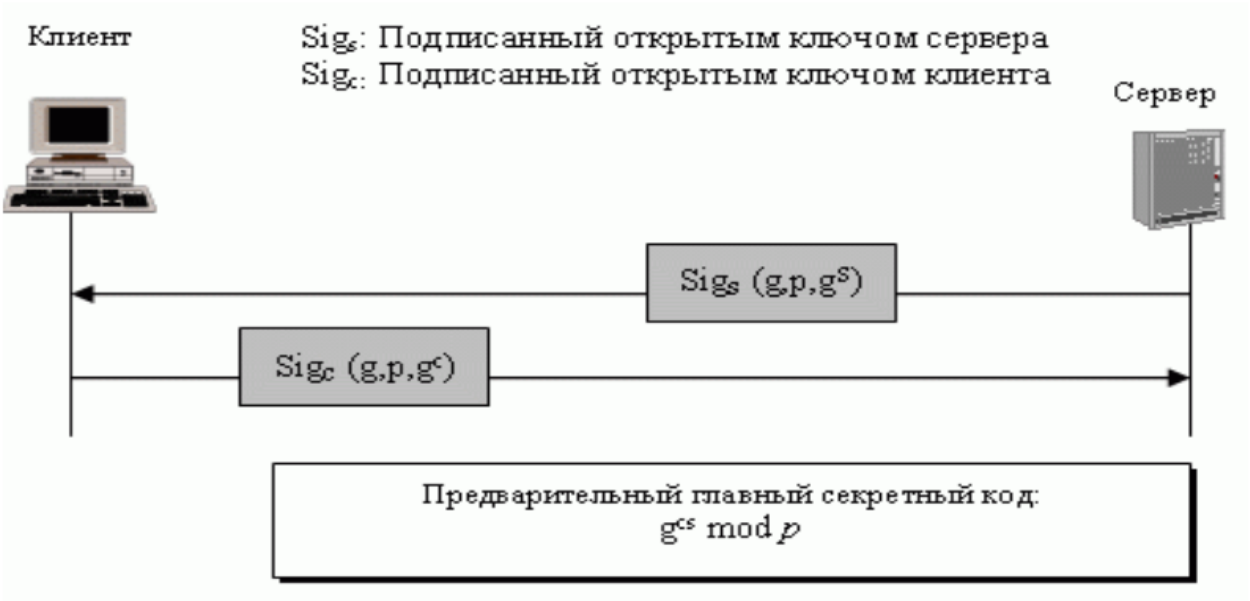


Рис. 4.3. Кратковременный протокол Диффи-Хеллмана смены ключей

Фиксированный метод Диффи-Хеллмана

Другое решение - фиксированный метод Диффи-Хеллмана. Все объекты в группе могут подготовить фиксированные параметры (g и p). Затем каждый объект может создать фиксированную половину ключа (gx). Для дополнительной безопасности каждая отдельная половина ключа Диффи-Хеллмана вставляется в сертификат, проверенный центром сертификации (CA). Другими словами, две стороны отдельно не обмениваются полуключами; CA передает полуключи в специальном сертификате RSA или DSS. Когда клиент должен вычислить *предварительный главный секретный код*, он использует свой собственный фиксированный полуключ и полуключ сервера, полученный в сертификате. Сервер делает то же самое, но в обратном порядке. Обратите внимание, что в этом методе не передаются сообщения смены ключей, а происходит только обмен сертификатами.

Алгоритмы шифрования/дешифрования

Есть несколько возможностей выбора алгоритма шифрования/дешифрования. Мы можем разделить алгоритмы на 6 групп, как это показано на рис. 2.4. Все протоколы блока используют 8-байтовый вектор инициализации (IV), кроме Fortezza, который применяет 20 байтов IV.

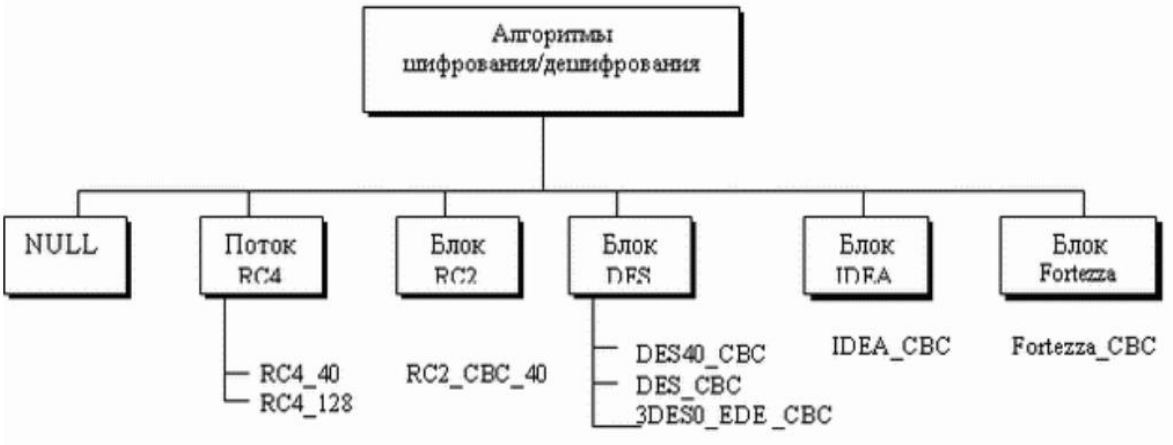


Рис. 4.4. Алгоритмы шифрования/дешифрования

NULL

NULL - категория, которая просто определяет отсутствие алгоритма шифрации/дешифрации.

Поток RC

В режиме потока RC определены два потока алгоритма: RC4-40 (ключ на 40 битов) и RC4-128 (ключ на 128 битов).

Блок RS

В режиме блока RC определен один алгоритм: RC2_CBC_40 (ключ на 40 битов). CBC (Cipher Block Chaining) - сцепление шифрованных блоков.

DES

Все алгоритмы DES определены в режиме блока. DES40_CBC использует ключ на 40 битов. Стандартные DES определены как DES_CBC. 3DES_EDE_CBC используют ключ на 168 битов.

IDEA

В режиме блока IDEA определен один алгоритм - IDEA_CBC, с ключом на 128 битов.

Fortezza

В режиме блока Fortezza определен один алгоритм - FORTEZZA_CBC, с ключом на 96 бит.

Алгоритмы хэширования

SSL использует алгоритмы хэширования, чтобы обеспечить целостность сообщения (установление подлинности сообщения). Имеются хэш-функции, показанные на рис. 2.5.

Рис. 4.5. Алгоритмы хэширования

Null (Пустой указатель)

Две стороны могут отказаться использовать алгоритм хэширования. В этом случае сообщение не заверено.

MD5

Две стороны могут выбрать MD5 как алгоритм хэширования. В этом случае используется алгоритм хэширования MD5 - 128-битовый.

SHA-1

Две стороны могут выбрать SHA как алгоритм хэширования. В этом случае используется алгоритм хэширования SHA-1 на 160 битов.

Алгоритмы сжатия

Как мы уже говорили, сжатие является дополнительной услугой в SSLv3. Для SSLv3 не определен алгоритм сжатия. Поэтому заданным по умолчанию методом сжатия служит NULL. Однако система может использовать любой алгоритм сжатия по выбору сторон.

Генерирование криптографических параметров

Чтобы обеспечить целостность и конфиденциальность сообщения, в SSL необходимо иметь: шесть криптографических объектов секретности, четыре ключа и два инициализирующих вектора (IV). Клиенту нужно: один ключ для передачи сообщения установления подлинности (HMAC - HASH-BASED MESSAGE AUTHENTICATION CODE), один ключ для шифрования и один IV для шифрования блока. Сервер нуждается в том же самом. SSL требует, чтобы ключи для одного направления отличались от ключей для другого направления. Если будет атака в одном направлении, она не затронет другое направление. Для генерации параметров используют следующую процедуру:

1. Клиент и сервер обмениваются двумя случайными числами, одно из которых создано клиентом, а другое - сервером.

2. Клиент и сервер обмениваются одним предварительным главным секретным кодом.

3. Создается 48-байтовый главный секретный код (master secret) из предварительного главного секретного кода (pre-master secret), с применением хэш-функций (SHA-1 и MD5), как это показано на рис. 4.6.

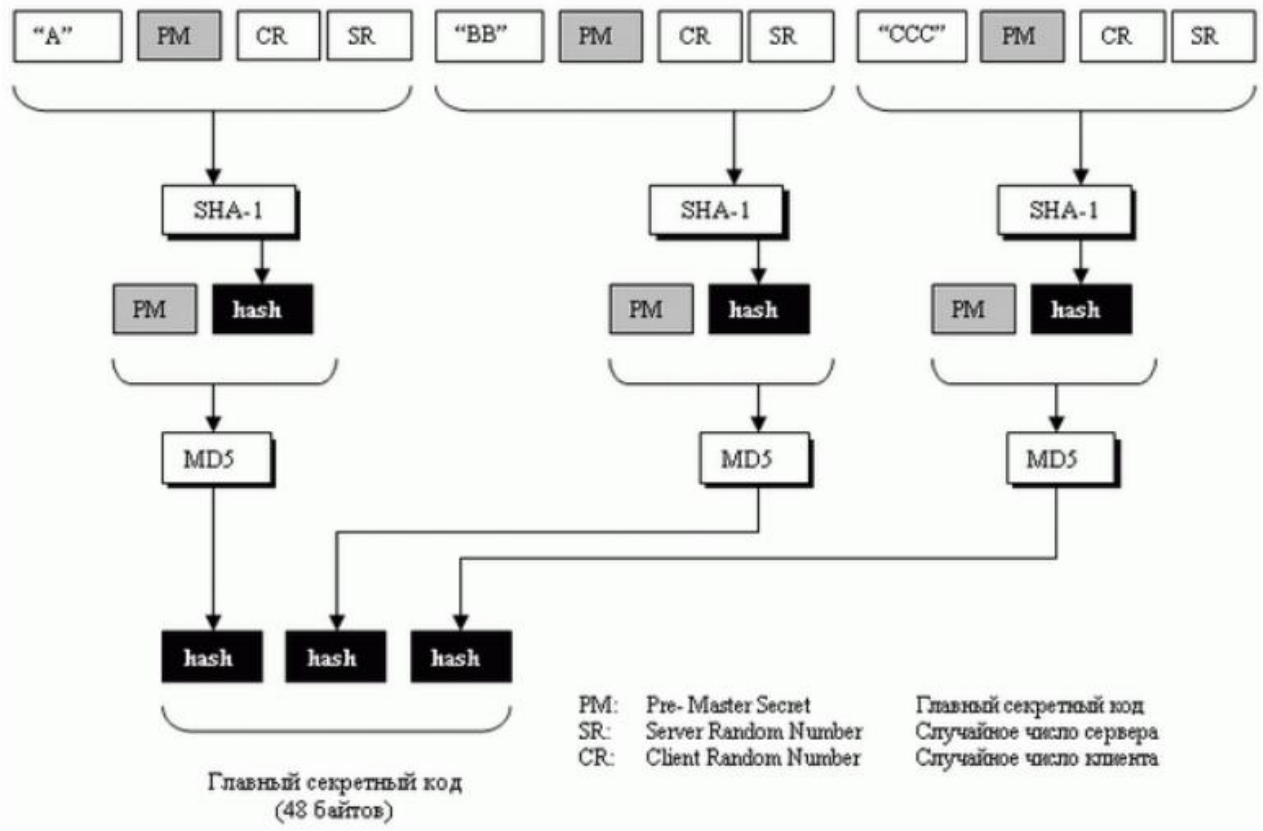


Рис. 4.6. Вычисление главного секретного кода из предварительного главного секретного кода

4. Главный секретный код используется для того, чтобы создать материал для ключей (key material), который имеет переменную длину. Для этого применяют то же самое

множество хэш-функций, что и в предыдущем случае, и подставляют спереди различные константы, как это показано на рис. 4.7. Алгоритм повторяется, пока не получится материал для ключа адекватного размера.

Длина блока материала для ключей зависит от выбранного набора шифра и размера ключей, необходимых для этого набора.

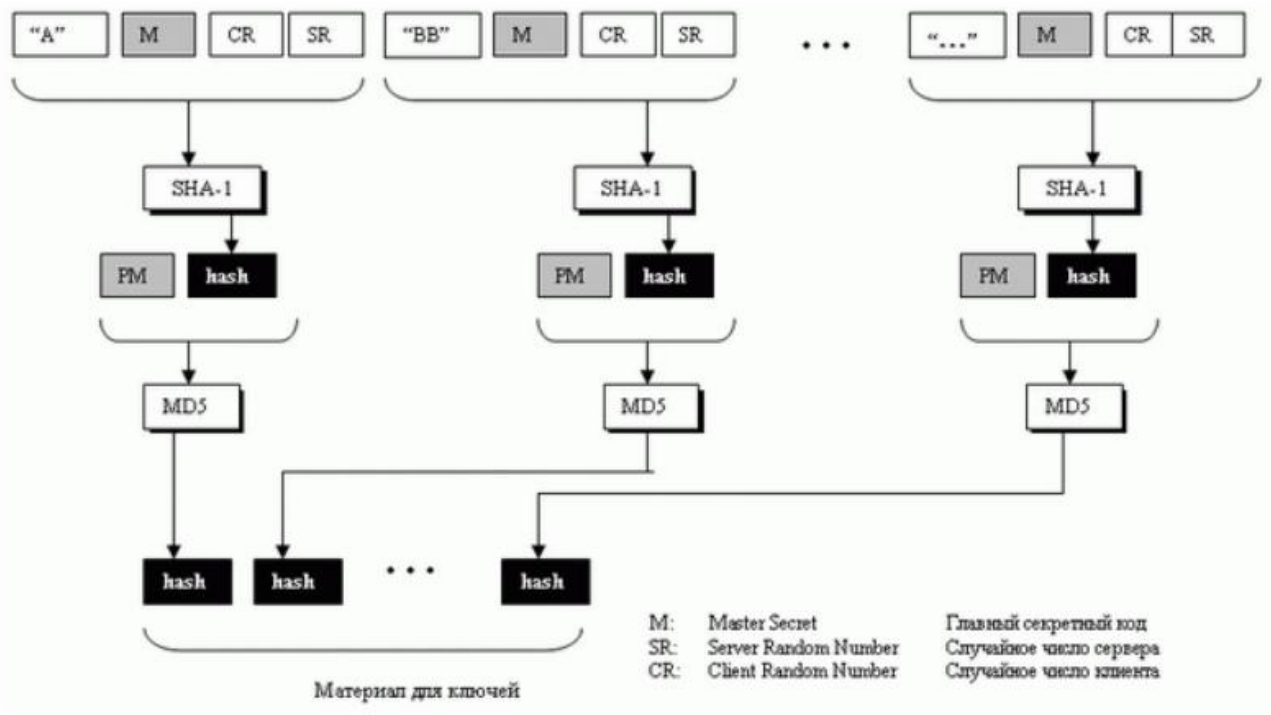


Рис. 4.7. Вычисление материала для ключей из главного секретного кода

5. Из материала для ключей извлекаются шесть различных ключей, как показано на рис. 2.8.

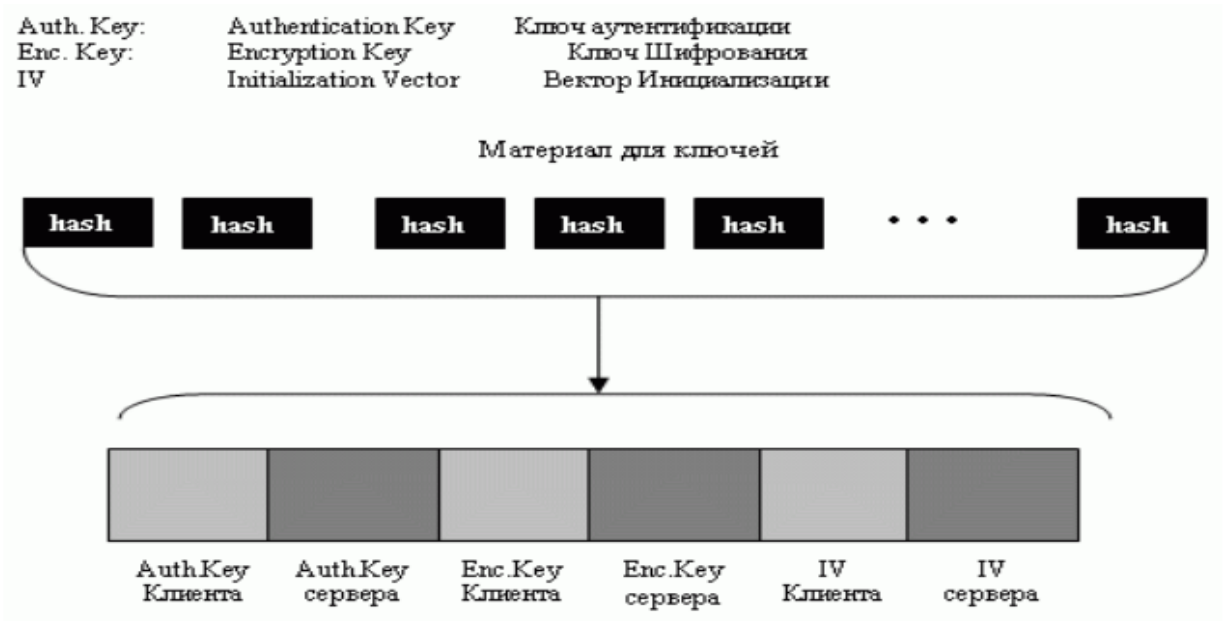


Рис. 4.8. Извлечение криптографических секретных кодов из материала

Сеансы и соединение

SSL отличает соединение от сеанса. Давайте рассмотрим эти два термина. Сеанс – связь между клиентом и сервером. После того как сеанс установлен, эти две стороны имеют общую информацию, такую как идентификатор сеанса, сертификат, подтверждающий подлинность каждого из них (в случае необходимости), метод сжатия(если необходимо), набор шифров и главный секретный код. Эта информация используется для того, чтобы создать ключи для сообщения, содержащего шифр установления подлинности.

Для двух объектов, чтобы начать обмен данными, установление сеанса необходимо, но не достаточно; они должны создать между собой соединение. Эти два объекта обмениваются двумя случайными числами и создают, используя главный секретный код, ключи и параметры, необходимые для того, чтобы обмениваться сообщениями, включая установление подлинности и секретность.

Сеанс может состоять из многих соединений. Соединение между двумя сторонами может быть закончено и восстановлено в пределах одного и того же сеанса. Когда соединение закончено, эти две стороны могут также закончить сеанс, но это необязательно. Сеанс может быть приостановлен и продолжен снова.

Чтобы создавать новый сеанс, эти две стороны должны пройти процесс переговоров. Чтобы возобновлять старый сеанс и создавать только новое соединение, эти две стороны могут пропустить часть переговоров, что уменьшает время вхождения в связь. Не надо создавать главный секретный код, когда сеанс продолжается.

Разделение сеанса от соединения предотвращает высокую стоимость создания главного секретного кода. Если мы разрешаем приостановления и продолжения сеанса, процесс вычисления главного секретного кода может быть устранен. рис. 2.9 иллюстрирует идею сеанса и соединения в этом сеансе.

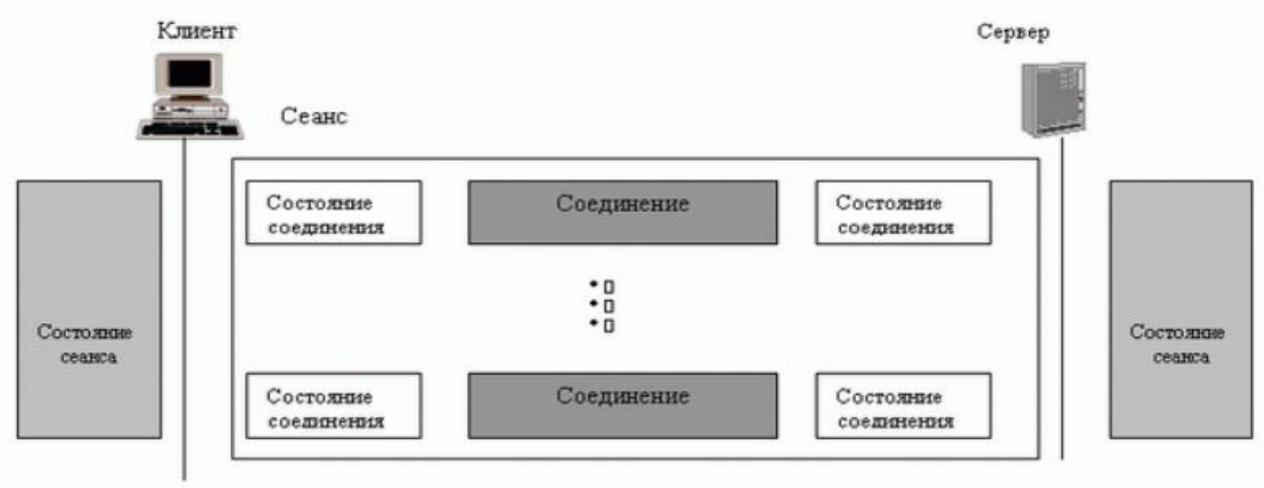


Рис. 4.9. Сеанс и соединение

В сеансе одна сторона играет роль клиента и другая - роль сервера. При соединении обе стороны имеют равные роли, они равны по уровню.

Четыре протокола

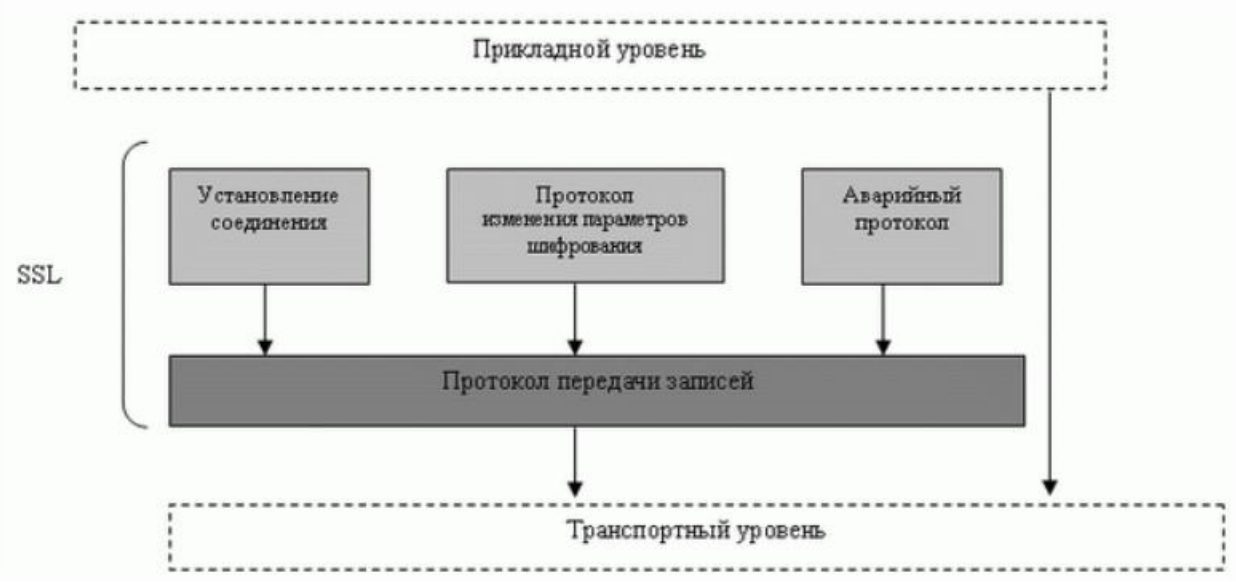


Рис. 4.10. Четыре протокола SSL

Протокол передачи записей - переносящий информацию. Он переносит на транспортный уровень сообщения от трех других протоколов, а также данные, поступающие от прикладного уровня. Сообщения из протокола записей - это полезная нагрузка для транспортного уровня, обычно TCP. Протокол установления соединения обеспечивает параметры безопасности для Протокола записей. Он устанавливает набор шифров и задает ключи и параметры безопасности.

Он также подтверждает, если необходимо, подлинность сервера клиенту и подлинность клиента серверу. Протокол изменения параметров шифрования используется, чтобы передавать сигналы для подготовки к криптографической безопасности. Аварийный протокол нужен, чтобы известить о ситуациях, отклоняющихся от нормы.

Протокол TLS

Безопасность транспортного уровня (TLS - Transport Layer Security) - протокол IEFT, стандартная версия протокола SSL. Эти два протокола очень похожи, но имеют небольшие отличия. Вместо того чтобы описывать TLS полностью, в этой секции мы только отметим отличия между протоколами TLS и SSL.

Генерация криптографической секретности

Генерация криптографической секретности в TLS более сложная, чем в SSL. TLS сначала определяет две функции: функцию расширения данных и псевдослучайную функцию.

Функция расширения данных

Функция расширения данных использует заранее заданный код аутентификации на основе хэширования (HMAC-HASH-BASED MESSAGE AUTHENTICATION CODE), или MD5, или SHA-1 для того, чтобы расширить информацию засекречивания. Эту функцию можно рассматривать как функцию, содержащую множество секций, где каждая секция

создает одно значение хэширования. Расширенная секретность – последовательное соединение значений хэширования. Каждая секция использует два HMAC, информацию засекречивания и начальное число. Функция расширения данных - это формирование цепочки в виде многих секций. Однако чтобы сделать следующую секцию зависимой от предыдущей, второе начальное число – фактически выход первого HMAC предыдущей секции, как это показано на рис. 4.11.

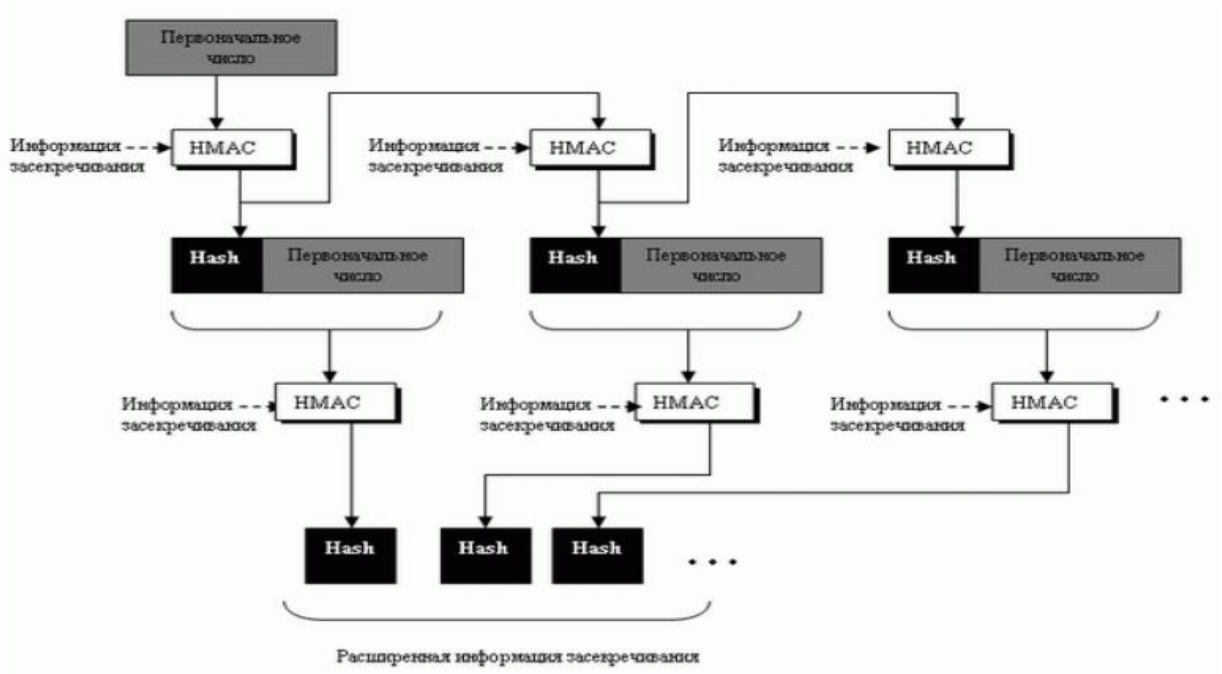


Рис. 4.11. Функция расширения данных

Псевдослучайная функция

TLS определяет псевдослучайную функцию (PRF - PseudoRandom Function), чтобы получить комбинацию двух функций расширения данных: одна из них использует MD5 и другая - SHA-1. На PRF поступает три части информации: секретный код, метка и начальное число.

Метка и начальное число связаны и служат начальным числом для каждой функции расширения данных. Информация засекречивания разделена на две части; каждая часть используется как информация засекречивания для каждой функции расширения данных. Выходы двух функций расширения данных складывают по модулю два, чтобы создать

конечную расширенную информацию засекречивания. Обратите внимание, что поскольку хэш создается MD5 и SHA-1, он имеет различные размеры, поэтому должны быть созданы дополнительные секции функций на базе MD5, чтобы сделать два вывода с одинаковым размером. рис. 4.12 показывает идею применения PRF.

Рис. 4.12. PRF

Главный секретный код

TLS использует функцию PRF, чтобы создать главный секретный код от предварительного главного секретного кода. Это можно сделать, используя предварительный главный секретный код как информацию засекречивания, строку "главный секретный код" - как метку и последовательное соединение информации (конкатенацию) случайного числа клиента и случайное число сервера - как начальное число. Обратите внимание, что метка - фактически код ASCII строки "главного секретного кода". Другими словами, метка определяет выход для создания главного секретного кода. рис. 3.3 иллюстрирует идею.

Рис. 4.13. Генерация главного секретного ключа

Материал для ключей

TLS использует функцию PRF, чтобы создать материал для ключей от главного секретного кода. На сей раз информация засекречивания содержит: главный секретный код; метку - это строка "расширение ключа"; и начальное число – конкатенацию случайного числа сервера и случайного числа клиента, как это показано на рис. 4.4.

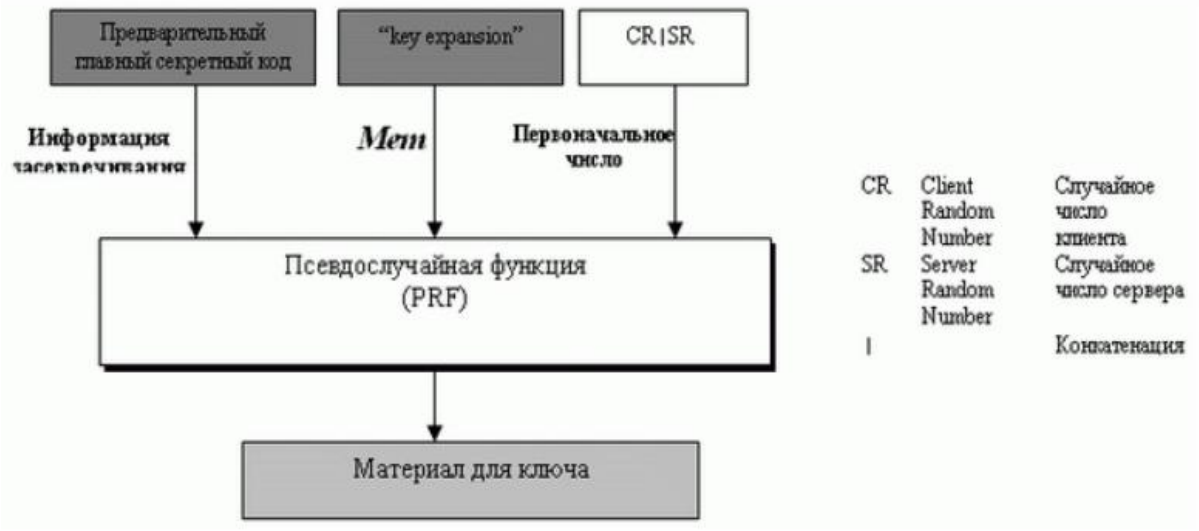


Рис. 4.14. Генерация материала для ключа

Протоколы

Аварийный протокол в TLS поддерживает все аварийные сигналы, определенные в SSL за исключением NoCertificate. TLS также добавляет к списку SSL некоторые новые.

Протокол установления соединения TLS вносит некоторые изменения в протокол установления соединения. Были специально изменены детали сообщения CertificateVerify и сообщения Finished.

Единственное изменение в *протоколе передачи записей* – использование HMAC, формируемы с помощью MAC, чтобы подписать сообщение.

Безопасность сети ПД на сетевом уровне IP SEC

IPSec является неотъемлемой частью IPv6 - Интернет-протокола следующего поколения, и расширением существующие версии Интернет-протокола IPv4. IPSec определён в RFC с 2401 по 2412.

Практически все механизмы сетевой безопасности могут быть реализованы на третьем уровне эталонной модели ISO/OSI в соответствии с рисунком 2.1. Кроме того, IP-уровень можно считать оптимальным для размещения защитных средств, поскольку при этом достигается удачный компромисс между защищенностью, эффективностью функционирования и прозрачностью для приложений.

Уровни TCP/IP	Уровни ISO/OSI
4. Прикладных программ	7. Прикладных программ 6. Представление данных
3. Транспортный	5. Сеансовый 4. Транспортный
2. Межсетевой	3. Сетевой
1. Доступа к сети	2. Канальный 1. Физический

Рис. 4.15. Модель OSI/ISO

Стандартизованными механизмами IP-безопасности могут (и должны) пользоваться протоколы более высоких уровней и, в частности, управляющие протоколы, протоколы конфигурирования и маршрутизации.

Средства безопасности для IP описываются семейством спецификаций IPSec, разработанных рабочей группой IP Security.

Протоколы IPSec обеспечивают управление доступом, целостность вне соединения, аутентификацию источника данных, защиту от воспроизведения, конфиденциальность и частичную защиту от анализа трафика.

Основополагающими понятиями IPSec являются:

- - аутентификационный заголовок (AH);
- - безопасное сокрытие данных (ESP);
- - режимы работы: туннельный и транспортный;
- - контексты (ассоциации) безопасности (SA);
- - управление ключами (IKE);

Основные составляющие архитектуры и их особенности

Архитектура средств безопасности для IP-уровня специфицирована в документе Security Architecture for the Internet Protocol. Ее основные составляющие представлены в соответствии с рисунком 4.2. Это, прежде всего протоколы обеспечения аутентичности (протокол аутентифицирующего заголовка - Authentication Header, AH) и конфиденциальности (протокол инкапсулирующей защиты содержимого - Encapsulating Security payload, ESP), а также механизмы управления криптографическими ключами. На более низком архитектурном уровне располагаются конкретные алгоритмы шифрования, контроля целостности и аутентичности. Наконец, роль фундамента выполняет так называемый домен интерпретации (Domain of Interpretation, DOI), являющийся, по сути, базой данных, хранящей сведения об алгоритмах, их параметрах, протокольных идентификаторах.

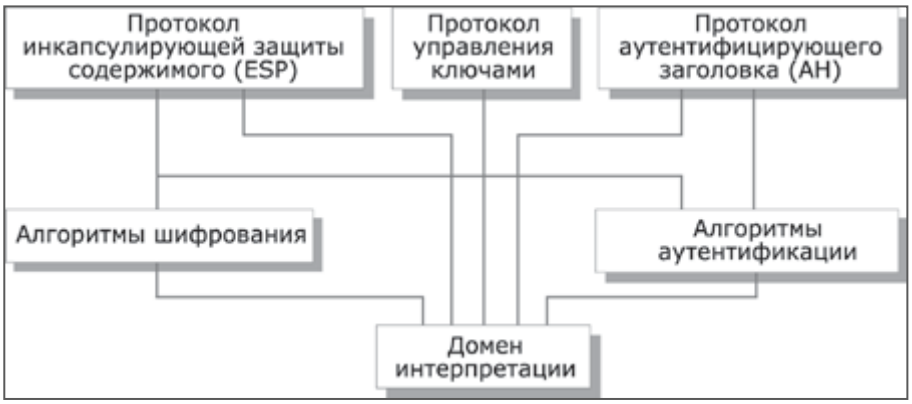


Рис. 4.16. Основные элементы архитектуры средств безопасности IP-уровня

Деление на уровни важно для всех аспектов информационных технологий. Там же, где участвует еще и криптография, важность возрастает вдвойне, поскольку приходится считаться не только с чисто техническими факторами, но и с особенностями законодательства различных стран, с ограничениями на экспорт и/или импорт криптосредств.

IPSec поддерживает две формы целостности: целостность соединения и частичную целостность последовательности. Целостность соединения является сервисом безопасности, который определяет модификацию конкретной IP датаграммы, безотносительно последовательности датаграмм в потоке трафика. Частичная целостность последовательности является anti-reply сервисом, с помощью которого определяется получение дубликатов IP датаграм.

Эти сервисы как раз и реализуются с использованием двух протоколов обеспечения безопасного трафика, Authentication Header (AH) и Encapsulating Security Payload (ESP), и с помощью процедур и протоколов управления криптографическим ключом. Множество применяемых IPSec протоколов и метод их использования определяются требованиями безопасности.

Когда данные механизмы установлены корректно, они не мешают пользователям, хостам и другим компонентам Internet, которые не применяют данные механизмы безопасности для защиты своего трафика. Протоколы обеспечения аутентичности и конфиденциальности в IPSec не зависят от конкретных криптографических алгоритмов. (Более того, само деление на аутентичность и конфиденциальность предоставляет и разработчикам, и пользователям дополнительную степень свободы в ситуации, когда к криптографическим относят только шифровальные средства.) В каждой стране могут применяться свои алгоритмы, соответствующие национальным стандартам, но для этого, как минимум, нужно позаботиться об их регистрации в домене интерпретации. Это означает возможность выбора

различного набора алгоритмов без воздействия на другие части реализации. Например, различные группы пользователей могут выбрать при необходимости различные наборы алгоритмов.

Определен стандартный набор алгоритмов по умолчанию для обеспечения интероперабельности. Использование этих алгоритмов совместно с защитой трафика на основе IPSec и протоколами управления ключа позволяет обеспечить высокую степень криптографической безопасности.

Алгоритмическая независимость протоколов имеет и оборотную сторону, состоящую в необходимости предварительного согласования набора применяемых алгоритмов и их параметров, поддерживаемых общающимися сторонами. Иными словами, стороны должны выработать общий контекст безопасности (Security Association, SA) и затем использовать такие его элементы, как алгоритмы и их ключи. SA подробно рассматривается далее. За формирование контекстов безопасности в IPSec отвечает особое семейство протоколов ISAKMP, которое рассматривается также в отдельном разделе.

Безопасность, обеспечиваемая IPSec, зависит от многих факторов операционного окружения, в котором IPSec выполняется. Например, от безопасности ОС, источника случайных чисел, плохих протоколов управления системой.

Размещение и функционирование IPSec

IPSec выполняется на хосте или шлюзе безопасности, обеспечивая защиту IP- трафика. Термин «шлюз безопасности» используется для обозначения промежуточной системы, которая реализует IPSec-протоколы. Защита основана на требованиях, определенных в Базе Данных Политики Безопасности (Security Policy Database- SPD), определяемой и поддерживаемой системным администратором. Пакеты обрабатываются одним из трех способов на основании соответствия информации заголовка IP или транспортного уровня записям в SPD. Каждый пакет либо отбрасывается сервисом безопасности IPSec, либо пропускается без изменения, либо обрабатывается сервисом IPSec на основе применения определенной политики.

IPSec обеспечивает сервисы безопасности на IP-уровне, выбирая нужные протоколы безопасности, определяя алгоритмы, используемые сервисами, и предоставляя все криптографические ключи требуемым сервисам. IPSec может использоваться для защиты одного или нескольких путей между парой хостов, между парой шлюзов безопасности или между шлюзом безопасности и хостом.

IPSec использует два протокола для обеспечения безопасности трафика - Authentication Header (AH) и Encapsulating Security Payload (ESP). Хотя бы один из этих сервисов должен быть задействован при использовании ESP.

Эти протоколы могут применяться как по отдельности так и в комбинации с друг другом для обеспечения необходимого набора сервисов безопасности в IPv4 и IPv6. Каждый протокол поддерживает два режима использования: режим транспорта и режим туннелирования. В транспортном режиме протоколы обеспечивают защиту главным образом для протоколов более высокого уровня; в режиме туннелирования протоколы применяются для скрытия IP-заголовков исходных пакетов. Разница между двумя режимами рассматривается дальше.

IPSec позволяет системному администратору управлять детализацией, с которой предоставляется сервис безопасности. Например, можно создать единственный зашифрованный туннель между двумя безопасными шлюзами, или для каждого TCP соединения может быть создан зашифрованный туннель между парой хостов. IPSec позволяет указывать следующие параметры:

а) какие сервисы используются, и в какой комбинации;

б) необходимый уровень детализации применяемой защиты;

в) алгоритмы, используемые для обеспечения безопасности на основе криптографии.

Существует несколько способов реализации IPSec на хосте или в соединении с роутером или firewall (для создания безопасного шлюза). Несколько общих примеров:

а) интеграция IPSec в конкретную реализацию IP, что требует доступа к исходному коду IP и применимо как к хостам, так и к шлюзам безопасности;

б) bump-in-the-stack (BITS) реализации, где IPSec действует внизу существующей реализации стека протоколов IP, между обычным IP и локальными сетевыми драйверами; доступа к исходному коду стека IP в данном контексте не требуется, что делает такой подход пригодным для встраивания в существующие системы, и реализации на хостах;

в) использование внешнего криптопроцессора (обычно в военных и в некоторых коммерческих системах), как правило, это является Bump-in-the-stack (BITS) реализацией, используется как на хостах, так и на шлюзах, обычно BITS-устройства являются IP-адресуемыми;

Транспортный режим работы

В этом варианте механизмы безопасности применяются только для протоколов, начиная с транспортного (TCP) уровня и выше, оставляя данные самого сетевого уровня (заголовок IP) без дополнительной защиты. Места размещения дополнительной информации, вставляемой протоколами в пакет, представлены в соответствии с рисунком

Рис. 4.17. Транспортный режим

Туннельный режим работы

Этот режим интересен тем, что обеспечивает защиту также и данных сетевого уровня путем добавления нового IP-заголовка. После определения ассоциаций безопасности (например, между двумя шлюзами) истинные адреса хостов отправления и назначения (и другие служебные поля) полностью защищаются от модификаций для AH или вообще скрываются для ESP, а в новый заголовок выставляются адреса и другие данные для шлюзов (отправления/получения). В соответствии с рисунком 2.4 видны преимущества и недостатки обоих протоколов. ESP обеспечивает сокрытие данных, но не полную аутентификацию всего пакета. AH полностью аутентифицирует, но не скрывает данные. В этом причина того, что для обеспечения высокого уровня безопасности, применение протоколов совмещается.

Рис. 4.18. Туннельный режим

Контексты безопасности и управление ключами

Формирование контекстов безопасности в IPSec разделено на две фазы. Сначала создается управляющий контекст, назначение которого - предоставить доверенный канал, т. е. аутентифицированный, защищенный канал для выработки (в рамках второй фазы) протокольных контекстов и, в частности, для формирования криптографических ключей, используемых протоколами AH и ESP.

В принципе, для функционирования механизмов IPSec необходимы только протокольные контексты; управляющий играет вспомогательную роль. Более того, явное выделение двух фаз утяжеляет и усложняет формирование ключей, если рассматривать последнее как однократное действие. Тем не менее, из архитектурных соображений управляющие контексты не только могут, но и должны существовать, поскольку обслуживают все протокольные уровни стека TCP/IP, концентрируя в одном месте необходимую функциональность. Первая фаза начинается в ситуации, когда взаимодействующие стороны не имеют общих секретов (общих ключей) и не уверены в аутентичности друг друга. Если с самого начала не создать доверенный канал, то для выполнения каждого управляющего действия с ключами (их модификация, выдача диагностических сообщений и т.п.) в каждом протоколе (AH, ESP, TLS и т.д.) этот канал придется формировать заново.

Общие вопросы формирования контекстов безопасности и управления ключами освещаются в спецификации «Контексты безопасности и управление ключами в Internet» (Internet Security Association and Key Management Protocol, ISAKMP). Здесь вводятся две фазы выработки протокольных ключей, определяются виды управляющих информационных

обменов и используемые форматы заголовков и данных. Иными словами строится протокольно-независимый каркас.

Существует несколько способов формирования управляющего контекста. Они различаются двумя показателями:

- используемым механизмом выработки общего секретного ключа;

- степенью защиты идентификаторов общающихся сторон;

В простейшем случае секретные ключи задаются заранее (ручной метод распределения ключей). Для небольших сетей такой подход вполне работоспособен, но он не является масштабируемым. Последнее свойство может быть обеспечено при автоматической выработке и распределении секретных ключей в рамках протоколов, основанных на протоколе Диффи-Хеллмана. Пример тому - «Протокол для обмена ключами в Internet « (The Internet Key Exchange, IKE).

Протокол Ди́ффи-Хе́ллмана (англ. Diffie-Hellman, DH) — криптографический протокол, позволяющий двум и более сторонам получить общий секретный ключ, используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования дальнейшего обмена с помощью алгоритмов симметричного шифрования. Алгоритм был впервые опубликован Уитфилдом Диффи (Whitfield Diffie) и Мартином Хеллманом в 1976 году.

При формировании управляющего контекста идентификаторы общающихся сторон (например, IP-адреса) могут передаваться в открытом виде или шифроваться. Поскольку ISAKMP предусматривает функционирование в режиме клиент/сервер (т. е. ISAKMP-сервер может формировать контекст для клиента), сокрытие идентификаторов в определенной степени повышает защищенность от пассивного прослушивания сети.

Последовательность передаваемых сообщений, позволяющих сформировать управляющий контекст и обеспечивающих защиту идентификаторов, выглядит в соответствии с рисунком 4.19.

Рис. 4.19. Формирование управляющего контекста

В первом сообщении (1) инициатор направляет предложения по набору защитных алгоритмов и конкретных механизмов их реализации. Предложения упорядочиваются по степени предпочтительности (для инициатора). В ответном сообщении (2) партнер информирует о сделанном выборе - какие алгоритмы и механизмы его устраивают. Для каждого класса защитных средств (генерация ключей, аутентификация, шифрование) выбирается только один элемент.

В сообщениях (3) и (4) инициатор и партнер отправляют свои части ключевого материала, необходимые для выработки общего секретного ключа (опускаются детали, специфичные для алгоритма Диффи-Хеллмана). Одноразовые номера (nonce) представляют собой псевдослучайные величины, служащие для защиты от воспроизведения сообщений.

Посредством сообщений (5) и (6) происходит обмен идентификационной информацией, подписанной (с целью аутентификации) секретным ключом отправителя и зашифрованной выработанным на предыдущих шагах общим секретным ключом. Для аутентификации предполагается использование сертификатов открытых ключей. В число подписываемых данных входят одноразовые номера.

В представленном виде протокол формирования управляющего контекста защищает от атак, производимых нелегальным посредником, а также от нелегального перехвата соединений. Для защиты от атак на доступность, для которых характерно прежде всего навязывание интенсивных вычислений, присущих криптографии с открытым ключом, применяются так называемые идентифицирующие цепочки (cookies). Эти цепочки, формируемые инициатором и его партнером с использованием текущего времени (для

защиты от воспроизведения), на самом деле присутствуют во всех ISAKMP-сообщениях и в совокупности идентифицируют управляющий контекст (в первом сообщении, по понятным причинам, фигурирует только цепочка инициатора). Согласно спецификациям, заголовок ISAKMP-сообщения имеет вид в соответствии с рисунком 4.20.

Если злоумышленник пытается «завалить» кого-либо запросами на создание управляющего контекста, подделывая при этом свой IP-адрес, то в сообщении (3) он не сможет предъявить идентифицирующую цепочку партнера, поэтому до выработки общего секретного ключа и, тем более, электронной подписи и полномасштабной проверки аутентичности дело попросту не дойдет.

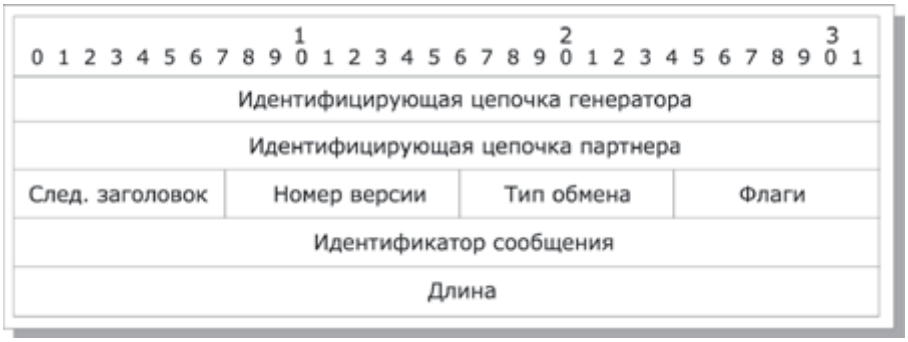


Рис. 4.20. Формат заголовка ISAKMP-сообщения

Управляющие контексты являются двунаправленными в том смысле, что любая из общающихся сторон может инициировать с их помощью выработку новых протокольных контекстов или иные действия. Для передачи ISAKMP-сообщений используется любой протокол, однако в качестве стандартного принят UDP с номером порта 500.

Протокольные контексты и политика безопасности

Системы, реализующие IPSec, должны поддерживать две базы данных:

- базу данных политики безопасности (Security Policy Database, SPD);

- базу данных протокольных контекстов безопасности (Security Association Database, SAD);

Все IP-пакеты (входящие и исходящие) сопоставляются с упорядоченным набором правил политики безопасности. При сопоставлении используется фигурирующий в каждом правиле селектор - совокупность анализируемых полей сетевого уровня и более высоких протокольных уровней. Первое подходящее правило определяет дальнейшую судьбу пакета:

- пакет может быть ликвидирован;
- пакет может быть обработан без участия средств IPSec;

- пакет должен быть обработан средствами IPSec с учетом набора протокольных контекстов, ассоциированных с правилом.

Таким образом, системы, реализующие IPSec, функционируют как межсетевые экраны, фильтруя и преобразуя потоки данных на основе предварительно заданной политики безопасности.

Далее рассматриваются контексты и политика безопасности, а также порядок обработки сетевых пакетов.

Протокольный контекст безопасности в IPSec - это однонаправленное соединение (от источника к получателю), предоставляющее обслуживаемым потокам данных набор защитных сервисов в рамках какого-то одного протокола (AH или ESP). В случае симметричного взаимодействия партнерам придется организовать два контекста (по одному в каждом направлении). Если используются и AH, и ESP, потребуется четыре контекста.

Элементы базы данных протокольных контекстов содержат следующие поля (в каждом конкретном случае некоторые значения полей будут пустыми):

- используемый в протоколе AH алгоритм аутентификации, его ключи и т.п.;
- используемый в протоколе ESP алгоритм шифрования, его ключи, начальный вектор и т.п.;
- используемый в протоколе ESP алгоритм аутентификации, его ключи и т.п.;
- время жизни контекста;
- режим работы IPSec: транспортный или туннельный;
- максимальный размер пакетов;
- группа полей (счетчик, окно, флаги) для защиты от воспроизведения пакетов.

Пользователями протокольных контекстов, как правило, являются прикладные процессы. Вообще говоря, между двумя узлами сети может существовать произвольное число протокольных контекстов, так как число приложений в узлах произвольно. В качестве пользователей управляющих контекстов обычно выступают узлы сети (поскольку в этих контекстах желательно сосредоточить общую функциональность, необходимую сервисам безопасности всех протокольных уровней эталонной модели для управления криптографическими ключами).

Управляющие контексты - двусторонние, т. е. любой из партнеров может инициировать новый ключевой обмен. Пара узлов может одновременно поддерживать несколько активных управляющих контекстов, если имеются приложения с существенно разными криптографическими требованиями. Например, допустима выработка части ключей на

основе предварительно распределенного материала, в то время как другая часть порождается по алгоритму Диффи-Хеллмана.

Протокольный контекст для IPSec идентифицируется целевым IP-адресом, протоколом (AH или ESP), а также дополнительной величиной - индексом параметров безопасности (Security Parameter Index, SPI). Последняя величина необходима, поскольку могут существовать несколько контекстов с одинаковыми IP-адресами и протоколами. Далее показано, как используются индексы SPI при обработке входящих пакетов.

IPSec обязывает поддерживать ручное и автоматическое управление контекстами безопасности и криптографическими ключами. В первом случае все системы заранее снабжаются ключевым материалом и иными данными, необходимыми для защищенного взаимодействия с другими системами. Во втором - материал и данные вырабатываются динамически, на основе определенного протокола - IKE, поддержка которого обязательна.

Протокольный контекст создается на базе управляющего с использованием ключевого материала и средств аутентификации и шифрования последнего. В простейшем случае, когда протокольные ключи генерируются на основе существующих, последовательность передаваемых сообщений выглядит в соответствии с рисунком.

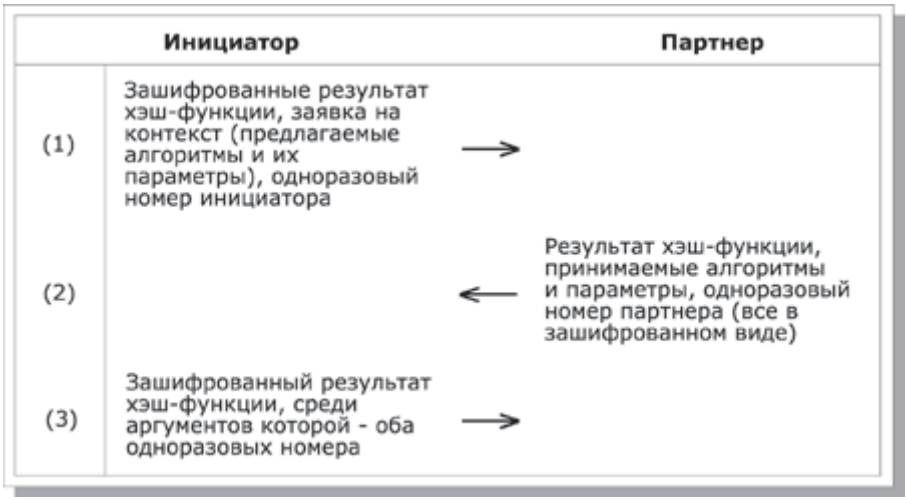


Рис. 4.21. Формирование протокольного контекста

Аутентификационный заголовок

Аутентификационный заголовок (англ. authentication header) AH предназначен для обеспечения аутентификации отправителя, контроля целостности данных и опционально для предотвращения повторной посылки (англ. replay) пакета - при условии, что принимающая сторона настроена производить проверку последовательного номера пакета. Поля IP-пакета, которые изменяются в пути следования, не подлежат контролю целостности. AH защищает

данные протоколов более высоких уровней и те поля IP-заголовков, которые не меняются на маршруте доставки или меняются предсказуемым образом (число «непредсказуемых» полей невелико - это prio. (Traffic Class), Flow Label и Hop Limit. Предсказуемо меняется целевой адрес при наличии дополнительного заголовка исходящей маршрутизации.).

Формат заголовка AH показан в соответствии с рисунком. поле «Следующий заголовок» (Next header) идентифицирует тип следующих значимых данных за аутентификационным заголовком; значения поля берутся из предопределенного множества установленных IANA (Internet Assigned Numbers Authority) номеров.

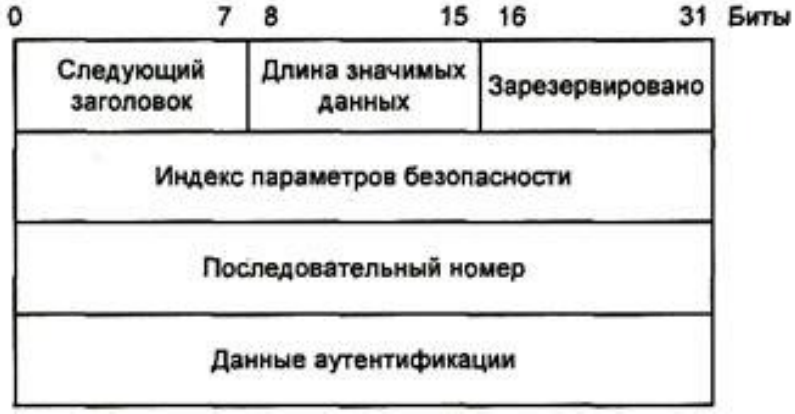


Рис. 4.22. Формат заголовка AH

Поле «Длина значимых данных» (Payload length) определяет длину самого AH в 32-битовых словах минус 2 слова, поскольку для заголовков, расширяемых для IPv6, установлено требование сокращения длины заголовка на 64 бита;

Зарезервированное поле должно быть нулевым;

Поле «Индекс параметров безопасности» (Security Parameters Index - SPI) - это значение, которое в совокупности с адресом назначения и самим протоколом (в данном случае AH) однозначно определяет ассоциацию безопасности (Security Association - SA) для данной датаграммы в виде 32-битного номера; номера с 1 по 255 зарезервированы IANA; SPI, равный нулю, означает, что SA не установлена; ассоциация безопасности - это набор параметров (версия алгоритмов шифрования и аутентификации, схема обмена ключами и т. п.), определяющих, каким образом будет обеспечиваться защита данных;

Поле «Последовательный номер» (Sequence number) - зто монотонно возрастающий от 0 (при установлении SA) номер пакета. Он используется для возможности контроля получателем ситуации повторной пересылки пакетов;

Поле «Данные аутентификации» (Authentication data) содержат значение контроля целостности (Integrity check value - ICV), рассчитанное по всем данным, которые не

изменяются в пути следования пакета или предсказуемы на момент достижения им получателя. Значение ICV рассчитывается в зависимости от алгоритма, определенного в SA, например код аутентификации сообщения (Message Authentication Code - MAC) с ключом симметричного или асимметричного алгоритма или хэшфункции;

Безопасное сокрытие существенных данных

Протокол инкапсулирующей защиты содержимого (Encapsulating Security payload, ESP) предоставляет три вида сервисов безопасности:

1. обеспечение конфиденциальности (шифрование содержимого IP-пакетов, а также частичная защита от анализа трафика путем применения туннельного режима);
2. обеспечение целостности IP-пакетов и аутентификации источника данных;
3. обеспечение защиты от воспроизведения IP-пакетов;

Функциональность ESP шире, чем у AH (добавляется шифрование); ESP не обязательно предоставляет все сервисы, но либо конфиденциальность, либо аутентификация должны быть задействованы. Формат заголовка ESP выглядит в соответствии с рисунком 2.9. Это не столько заголовок, сколько обертка (инкапсулирующая оболочка) для зашифрованного содержимого. Например, ссылку на следующий заголовок нельзя выносить в начало, в незашифрованную часть, так как она лишится конфиденциальности.

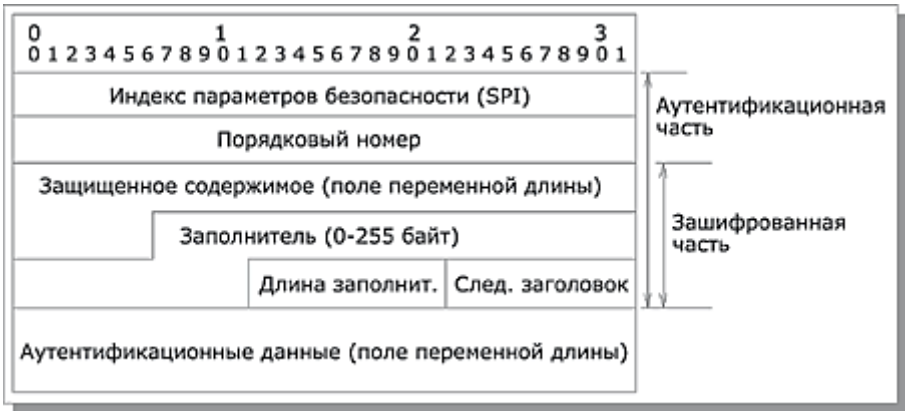


Рисунок 4.23. Формат заголовка ESP

Поля "Индекс параметров безопасности (SPI)", "Порядковый номер" и "Аутентификационные данные" (последнее присутствует только при включенной аутентификации) имеют тот же смысл, что и для AH. ESP аутентифицирует лишь зашифрованную часть пакета (плюс два первых поля заголовка).

Применение протокола ESP к исходящим пакетам можно представлять себе следующим образом. Пусть остаток пакета та его часть, которая помещается после предполагаемого места вставки заголовка ESP. При этом не важно, какой режим используется - транспортный или туннельный. Шаги протокола таковы:

1. - остаток пакета копируется в буфер;
2. - к остатку приписываются дополняющие байты, их число и номер (тип) первого заголовка остатка, с тем чтобы номер был прижат к границе 32-битного слова, а размер буфера удовлетворял требованиям алгоритма шифрования;
3. - текущее содержимое буфера шифруется;
4. - в начало буфера приписываются поля "Индекс параметров безопасности (SPI)" и "Порядковый номер" с соответствующими значениями;
5. - пополненное содержимое буфера аутентифицируется, в его конец помещается поле "Аутентификационные данные";
6. - в новый пакет переписываются начальные заголовки старого пакета и конечное содержимое буфера;

Если в ESP включены и шифрование, и аутентификация, то аутентифицируется зашифрованный пакет. Для входящих пакетов действия выполняются в обратном порядке, т. е. сначала производится аутентификация. Это позволяет не тратить ресурсы на расшифровку поддельных пакетов, что в какой-то степени защищает от атак на доступность.

Протокол обмена ключами – IKE

Поскольку основным механизмом обеспечения безопасности данных протокола являются криптографические методы, участники защищенного соединения должны наладить обмен соответствующими криптографическими ключами. Обеспечить настройку процесса такого обмена можно вручную и автоматически. Первый способ допустим для небольшого количества достаточно статичных систем, а в общем случае это производится автоматически.

Для автоматического обмена ключами по умолчанию используется Протокол управления ключами в Интернете (Internet Key Management Protocol - IKMP), иначе называемый Обмен ключами в Интернете (Internet Key Exchange - IKE). Дополнительно или альтернативно могут быть применены другие протоколы, такие как Kerberos или SKIP.

IKE совмещает в себе три основных направления (отдельных протокола):

- ISAKMP (Internet Security Association and Key Management Protocol) - протокол ассоциаций безопасности и управления ключами в интернете; это общее описание

(framework) для обеспечения аутентификации и обмена ключей без указания конкретных прикладных алгоритмов;

- Oakley (Oakley key determination protocol) - протокол определения ключей Окли; он описывает последовательности обмена ключами - моды (mode) и описывает предоставляемые ими функции;
- SKEMI (Secure Key Exchange Mechanism for Internet) - механизм безопасного обмена ключами в Интернете; он описывает многофункциональные технологии, предоставляющие анонимность, неотрекаемость (аппелируемость) и быстрое обновление ключей;

IKE содержит две фазы согласования ключей. В первой фазе происходит создание защищенного канала, во второй - согласование и обмен ключами, установление SA. Первая фаза использует один из двух режимов: основной (англ. Main Mode) или агрессивный (англ. Aggressive Mode). Различие между ними в уровне защищенности и скорости работы. Основной режим, более медленный, защищает всю информацию, передаваемую между узлами. Агрессивный режим для ускорения работы оставляет ряд параметров открытыми и уязвимыми для прослушивания, его рекомендуется использовать только в случае, когда критическим вопросом является скорость работы. Во второй фазе используется быстрый режим (англ. Quick Mode), названный так потому, что не производит аутентификации узлов, считая, что это было сделано в первой фазе. Эта фаза обеспечивает обмен ключами, с помощью которых происходит шифрование данных.

Расширенный обзор безопасных ассоциаций

Понятие "безопасные ассоциации" (Security Association - SA) является фундаментальным в IPSec.

SA есть симплексное (однонаправленное) логическое соединение, создаваемое для обеспечения безопасности. Весь трафик, передаваемый по SA, некоторым образом обрабатывается в целях обеспечения безопасности. И AH, и ESP используют в своей работе SAs. Одной из основных функций IKE является установление SA. Далее приводятся различные аспекты управления SA, определим требуемые характеристики управления политикой SA, обработку трафика и технологии управления SA.

SA есть совокупность параметров соединения, которые дают возможность сервисам обеспечивать безопасный трафик. SA определяет использование AH или ESP. Если к потоку трафика применяются оба протокола, AH и ESP, то создаются две SAs. При

двунаправленном соединении между двумя хостами или между двумя шлюзами безопасности требуется два SA (по одному на каждое направление).

SA однозначно определяется тройкой, состоящей из Security Parameter Index (SPI), IP Destination Address (адресом назначения) и идентификатора протокола безопасности (AH или ESP). В принципе адрес назначения может быть единственным адресом, широковещательным (broadcast) адресом или групповым (multicast) адресом. Однако механизм управления SA в настоящее время определяется только для единственной SA. Следовательно, SAs будут описаны в контексте point-to-point соединения, даже если концепция также применяется в случае point-to-multipoint.

Определены два режима SA: режим транспорта и режим туннелирования. Транспортный режим SA обеспечивает безопасную связь между двумя хостами. В IPv4 заголовок протокола безопасности транспортного режима появляется сразу после IP заголовка и всех опций и перед любыми протоколами более высокого уровня (TCP или UDP). В случае ESP транспортный режим SA обеспечивает сервисы безопасности только для протоколов более высокого уровня, но не для IP-заголовка. В случае AH защита также распространяется на отдельные части IP-заголовка.

Другим режимом SA является режим туннелирования. Если хотя бы одним из концов соединения является шлюз безопасности, то SA обязательно должна выполняться в туннелирующем режиме. SA между двумя шлюзами безопасности всегда находится в туннелирующем режиме, так же, как и SA между хостом и шлюзом безопасности. Заметим, что когда трафик предназначен для шлюза безопасности, например, в случае SNMP-команд, шлюз безопасности рассматривается как хост, и допустим транспортный режим. Два хоста могут при желании так же устанавливать туннелирующий режим.

В туннелирующем режиме SA существует "внешний" IP-заголовок, который определяет пункт назначения IPSec, и "внутренний" IP-заголовок, который определяет конечный пункт назначения для пакета. Заголовок протокола безопасности расположен после внешнего IP-заголовка и перед внутренним IP-заголовком. Если AH используется в туннелирующем режиме, части внешнего IP заголовка являются защищенными, как и весь туннелируемый IP-пакет, т.е. все внутренние заголовки защищены, как и все протоколы более высокого уровня. Если применяется ESP, зашита обеспечивается только для туннелируемого пакета, а не для внешнего IP-заголовка.

Программная реализация

Для создания защищенного соединения необходимо получить сертификат или создать его самому. Создать SSL сертификат самому можно средствами свободно распространяемого пакета IIS6 Resource Kit Tools. Может потребоваться установка служб. Для этого в Панели управления необходимо выбрать раздел «Программы и компоненты» и в открывшемся окне в меню слева нажать на ссылку «Включение или отключение компонентов Windows». Далее в открывшемся окне нужно включить компонент «Службы IIS».

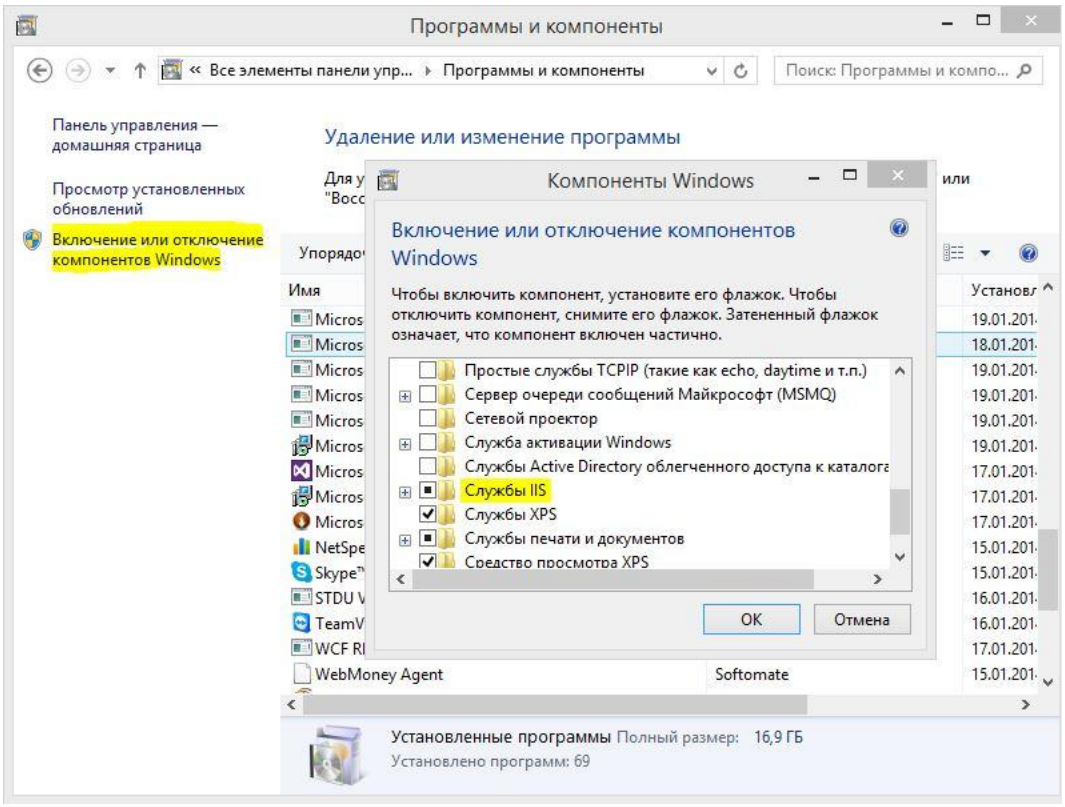


Рис. 4.24. Активация службы ISS

Выбираем службу ISS

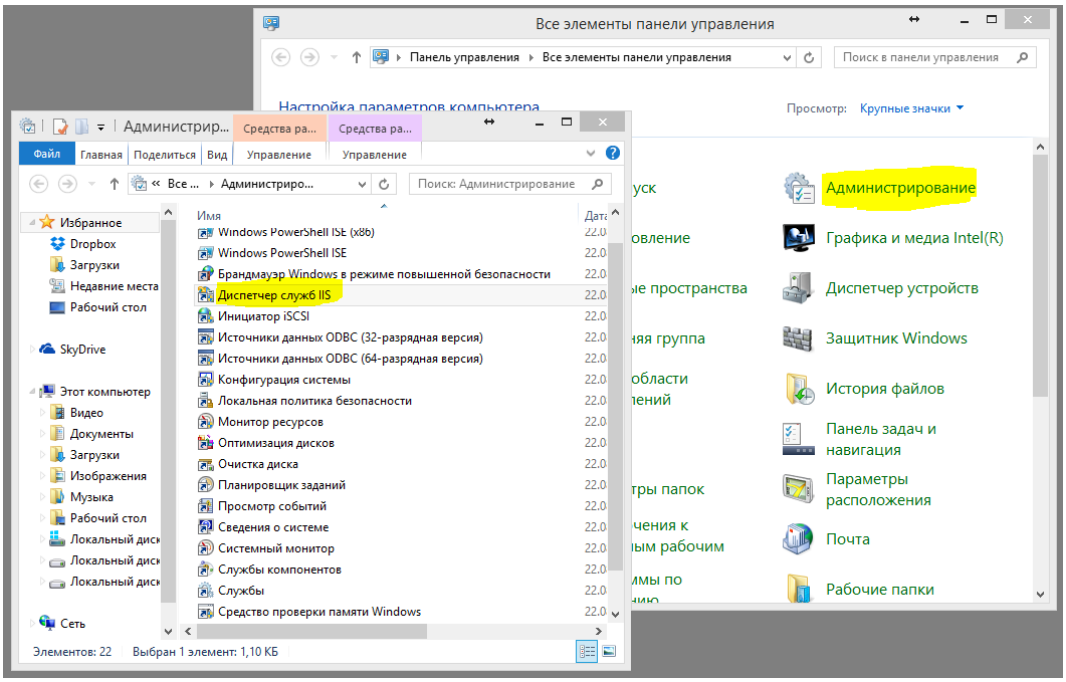


Рис. 4.25. Выбор службы ISS

В открывшемся окне жмем на «Сертификаты сервера»

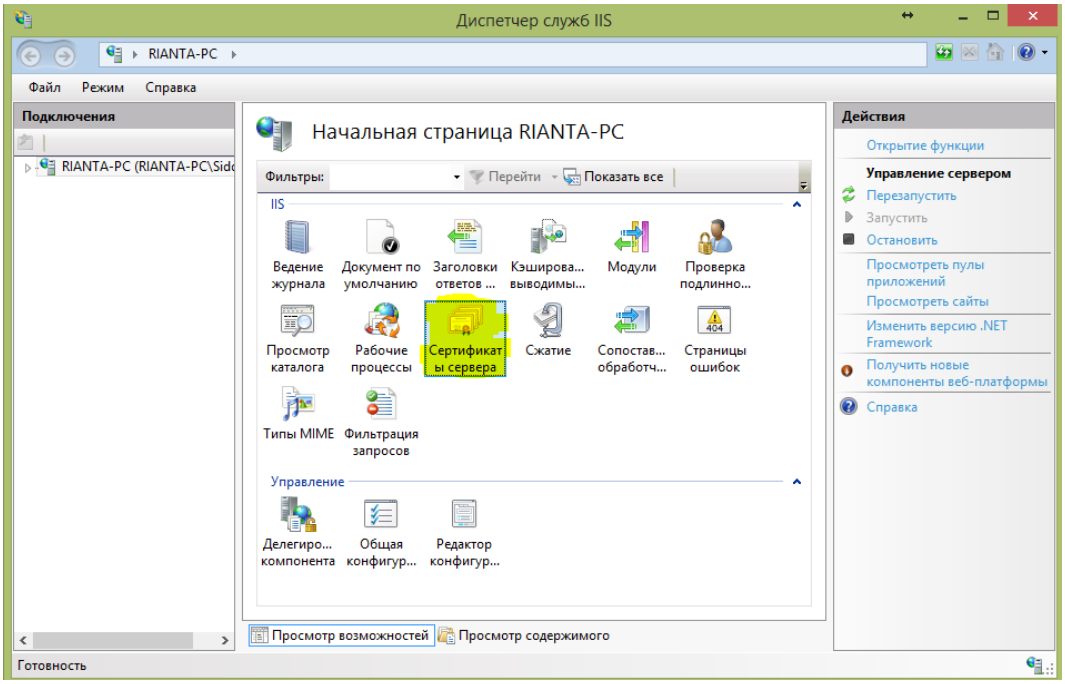


Рис. 4.26. Создание сертификата

Создаем самозаверенный сертификат

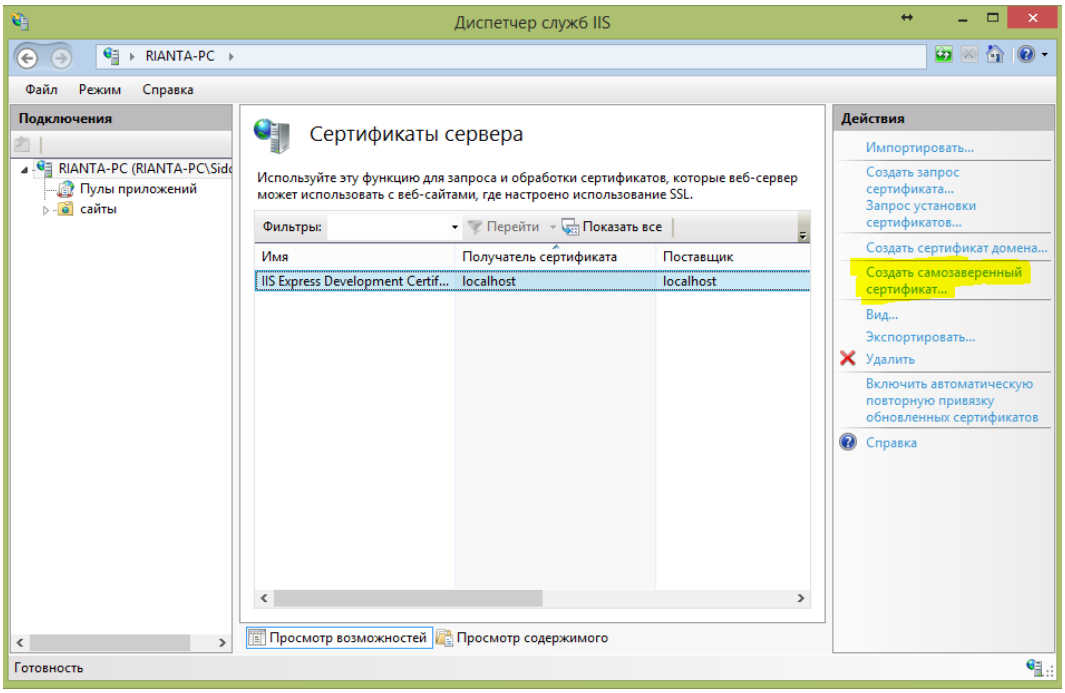


Рис. 4.27. Создание самозаверенного сертификата

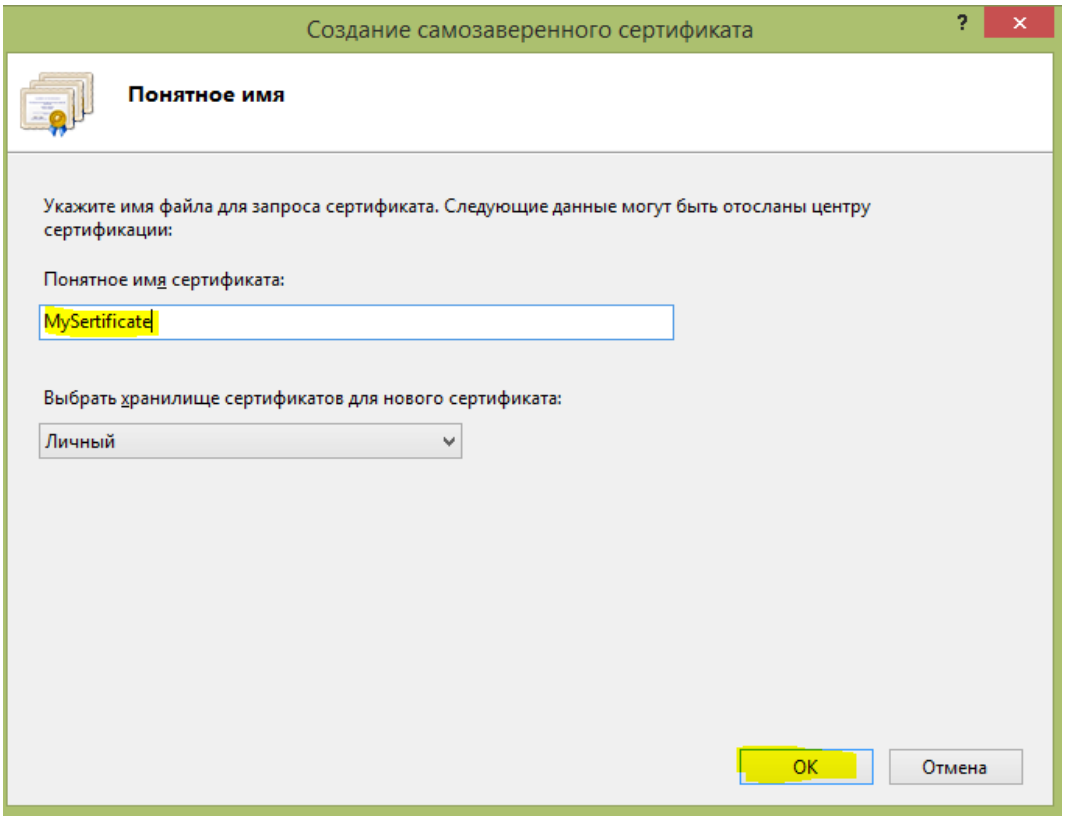


Рис. 4.28. Ввод имени сертификата

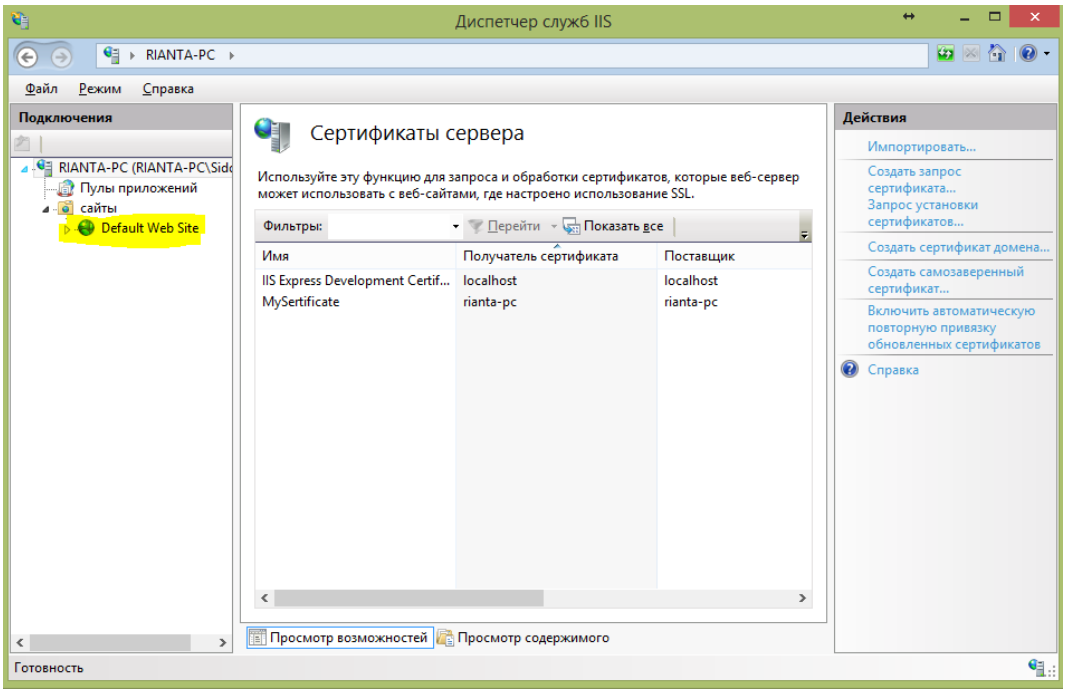


Рис. 4.29. Отображение сертификата

Привязываем сертификат к нужному IP

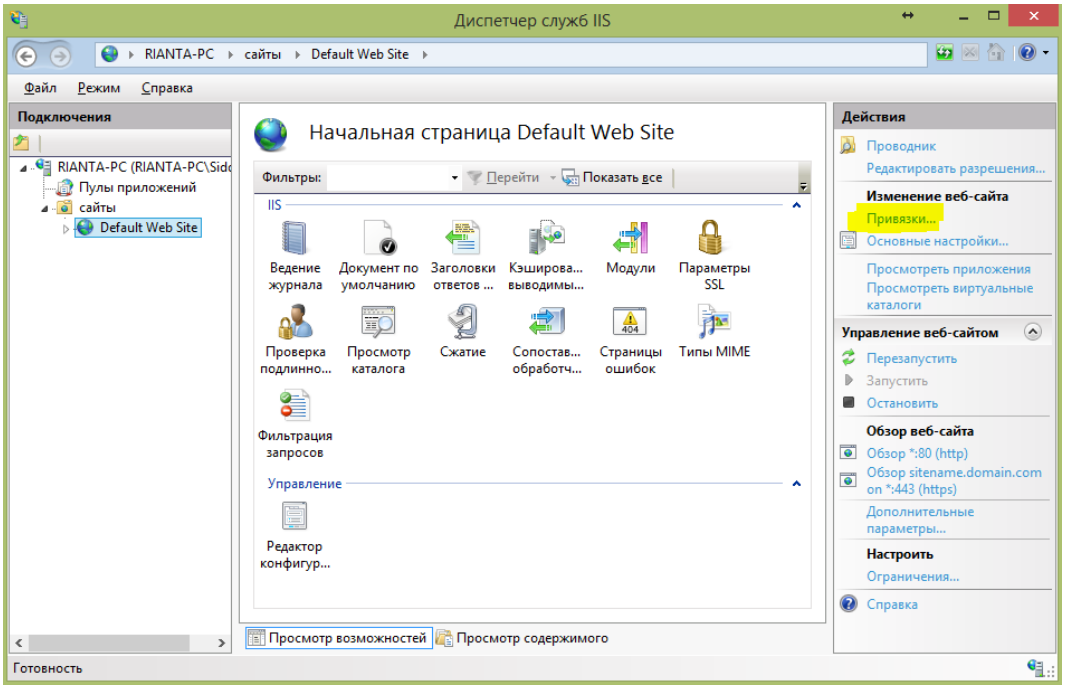


Рис. 4.30. Привязка сертификата к сайту (компьютеру)

Затем с помощью программы THEGREENBOW устанавливаем защищенное SSL – соединение.

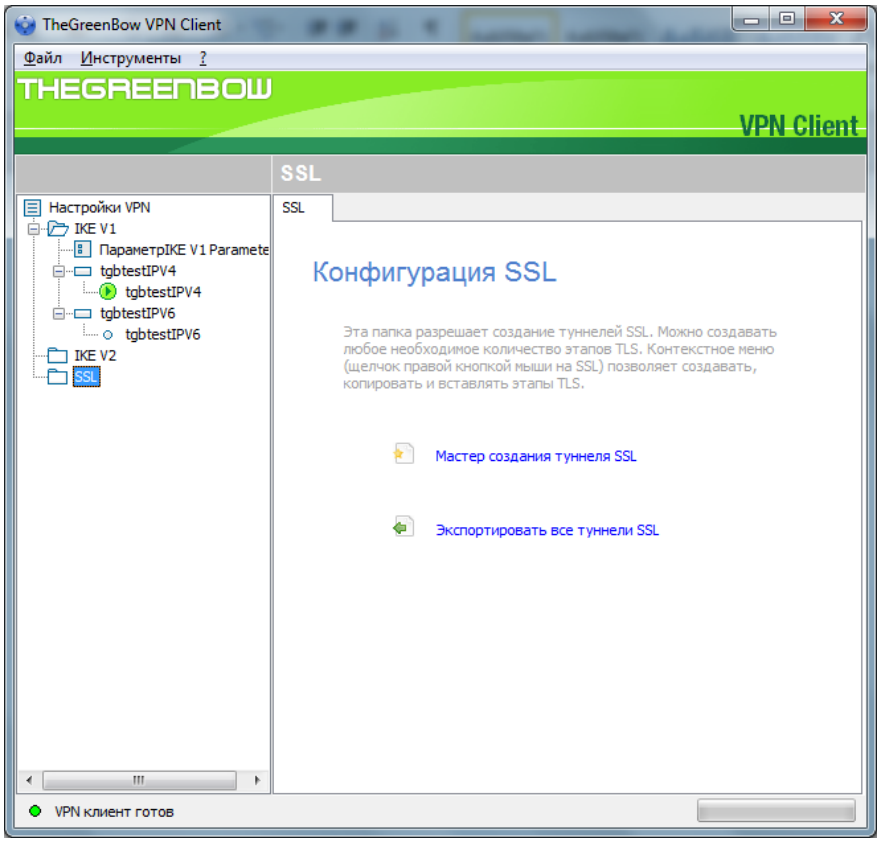


Рис. 4.31. Окно раздела SSL

Указываем путь к сертификату

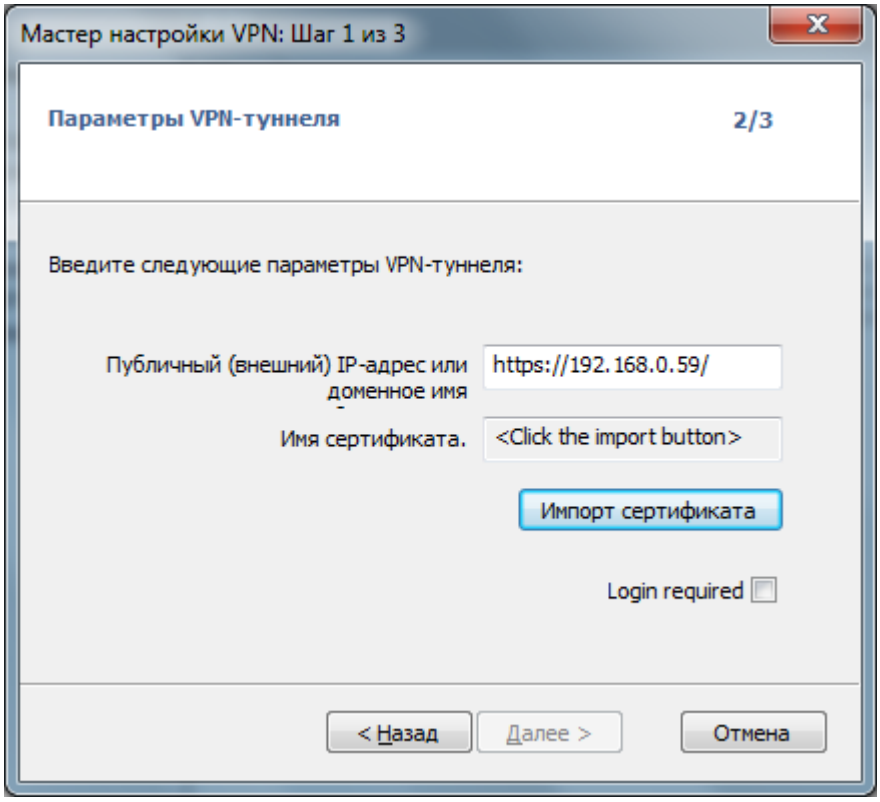


Рис. 4.32. Окно мастера настройки VPN-туннеля

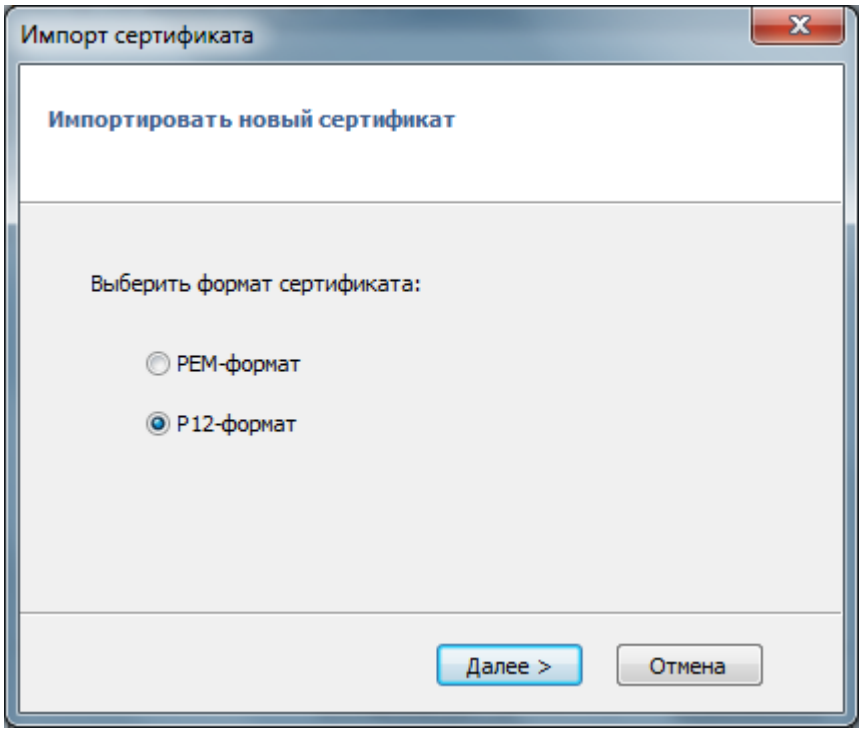


Рис. 4.33. Импорт сертификата

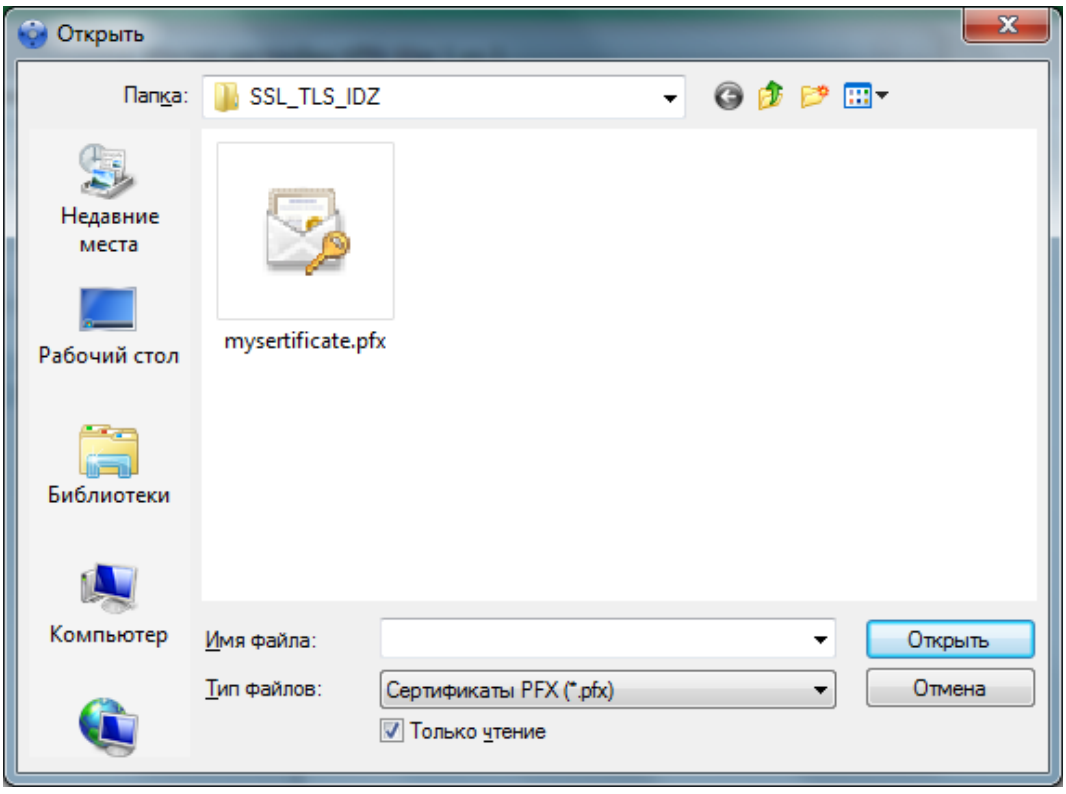


Рис. 4.34. Выбор созданного сертификата

Вводим пароль для доступа к сертификату

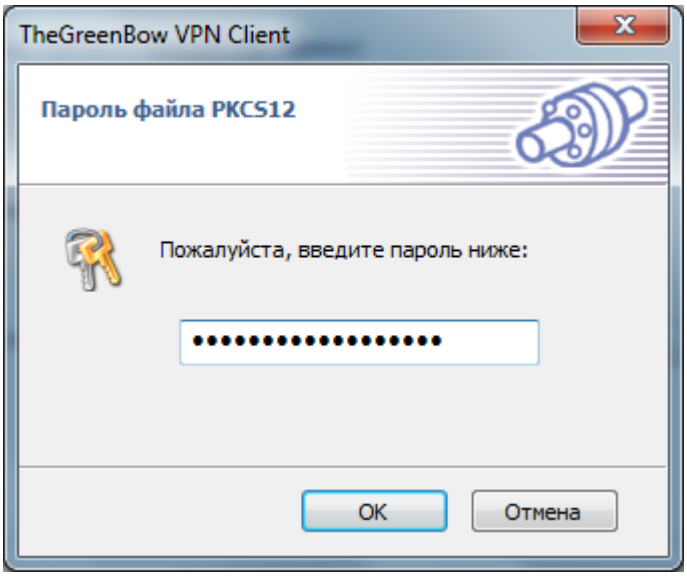


Рис. 4.35. Окно ввода пароля для доступа к сертификату

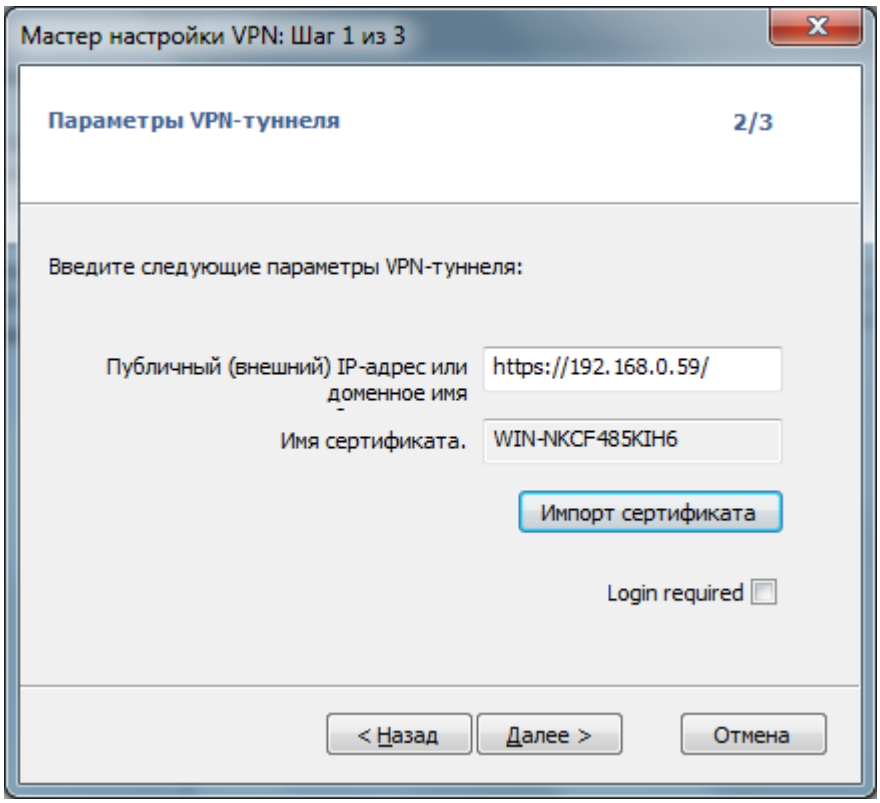


Рис. 4.36. Окно мастера настройки VPN-туннеля

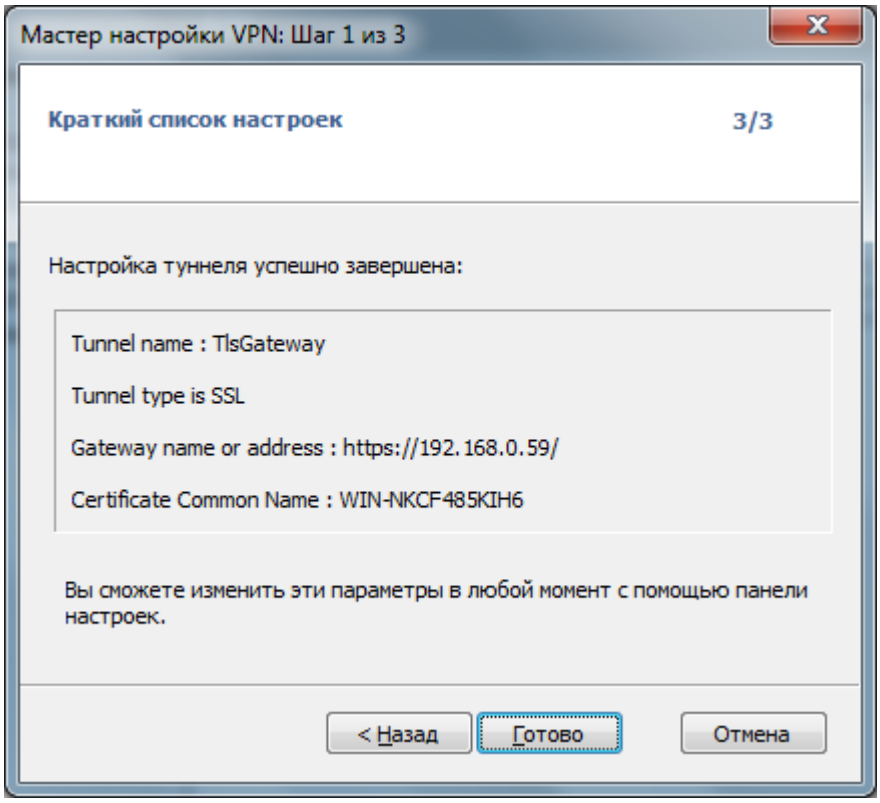


Рис. 4.37. Подтверждение об успешно созданном туннеле

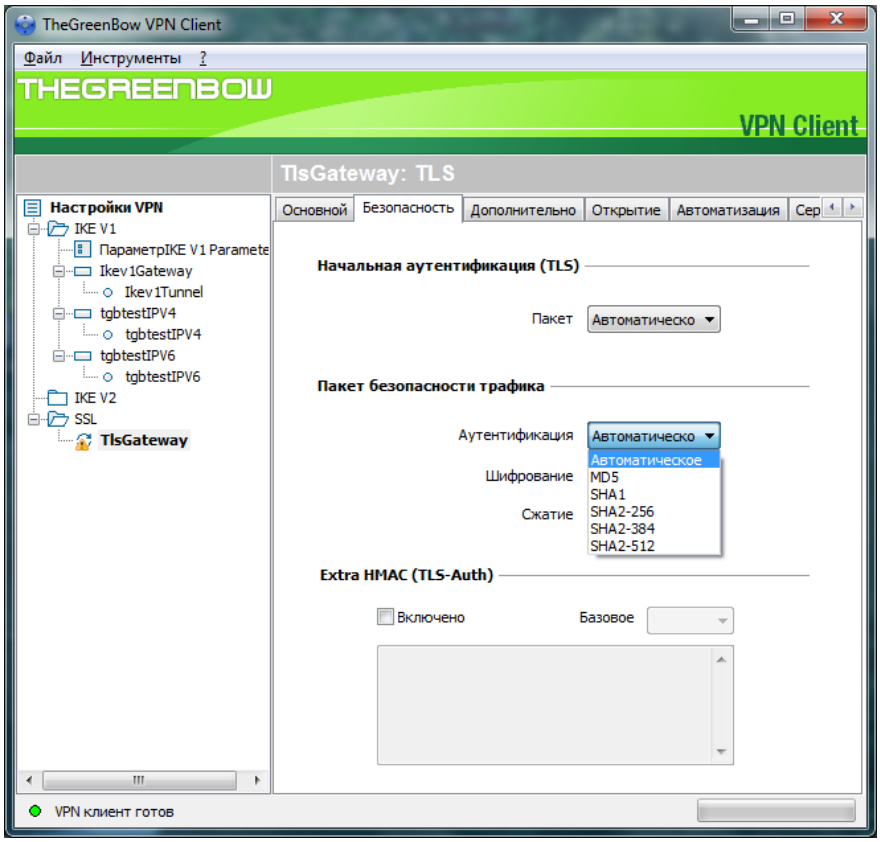


Рис. 4.38. Параметры аутентификации трафика

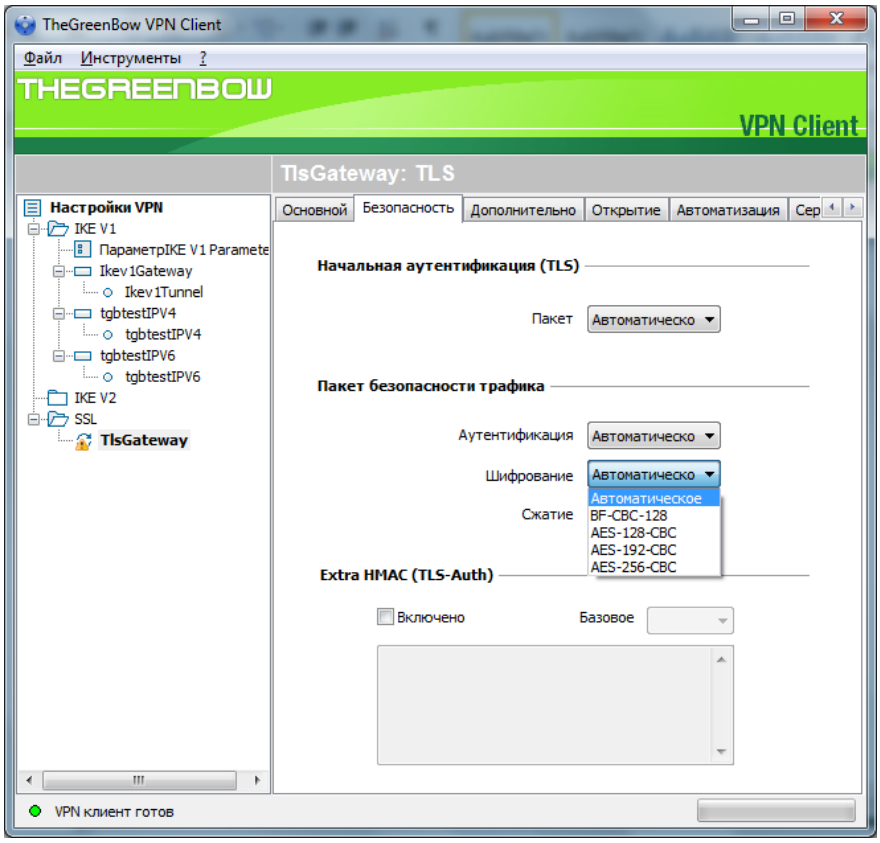


Рис. 4.39. Параметры шифрования

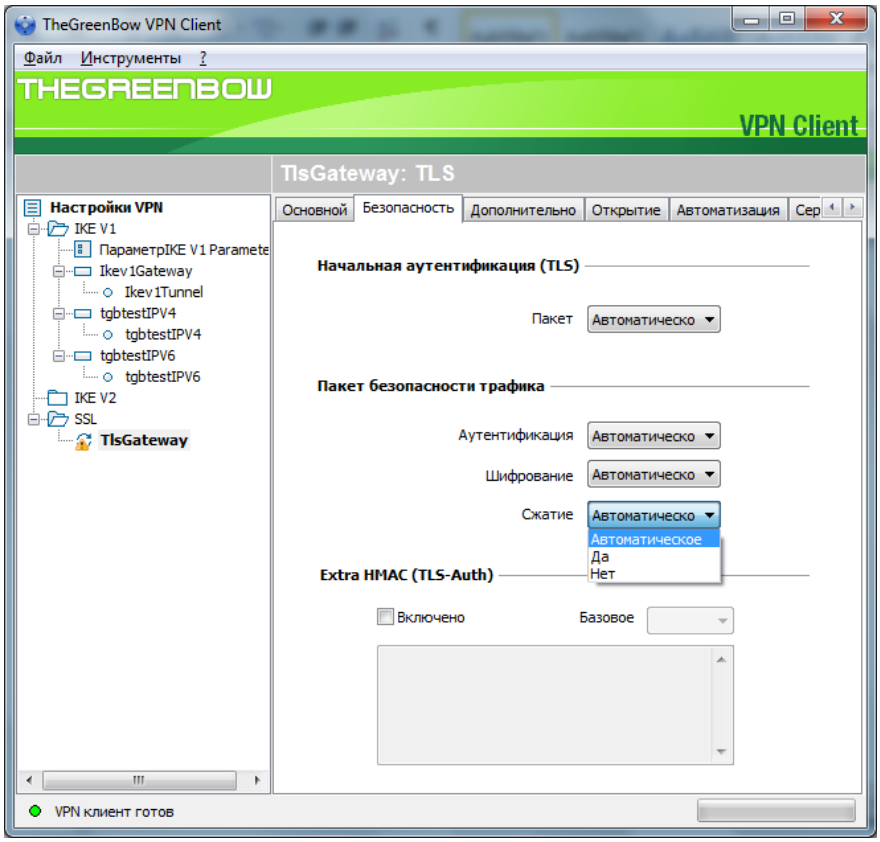


Рис. 4.40. Параметры сжатия

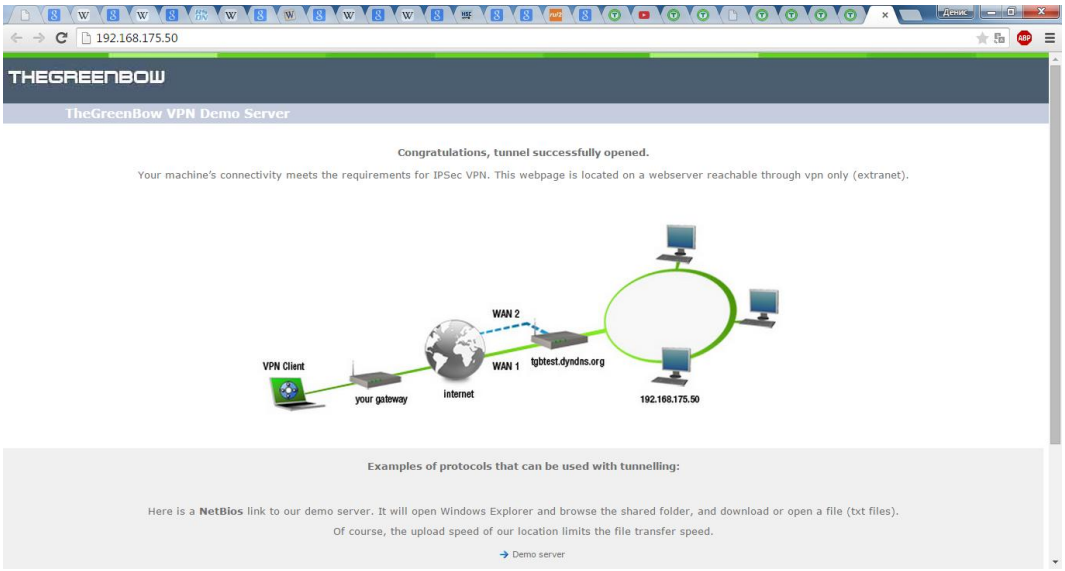


Рис. 4.41. Окно в браузере об успешном соединении

Протокол безопасности транспортного уровня обеспечивает услуги безопасности из конца в конец для приложений, которые пользуются протоколами транспортного уровня, такими как, например, TCP. На сегодняшний день преобладает применение двух протоколов: *протокол "Уровень Безопасных Розеток" (SSL - Secure Sockets Layer)* и *протокол "Безопасность Транспортного уровня"* (TLS - Transport Layer Security).

• SSL или TLS обеспечивают такие услуги, как фрагментация, сжатие, целостность сообщения, конфиденциальность и создание кадра данных, полученных от прикладного уровня. Как правило, SSL (или TLS) может получить прикладные данные от любого протокола прикладного уровня, но работает протокол обычно с *HTTP*.

• Комбинация алгоритмов смены ключей, хэширования и алгоритм шифрования определяют *набор шифров* для каждого сеанса.

• Для того чтобы обмениваться заверенными и конфиденциальными сообщениями, клиенту и серверу необходимо иметь шесть единиц криптографической секретности (четыре ключа и два вектора инициализации).

• В SSL (или TLS) отличают *подключение* и сеанс. В сеансе одна сторона играет роль клиента, а другая - роль сервера; при *подключении* обе стороны играют одинаковые роли, на равном подуровне.

• SSL (или TLS) определяет четыре протокола на двух уровнях: *протокол установления соединения*, *протокол изменения параметров шифрования, аварийный протокол* и *протокол передачи записей*. *Протокол установления соединения* использует

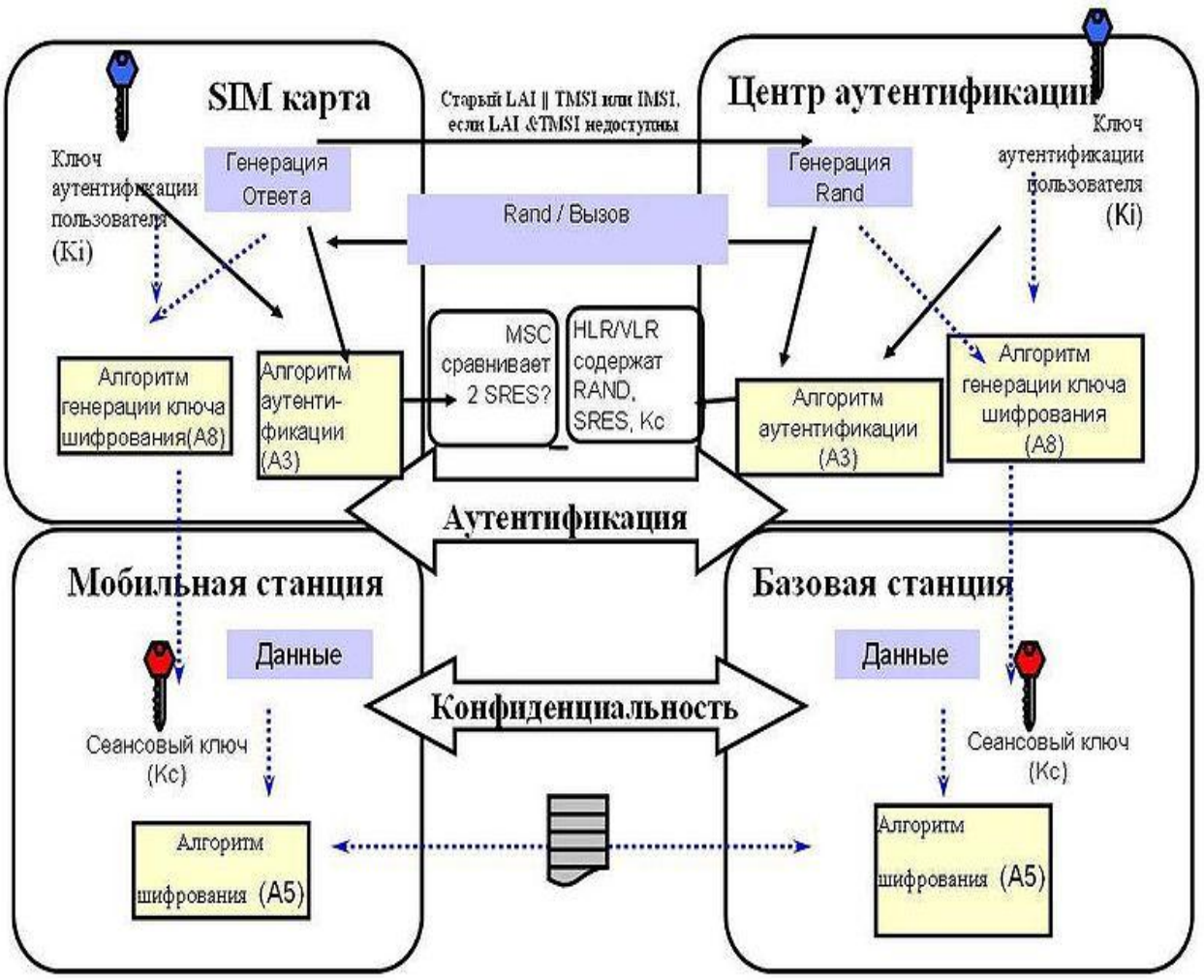
несколько сообщений, чтобы договориться о *наборе шифров*, подтвердить подлинность сервера для клиента и клиента для сервера, если это необходимо, и обмениваться информацией для организации криптографической секретности. *Протокол изменения параметров шифрования* определяет процесс перемещения информации между состоянием ожидания и активным состоянием. *Аварийный протокол* передает извещения об ошибках и ситуациях, отклоняющихся от нормальных. *Протокол передачи записей* доставляет сообщения от верхнего уровня (протокол установления соединения, аварийный протокол, ChangeCipherSpec-протокол) или прикладного уровня.

5. ШИФРОВАНИЕ В СОВРЕМЕННЫХ СИСТЕМАХ СВЯЗИ

Безопасность GSM сетей

Прежде чем приступить к описанию алгоритма шифрования используемого в GSM сетях рассмотрим каким образом происходит аутентификация пользователя и формирования

ключа шифрования. Для этого воспользуемся картинкой.



На данном рисунке схематично представлены следующие шаги:

1. Телефон оператора подключается к сети.
2. Для подтверждения своей подлинности телефон посылает специальный идентификационный код, называемый TMSI(Temporary Mobile Subscriber Identity).
3. Центр Аутентификации(ЦА) генерирует 128-битное случайное число RAND и посылает его на Мобильную Станцию(МС).
4. МС зашифровывает полученное число RAND, используя свой секретный ключ K_i и алгоритм аутентификации A3.
5. МС берет первые 32 бита из последовательности, полученной на предыдущем шаге(назовем их SRES(signed response)) и отправляет их обратно на ЦА.
6. ЦА проделывает ту же операцию и получает 32 битную последовательность XRES(expected response).

7. После чего ЦА сравнивает SRES и XRES. В случае, если оба значения равны, телефон считается аутентифицированным.

8. МС и ЦА вычисляют сессионный ключ шифрования, используя секретный ключ K_i и алгоритм формирования ключа A8 $K_c=A8_{ki}(RAND)$

Говоря об алгоритмах аутентификации A3 и алгоритме формирования ключа A8, следует отметить что на практике большинство сотовых операторов используют для этих целей один алгоритм, называемый COMP128(он имеет множество модификаций COMP128-1, COMP128-2, COMP128-3).

COMP128 представляет собой обыкновенную хэш-функцию, на входе которая принимает 128-битную последовательность и на выходе возвращает 96-битную.

Как всегда в криптографии, попытка сэкономить время разработчикам обернулась полным провалом. Безопасность GSM сетей изначально основывалась на принципе «безопасность за счёт неизвестности». И когда в 1998 году алгоритм был вскрыт группой исследователей состоящих из Marc Briceno, Ian Goldberg и David Wagner обранужилась одна занятная особенность: последние 10 бит секретного ключа K_i всегда равнялись нулю. Используя это любопытное свойство, а так же уязвимость COMP128 к «атаке дней рождений» Marc Briceno, Ian Goldberg и David Wagner смогли извлечь секретный ключ K_i из SIM-карты.

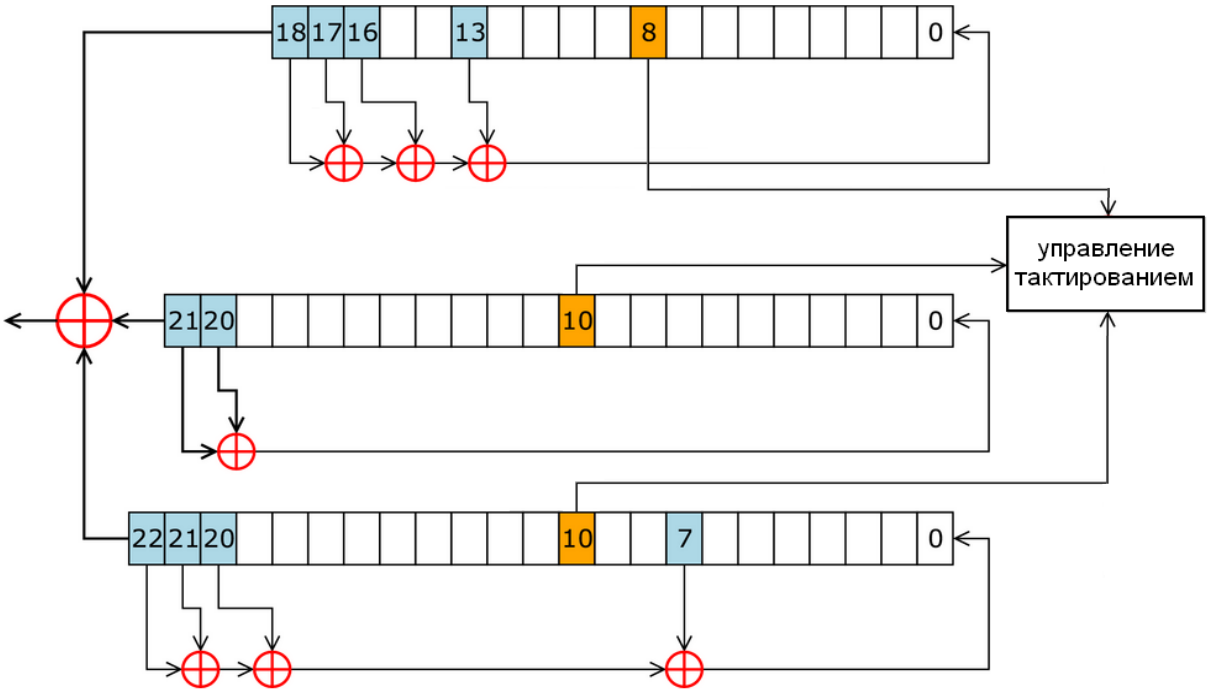
Результатом этого исследования стал повсеместный отказ от алгоритма COMP128 и его замена на более надежные модификации COMP128-2 и COMP128-3, технические детали которых держатся в тайне.

Алгоритм шифрования A5/1

В качестве алгоритма шифрования в GSM используются алгоритмы из семейства A5. На сегодняшний день их всего 3:

- **A5/1** — поточный шифр, наиболее распространенный на сегодня.
- **A5/2**-вариант предыдущего алгоритма «для бедных». Очень похож на своего «старшего брата», но изначально задумывался, как сильно ослабленная версия A5/1. В настоящее время не используется
- **A5/3**-блочный шифр. Разработан в 2002 году с целью заменить устаревший A5/1. Однако в настоящее время используется только в 3GPP сетях. У алгоритма найден ряд уязвимостей, но о практических атаках речи пока не идет.

Рассмотрим подробнее алгоритм A5/1.



Внутреннее состояние шифра A5/1 состоит из трех линейных регистров сдвига с обратной связью R1, R2, R3, длиной 19, 22 и 23 бита соответственно(всего 64 бита).

Сдвиг в регистрах R1, R2, R3 происходит только при выполнении определенного условия. Каждый регистр содержит " бит управления тактированием". В R1 это 8-й бит, а в R2 и R3 — 10-й. На каждом шаге сдвигаются только те регистры у которых значение бита синхронизации равно большинству значений синхронизирующих битов всех трех регистров.

На сегодняшний день известно большое количество успешных атак на GSM шифрование и все они относятся к атакам типа known-plaintext, т.е. для восстановления ключа атакующему помимо зашифрованных фреймов необходимо знать так же незашифрованные данные, которые соответствуют этим фреймам. На первый взгляд такое требование может показаться фантастическим, однако из-за специфики стандарта GSM, в котором помимо голосового трафика передаются различные системные сообщения, такого рода атаки из разряда теоретических переходят в разряд практических.

Системные сообщения GSM содержат повторяющиеся данные и могут использоваться злоумышленником. В частности метод, предложенный Karsten Nohl в 2010 году основан как раз таки на поиске такого рода данных в шифротексте и простом переборе различных вариантов ключей, хранящихся в радужных таблицах, до тех пор пока не будет найден ключ, порождающий нужный шифротекст для известного заранее системного сообщения.

Криптографическая защита беспроводных сетей стандартов LTE

Стандарт сетей LTE – стандарт беспроводной высокоскоростной передачи данных для мобильных телефонов и других терминалов, работающих с данными. Он основан на GSM/EDGE и UMTS/HSPA сетевых технологиях, увеличивая пропускную способность и скорость за счёт использования другого радиоинтерфейса вместе с улучшением ядра сети.

На рисунке 5.1 представлена структура сети стандарта LTE.

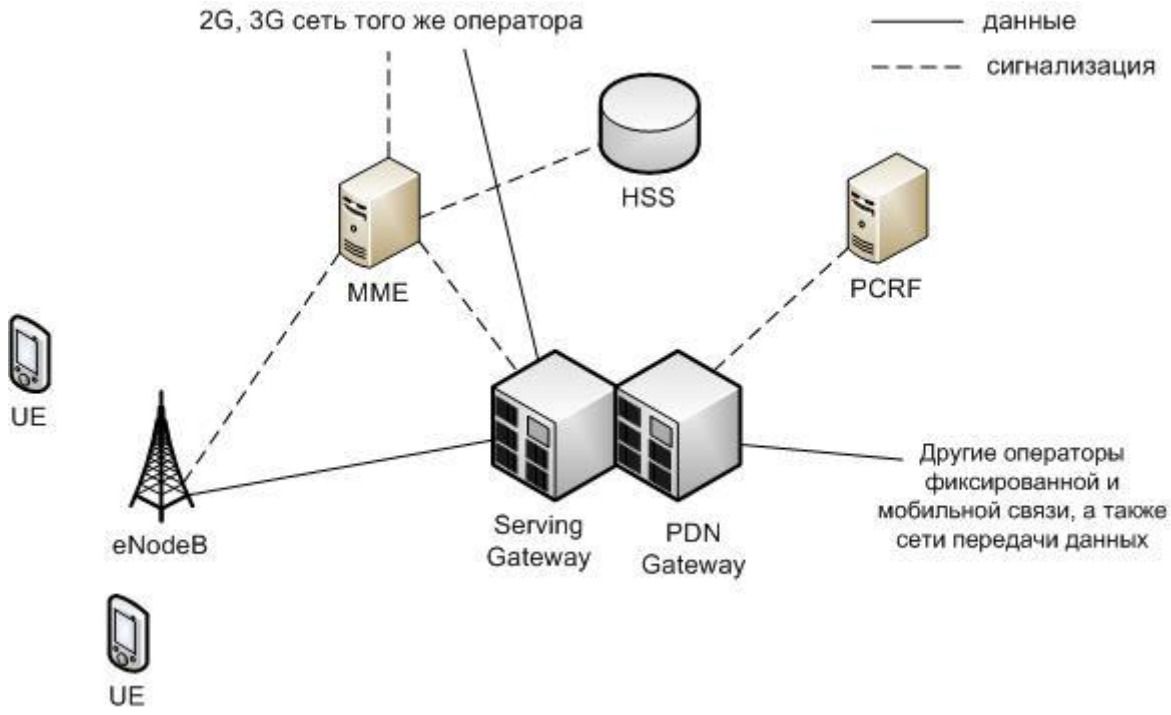


Рис. 5.1. Структура сети стандарта LTE

Из этой схемы видно, что структура сети сильно отличается от сетей стандартов 2G и 3G. Существенные изменения претерпела и подсистема базовых станций, и подсистема коммутации. Изменена технология передачи данных между оборудованием пользователя и базовой станцией. Также подверглись изменению и протоколы передачи данных между сетевыми элементами. Вся информация (голос, данные) передается в виде пакетов. Таким образом, уже нет разделения на части обрабатывающие либо только голосовую информацию, либо только пакетные данные.

Можно выделить следующие основные элементы сети стандарта LTE:

- **Serving SAE Gateway** или просто **Serving Gateway** *(SGW)* – обслуживающий шлюз сети LTE. Предназначен для обработки и маршрутизации пакетных данных поступающих из/в подсистему базовых станций. SGW имеет прямое соединение с сетями второго и третьего поколений того же оператора, что упрощает передачу соединения в /из них по причинам ухудшения зоны покрытия, перегрузок и т.п. В SGW нет функции

коммутации каналов для голосовых соединений, т.к. в LTE вся информация, включая голос коммутируется и передается с помощью пакетов.

- **Public Data Network SAE Gateway** или просто **PDN Gateway** *(PGW)* – шлюз к сетям передачи данных других операторов для сети LTE. Основная задача PGW заключается в маршрутизации трафика сети LTE к другим сетям передачи данных, таких как Интернет, а также сетям GSM, UMTS.
- **Mobility Management Entity** *(MME)* – узел управления мобильностью сети сотовой связи стандарта LTE. Предназначен для обработки сигнализации, преимущественно связанной с управлением мобильностью абонентов в сети.
- **Home Subscriber Server** *(HSS)* – сервер абонентских данных сети сотовой связи стандарта LTE. Представляет собой большую базу данных и предназначен для хранения данных об абонентах. Кроме того, HSS генерирует данные, необходимые для осуществления процедур шифрования, аутентификации и т.п. Сеть LTE может включать один или несколько HSS. Количество HSS зависит от географической структуры сети и числа абонентов.
- **Policy and Charging Rules Function** *(PCRF)* – элемент сети сотовой связи стандарта LTE, отвечающий за управление начислением платы за оказанные услуги связи, а также за качество соединений в соответствии с заданными конкретному абоненту характеристиками.

Для того чтобы данные могли быть транспортированы через интерфейс радио LTE, используются различные «каналы». Они используются для того, чтобы выделять различные типы данных и позволить им транспортироваться через сеть доступа более эффективно. Использование нескольких каналов обеспечивает интерфейс более высокого уровня в рамках протокола LTE и включают более чёткую и определенную сегрегацию данных.

Есть три категории, в которые могут быть сгруппированы различные каналы передачи данных:

Логические каналы – предоставляет услуги среднего уровня управления доступом MAC *(Medium Access Control)* в пределах структуры протокола LTE. Логические каналы по типу передаваемой информации делятся на логические каналы управления и логические каналы трафика. Логические каналы управления используются для передачи различных сигнальных и информационных сообщений. По логическим каналам трафика передают пользовательские данные.

Транспортные каналы — транспортные каналы физического уровня предлагают передачу информации в MAC и выше. Информацию логических каналов после обработки на RLC/MAC уровнях размещают в транспортных каналах для дальнейшей передачи по

радиоинтерфейсу в физических каналах. Транспортный канал определяет как и с какими характеристиками происходит передача информации по радиоинтерфейсу. Информационные сообщения на транспортном уровне разбивают на транспортные блоки. В каждом временном интервале передачи (Transmission Time Interval, TTI) по радиоинтерфейсу передают хотя бы один транспортный блок. При использовании технологии MIMO возможна передача до четырех блоков в одном TTI.

Физические каналы – это каналы передачи, которые переносят пользовательские данные и управляющие сообщения. Они изменяются между восходящим и нисходящим потоками, поскольку каждый из них имеет различные требования и действует по-своему.

Существующие методы и стандарты защиты беспроводных сетей LTE

Безопасность в сетях LTE заключается в нескольких видах:

- Защита абонентов.
- Защита передаваемых сообщений.
- Шифрование сообщений.
- Аутентификация и абонента, и сети.

Защита абонента заключается в том, что в процессе обслуживания его скрывают временными идентификаторами.

Для закрытия данных в сетях LTE используется потоковое шифрование методом наложения на открытую информацию псевдослучайной последовательности (ПСП) с помощью оператора XOR (исключающее или). В этих сетях для обеспечения безопасности внутри сети применяется принцип туннелирования соединений. Шифрации можно подвергать пакеты S1 и X2 при помощи IPsec ESP, а также подвергаются шифрации сигнальные сообщения этих интерфейсов.

В момент подключения или активизации абонентского оборудования (UE) в сети, сеть запускает процедуру аутентификации и соглашения о ключах AKA (Authentication and Key Agreement). Целью этой процедуры является взаимная аутентификация абонента и сети и выработка промежуточного ключа K_{ASME}. Работа механизма AKA занимает доли секунды, которые необходимы для выработки ключа в приложении USIM и для установления соединения с Центром регистрации (HSS). Вследствие этого, для достижения скорости передачи данных сетей LTE необходимо добавить функцию обновления ключевой информации без инициализации механизма AKA. Для решения этой проблемы в сетях LTE предлагается использовать иерархическую ключевую инфраструктуру. Здесь также, как и в сетях 3G, приложение USIM и Центр аутентификации (AuC) осуществляет предварительное распределение ключей. Когда механизм AKA инициализируется для осуществления двусторонней аутентификации пользователя и сети, генерируются ключ шифрования CK и

ключ общей защиты, которые затем передаются из ПО USIM в Мобильное оборудование (ME) и из Центра аутентификации в Центр регистрации (HSS). ME и HSS, используя ключевую пару (CK;IK) и ID используемой сети, вырабатывает ключ K_{ASME}. Установив зависимость ключа от ID сети, Центр регистрации гарантирует возможность использования ключа только в рамках этой сети. Далее K_{ASME} передается из Центра регистрации в устройство мобильного управления (MME) текущей сети, где он используется в качестве мастер-ключа. На основании K_{ASME} вырабатывается ключ $K_{nas\text{-}enc}$, который необходим для шифрования данных протокола NAS между мобильным устройством (UE) и MME, и $K_{nas\text{-}int}$, необходимый для защиты целостности. Когда UE подключается к сети, MME генерирует ключ KeNB и передает его базовым станциям. В свою очередь, из ключа KeNB вырабатывается ключ Kup-enc, используемый для шифрования пользовательских данных протокола U-Plane, ключ Krrc-enc для протокола RRC (Radio Resource Control - протокол взаимодействия между Мобильными устройствами и базовыми станциями) и ключ Krrc-int, предназначенный для защиты целостности.

Алгоритм аутентификации и генерации ключа представлен на рис 10.2

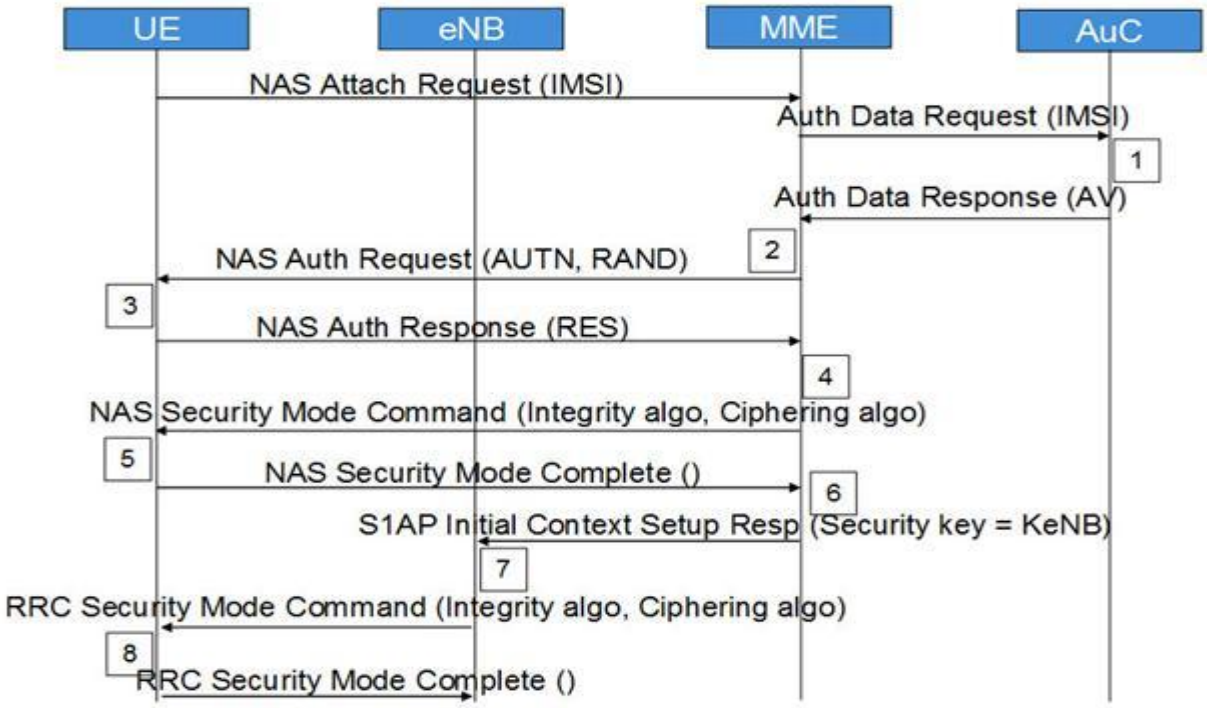


Рис. 5.2. Алгоритм аутентификации и генерации ключа

Здесь:

Шаг 1. Запрос о подключении к сети от мобильной станции (UE). MME запрашивает аутентификационные данные, относящиеся к конкретному IMSI, отправляя Authentication

Data Request. AuC/HSS выбирает PSK, относящийся к конкретному IMSI и вычисляет аутентификационные данные по PSK. AuC/HSS отправляет обратно AV с Authentication Data Response.

Шаг 2. MME получает IK, CK, XRES, RAND и AUTH из AV. MME отправляет AUTH и RAND при помощи Authentication Request к UE.

Шаг 3. UE аутентифицирует NW, проверяя полученный AUTH. После чего вычисляет IK, CK, RES, XMAC из своего ключа защиты, AMF, (OP), AUTH и RAND. Она отправляет RES с Authentication response.

Шаг 4. После получения RES, MME сравнивает его с XRES и если они совпадают, то аутентификация прошла успешно, в противном случае, MME отправляет сбой аутентификации (Authentication failure) к UE. MME сбрасывает счетчик DL NAS. Рассчитывает KASME, KeNB, Knas-int, Knas-enc. Отправляет NAS команду режима безопасности (алгоритм целостности, алгоритм шифрования, NAS набор ключей ID, функцию безопасности UE) с целостностью охраняемых, но не зашифрованных, используя Knas-inc.

Шаг 5. После получения NAS команды режима безопасности, UE вычисляет KASME, KeNB, Knas-int, Knas-enc. UE отправляет NAS режима безопасности выполнен с целостностью, защищенных и зашифрованных.

Шаг 6. После получения NAS команды режима безопасности от UE, MME отправляет KeNB в eNB с S1AP первоначальная установка начального контекста (ключ защиты).

Шаг 7. После получения KeNB, eNB вычисляет Krrc-int, Krrc-enc, Kup-enc. Затем оно отправляет RRC ключ защиты команду с AS целостностью алгоритма и AS шифрующий алгоритм.

Шаг 8. После получения RRC команды ключа защиты UE вычисляет Krrc-int, Krrc-enc, Kup-enc. UE отправляет RRC выполненный ключ шифрования на eNB.

После всех описанных действий, все NAS и AS сообщения будут надежно защищены и зашифрованы, в отличие от пользовательских данных, которые будут только шифроваться.

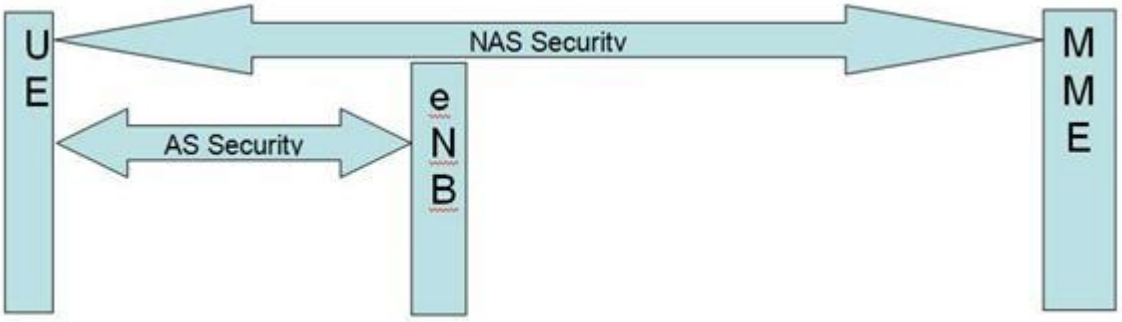


Рис. 5.3. Слои безопасности

Архитектура безопасности LTE определяет механизм безопасности и для уровня NAS и для уровня AS.

Безопасность NAS (слоя без доступа): Выполнена для NAS сообщений и принадлежит области UE и MME.

В этом случае необходима при передаче сообщений NAS между UE и MME – целостность, защищенная и зашифрованная с дополнительным заголовком безопасности NAS.

Безопасность AS (слоя с доступом): Выполнена для RRC и плоскости пользовательских данных, принадлежащих области UE и eNB. Уровень PDCP на сторонах UE и eNB отвечает за шифрование и защиту целостности.

RRC сообщения защищены целостностью и зашифрованы, однако данные U-Plane только зашифрованы.

Для генерации векторов аутентификации используется криптографический алгоритм с помощью однонаправленных функций (f1, f2, f3, f4, f5) когда прямой результат получается путем простых вычислений, а обратный результат не может быть получен обратным путем, то есть не существует эффективного алгоритма получения обратного результата. Для этого алгоритма используется случайное 128 битное случайное число RAND, мастер-ключ K абонента, также 128 бит и порядковый номер процедуры SQN (Sequence Number). Счетчик SQN меняет свое значение при каждой генерации вектора аутентификации. Похожий счетчик SQN работает и в USIM. Такой метод позволяет генерировать каждый раз новый вектор аутентификации, не повторяя предыдущий уже использованный вектор аутентификации.

Помимо этих трех исходных величин: SQN, RAND и K в алгоритме f1 участвует поле управления аутентификацией Authentication Management Field (AMF), а в алгоритмах f2 – f5 исходные параметры – RAND и K, что и продемонстрировано на рис. 2.3, 2.4. На выходах соответствующих функций получают Message Authentication Code (MAC) - 64 бита; XRES – eXpected Response, результат работы алгоритма аутентификации <32 – 128 бит>; ключ шифрации CK, генерируемый с использованием входящих (K,RAND)->f3->CK; ключ целостности IK, сгенерированный с использованием входящего (K,RAND)->f4->IK; и промежуточный ключ Anonymity Key (AK), генерируемый с помощью (K,RAND)->f5->AK - 64 бита.

При обслуживании абонента сетью E-UTRAN ключи CK и IK в открытом виде в ядро сети не передают. В этом случае HSS генерирует K_{ASME} с помощью алгоритма KDF (Key Derivation Function), для которого исходными параметрами являются CK и IK, а также

идентификатор обслуживающей сети и SQNÅAK. Вектор аутентификации содержит RAND, XRES, AUTN и K_{ASME}, на основе которого происходит генерация ключей шифрации и целостности, используемых в соответствующих алгоритмах.

Когда мобильная станция получает из ядра сети три параметра (RAND, AUTN и KSI_{ASME}, где KSI – Key Set Identifier, индикатор установленного ключа, однозначно связанный с K_{ASME} в мобильной станции).

После чего используя RAND и AUTN, USIM на основе алгоритмов безопасности, тождественных хранящимся в HSS, производит вычисление XMAC, RES, CK и IK.

Затем в ответе RES UE передает в MME вычисленное RES, которое должно совпасть с XRES, полученным из HSS. Так сеть аутентифицирует абонента. Вычислив XMAC, UE сравнивает его с MAC, полученным ею в AUTN. При успешной аутентификации абонентом сети (MAC = XMAC) UE сообщает об этом в ответе RES. Если аутентификация сети не удалась (MAC ≠ XMAC), то UE направляет в MME ответ CAUSE, где указывает причину неудачи аутентификации.

При успешном завершении предыдущего этапа MME, eNB и UE производят генерацию ключей, используемых для шифрации и проверки целостности получаемых сообщений. В E-UTRAN имеется иерархия ключей, которая приведена на рис. 2.5.

Векторы аутентификации (рис. 2.3, 2.4):

Ключи IK и CK генерируются и в центре аутентификации, и в USIM;

Ключ AK генерируется только в центре аутентификации;

Ответ XRES генерируется только в центре аутентификации, а RES генерируется в USIM;

Код MAC генерируется только в центре аутентификации, а соответствующий ему параметр XMAC генерируется в USIM;

Маркер AUTH генерируется только в центре аутентификации.

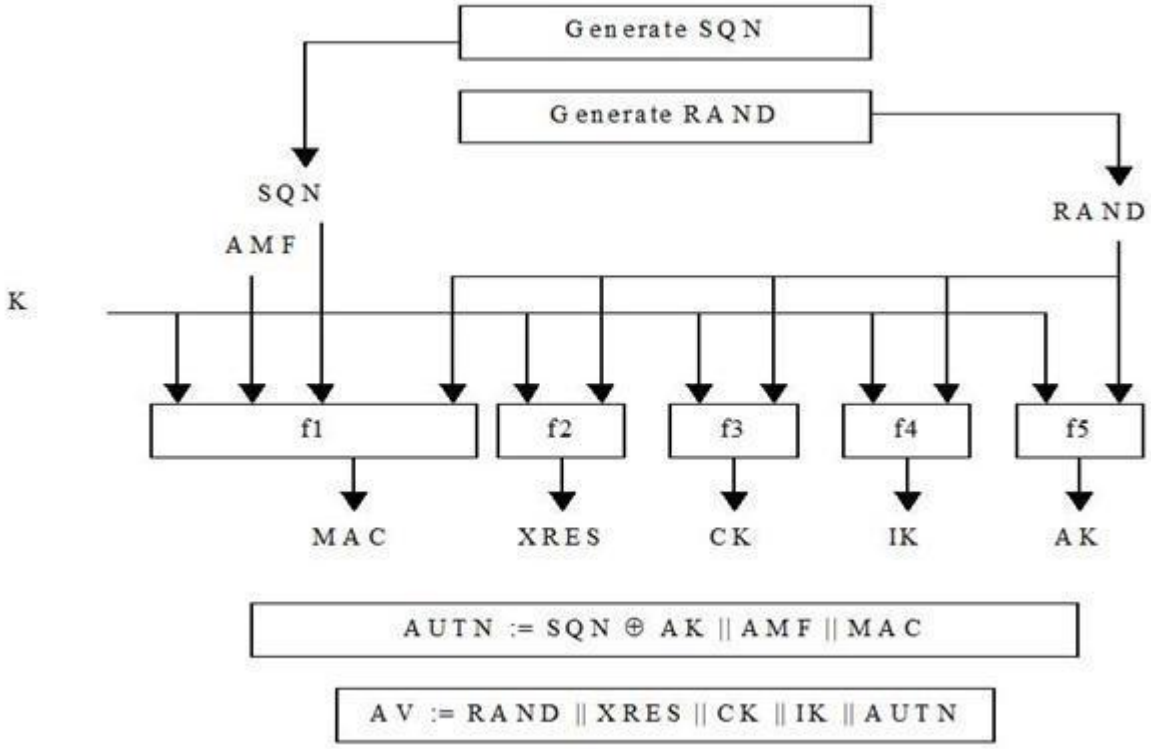


Рис. 5.4. Создание векторов на передающей стороне

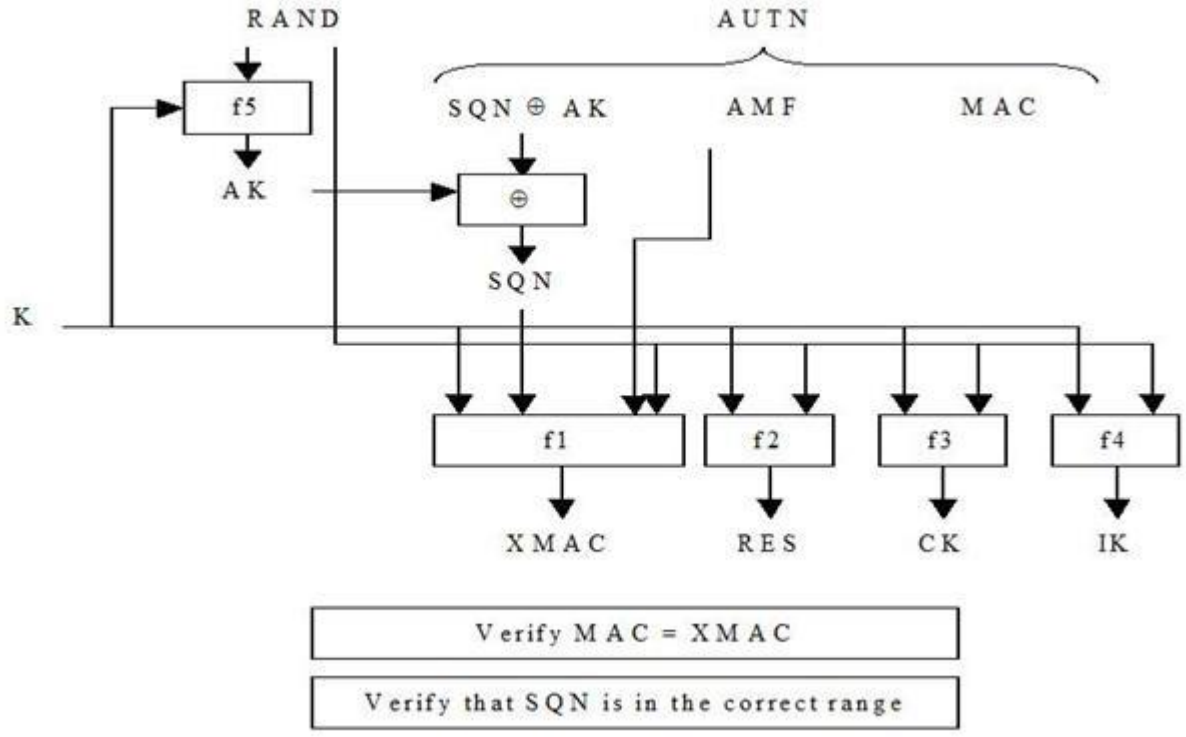


Рис. 5.5. Преобразование векторов на принимаемой стороне

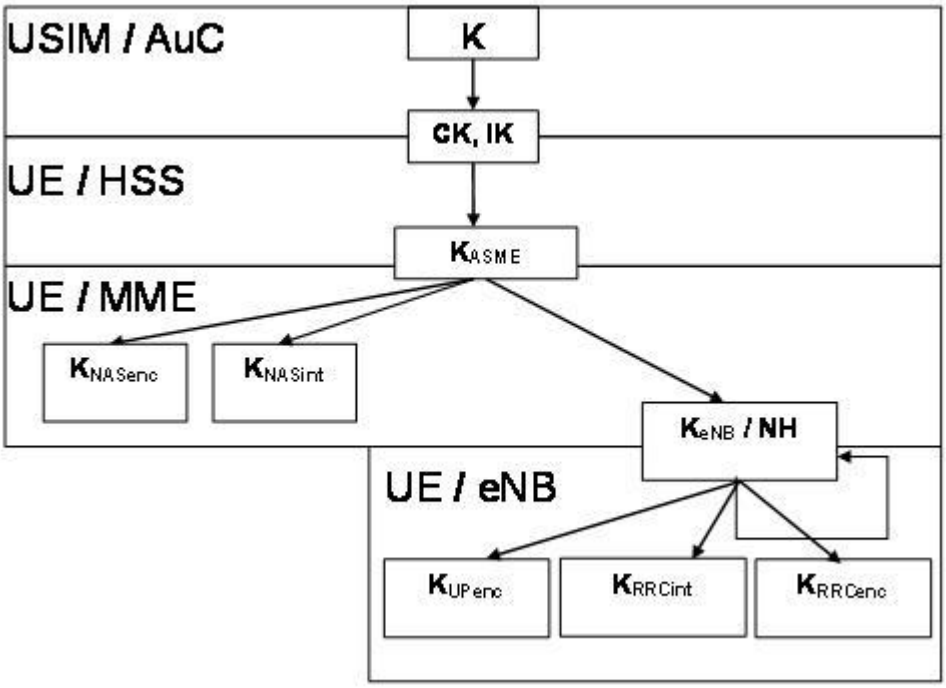


Рис. 5.6. Иерархия ключей в E-UTRAN

Исходным ключом для всей цепочки является K_{ASME} (256 бит). При передаче в радиоканале защиту обеспечивают для сигнального трафика (Control Plane) и для пользовательских пакетов (User Plane). При этом все сообщения сигнализации разделяют на сквозные сигнальные сообщения между UE и MME протоколов MM и SM (NAS – Non Access Stratum) и сигнальные сообщения между eNB протокола RRC (AS – Access Stratum). Для шифрации и защиты целостности можно использовать разные базовые алгоритмы:

- UEA2 (UMTS Encryption Algorithm 2) и UIA2 (UMTS Integrity Algorithm 2);
- разработанные для стандартов 3G,AES (Advanced Encryption Standard).

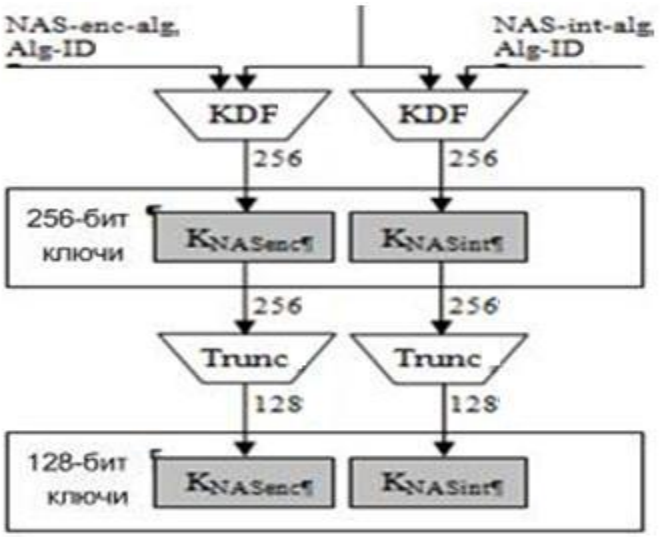


Рис. 5.7. Генерирование ключей шифрации и целостности для NAS сигнализации

Сигнальные сообщения протокола RRC (AS) также шифруют и обеспечивают их целостность. Пакеты трафика только шифруют. Эти операции производят в обслуживающей eNB и UE. Схема получения ключей шифрации и целостности (рис. 7) для AS и UP трафика отличается от предыдущего случая тем, что исходным параметром здесь служит вторичный промежуточный ключ KeNB (256 бит). Этот ключ генерируют, также используя KDF, где входными параметрами являются: KASME, счетчик сигнальных сообщений NAS вверх, прежнее значение KeNB, идентификатор соты и номер частотного канала в направлении вверх. Следовательно, при каждой периодической локализации UE происходит изменение KeNB.

Также KeNB меняется и при хэндовере; при этом в алгоритме генерации нового KeNB можно использовать дополнительный параметр NH (Next Hop), фактически счетчик числа базовых станций, по цепочке обслуживающих абонента. Все реализуемые процедуры безопасности в сети E-UTRAN продемонстрированы на рис. 2.8.

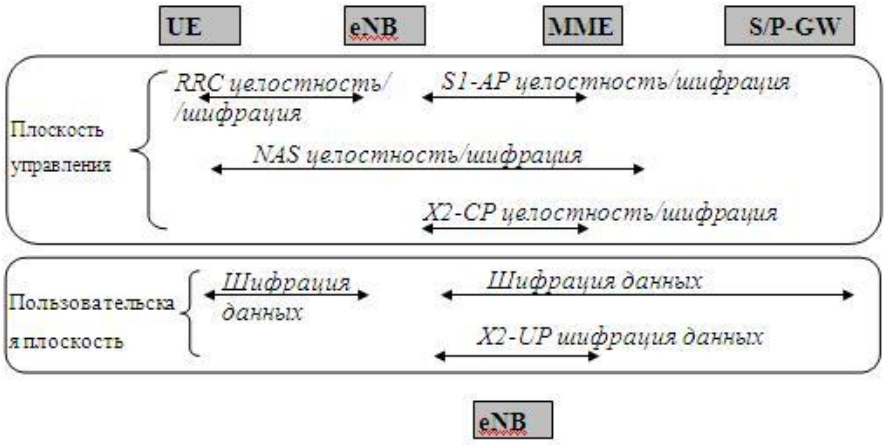


Рис. 5.8. Реализуемые процедуры безопасности в сети E-UTRAN

Алгоритм шифрации и дешифрации сообщений представлен на рис. 2.9.

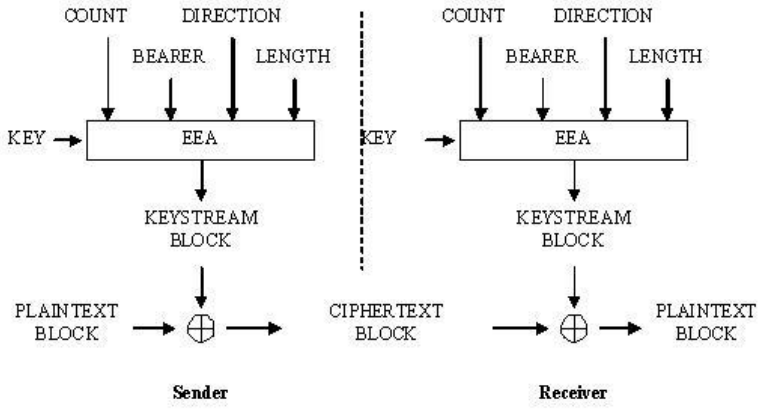


Рис. 5.9. Алгоритм шифрации в E-UTRAN

Исходными параметрами в этом алгоритме являются шифрующий ключ KEY (128 бит), счетчик пакетов (блоков) COUNT (32 бита), идентификатор сквозного канала BEARER (5 бит), указатель направления передачи DIRECTION (1 бит) и длина шифрующего ключа LENGTH. В соответствии с выбранным алгоритмом шифрации EEA (EPS Encryption Algorithm) вырабатывается шифрующее число KEYSTREAM BLOCK, которое при передаче складывают по модулю два с шифруемым исходным текстом блока PLAINTEXT BLOCK. При дешифрации на приемном конце повторно совершают эту же операцию.

Процедура защиты целостности сообщения состоит в генерации “хвоста“ MAC (Message Authentication Code) (32 бита), присоединяемого к передаваемому пакету. Алгоритм генерации MAC и проверки целостности полученного пакета путем сравнения XMAC с MAC (они должны совпасть) отображен на рис. 5.10.

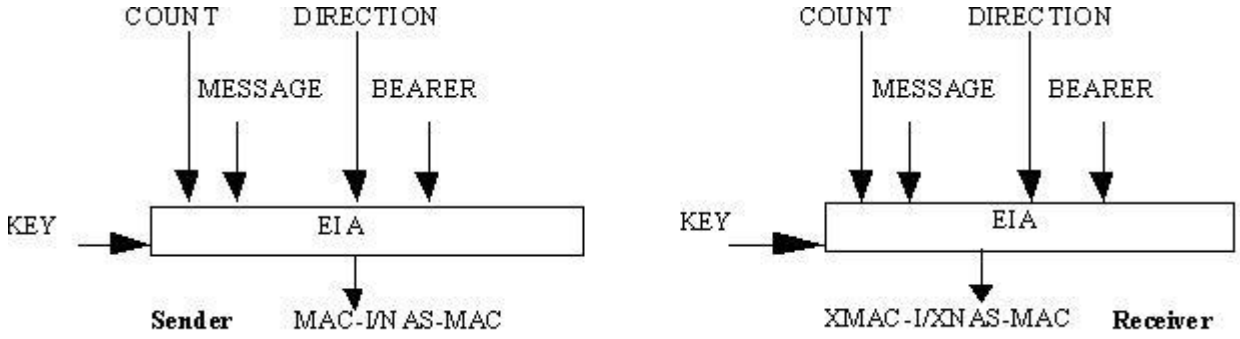


Рис. 5.10. Алгоритм проверки целостности E-UTRAN

В алгоритме EIA (EPS Integrity Algorithm) использован ключ целостности KEY (128 бит), счетчик сообщений COUNT (32 бита), идентификатор сквозного канала BEARER (5 бит), указатель направления передачи DIRECTION (1 бит) и само сообщение MESSAGE.

Моделирование технологии LTE в среде MATLAB с использованием встроенного пакета LTE System Toolbox

LTE System Toolbox™ предоставляет соответствующие стандарту функции и приложения для проектирования, моделирования и проверки коммуникационных систем стандартов LTE и LTE-Advanced. Данный инструмент ускоряет разработку LTE-алгоритмов и физического уровня (PHY), предоставляет эталонный образец для проверки и тестирования на соответ- ствие стандарту, а также позволяет генерировать тестовые сигналы. С помощью LTE System Toolbox можно производить настройку, моделирование, измерения и анализ канала связи. Также можно создавать и повторно использовать сценарии тестов для подтверждения того, что проекты, прототипы и разработки соответствуют стандарту LTE.

Основные особенности:

• Модели, соответствующие стандартам LTE и LTE-Advanced (Release 8, 9, 10 и 11). • Функции обработки на канальном уровне, поддержка от 1 до 10 режимов передачи по нисходящему каналу, образцы проектов, в том числе CoMP.

• Тестовые модели (E-TM), эталонный измерительный канал (RMC) для LTE, LTE-A, а также генератор UMTS-сигналов.

• Интерактивные инструменты для проверки на соответствие стандарту и BER тестов.

• Передача и приём сигналов при помощи радиоустройств для тестирования систем в реаль- ном эфире.

• Выделение системных и контрольных параметров из принятого сигнала, в том числе cell ID, MIB и SIB1. • Оценка канала связи.

Сквозное моделирование LTE

System Toolbox даёт возможность моделировать и имитировать физический уровень стандарта LTE. Моделирование системы на канальном уровне позволяет добиться требуемых значений характеристик системы, в том числе пропускной способности и BER, а также опреде- лить конкретные реализации системы на основе производимых измерений.

LTE System Toolbox также позволяет улучшить планирование системы, облегчая моделирование канального уровня, которое предоставляет некоторые параметры, необходимые для проектирования базовых станций с заданной геометрией и характеристиками распространения сигнала.

Набор поддерживаемых функций для моделирования режимов передачи и приёма, а также канала связи, включает в себя:

• режимы FDD и TDD на несущих частотах; • все полосы передачи LTE-сигналов от 1,4 до 20 МГц, LTE-A до 100 МГц с агрегацией несущей; • различные типы LTE-сигналов, включая нисходящие и восходящие опорные сигналы и сигналы синхронизации;

• физические LTE-каналы, в том числе каналы управления и каналы общего доступа; • готовую процедуру обработки нисходящего канала, в том числе формирование нисходящего общего канала и канала управления, все возможные MIMO-режимы и генерацию OFDM-сигналов;

• готовую процедуру обработки восходящего канала, в том числе формирование восходящих общего канала и канала управления, SU-MIMO и MU-MIMO режимы и генерацию SC-FDMA сигналов;

• адаптацию к каналу связи, включая схемы выбора типов модуляции и кодирования (MCS) в соответствии с оценкой качества канала связи (CQI), индикатора ранга (RI) и индикации матрицы прекодера (PMI);

• возможности и примеры построения LTE-Advanced, в том числе приём и передача с несколькими eNB (CoMP) и с агрегацией несущей;

• модели распространения LTE-сигналов, в том числе модель пешехода (EPA), модель автомобиля (EVA), модель типичный городской застройки (ETU), модель распространения в движении, а также модели MIMO-каналов в скоростном поезде. LTE System Toolbox даёт возможность создавать тесты, измеряющие пропускную способность PDSCH-канала в соответствии с указанными в стандарте LTE (TS 36.101) условиями испытаний. Структуры данных в LTE System Toolbox позволяют удобно отображать все параметры системы. Функции данного инструмента отражают любые возможные комбинации режимов работы передатчиков, моделей каналов и приемников. Используя этот инструмент для тестирования на соответствие стандарту и BLER-тестирования, вы можете измерять характеристики системы и сравнивать их с указанными в спецификации к стандарту

На рисунке 5.12 изображена структурная схема программного комплекса.

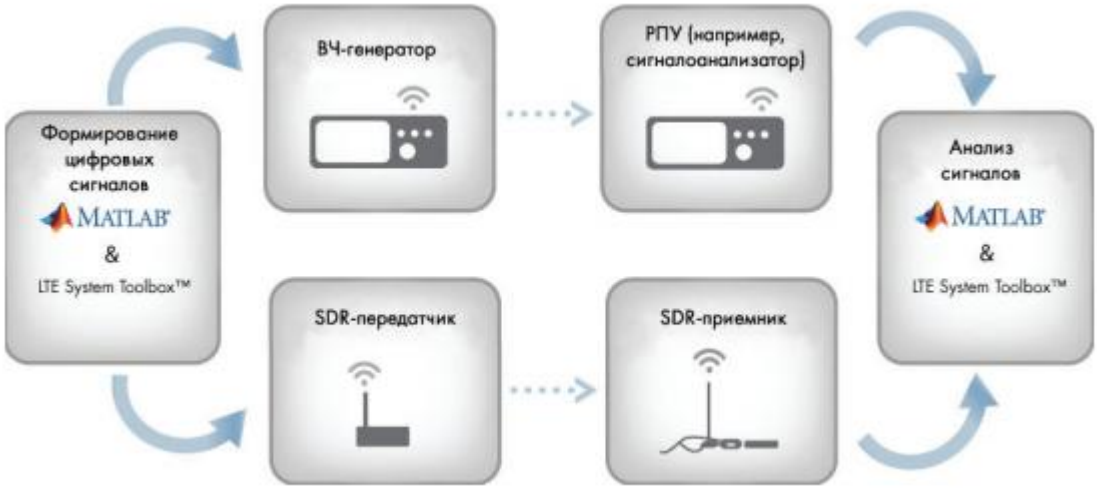


Рис. 5.12. Структурная схема программного комплекса

Для генерации тестового сигнала от базовой станции к абоненту используется генератор LTE-Downlink E-TM Generator.

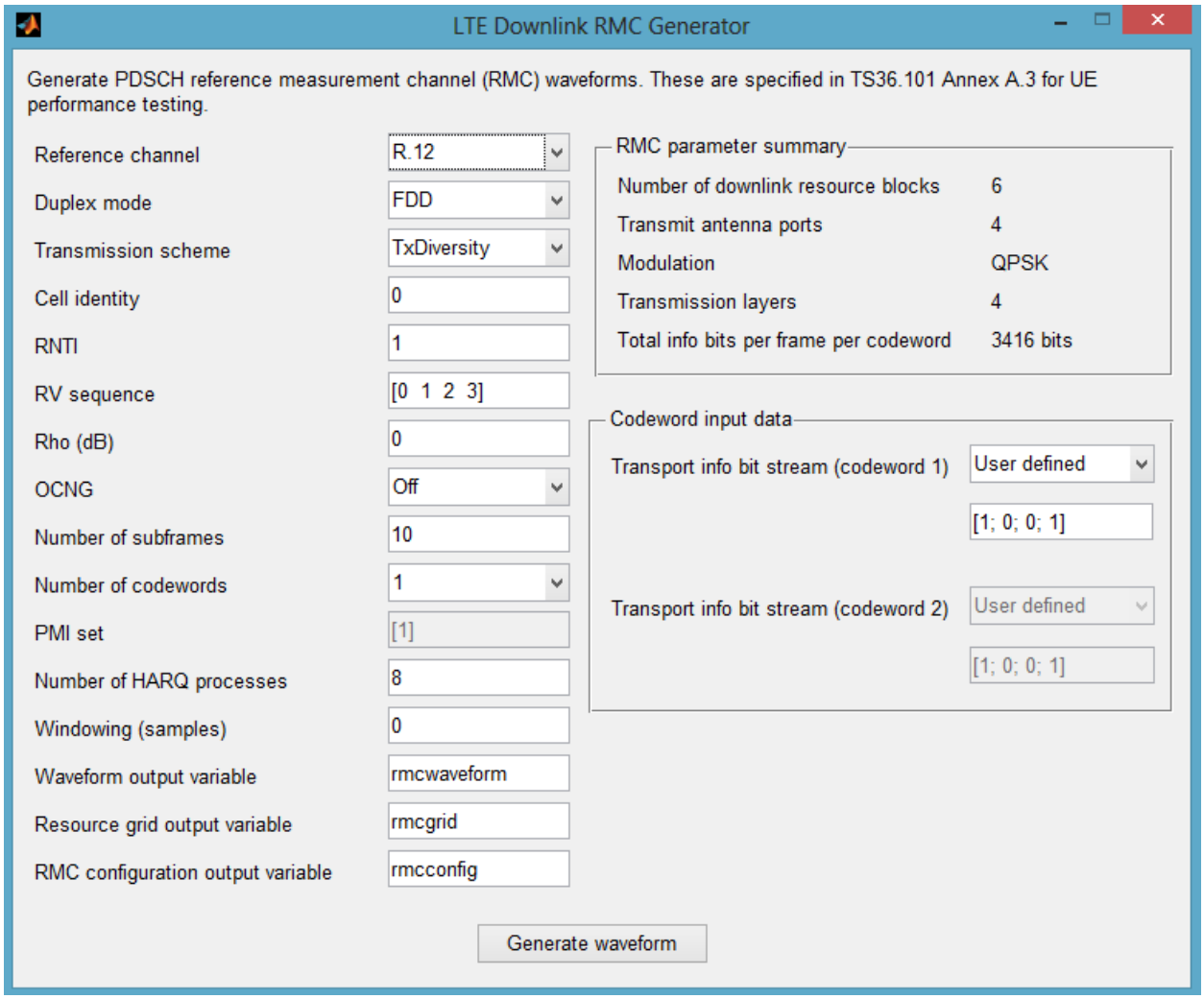


Рис. 5.13. Окно генератора от базовой станции к абоненту

В данном окне задаются параметры генерируемого сигнала на базовой станции, такие как: Количество каналов, вид модуляции, количество обслуживаемых абонентов в секунду, число кодовых слов, вектор инициализации.

На рисунке 5.14 показан вид сгенерированного сигнала, а на рисунке 5.15 трех мерный спектр полученного сигнала.

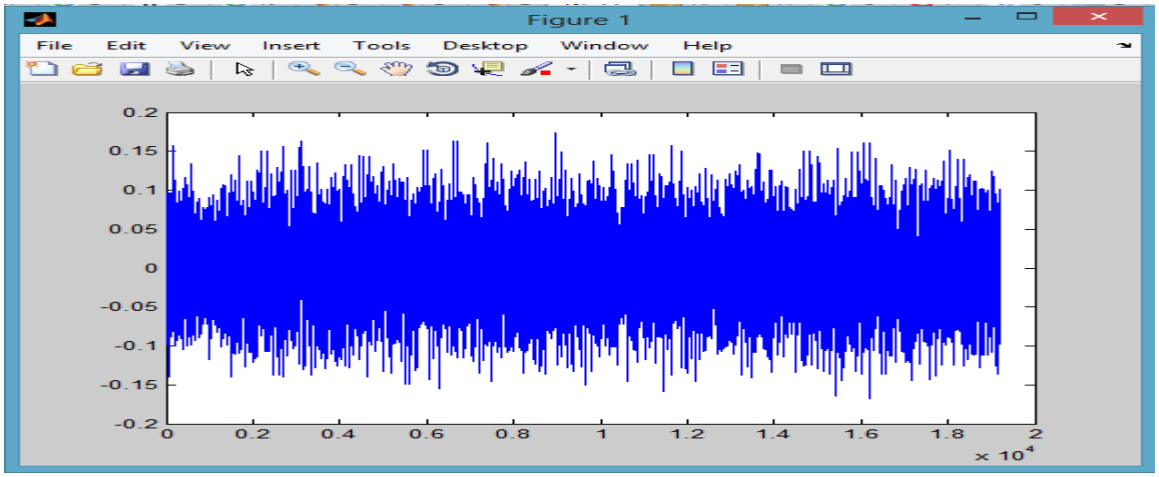


Рисунок 5.14. Сгенерированный сигнал станцией

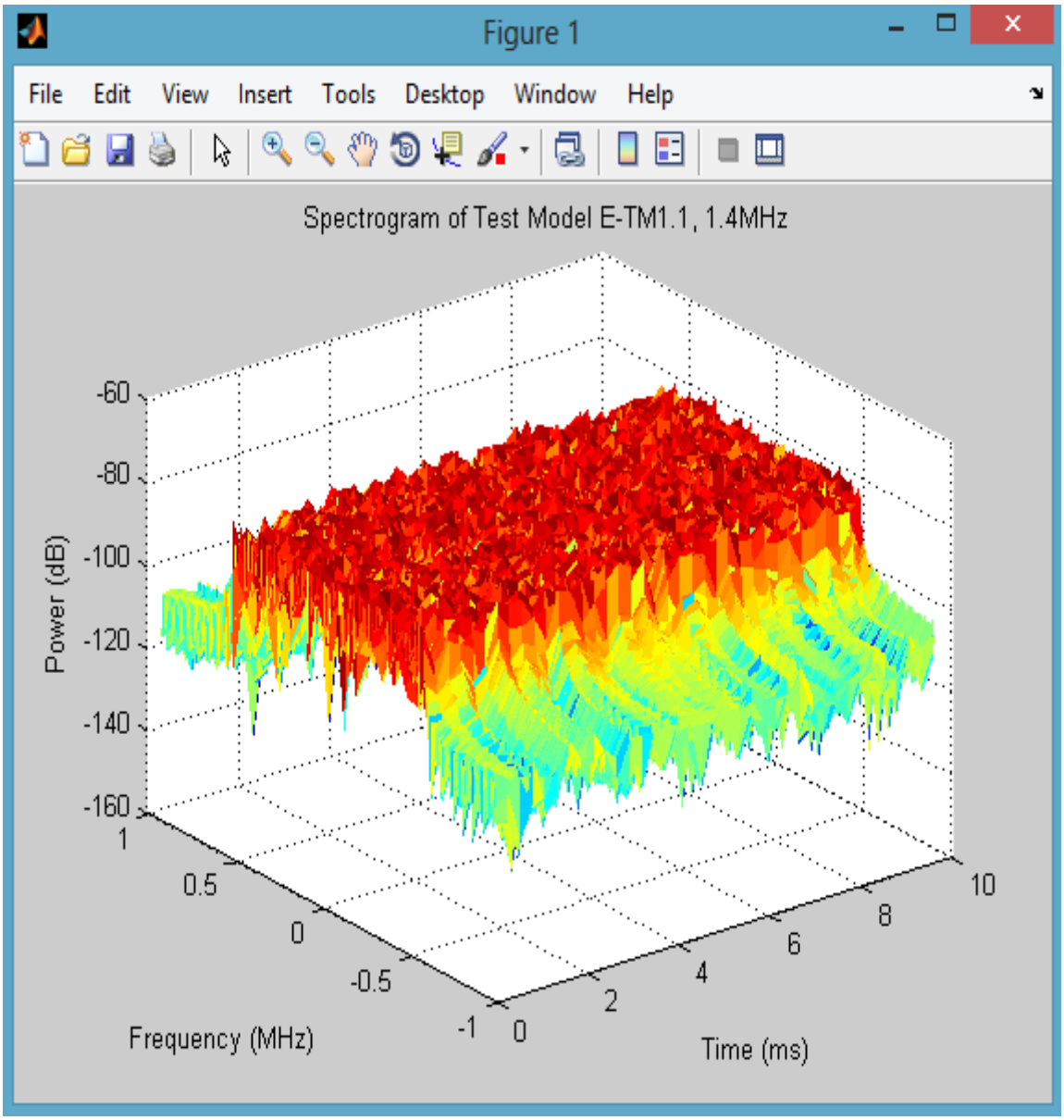


Рис. 5.15. Спектр сгенерированного сигнала станцией в течении 10 секунд

Для генерации тестового сигнала от абонента к базовой станции используется генератор LTE-Uplink RMS Generator.

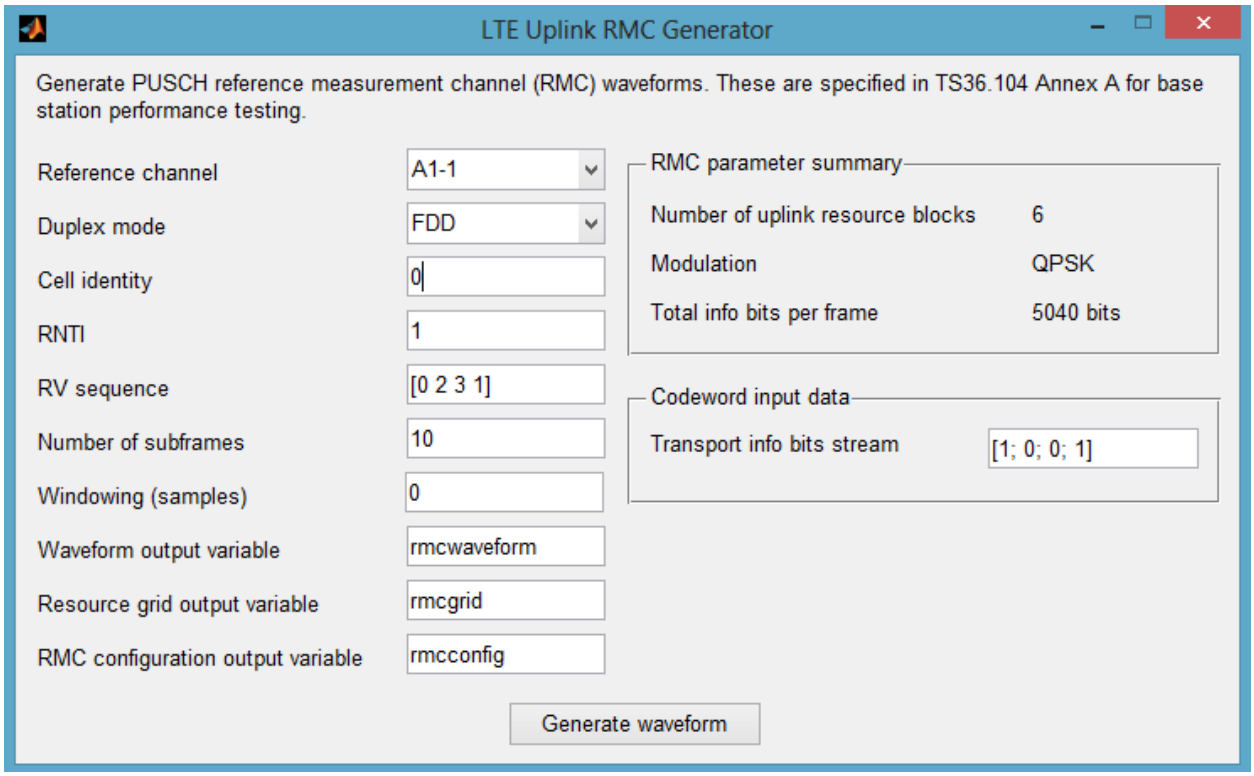


Рис. 5.16. Окно генератора LTE-Uplink RMS Generator

В окне на рисунке 10.16 производятся настройки параметров генерирования от абонента к базовой станции, такие как: Количество каналов для передачи потока пакетов, тип модуляции, вектор инициализации, количество передаваемых пакетов в секунду.

Сигнал, генерируемый генератором LTE-Uplink RMS Generator, представлен на рисунке 5.17.

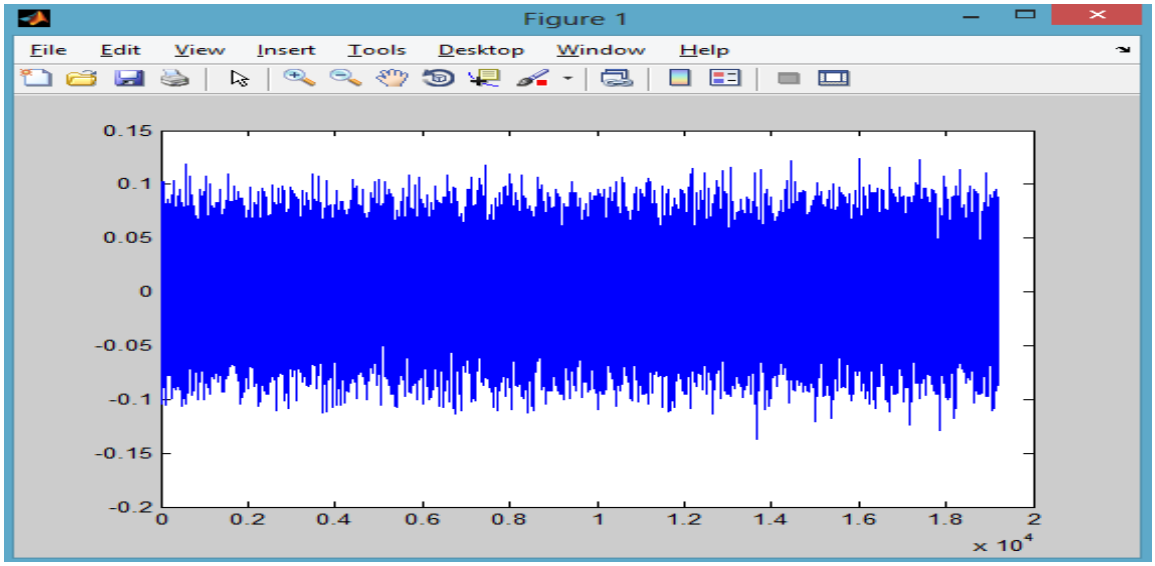


Рис. 5.17. Сигнал генерируемый генератором LTE-Uplink RMS Generator

Для вычисления потерь и пропускной способности системы передачи можно использовать блок LTE PDSCH Conformance Testing.

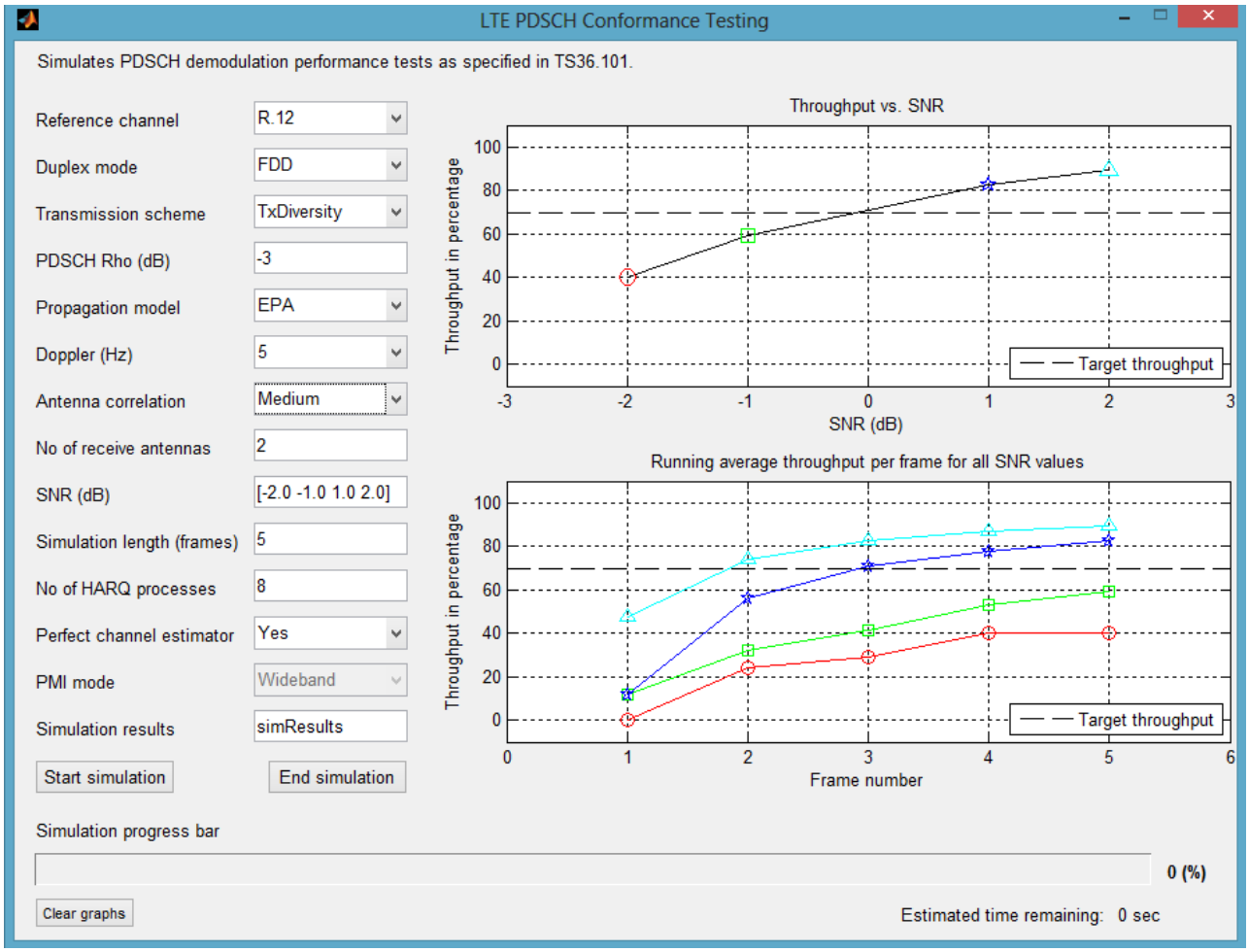


Рис. 5.18. Окно LTE PDSCH Conformance Testing

В данном окне можно произвести настройку параметров линии передачи, таких как: количество каналов, тип модуляции, доплеровскую частоту, уровень шума, настройки антенны и другие. А так же наглядно пронаблюдать изменение количества потерь в линии передачи и пропускную способность системы.

В результате проделанной работы были изучены основные понятия беспроводных сетей LTE, физическая структура построения беспроводной сети LTE. Изучена существующие методы и средства защиты беспроводных сетей LTE. Построена программная структурная схема беспроводной сети LTE среде Matlab с использование встроенного пакета LTE System Toolbox и проведены исследования основных узлов сети LTE.

ЗАКЛЮЧЕНИЕ

Рассмотрены классические шифры, теория классических шифров, компьютерный практикум для классических шифров и задания на самостоятельную работу по классическим шифрам. Преставлены современные шифры с секретным ключом, теория шифров с секретным ключом, компьютерный практикум для шифров с секретным ключом и задания на самостоятельную работу по шифрам с секретным ключо. Рассмотрены отечественные и зарубежные шифры с открытым ключом, теория шифров с открытым ключом, компьютерный практикум для шифров с открытым ключом и задания на самостоятельную работу по шифрам с открытым ключом. Представлены криптографические протоколы в сетях передачи данных и компьютерный практикум для исследования протоколов SSL и TSL. Рассмотрено шифрование в современных системах связи стандартов GSM и LTE и компьютерный практикум для исследования стандарта LTE в MATLAB.

ЛИТЕРАТУРА

1. Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. Основы криптографии – М.: Гелиос АРВ, 2001. – 480 с.
2. Метлицкий Ю.В. Разработка программного комплекса для визуализации и анализа стандарта криптографической защиты AES, МИФИ 2003 г.
3. Зензин О.С., Иванов М.А. Стандарт криптографической защиты – AES. Конечные поля. – М: КУДИЦ-ОБРАЗ, 2002. -176 с.
4. Б. Шнайер «Прикладная криптография. 2-е издание. Протоколы, алгоритмы и исходные тексты на языке С». - М.: Изд-во "Триумф", 2002. - 816 с.
5. http://www.des-crypto.ru/cryptography/rc4/
6. Асосков А.В. и др. Поточные шифры. - М.: КУДИЦ-ОБРАЗ, 2003. - 336 с.
7. http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html
8. Романец Ю.В. Защита информации в компьютерных системах и сетях / Под ред. В.Ф. Шаньгина. - 2-е изд., перераб. и доп. - М.: Радио и связь, 2001. - 376 с.
9. RSA Laboratories // http://www.rsa.com/rsalabs/node.asp?id=2009
10. Алгоритм RSA : метод. указания к выполнению лабораторных работ для студентов спец. 090105 «Комплексное обеспечение информационной безопасности автоматизированных систем» очной формы обучения / сост.: О. Н. Жданов, И. А. Лубкин ; Сиб. гос. аэрокосмич. ун-т. – Красноярск, 2007. – 38 с.
11. Diffie, D. New directions in Cryptography / D. Diffie, M.Hellman // IEEE Transactions on information theory. November. 1976.
12. Rivest, R. A Method for obtaining digital signatures and public keyCryptosystems / R. Rivest, A. Shamir, L. Adleman // Communications of the ACM. February. 1978.
13. Столингс, В. Криптография и защита сетей: принципы и практика / В. Столингс ; пер. с англ. . – 2-е изд. – М. : Изд. дом «Вильямс», 2001. – 672 с.
14. Коблиц, Н. Курс теории чисел и криптографии / Н. Коблиц ; пер. с англ. М. А. Михайловой и В. Е. Тараканова ; под. ред. А. М. Зубкова. – М. : Науч. изд-во ТВП, 2001. – 254 с.
15. Безопасность на транспортном уровне: SSL и TLS | Лекция | НОУ ИНТУИТ [Электронный ресурс]. – Режим доступа http://www.intuit.ru/studies/courses/553/409/lecture/9387 (дата обращение 13.04.2015).
16. http://matlab.ru/products/LTE-System-Toolbox/lte-system-toolbox_rus_web.pdf (дата запроса 05.04.2016)